AutoCut Image Segmentation

Evan Rosen and Nikil Viswanathan
Computer Science Department
Stanford University
{emrosen,nikil}@stanford.edu

Abstract

Image segmentation with minimal user input is still a challenging task with many practical implications. The popular GrabCut [4] algorithm approaches this task from an iterative optimization perspective using image color and contrast information to develop a system that only requires a user to draw a bounding box around the foreground. Our approach extends the state of the art GrabCut system by inspecting the image to discover the optimal number of color clusters, creating a local contrast map instead of using a global constant, and automatically learning the size of the foreground bounding box. While eliminating the need for user input and minimizing the amount of parameter tuning, we are still able to maintain competitive performance. Our system creates state of the art segmentations for fairly difficult images completely on its own.

1. Introduction

We approach the image segmentation challenge through an energy minimization method with graph cuts [2]. Building off of the GrabCut technique, we extend the algorithm with an automatic selection of the number of components in the Gaussian mixture model, a local contrast model, and automatic foreground boxing. Our goals were to minimize the amount of parameter tuning and user interaction required. We were pleased to find that our implementation achieved the state of the art results over a wide array of images.

2. Segmentation

The problem of image segmentation in computer vision is much like the task of automatic parsing in natural language processing: it provides the fundamental building blocks with which we may approach a variety of higher level visual questions such as scene classification, object recognition or depth reconstruction. While there is no single definition of what constitutes a correct segmentation, we generally expect a good segmentation to divide an image along region and object boundaries. In their normalized cuts paper [6], Shi and Malik tackle image segmentation through a graph partitioning approach. Normalized cuts account for the tendency of minimum cut to choose a single vertex on one side by scaling the value of the cut by the inverse of the total weight in each cut. This graph cut is computed through optimization by formulation as a generalized eigenvalue problem. Arbalez et al. [1] augment the global normalized cuts method with a local contour detector. A local contour model, incorporating brightness, color, and texture boundary cues serve as the edges weights used for the normalized cuts algorithm. Russel et al. [5] take yet another approach by incorporating semantic level information about feature co-occurence from large unlabeled corpora. By adapting topic models such as pLSA and LDA from the text analysis community, they are able to identify good segmentations by the fact they they consist of coherent groupings of features, or topics which have been previously observed with great regularity across the corpus. Unlike some of the methods above, *Graph Cuts* [2] makes the simplifying assumption that an image can only consist foreground and background segments. This is geared towards interactive segmentation in which a user often just wants to separate an object from the background. The graph cut method casts the segmentation problem is cast as a maximum a posteriori (MAP) estimation problem for a Markov random field (MRF). The MRF is built from a grid of segmentation indicator variables for each pixel in the image, denoted as α_{ij}. Then for each such indicator variable, a unary potential is defined which encodes the compatibility of a given segmentation assignment with the appearance information (grayscale intensity). A set of pairwise potentials are also defined over each pair of neighboring pixels, which enforce how strongly the pixels prefer to be segmented to the same region rather than separated by a segmentation boundary. The unary potentials, or appearance models, are learned on a per image basis by asking the user to brush pixels from the foreground and background. Given these pixels, a histogram over intensity values is learned for each region. The unary energy for a particular segmentation as-

signment of a pixel is then the probability associated with the histogram bin into which it falls. The pairwise potentials on the other hand are just the between-pixel intensity differences, normalized by the amount of contrast within a given image. Given these potentials, the optimal segmentation is then defined as MAP assignment to the α variables, which can be solved using an efficient min-cut/max-fliw algorithm proposed in a previous paper [3].

3. Algorithm

The GrabCut algorithm introduced in [4] extends the graph cut technique proposed in [2] through an iterative energy minimization process. GrabCut still uses the MRF formulation of the segmentation problem and employs the min-cut/max-flow algorithm of [2]. The main difference lies in the user interface, which only requires a single bounding box around the foreground region. Because this reduces the quality of the initial pixel labels used to learn the unary potentials, they introduce an iterative method which still yields final segmentations comparable to the original graph cut results of [2]. Another important difference between the GrabCut algorithm and [2] is that it uses color information rather than grayscale intensity. Thus, the unary potentials must to model distributions within colorspace. To adjust to this higher dimension, [4] use two Gaussian mixture models in the place of the histograms which avoids the exponential increase in the number of histogram bins and the associated problems of over-fitting. The switch from histograms to a mixture models also requires a slight adjustment to the unary potentials. In GrabCut, the unary energy of a segmentation assignment is the maximum probability under any of the Gaussian mixture components. The iterative component of GrabCut is key because unlike the precise brushing in [2] the bounding box will incorrectly label some background pixels as foreground. To overcome this problem, [4] repeatedly updates the unary potentials and then the segmentation, at each iteration, re-learning the appearance models from the current segmentation. Once this process converges to a stable segmentation, the user is presented with the option to manually "brush" the image and specify regions the algorithm misclassified. The incorrect pixel assignments are then enforced to be hard labelling constraints in the segmentation step

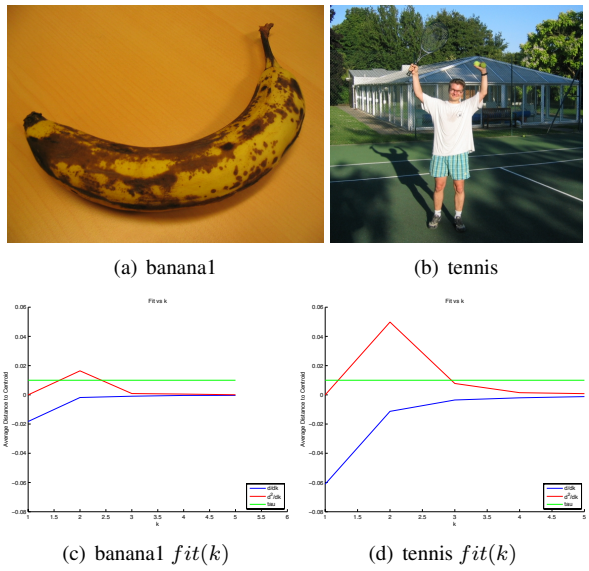
4. Our Implementation

4.1. Appearance Models

For our appearance model we used the RGB Gaussian mixture model (GMM) as described in [4]. That is, we created two $k-$mixtures of multivariate Gaussians in RGB-space: one for the foreground region and one for the background region. In addition to the actual Gaussian parameters (Σ and μ) there is also a mixture weight associated with each component of each model, $\pi_{\alpha,k}$. Intuitively each Gaussian corresponds to some object or region which resides in either the foreground or background. We initialize the parameters for each Gaussian component by running $k-$means on the color vectors for each pixel and then estimating each component from the pixels assigned to the k^{th} cluster. At each stage of the iterative algorithm, we then re-estimate these parameters according to the current assignment of pixels to regions. For each pixel z_n, assigned to region α_n, we assign it to the mixture component k which minimizes the score

$$D(z_n, \alpha_n, k) = -\log \pi_{\alpha_n,k} - \log p(z_n|\Sigma_{\alpha_n,k}, \mu_{\Sigma_n,k})$$

Then, given these new assignments of pixels to mixture components, we learn a new multivariate Gaussian using a standard ML estimation and set the new $\pi_{\alpha,k}$ to be the proportion of pixels in region α which are assigned to the k^{th} component. It is worth noting that these mixture weights play an important role at segmentation time because both the foreground and background GMMs will likely contain components which describe the same colors. This can happen due to the fact that the foreground bounding box often includes some of the background. Though each GMM may learn similar components for those colors, the mixture weights can help push ambiguous pixels to the correct segmentation by recognizing that those colors show up significantly more often in a particular region. We found that the quality of our results depended heavily upon the number of components, k, in each GMM. In simple images such as "banana1", for example, the use of too many mixture components led to mistakenly treating the portion of the background region included in the initial foreground bounding box as foreground. We hypothesize that this was the result of effectively allocating a specific mixture component to the unique color of the table which was initially labelled as foreground by inclusion in user drawn the bounding box. Though this part of the table was very similar in color to the parts of the table initially labelled as background, there always remained a dedicated mixture component in the foreground model, which was not needed to describe the banana. To remedy this problem we experimented with a variety of different automatic methods for choosing k. We first ran k-means clustering the pixels in a down-sampled version of the image for values of $k \in [2, 6]$. Due to the sensitivity of k-means to initialization conditions, we repeat this 3 times for each value of k and use the mean for each value of k. We then use the average distance to centroids for given value of k as an indicator of goodness of fit, which we denote as $fit(k)$. Our general aim is to find the value of k which corresponds to a significant drop off in $fit(k)$. Our first attempt used the value of k below which we observed largest change in average distance to centroid, $\max_k\{\frac{d}{dk} fit(k)\} + 1$. Interestingly, this almost always



(a) banana1 (b) tennis

(c) banana1 $fit(k)$ (d) tennis $fit(k)$

Figure 1. Plots of $\frac{d}{dk}fit(k)$, $\frac{d^2}{dk}fit(k)$ and τ for banana1 and bool. This led to the choice of $K = 3$ based on the threshold metric.

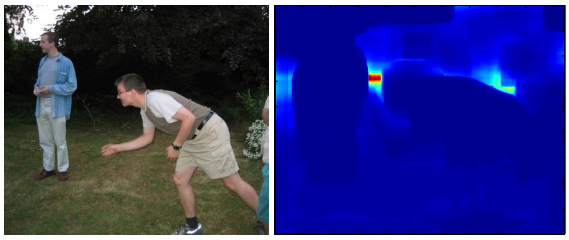
led to setting $k = 2$ even for images which clearly contained several distinctly colored objects (see Figures 1(b) and 1(d)). We then realized that we need not take the value of k associated with maximum change in $fit(k)$ but perhaps any value of k below which there was a drop in fit larger than some threshold. This cannot be directly implemented however due to the fact that the color distributions of some images are just inherently harder to model than others. This means that the first $\frac{d}{dk}fit(k)$ might be difficult to compare between images. Nonetheless, the plots of $\frac{d}{dk}fit(k)$ in Figures 1(c) and 1(d) do seem to exhibit a useful difference in shape, where the plot of $\frac{d}{dk}fit(k)$ in the tennis image begins to drop in a meaningful way between $k = 3$ and $k = 2$. To capture this difference, we realized that we could use the second derivative of $fit(k)$ to pick out the point at which the fit began to drop off, scaled to the rate of change with respect to a given image. Thus, we finally used a hand set a threshold τ on the second derivative so that we chose the largest value of k for which $\frac{d^2}{dk}fit(k) > \tau$. We found values of $\tau \approx 1.0\text{E}-2$ worked well for assigning a range of values to k for images of varying color complexity. An example plot of $fit(k)$ for banana2 is shown in Figure 1. We also experimented with color space and transformed the RGB images into HSV (hue, saturation and value) space in the hope that the color distributions for a given region cluster more naturally if the saturation and intensity are explicitly factored out. This might help deal with shadow lines which presumably mostly affect saturation and intensity. Interestingly we had to refit our goodness of fit threshold τ because the HSV image data had vastly different variance, despite the fact that the parameters were all scaled to be within the same range $[0, 1]$. Unfortunately, this had no strong effect, and only demonstrated that the algorithm could produce comparable results in a different color space. We have omitted a demonstration of nearly identically segmentation for the sake of space.

4.2. Smoothness Constraints

The second key component in the GrabCut model is the presence of a smoothness constraint which enforces that neighboring pixels tend to have similar α values, thus restricting the total number of edge cuts. In the min cut framework, this corresponds to placing a pairwise potential on adjacent pixels which is 0 when they are assigned the same α value and some positive value in the case they are assigned different α values. While a constant edge weight would accomplish the general intuition, it can be refined by using appearance information to pick out which pairs of adjacent pixels are more likely to be separated by a true segmentation boundary. By setting the pairwise penalty be the negative Euclidean distance in RGB-space between two pixels, we can enforce that the segmentations tend to fall along color discontinuities; that is, the more distant the color values, the smaller the pairwise potential, and the more appealing an edge is to be part of the cut set. This model assumes that all color differences of a given value provide equal evidence for a segmentation boundary. This is problematic however, for images which contain highly textured regions along with relatively untextured regions. For example, the small-scale discontinuities between sun and shade within a grass region should not be treated the same as large color discontinuities separating the sky from the horizon. To encode this intuition, similar to [4], we normalize the edge penalties by a region-specific measure of the amount of variation within a $p \times p$ square surrounding the edge in question, $\beta(u, v)$. This then makes pairwise penalties for edges in high constrast regions larger (less negative) and the pairwise penalties for edges in low contrast regions smaller (more negative). We construct β by convolving each channel of the image against 8 different 3×3 filters with +1 in the center and -1 in each of the other 8 positions. We then convolve the sum of squares of these 8 convolutions with a $p \times p$ mask where each cell has value $1/p^2$, to efficiently compute the amount of variation surrounding each pixel. Then, for a given edge we normalize the pairwise penalty by the average of these values for each pixel in the pair. This yields an $m \times n$ map of β values like the one shown in Figure 2 for which

$$C_{x,y} = \frac{1}{p^2} \sum_{i,j \in P} \sum_{r=1}^{8} (d_r(x+i, y+j))^2$$

where $d_r(x, y)$ is the squared euclidean distance between the color value at pixel (x, y) and one of the eight possible immediate neighbor pixels indexed by r. The normalization



(a) bool (b) β map

Figure 2. Example map of β values for the bool image. Note that the β value is larger for the grass and tree regions in which there is greater between-pixel contrast. The highest values occur for the points whose surrounding region includes the boundaries of both people and the background.

term for each edge is then computed by simply taking the average of the C values for both pixels so that

$$\beta_{uv} = \frac{1}{2}(C_{x_u,y_u} + C_{x_v,y_v})$$

4.3. Implementation Details

The main file in our implementation is grabcut.m which, given an image name, asks the user for a bounding box and then computes the GrabCut algorithm [4]. In a slight departure from the terminology of [4], the variable `tmap` holds the α values for each pixel. We also keep a map of the pixels originally set to be background by the user in the variable `bbox_map`. We iterate between updating the appearance model and the segmentation until the change in the flow, or cut value, between iterations is less than `1E2` or we have reached a the maximum number of iterations (40). In order to enforce the hard constraint that user labeled background pixels be segmented correctly, we tried several approaches. We first ran a max-flow/min-cut [1] [2] algorithm on the entire image and then only updated the α values for those pixels which were within the bounding box, keeping the remaining pixels set to background. This had several undesirable effects, however. For one, it allowed edges to cross the bounding box, which should not be allowed. Moreover, it ignores the fact that any pixel on the edge of the bounding box is necessarily going be adjacent to a background pixel, and should be segmented accordingly. By letting all of the pixels be segmented according to their model-based unary and pairwise weights it allows pixels outside of the bounding box to be segmented as foreground, which can then propagate across the bounding box and encourage pixels on the edge of the bounding box to be segmented foreground. To remedy this, we simply set the appearance model weights for the pixels outside of the bounding box to be a large positive number for $\alpha = 1$ and a large negative number for $\alpha = 0$. This reliably enforces hard constraints on the segmentation because the smoothness constraints never outweigh these very large appearance model weights. One downside of this approach is the significant runtime cost associated with placing very high appearance weights on some pixels. Though simple fix for this might be to set the unary weights for pixels only on the edge bounding box, we opted to keep the stick with the less efficient approach because it was more compatible with our user editing steps described in Section 4.4. The file `evaluate.m` evaluates our model on the entire data set. We created a file called `names_GT` which stores the image names and files extension. The evaluation code then loads the image and ground truth segmentation and runs `grabcut.m` for each file, reporting the accuracy, precision and recall. To allow for overnight runs without user interaction, we also created a bounding box cache scheme where each bounding box is saved to a directory called `bbox_GT/` which if present, is loaded by `evaluate.m` and passed into `grabcut.m` which skips the user prompt to draw a bounding box.

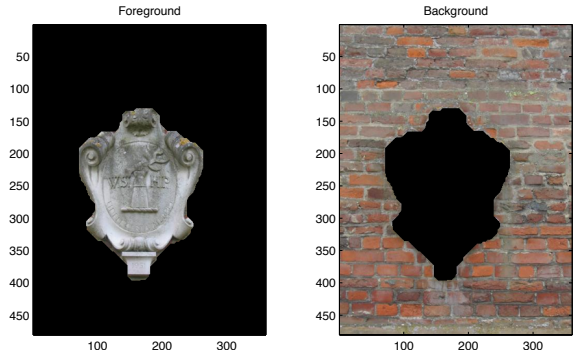
[1] We modified the Boykov min-cut/max-flow c++ implementation from http://vision.csd.uwo.ca/code/

[2] We also used a matlab min-cut/max-flow mex file wrapper from http://www.mathworks.com/matlabcentral/fileexchange/21310-maxflow

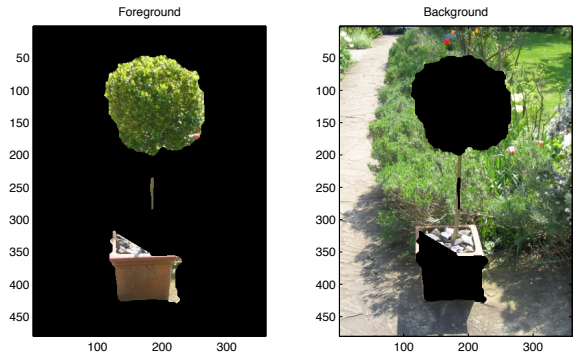
4.4. User Editing

We chose to extend the basic GrabCut [4] algorithm with the option to update the segmentation after the the iterative re-estimation has converged. We allow the user to specify a region of pixels which have been mislabeled by drawing an arbitrary number of enclosed freeform shapes. Any pixels within these shapes are then forced to have the label opposite that of the current segmentation. This is done using the same technique described for enforcing that pixels outside the bounding box have $\alpha = 0$. The final segmentation is thus computed with the original appearance models but with the added constraint that certain pixels have fixed α values. A successful example of user editing can be seen in Figure 3(c). This method has several limitations worth noting. For one, it can really only update regions within the bounding box, because even though it allows the user to change the α values for the selected pixels which may reside in the initial background, it cannot propagate this change to the neighboring pixels which are have fixed α values of 0. One solution would be to remove the hard constraint on background segmentation for all the pixels in the region surrounding the user edit. However, this can be easily avoided by simply making the bounding boxes large enough in the first place. Another limitation of this approach is that it does not update the appearance model according to the user edit, but simply fixes the α values for some pixels by overwriting their unary weights and leaves all other unary weights unchanged. This means that in cases where one of the appearance models has learned to describe pixels in the other region with high

(a) memorial user interaction (b) bush user interaction



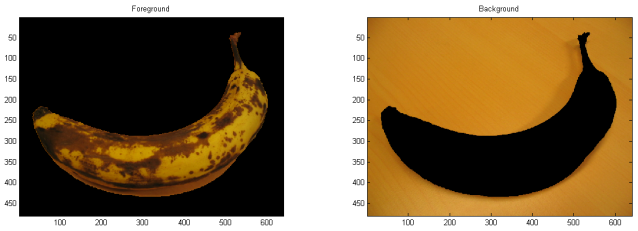
(c) memorial results



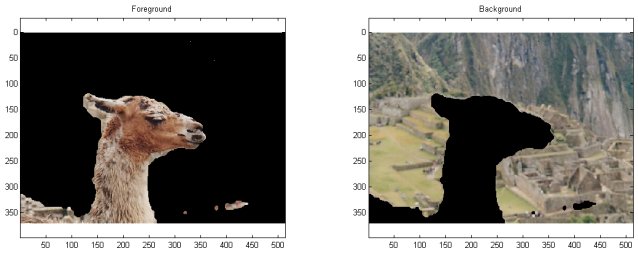
(d) bush results

Figure 3. User brushing results and interface. 6 and 3(b) demonstrate our user "brushing" interface which allows the user to update both incorrectly segmented foreground and background pixels.

confidence, it will continue to mislabel any of the pixels not explicit marked by the user as incorrect. Because the pixels not marked by the user still have a low weight (high probability) according for the wrong appearance model, the forced α assignment cannot always overcome strong unary weights via the smoothness penalties. This can be seen in Figure 3(d) in which, though we mark a large number of pixels on the top of the planter as mistakenly assigned to the background, these changes do not propagate to other



(a) Banana1



(b) Llama

Figure 4. Example of segmentations using the automatic bounding box with aspect ratio orientation

mislabeled pixels on the top of the planter. In the current theory, this is because they still receive a high probability from the background appearance model which recognizes the strong similarity between the bright white path (which is entirely in the background region) and the sunlit portion of the planter.

4.5. Auto Bounding Box

A core feature of the GrabCut algorithm is the reduced amount of user input required in comparison to contemporary segmentation methods. We take this one step further and attempt to eliminate the need for user input completely. Our initial approach set a 10%-20% region of padding on the inside of the image border to use as the bounding box. While this idea came from viewing the our dataset, the task of image segmentation in general applies to images in which a foreground object is surrounded by background. An image without background would not need to be segmented. Thus the foreground object must be a subset of the image and most likely a portion of the foreground will be inside the central 80%-90% of the image. One consequence of this technique is that since this bounding box is approximate we can no longer enforce the hard constraint that no pixel outside of the bounding box is labeled as foreground thus this initial segmentation is only used to initialize the GMMs. Deciding that learning from our data would be a better approach than human inference, we then estimated the bounding box by taking the mean of the human selected bounding boxes for the dataset. Inspecting the results for the mean bounding box coordinates over all images (as can be seen in Table 5), were surprised to see that the bounding

Mean Bounding Box Dimensions (%)				
Dimension	x Min	y Min	x Max	y Max
Average	25.2	16.9	78.6	88.4
Wide	23.0	15.5	83.3	92.0
Tall	29.0	19.3	70.3	82.2

Figure 5. Mean bounding box coordinates in % of width and height. Average uses all images. Wide is bounding box mean over images that have a greater width than height and tall vice versa.

was inset so far into the image (as can be seen from Table 5, the mean lefthand x coordinate is 25.2% of the width in). Drilling down into the data, we postulated that wide versus tall images have different mean bounding box dimensions and to explore this we calculated the mean with respect to aspect ratio orientation. As can be seen (Wide and Tall rows of Table 5), this is definitely true and inspecting the results led us to the realization that vertical pictures tend to be taken of objects that are tall and skinny and thus these objects are likely to be closer to the center of the image and vice versa for horizontal pictures. Using this aspect ratio model, we estimated the bounding box a test image by calculating the mean bounding box over all of the other images with the same aspect ratio orientation. The auto bounding box segmentation performed surprisingly well even given the tiny training dataset (30 images in total). Banana1 in Figure 4(a) was segmented as successfully as with a human bounding box, and with llama the new technique actually discovered the piece of the llama llama's back (Figure 4(b)) that was mislabeled by both our human generated ground truth and the Grabcut [4] paper.

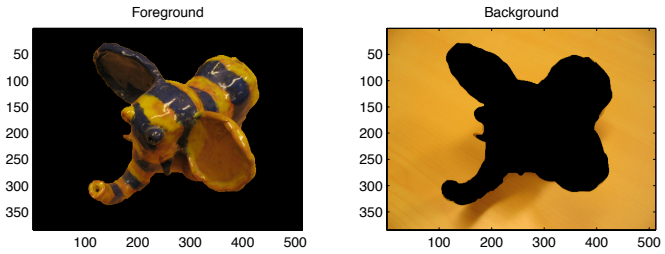
5. Results

Overall we obtained good segmentation on most of the images in the dataset. Upon inspection of the images that our algorithm did not label perfectly, we discovered that in many cases, we did not believe that the ground truth was a correct segmentation. In images, bool and person5, for example, only one of the two people in each image is chosen for the foreground. The average statistics on the 20 test images are given below and a complete table of our results is given in Appendix A.

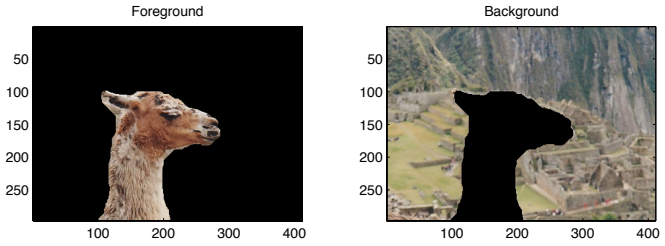
Results over 30 dataset images		
Avg. Acc	Avg. Pr	Avg. Rec
0.954	0.867	0.926

6. Conclusion

The Grabcut algorithm presents a powerful method for segmenting images that requiring minimal user input. Our system uses the core features of the Grabcut algorithm and extends the functionality by reducing the amount of parameter tuning and eliminating user input. In order to achieve this we have created algorithms that attempt to auto tune parameters with respect to a given image and emulate user behavior by examining user interactions on other images. We have found that these extensions maintain the Grabcut performance for many images and believe that with an increased dataset size we can further enhance robustness of our system.



(a) ceramic



(b) llama

Figure 6. Results With User Bounding Box

extends the functionality by reducing the amount of parameter tuning and eliminating user input. In order to achieve this we have created algorithms that attempt to auto tune parameters with respect to a given image and emulate user behavior by examining user interactions on other images. We have found that these extensions maintain the Grabcut performance for many images and believe that with an increased dataset size we can further enhance robustness of our system.

7. Conclusion

The Grabcut algorithm presents a powerful method for segmenting images that requiring minimal user input. Our system uses the core features of the Grabcut algorithm and extends the functionality by reducing the amount of parameter tuning and eliminating user input. In order to achieve this we have created algorithms that attempt to auto tune parameters with respect to a given image and emulate user behavior by examining user interactions on other images. We have found that these extensions maintain the Grabcut performance for many images and believe that with an increased dataset size we can further enhance robustness of our system.

Appendix

A. Full Results

Image Name	Acc	Pr	Rec
banana1	0.98	0.95	0.98
banana2	0.99	0.97	0.98
banana3	0.99	0.99	0.97
book	0.96	0.91	0.99
bool	0.81	0.41	0.84
bush	0.95	0.92	0.80
ceramic	0.88	0.99	0.63
cross	0.68	0.59	0.44
doll	0.99	0.99	0.99
elefant	0.96	0.89	0.98
flower	1.00	0.99	1.00
fullmoon	0.98	0.78	1.00
grave	0.96	0.78	0.96
llama	0.99	0.95	0.98
memorial	0.98	0.94	0.98
music	0.98	0.96	0.99
person1	0.99	1.00	0.97
person2	0.98	0.90	0.97
person3	0.99	0.96	0.99
person4	0.91	0.70	0.95
person5	0.94	0.54	1.00
person6	0.98	0.98	0.82
person7	0.99	0.98	0.89
person8	0.99	0.98	0.95
scissors	0.87	0.47	0.87
sheep	1.00	1.00	0.95
stone1	0.98	0.93	0.99
stone2	1.00	1.00	0.99
teddy	0.99	0.95	1.00
tennis	0.94	0.60	0.94
Mean	0.954	0.867	0.926

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. Technical Report UCB/EECS-2010-17, EECS Department, University of California, Berkeley, Feb 2010.

[2] Y. Boykov and M. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in ND images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112. IEEE, 2001.

[3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26:1124–1137, September 2004.

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

[5] B. C. Russell, A. A. Efros, J. Sivic, W. T. Freeman, and A. Zisserman. Using multiple segmentations to discover objects and their extent in image collections. In *Proceedings of CVPR*, June 2006.

[6] J. Shi and J. Malik. Normalized cuts and image segmentation. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 731–, Washington, DC, USA, 1997. IEEE Computer Society.