B657 Assignment 3: Markov Random Field Models

Spring 2014
Due: Sunday April 20

In this assignment you'll implement two different computer vision algorithms that can be posed using the same underlying mathematical and algorithmic framework: Markov Random Fields (MRFs).

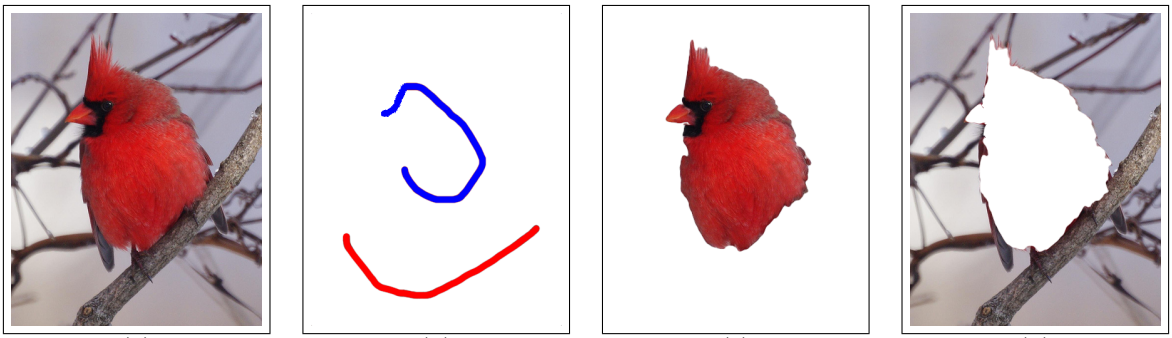
The first algorithm is for semi-supervised segmentation. The program will take a color image and a set of "seed points" that indicate where some of the foreground and background pixels in the image are located. These seed points could be quick strokes drawn manually by a human, like in Figure 1. The program will then use these seeds in an MRF model to produce a complete segmentation of the image, partitioning the image into foreground and background (Figure 1). The second algorithm is for resolving stereo: given two images of the same scene taken from two different camera angles, infer the depth of each pixel by computing disparities between the two images.

Despite the fact that these two algorithms seem quite different, they can both be posed as MRF inference problems "under the hood," so that much of the code can be shared between the two programs.

You may work on this project individually or in groups of two. If you work in a group, you should turn in a single project report and single set of source code files. Both members of a group will receive the same grade.

We recommend using C/C++ for this assignment, and we have prepared some skeleton code that may help you get you started (see details below). We recommend using C/C++ because computer vision algorithms tend to be compute-intensive, and your code may be frustratingly slow when implemented in higher-level languages (like Matlab, Java, Python, etc.). However you may choose to use a different programming language, with the restriction that you must implement the image processing and computer vision operations yourself. You do not have to re-implement image I/O routines (i.e. you may use Matlab's imread and imwrite functions). You may use built-in or third party libraries for routines not related to image processing (e.g. data structures, sorting/searching algorithms, etc.).

Please use the Forums on the OnCourse site to ask questions about this assignment, and monitor the forums for answers to frequently asked questions.



(a) (b) (c) (d)

Figure 1: Sample semi-supervised segmentation problem and results: (a) input image, (b) foreground (blue) and background (red) strokes drawn by a human, indicating roughly where the foreground and background regions of the image are, (c) foreground image produced by segmentation, (d) background image produced by segmentation.

What to do

1. Download the skeleton code and test images from OnCourse and make the sample program:

   ```
   tar -xzf a3-skel.tar.gz
   cd a3/
   make
   ```

   The code has been tested in the Linux machines in the School of Informatics, so we suggest using one of those. You may use another development platform (Windows, MacOS, etc.), but you may have to modify the skeleton code to get it to compile.

   There are two skeleton programs here. One is called "segment" and takes an input to be segmented, and then a second image that specifies seed points. The seed points image is just a blank image with a few red and blue areas, that show the background and foreground seed points, respectively. This program should run foreground vs background segmentation on the input image using the seed points, and then output two images, one containing the foreground pixels and the other the background pixels. Run the program like this:

   ```
   ./segment input.png seeds.png
   ```

   The second program is called "stereo" and takes a left image, a right image, and (optionally) a third image which is a ground-truth disparity map. The goal of this program is to compute a disparity map between the left and right images, output the disparity map as a png file, and compute a quantitative error measure by comparing the disparity map to the ground-truth. The current skeleton code computes the disparity map completely randomly. Run the program like this:

   ```
   ./stereo left.png right.png gt.png
   ```

   and then look at the resulting output files.

2. Let's work on `segment` first. What we'd like to do here is to assign each pixel a label that is either a 1 or 0: 1 if the pixel is in the foreground and 0 if in the background. The existing skeleton code doesn't do very much – it just computes a random segmentation.

   As a first step towards a better segmentation algorithm, we should at least make use of the seed points provided to the program. The seed points are a set $\mathcal{F} \in I$ of pixels of image I known to be foreground points, and a set $\mathcal{B} \in I$ of points known to be background points. Based on the set of points in $\mathcal{F}$, we can build a simple appearance model of what foreground pixels "look like," for instance by simply estimating means $\mu = [\mu_R \;\; \mu_G \;\; \mu_B]^T$ and variances $\Sigma = \text{diag}(\sigma_R, \sigma_G, \sigma_B)$ of the RGB color values in $\mathcal{F}$. Let $L(p)$ denote a binary label assigned to pixel p. Then we can define a function that measures the "cost" of giving a label $L(p)$ to pixel p,

$$D(L(p), I(p)) = \begin{cases} 0 & \text{if } L(p) = 0 \text{ and } p \in \mathcal{B} \\ 0 & \text{if } L(p) = 1 \text{ and } p \in \mathcal{F} \\ \infty & \text{if } L(p) = 0 \text{ and } p \in \mathcal{F} \\ \infty & \text{if } L(p) = 1 \text{ and } p \in \mathcal{B} \\ -\log N(I(p); \mu, \Sigma) & \text{if } L(p) = 1 \text{ and } p \notin \mathcal{F} \text{ and } p \notin \mathcal{B} \\ \beta & \text{if } L(p) = 0 \text{ and } p \notin \mathcal{F} \text{ and } p \notin \mathcal{B}, \end{cases}$$

   where β is a constant and $N(I(p); \mu, \Sigma)$ is a Gaussian probability density function evaluated with the RGB color values at $I(p)$. Intuitively, this says that one pays no cost for assigning a 1 to a pixel in $\mathcal{F}$ or a 0 to a pixel in $\mathcal{B}$, but we pay a very high cost for assigning the wrong label to one of these known pixels. For a pixel not in $\mathcal{B}$ or $\mathcal{F}$, we pay some fixed cost β to assign it to background, and a cost for assigning it to foreground that depends on how similar its color is to the known foreground pixels.

Implement a function called `naive_segment()` that chooses a most-likely pixel for each pixel given only the above energy function, i.e. computes the labeling,

$$L* = \arg\min_L \sum_{p \in I} D(L(p), I(p)).$$

To help you test your code, we've given you some sample images and seed files in the images/ directory of the skeleton code archive.

3. The above formulation has a major disadvantage, of course: it doesn't enforce any sort of spatial coherence on the segmentation result. To fix this, let's define a more complicated energy function,

$$E(L, I) = \sum_{p \in I} D(L(p), I(p)) + \alpha \sum_{p \in I} \sum_{q \in \mathcal{N}(p)} V(L(p), L(q))$$

where α is a constant, $\mathcal{N}(p)$ is the set of 4-neighbors of p (i.e. $\mathcal{N}(p) = \{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\}$ for $p = (i, j)$ not on the image boundary), and $V(\cdot, \cdot)$ is a pairwise cost function. We could use various forms for this pairwise cost, but intuitively we want to penalize disagreement between labels,

$$D(L(p), L(q)) = (L(p) - L(q))^2.$$

Now to actually perform segmentation, we simply need to minimize the energy; i.e. find $L*$ such that,

$$L* = \arg\min_L E(L, I).$$

As we saw in class, this is an MRF energy minimization problem. Implement loopy belief propagation to (approximately) minimize this energy function. Your final program should produce four output images: foreground and background images produced by the simple procedure of step 2, and foreground and background images produced by loopy BP in this step.

4. Now to stereo. Recall from class that, to make things easier for the stereo algorithm, image pairs are typically rectified so that the epipolar lines are parallel and correspond to horizontal scan lines of the images. Also recall that depth is inversely proportional to disparity; i.e. the depth of a pixel is given by $z = fb/d$, where d is the disparity between left and right images at that pixel, f is the focal length of the camera (assumed to be the same for both images), and b is the distance between camera centers. The hard part is finding a good estimate of d for each pixel. We'll thus ignore f and b and instead concentrate on finding good disparity maps; the output of your code will be a disparity map instead of a depth map.

   As we saw in class, stereo can also be posed as an MRF inference problem. We can define a stereo energy function that turns out to be nearly identical to that of semi-supervised segmentation,

$$F(L, I) = \sum_{p \in I} D_2(L(p), I_L, I_R, p) + \alpha \sum_{p \in I} \sum_{q \in \mathcal{N}(p)} V_2(L(p), L(q)),$$

   except that we'll define the cost functions D_2 and V_2 differently, and the set of possible labels is no longer binary but instead the set of possible disparity values (i.e. an integer greater than or equal to 0 and less than the width of the image). For the pairwise function, one could use the same quadratic cost function defined in step 3, or one of the variants we saw in class (e.g. Potts or truncated quadratic).

   For the unary cost, we need a function that uses image data to infer the disparities — i.e. for each pixel (i, j) in the left image, estimates the likelihood that it corresponds to pixel $(i, j + d)$ in the right image, for all possible values of d. As we saw in class, a reasonable way of doing this is to compute sum-squared differences. The cost function then becomes,

$$D_2(d, I_L, I_R, (i, j)) = \sum_{u=-W}^{u=W} \sum_{v=-W}^{v=W} (I_L(i+u, j+v) - I_R(i+u, j+v+d))^2.$$

Complete `stereo` by implementing loopy belief propagation to minimize the energy function F. To help you test your code, we've included five stereo pairs in the images/ directory of a3.tar.gz, named Aloe, Baby, Bowling, Flowerpots, and Monopoly. These pairs have already been rectified. For each pair, we've included left and right images, as well as two ground truth disparity maps. The skeleton code can compute an error measure (the mean squared difference) between the disparity map you produce and the ground truth. What is the effect of the window size parameter W on the quality of the disparity maps, both qualitatively and quantitatively?

Academic integrity

You and your partner may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about C/C++ syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials in the documentation of your source code. However, the code that you (and your partner, if working in a group) submit must be your own work, which you personally designed and wrote. You may not share written code with any other students except your own partner, nor may you possess code written by another student who is not your partner, either in whole or in part, regardless of format.

What to turn in

Turn in two files, via OnCourse:

1. Your C++ source code compressed as a .zip or .tgz file. Make sure your code is thoroughly debugged and tested. Make sure your code is thoroughly legible, understandable, and commented. Please use meaningful variable and function names. Cryptic or uncommented code is not acceptable.

2. A separate text report in PDF format. Explain how you implemented it (e.g. what data structures you used), what design decisions and other assumptions you made. Give credit to any source of assistance (students with whom you discussed your assignments, instructors, books, online sources, etc.). Make sure to show the results of your program on our sample test images, by presenting qualitative (i.e. figures showing the output) and quantitative (i.e. error metrics with respect to ground truth for `stereo`.

Grading

We'll grade based on the correctness, style, and the project report. In particular: Does the code work as expected? Does it use reasonable algorithms as discussed in class? Did it produce reasonable results? Was it tested thoroughly? Does it check for appropriate input and fail gracefully? Is the code legible and understandable? Is the code thoroughly and clearly commented? Is the project report adequate?