IMAGE FOREGROUND/BACKGROUND SEGMENTATION USING MAX FLOW ALGORITHM

Ayush Shirsat & Sarthak Arora

Dec. 13, 2019

Boston University

Department of Electrical and Computer Engineering

BOSTON UNIVERSITY

IMAGE FOREGROUND/BACKGROUND SEGMENTATION USING MAX FLOW ALGORITHM

Ayush Shirsat & Sarthak Arora

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

Dec. 13, 2019

Contents

List of Figures

1 Overview

Image segmentation is a very common topic in computer vision in which an object is separated/segmented from the background. Generally, similar pixels are clustered together which helps to differentiate the foreground from the background. In this project, a user uses an interactive command line interface (CLI) to draw lines on foreground and background. The foreground is an object the user wishes to segment out.

All pixels in an image act as vertices of a graph and they are connected with their neighboring vertices using a weighted edge. The weights of the edges are determined by a boundary penalty formula and the log likelihood ratio of the probabilities given by the Gaussian Mixture Models. Once the graph is set, Max-flow algorithm is used to create a min-cut between the foreground and background pixels. The area within the cut is the foreground which is segmented out and displayed on a white background.

Edmond-Karp algorithm, which is an implementation of Ford Fulkerson method, is used to segment out the object. The time complexity of this algorithm is $O(VE^2)$. Once graph-cut is performed, some image processing techniques are used for segmentation.

2 Dataset

Dataset used for this project is Berkeley Segmentation Dataset [3]. It is designed specifically for testing segmentation tasks and data is labelled incase Machine Learning has to be used. All images are in RGB format so we converted them to grayscale to adhere to the scope of the project.

3 Methodology

3.1 Command Line Interface

Here the user marks the foreground pixels first on the image and then presses the ‘b’ key to mark the background ones. Once all markings are done, ‘q’ is pressed to stop marking.

As per our implementation, user marked pixels are always in a straight line. The code picks up initial and final coordinates of the points marked by the user and a straight line is plotted to connect these two points. All pixels belonging to the line are stored in format of {x coordinate, y coordinate, pixel intensity}. The image with all markings is called the seeded image and is saved.

Fig. 1 demonstrates the original image and interactive window for user. Fig. 2 shows the user markings or seeded image.

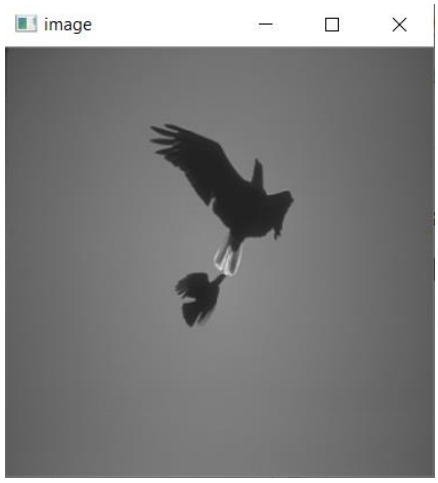

Fig. 1: Interactive interface for user

Fig. 2: Seeded image

3.2 Graph Setup

Each pixel is treated as a node in the graph. The weights of the edges in the graph are assigned using a boundary penalty formula and the log likelihood ratio of the probabilities returned by the GMM model, which is explained further in the next subsection.

For max flow problems, we have two specific nodes called the source and the sink. These can also be called terminals. For our graph setup, besides the existing pixels in the image, we need two separate nodes in our graph to serve as the source and the sink.

Now we have all the nodes. Each node or pixel shall have two types of edges:

1. Neighborhood edges
2. Terminal Edges

The Neighborhood edges suggest the neighboring pixels we are looking at for the pixel currently at hand. In our implementation we only look at 2 neighbors for any pixel. One of these neighbors is on the immediate right and the other one is right below the pixel, whose neighbors are currently being considered.

The terminal edges suggest the connection of each pixel with either the source terminal or the background terminal.

Figure 3 provides a reference for the edge weights in this graph that we have built. This table[1], refers to neighborhood edge weights as Boundary properties, which is denoted by B_{pq}. The ‘p’ subscript denotes the pixel or node being looked at. The ‘q’ subscript denotes the neighbor of the pixel ‘p’. The boundary properties are set using the formula as shown below:

$$B_{\{p,q\}} \propto exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right)$$

I_p and I_q denote the pixel intensities of the p and q pixels respectively. The B_{pq} formula penalizes similar pixel intensities and therefore a capped weight is obtained at the edge where we have a boundary when looking at the neighbors of the pixels.

The table[1], refers to terminal edges as Region properties, denoted by R_p. These properties are determined by the following formula:

$$Log\ Likelihood\ Ratio\ (llr_p) = log(\mathcal{K} \cdot\ p_f/p_b)$$

Here K is a constant, whereas p_f and p_b are the probabilities of the pixel belonging to the foreground and the background respectively. If $p_f > p_b$, the weight is assigned to the pixel with an edge from the Source. If $p_b > p_f$, the weight is assigned with the llr_p formula having the ratio of the probabilities as flipped, i.e. p_b/p_f. This weight is given to the edge connecting the pixel p to the Sink.

edge	weight (cost)	for
$\{p, q\}$	$B_{\{p,q\}}$	$\{p, q\} \in \mathcal{N}$
$\{p, S\}$	$\lambda \cdot R_p(\text{“bkg”})$	$p \in \mathcal{P},\ p \notin \mathcal{O} \cup \mathcal{B}$
	K	$p \in \mathcal{O}$
	0	$p \in \mathcal{B}$
$\{p, T\}$	$\lambda \cdot R_p(\text{“obj”})$	$p \in \mathcal{P},\ p \notin \mathcal{O} \cup \mathcal{B}$
	0	$p \in \mathcal{O}$
	K	$p \in \mathcal{B}$

Fig. 3 Reference table for edge weights

3.3 Gaussian Mixture Model

All the pixels selected by the user are used to train a Gaussian Mixture Model (GMM). Here foreground pixels are labelled as 1 and background pixels as 0. The trained GMM model is then used to predict probabilities of the remaining unmarked pixels. The sklearn library was used to run the Gaussian Mixture Model with 2 components(parameter for segmentation into foreground and background) over the labelled data to return two predicted probabilities for each pixel. One of them being the probability of the pixel belonging to the background(p_b) and the other being the probability of the pixel belonging to the foreground(p_f). Both these probabilities for each pixel sum up to 1.

With the weights as found above, the graph is now ready for min cut/max flow.

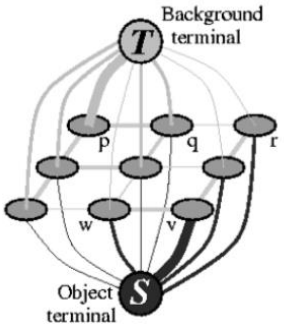

Fig. 4 After the graph setup

In our code implementation, we have two files:

1. main.py
2. mainGMM.py

In main.py the terminal edge weights are constant and only assigned to the seeded pixels. In mainGMM.py the terminal edge weights are constant for the seeded pixels but for the unlabeled pixels we have edges based on the log likelihood ratio of the probabilities.

3.4 Edmond Karp Algorithm

We run the implementation of Ford Fulkerson method which is called the Edmonds Karp algorithm. This algorithm finds the augmenting paths in the residual graph using Breadth First Search. Iteratively, we find augmenting paths and increase the flow through the graph. Once there are no more augmenting paths to be found, we can conclude the graph is ready to be cut. The cuts are those edges in the graph with the least amount of weight.

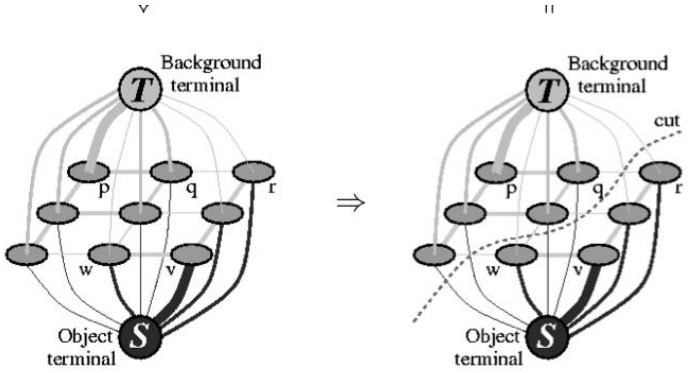

Fig. 5: Max flow

3.5 Segmentation

Graph Cut gives us the boundary around the object. This boundary is currently on an image of size 30x30. We need the entire object to be enclosed so as to create a mask. The cut is plotted on a binary image with the boundary being in white color and background in black. The following steps are implemented to create the mask:

1. Perform dilation to increase the cut size. This is done so that there is no space between the cut and we get a uniform boundary.
2. The area within the boundary is filled using convex polygon fill function of OpenCV.
3. Erosion is performed so that the mask object shrinks down to its original size.

The kernel size for dilation is slightly higher than that of erosion to ensure we do not erode the region enclosed by the cut. Once the mask is created it is resized to original size of 300x300. A bitwise AND operation is performed between the mask and original image which segments out the foreground from the background. The foreground is then put on top of a white background.

4 Results

4.1 Parameters

- Image size: 30 x 30
- Graph Nodes: 902 x 902 (+2 for Source & Sink)
- Graph Edges: N-edges + T-edges = 1740 + 900 = 2640
  (All pixels Source or Sink)
- Time Taken: 3 min 43 seconds

4.2 Segmentation results

We were able to get the segmented image and put it on a white background. While performing graph cut it was observed that the cut may or may not be closed. If the boundary was closed segmenting out the object was easy just by filling the region enclosed by the cut.

Fig. 5 shows the cut along the object. Here, it is a closed cut and object is well enclosed by the boundary. Fig. 6 shows the mask and the segmented image.

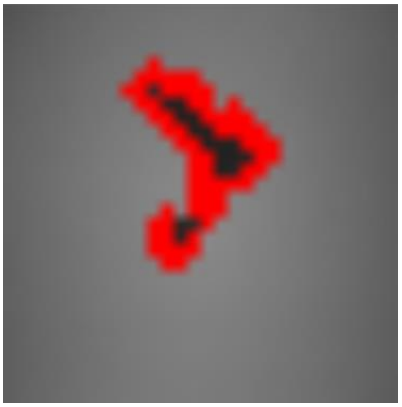

Fig. 6: Graph Cut (closed boundary)

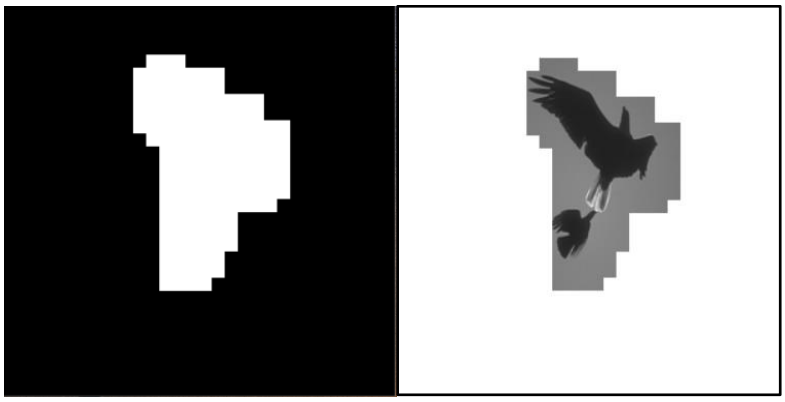

Fig. 7: Mask and Segmentation

If the boundary is not closed then some morphological operations need to be performed. As discussed in the methodology, we perform dilation and erosion. This operation was eventually used on all images, whether boundary is closed or not.

Fig. 7 demonstrates graph cut which is not closed. Fig. 8 shows the mask and segmented image. Segmentation can be improved by marking more foreground pixels. However, the general aim is to segment out the object in least possible user markings.

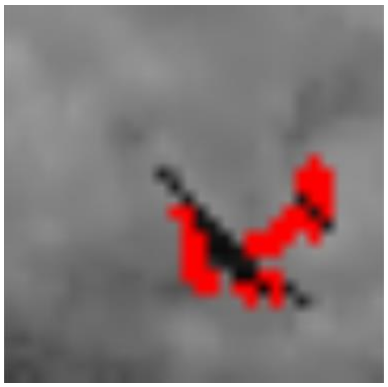

Fig. 8: Graph cut (boundary not closed)

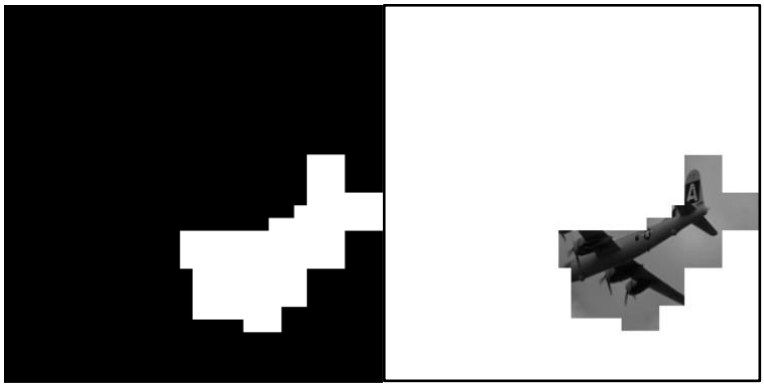

Fig. 9: mask and Segmented Image

Difficulties faced:

- If the graph cut is not closed then the segmentation is not very clean either and some important parts of an object are not segmented.
- The downscaling and upscaling while creating mask make the mask pixelated and part of background that is not required is also segmented.
- Repeating the results majorly depends on the seeds chosen. Hence seeds could be saved and loaded for better debugging.

5 Conclusion

- Interactive command line interface was implemented for the user to mark foreground and background pixels.
- Graph weights were initialized using:
  - Terminal Edge Weights: Gaussian Mixture model or constant.
  - Neighborhood Edge Weights: Boundary penalty
- Graph cut can be performed on any size image. As we resize it to 30*30 for computational purposes.
- As discussed during our presentation, our implementation of graphs was not efficient as we looked at the entire Adjacency matrix, which for even 30*30 images amounted to 902*902 size. In future, Use adjacency list to reduce the computation time.
- We worked with 2 neighbors (Right and bottom pixels). Look at more neighbors to improve segmentation results
- GMM was used for probabilities. K-Means can be used too to group pixels in foreground and background clusters
- For max flow we used Edmonds Karp. Boykov Algorithm to be attempted
- Worked on Grayscale Images. In future move towards Colored Images.

6 Work Breakdown

Ayush Shirsat: CLI, image processing and Edmonds Karp algorithm

Sarthak Arora: GMM, Edmonds Karp algorithm and image processing

References

[1] Boykov, Yuri Y., and M-P. Jolly. "Interactive graph cuts for optimal boundary & region segmentation of objects in ND images." *Proceedings eighth IEEE international conference on computer vision. ICCV 2001*. Vol. 1. IEEE, 2001.

[2] Kulkarni, Mayuresh, and Fred Nicolls. "Interactive image segmentation using graph cuts." *Pattern Recognition Association of South Africa* (2009): 99-104.

[3] Martin, David, et al. "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics." Vancouver:: Iccv, 2001.

[4] https://julie-jiang.github.io/image-segmentation/#algos