MIS 381N

Stochastic Control and Optimization: Project 4

Introduction

By interpreting pixels of an image as nodes of a graph with edges connecting neighboring pixels, we can take advantage of several network optimization methods to do image analysis. In class, we went through several examples that ended up being formulated as network optimization problems. One particular example discussed in class was the max-flow problem. It turns out that the max-flow problem is equivalent to a **min-cut problem**. A later section in this document throws some light into this equivalence.

In this project, we will use the min-cut formulation to do image segmentation. Image segmentation is, in general, a very hard problem but is very useful in a variety of applications including medical diagnosis, autonomous driving and face recognition. Several variants of the problem exist (like hierarchical image segmentation and image labeling). Though several competing methods have been proposed, graph-partitioning methods play a central role and have been found to be very successful. The min-cut is essentially a graph partitioning problem that seeks to cut a graph/network into disjoint pieces in such a way that it minimizes an objective.

Our objective is to segment an image into the foreground and the background. We first show how we can segment an image using naïve *thresholding*. Then we detail how we can formulate a min-cut optimization problem that attempts to correct the disadvantages of thresholding. It turns out that once we do this formulation, the challenge would be in coming up with a right set of parameter choices (a,b,c) for our formulation. Much of the research in this area has focused on choosing these parameters.

For this project, we will start with a simple set of parameters that this document provides. We will also restrict all our attention to 128x128 grey scale images. Obviously, an extension to larger and color images would be trivial, and only computationally more time-consuming.

Image segmentation via Thresholding

A 128x128 grey-scale image is essentially a 128x128 matrix with each element denoting the intensity of a pixel. A histogram of all the intensities would be a good place to start for segmentation. For example, consider an easy to segment image like Figure 1.

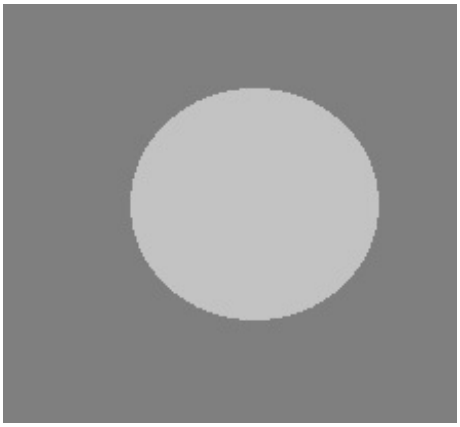

Figure 1: Simple Image

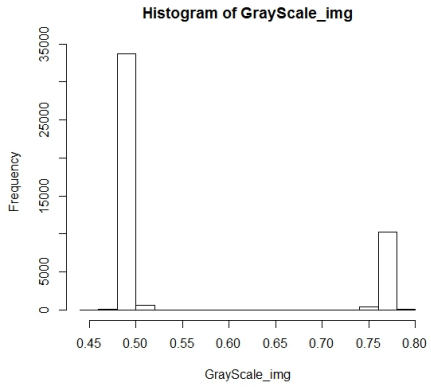

Figure 2: Histogram of Intensities in Figure 1

Such an image would have an intensity histogram presented in Figure 2.

The two modes in this distribution correspond to the two color patterns in this picture. A threshold of 0.65 can segment the image. We can obviously have our code, take a grey-scale image matrix as input and then choose a threshold. Once the threshold is chosen, the output segmented image is another greyscale matrix of the same size with pixels in black for those pixels below the threshold and vice versa. For the simple example in Figure 1, the segmentation would be:

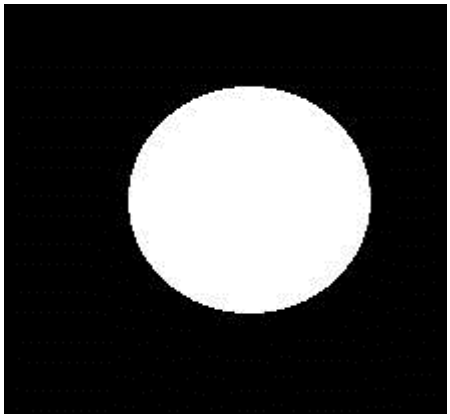

Figure 3: Background and Foreground for Image in Fig 1

Your first task: Implement a function that uses thresholding to segment an image. Any reasonable way of choosing the threshold is fine. Try your function on the sample images provided.

Issues with Thresholding

As you might have realized, thresholding is too naïve to work on complicated images, especially because there is no incentive built to mark pixels closer to one another with the same background or foreground label. For example, the image to left below (Figure 4) has an intensity histogram given by the image to the right below (Figure 5).

Figure 4: A slightly more complicated image

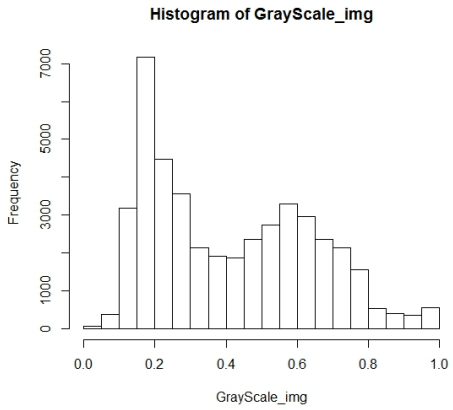

Figure 5: Histogram for Image in Fig 4

The intensity histogram, in this case, has no clear threshold. Suppose we use a threshold of 0.6, we would end up getting the shirt belonging to the foreground with the pants belonging to the background! And too many orphan foreground and background spots on the ground.

Figure 6: Segmentation using thresholding for Fig 4

Image Segmentation Using Min-cut/Max-flow

Say for any given image we can compute matrices *a, b,* and *c* in a way that makes a_i the likelihood that pixel *i* is a foreground pixel and b_i be the likelihood that it is a background. Each pixel is considered a separate node and the c_{ij}'s are the weights on the edges connecting node pairs.

The simplest way to obtain *a, b* is to use thresholding and set *a=1*, b=0 when pixel intensity is above the threshold and vice versa. And set $c_{ij} = 1$ if *i* and *j* are neighboring pixels and 0 otherwise.

For this project though, better choices are described as follows:

We first create a good bounding box, which encapsulates the foreground. It is an input in the form of two (x, y) coordinate pairs in the image such that the area outside the box is mostly background and all the area inside the box is foreground. Let p_i be pixel intensities, $\overline{p_f}$ be the average intensity of the pixels in the temporary foreground and $\overline{p_b}$ be the average intensity of the pixels in the temporary background. Now calculate a_i and b_i for each pixel *i* in your image as follows,

$$a_i = -log \frac{abs(p_i - \overline{p_f})}{abs(p_i - \overline{p_f}) + abs(p_i - \overline{p_b})} \text{ and } b_i = -log \frac{abs(p_i - \overline{p_b})}{abs(p_i - \overline{p_f}) + abs(p_i - \overline{p_b})}$$

The weight for an edge connecting pixels *i* and *j* is given by c_{ij},

$$c_{ij} = K \exp\left(\frac{-(p_i - p_j)^2}{\sigma^2}\right)$$

when pixels *i* and *j* are adjacent to each other (You don't need to consider diagonal links). Use K = 0.01 and $\sigma = 1$.

Our goal now is to now refine the partition (the background and the foreground) of the set of pixels into two sets A and B such that they maximize q(A,B),

$$q(A, B) = \sum_{i \varepsilon A} a_i + \sum_{j \varepsilon B} b_j - \sum_{(i,j) \varepsilon E} c_{ij}$$

where E is the set of adjacent nodes (i, j) such that nodes *i* and *j* are adjacent to each other and exactly one of them is in the foreground.

However, in our network, we are still missing the source and the sink nodes. We will add a source node s and a sink node t. Also, we will add a directed edge (s, i) for every pixel i and another directed edge (i, t) for the same pixel. The weight of an edge connecting the source to pixel i is a_i, and the weight of the edge connecting the pixel i to the sink is b_i. You may have guessed that all pixels connected to s in our final network will be foreground pixels and t will be background pixels. Lastly, all edges between two adjacent pixels (i, j) are bi-directional with forward and backward edges having the same weight, c_{ij} .

You can think of c_{ij} as the penalty for putting one of i, j in the foreground and the other in the background.

The Max-flow and the min-cut problem

Consider a directed graph with source s sink t and positive edge capacities c_{ij} between two connected nodes i and j. A flow in a graph is a routing specification from source to sink such that no edge is used beyond its capacity. Formally, it is an assignment of a non-negative number to every edge no bigger than its capacity such that the sum of outflow is less than inflow at every node in the graph. The cost of this flow is the sum of all non-negative numbers assigned to the edges along a path from source to sink.

Consider any set of edges whose removal disconnects the source from the sink. These set of edges constitute a cut. Concretely, a cut is a partition of the graph into two disjoint set of vertices. The capacity of the cut is the sum of capacities of the removed edges. A minimum cut is a cut whose capacity is minimum over all cuts of the graph. Intuitively, the removed edges of the minimum cut can be thought of as a bottleneck of the system. They are the ones that constrict the capacity of the system, and hence their sum must equal the maximum flow possible.

It is not too difficult to show that the cost of the flow is at most the capacity of the cut. To maximize the flow, we must minimize the cut capacity. These are competing objectives, or more formally, these two problems are duals of each other. By the principle of strong duality, dual programs for convex problems attain the same objective value at their respective optimum solutions. Maximum flow and minimum cut are linear programs, convex, hence by the principle of strong duality, the optimal value of the maximum flow problem must be equal to the optimal value of the minimum cut problem. This is formally proved in the Max-flow Min-Cut theorem, which says there exists a cut whose capacity is exactly equal to the cost of the flow.

Implementation

We are interested in the minimum cut, which will give us a separation of our image graph into two disjoint set of nodes. Foreground nodes are connected to the source, and background nodes are connected to the sink.

Now construct the graph as described above using the *igraph* package in R. Take a look at the *max_flow* function in the library. While *max_flow* will give us the path and the maximum possible flow, it will not directly give us the separation. We are interested in the minimum cut, which will give us the necessary separation. If you take a look the values returned from the *max_flow* function, you will notice that it also returns the cut information. Explore how to use the cut information and color all the foreground pixels in the input image with the color blue and all background pixels in the image using the color red. Try with the different images provided.

Submission Instructions:

Name your report as **project4_gZ.pdf** (Z is your group number). Upload your report and your code as two separate files. Include the results of your code in your report.

(We want to use speedgrader that requires separate files.)

Side Note 1:

List of Useful R libraries and functions

Libraries

- igraph – a package that helps in handling large networks and includes functions to optimize network flow problems
- imager: To read and pixelate the images
- dplyr – A fast, consistent tool for working with dataframe objects

Functions

- load.image ()– a function to load an image as a matrix of pixels containing the information about each pixel
- graph_from_data_frame – a function that creates the graph from date frames.
- max_flow – a function to solve for the optimal solution to the network problem. If set up correctly, returns the partition of the nodes.

Side Note 2:

We expect the results of your code to show the image and the separation. Ideally, we would expect something like this: