Natalia Frumkin

EC504 Project

12/13/2019

A Survey of Segmentation Algorithms

Problem Statement

Image Segmentation is a topic in computer vision with many applications, from image augmentation to object classification to medical imaging. The problem of accurate labelling and grouping of pixels can be used for a variety of tasks. The objective of this project is to separate the foreground and background of an image. To do this, I am using two techniques for image segmentation: automatic thresholding and max-flow algorithms.

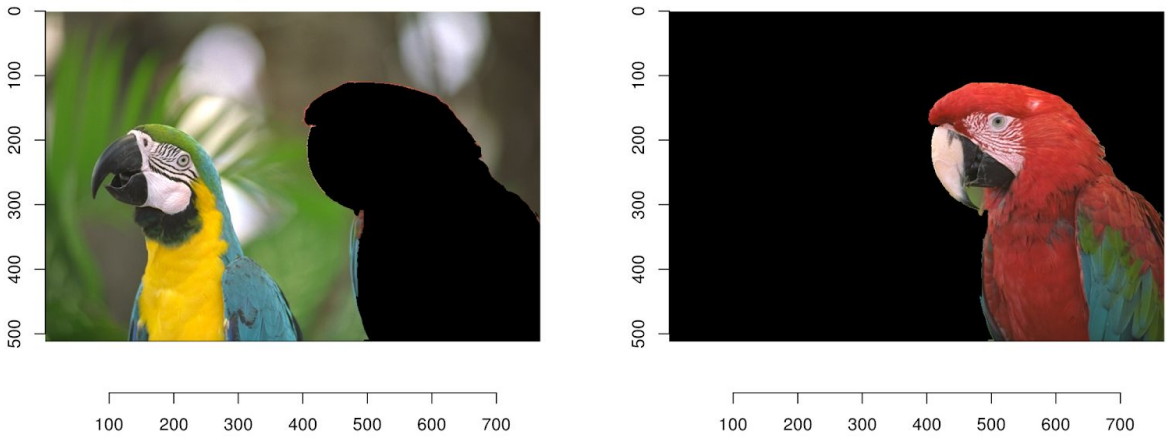

Figure 1: An example of foreground/background image segmentation. source: https://dahtah.github.io/imager/foreground_background.html

Automatic Thresholding

Image Thresholding is effective for 2-class image segmentation, because it simply entails finding a suitable decision boundary between different parts of the image. I used Otsu's method for automatic image thresholding. The objective for Otsu's method is defined as:

$$\sigma_w^2(t) = \omega_0(t) \cdot \sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

where: t is the threshold, and σ_0 and σ_1 correspond to the variances of class 0 and class 1 respectively. The goal in Otsu's method is to minimize the intra-class variance as defined above. The weights $\omega_0(t)$ and $\omega_1(t)$ are defined by class probabilities taken from a histogram binning.

The [wikipedia page] provides a well-defined algorithm to compute this. I used the built-in implementation of Otsu's method in Python3's Scikit-Image package.

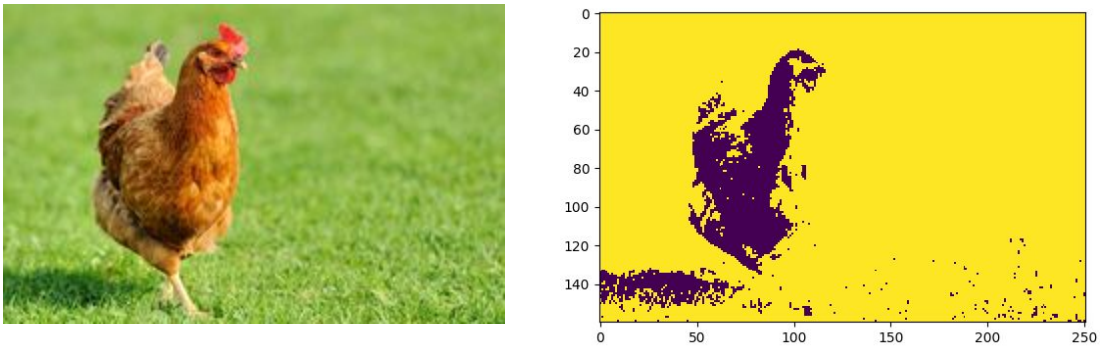

Figure 2: Thresholding a picture of a chicken using Otsu's method. Original image (left) and Segmented pixels (right).

Max-Flow Algorithms

The classical foreground-background image segmentors use a class of graph algorithms called max-flow algorithms. The maximum flow problem involves finding the path and value of the maximum feasible flow in a flow network with respect to certain constraints. We formulate this algorithm in terms of graphs by naming a source node and a sink node in a directed graph, and finding the path of maximum flow from the source to the sink. The maximum flow problem is equivalent to finding the minimum cut severing flow between the source and the sink. In image segmentation, this minimum cut is equivalent to the segmentation of the image into a foreground and background.

Using max-flow algorithms for image segmentation consists of four stages:

1. Image Annotation
2. Constructing a graph with appropriate inter-pixel and source/sink weights
3. Applying a max-flow algorithm on this graph
4. Identifying the connected components of the residual graph

Image Annotation

The image segmentation problem can be formulated using an unsupervised algorithm, such as thresholding, where the components are not explicitly specified by the user. It can also

be solved using a semi-supervised approach, where one annotates and image to identify foreground and background components. Image annotation is important with max-flow based algorithms because we can assign labels and appropriate probability distributions based on pixels belonging to the background and those belonging to the foreground. For my project, I used a graphics API called Tkinter for Python3. I designed a basic pop-up window with appropriate annotation buttons on the top panel. From there, the chosen image is displayed along with a painting overlay. The painting overlay is a component of Tkinter and serves as a mechanism for the user to annotate the foreground as green and the background as red. These annotations are then transformed into relative coordinates to the image, and stored pickle files for the graph initialization component.

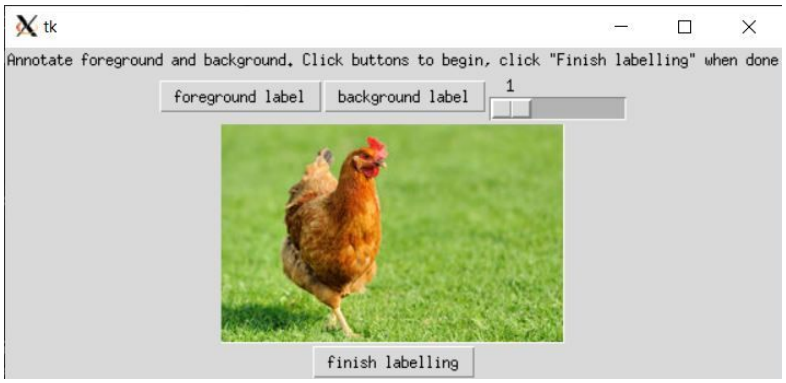

Figure 3: The Graphical User Interface (GUI) for image annotation. This GUI includes a foreground/background label choice, a sliding bar for drawing width, and a button to save the current annotation state.

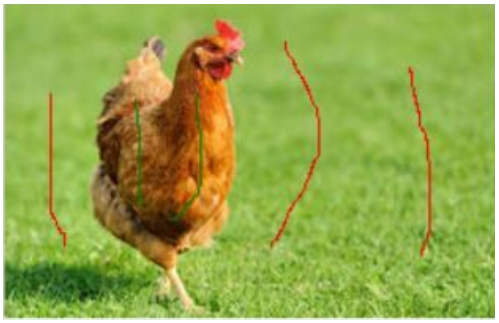

Figure 4: An example annotation for the chicken photo. The green is represents the foreground and the red represents the background as defined by the user.

Initializing graph

In order to apply a max-flow algorithm to our image, we must specify a graph topology along with corresponding weights. According to Boykov and Kolmogorov's paper on Max-Flow algorithms for vision, the max-flow graph initialization for image segmentation is defined in the following manner. We define a graph node corresponding to each pixel in the graph and attach edges to the neighboring nodes in the horizontal and vertical direction. The weights on these edges are defined by an inter-pixel distribution based on relative intensity. Two additional nodes corresponding to the source and the sink are included in the graph, and attached to each of the pixel nodes. Each pixel node should have exactly one edge from a source and one edge to a sink. The edges from/to the source/sink are defined using weights based on individual intensities. The source/sink edge are defined by the foreground/background annotations defined by the user. The graph described above is visualized in Figure 5.

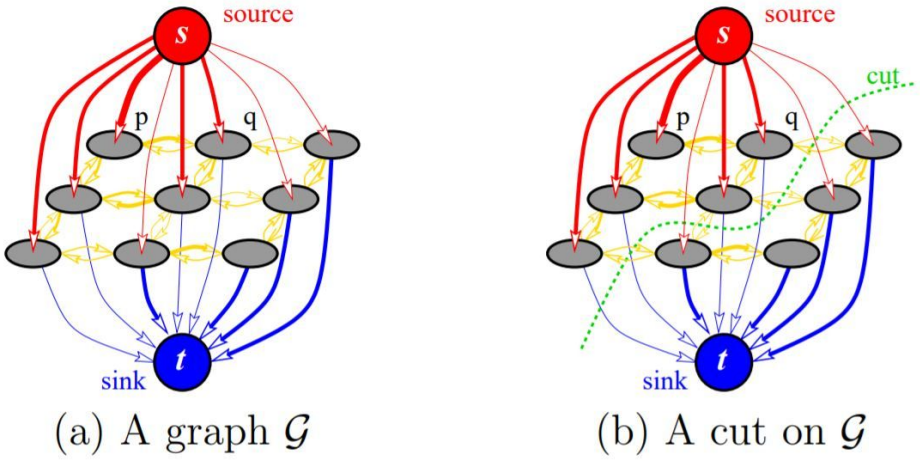

Figure 5: Graph setup for max-flow. source: https://pub.ist.ac.at/~vnk/papers/BK-PAMI04.pdf

To implement the graph appropriate weights, I relied heavily on an interactive graph-cut segmentation tutorial. I chose a closed-form method to solve for the underlying source/sink weight distributions and the standard interpixel distribution formulation as described.

Ford Fulkerson Algorithm

I implemented the standard Ford Fulkerson algorithm in Python using a classic Node class. While the Ford Fulkerson algorithm passed basic tests for a sample graph, it never converged for the image segmentation graph it was given.

Boykov's Algorithm

I used an implementation of Boykov's algorithm in Python's max-flow API package.

Results

Otsu's Method

complexity: $O(\max(N_{pixels}, N_{bins} \cdot N_{bins}))$

real runtime: 0.0017 sec

Ford-Fulkerson Method

complexity: $O(V \cdot E^2)$

Boykov-Kolmogorov Method

complexity: $O(V^2 \cdot E \cdot |C|)$

real runtime: 10.6+0.00088 sec

Conclusion

While the Boykov-Kolmogorov algorithm took a significant time to run versus Otsu's method, it is important to note that Otsu's method is implemented using Scipy, a Python3 API, which is most likely highly optimized for best performance. The algorithms in C++ may run much faster than their Python counterparts, and as a result, it is difficult to accurately compare the runtime between Otsu thresholding and the Max-Flow algorithms implemented here.

References

https://en.wikipedia.org/wiki/Otsu%27s_method

https://en.wikipedia.org/wiki/Maximum_flow_problem

https://www.datasciencecentral.com/profiles/blogs/interactive-image-segmentation-with-graph-cut-in-python

https://pub.ist.ac.at/~vnk/papers/BK-PAMI04.pdf