Problem Set 4

Assigned: March 23, 2006
Due: April 17, 2006

Problem 1 *(6.882) Belief Propagation for Segmentation*

In this problem you will set-up a Markov Random Field (MRF) for which the solution gives a segmentation of an image into foreground and background. To solve the MRF and segment the image, you'll implement belief propagation (BP), which will give an approximation to the minimum mean squared error (MMSE) solution at every pixel. We'll guide you through the steps.

1.1 Setting up the MRF

The Markov Random Field has this probabilistic model:

$$p(X,Y) = \frac{1}{Z}\prod_i \Phi(x_i, y_i) \prod_{(i,j)\in E} \Psi(x_i, x_j), \tag{1}$$

where Y is the observed color image, and $X = \{x_i\}, x_i \in \{1, 0\}$ is foreground(1)/background(0) labeling. $\Phi(\cdot,\cdot)$ and $\Psi(\cdot,\cdot)$ are compatibility functions. Z is a normalization constant to make $p(X,Y)$ a pdf, but we can ignore it for this problem. $(i,j) \in E$ means all neighboring pixels i and j.

We'll assume that the user has labelled some pixel colors that are to be foreground, and some that are to be background. Load the image `flower.bmp`, a partial labeling of foreground pixels, `cfgmask.bmp`, and a partial labeling of background pixels, `cbgmask.bmp`. In each label image the white pixels indicate a valid sample of foreground or background pixels. We'll use these labellings to estimate the color distributions of the foreground and background pixels. This will let us find the local evidence that a pixel is in the foreground segment or the background segment, based on local color information alone.

(1) $\Phi(\cdot,\cdot)$. From the labelled pixels and the original image, estimate the mean and covariance of the foreground and background pixels (RGB), denoted as $\mu_f, \mathbf{\Sigma}_f$ and $\mu_b, \mathbf{\Sigma}_b$.

We'll define the likelihood function to be

$$p(y_i|x_i = 1) = \frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}_f|^{1/2}} \exp\{-\frac{1}{2}(y_i - \mu_f)^T \mathbf{\Sigma}_f^{-1}(y_i - \mu_f)\} + \varepsilon, \quad (2)$$

and

$$p(y_i|x_i = 0) = \frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}_b|^{1/2}} \exp\{-\frac{1}{2}(y_i - \mu_b)^T \mathbf{\Sigma}_b^{-1}(y_i - \mu_b)\} + \varepsilon. \quad (3)$$

We have add the term ε to account for outliers from the overly-simple Gaussian model for the color distribution. Set $\varepsilon = 0.01$ in this problem.

The local evidence compatibility function Φ can be simply

$$\begin{cases} \Phi(y_i, x_i = 1) = \dfrac{p(y_i|x_i = 1)}{p(y_i|x_i = 1) + p(y_i|x_i = 0)} \\ \Phi(y_i, x_i = 0) = \dfrac{p(y_i|x_i = 0)}{p(y_i|x_i = 1) + p(y_i|x_i = 0)} \end{cases} \quad (4)$$

(2) $\Psi(\cdot, \cdot)$. To allow local evidence to propagate over the image, we set the compatibility function for neighboring nodes i and j, $\Psi(x_i, x_j)$ to be the symmetric matrix

$$\begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}. \quad (5)$$

This encourages neighboring nodes to have the same foreground/background labelling. This defines a Markov Random Field, giving a probability to any configuration of pixel foreground/background labels, through Eq. (1). Next, we want to find an approximation to the optimal labelling.

1.2 Solving the MRF

In this homework we ask you to implement MMSE estimate for $\hat{x}_i$ by marginalizing over the other variables in the pdf Eq. (1) using Belief Propagation. The message update rule is (for the message from node i to j):

$$m_{ij}(x_j) \leftarrow c \sum_{x_i} \Psi(x_j, x_i)\Phi(x_i, y_i) \prod_{k \in N(i) \backslash j} m_{ki}(x_i). \quad (6)$$

In words: you multiply together all the incoming messages (except the one coming from the node you're sending the message to) times the local evidence

term for this node. Then multiply that times the compatibility matrix connecting this to the neighboring node, and sum over the states of this node, x_i. (By the way, the message update rule for MAP estimate is the same, except with the sum over x_i changed to $\max_{x_i}$. That fact is not needed for this problem).

The belief, or the approximated marginal distribution is

$$b(x_i) = c\ \Phi(x_i, y_i) \prod_{j \in N(i)} m_{ji}(x_i), \tag{7}$$

which can be computed at the end of the message updating.

You can assume that each pixel has four neighbors, bottom, top, left, right, except at the image boundaries. So there are four directional messages, M_{tb}, M_{bt}, M_{lr} and M_{rl}. You will need to store them separately. In each step, allocate buffers for new messages M'_{tb}, M'_{bt}, M'_{lr} and M'_{rl} so that all messages get updated at the same time.

For each BP iteration, you can write a loop (slow!!) over all pixels to compute the new messages. Or (faster!!), you can write everything in matrix multiplication form, although that's not required for the assignment.

To avoid any over or underflow problems, the new messages should be normalized so that the terms sum to one. Also, the algorithm will converge better if, instead of updating the messages as in Eq. (6), you update each message with a weighted combination of the previous iteration's message and the new one computed by Eq. (6). Ie, instead of directly updating each message as $M_{tb} = M'_{tb}$, please use $M_{tb} = \alpha M'_{tb} + (1 - \alpha) M_{tb}$, with $\alpha = 0.7$.

Implement MMSE estimate for X. You can use the estimated belief, $b(x_i)$, as an approximate alpha map for the foreground object. Paste the foreground into `leaves.bmp` using the calculated beliefs as an alpha map.

Problem 2 *(Extra Credit) Efros and Leung Texture Synthesis for Inpainting*

In this problem, we show an effective way to make-up pixel values that are missing over regions of an image. This is called *inpainting*.

There are many criteria for inpainting. We ask you to implement texture synthesis-based approach which is particularly suitable for a textured background. Read Efros and Leung's ICCV 1999 paper "Texture Synthesis by Non-parametric Sampling ". The pseudo code for the corresponding inpaint-

ing algorithm can be found at this link `http://graphics.cs.cmu.edu/people/efros/research/NPS/alg.html`.

Write a MATLAB function `ImSyn=texturesyn_inpaint(im,mask,psize)` to synthesize a new image which has the same dimension as `im` (either grayscale or color). In `mask` the pixels marked as 1 are "search area", and those marked as 0 are "synthesis area". A `psize` x `psize` patch is used to find the best match for each pixel.

Here are some tips for this MATLAB implementation. For each pixel you need to generate a template of both mask and pixel values. Fix the size of the template for each pixel. Treat both the mask and pixel values as zero for those outside image boundary.

(a) Load grayscale image `rings.bmp`. Put it at the center of a 64×64 image and grow texture. Try patch size 17 and 21, and see how the result varies. Then load color image `net.jpg` and grow it to a 100×100 image with patch size 23. Your code may need to run for a while.

(b) Load `horse.bmp` and `mask.bmp`. Use the background pixels on the grass to remove the house. You do not have to use all the background pixels but only those close to the boundary. Run this code before you go to bed and see the result in the morning. You may also try your favorite way to accelerate searching.

Problem 3 *Image Morphing*

In this problem, you will implement image warping using radial basis functions, and will then use it to perform morphing between two images. In image warping, a smooth displacement field is applied to deform an image. The displacement field that we will manipulate is a 2D to 2D function which, given a position in the **output** image, provides the original position in the **input** image. Note that this is the inverse of the displacement undergone by the image, because we need to look up for each new pixel the corresponding color in the old image.

In this problem, the field $f : \mathbb{R}^2 \mapsto \mathbb{R}^2$ will be sparsely specified by the user who will provide the before and after position of a sparse set of points as a set of pairs of 2D vectors, denoted as x_i for the position before and y_i for the positions after. The main task is, given these sparse correspondences, to

interpolate a smooth displacement field f. For this, we will use radial basis functions, that is, smooth kernel functions centered on the sparse data points. We will use thin-plate RBFs that depend on the spatial distance to the data points (we will ignore the polynomial term sometimes included in RBFs). The basis function centered on data point x_i evaluated at an arbitrary point z is:

$$R(z, x_i) = \frac{1}{\sqrt{c + \|z - x_i\|^2}} \tag{8}$$

where $\|z - x_i\|$ is the distance between position z and data point x_i, and c is a parameter that controls the falloff of the function. Set $c = 10$ at the beginning and experiment with it only once your code fully works.

Your main task is then to determine the coefficient (weight) of the basis function centered on each data point. These weights are found by solving a linear system that enforces interpolation. That is, the value of the RBF at each data point x_k should be the provided vector y_k, i.e.

$$\sum_i \alpha_i R(x_k, x_i) = y_k, \quad k = 1, \cdots, n \tag{9}$$

where $\alpha_i \in \mathbb{R}^2$.

Once you have computed an RBF model, you can warp the image by computing the color of each new pixel. Compute the displacement field at the location z by evaluating the RBF model.

$$f(z) = \sum_i \alpha_i R(z, x_i) \tag{10}$$

(You will notice a computational disadvantage of RBFs: each kernel needs to be evaluated, making the cost linear in the number of provided data points). Then perform a simple lookup and use the color of the displaced pixel $f(z)$ in the input image. You may use MATLAB function `interp2`.

(a) Load `chessboard.bmp`. Generate the the warped image according to the following specified displacement:

$$\begin{aligned} (32, 32) &\mapsto (88, 58) \\ (224, 32) &\mapsto (229, 82) \\ (224, 224) &\mapsto (165, 212) \\ (32, 224) &\mapsto (61, 178) \end{aligned}$$

You may change c and see how it affects the warp field.

(b) Load two face images `face1.bmp` and `face2.bmp`, and the labeled feature points `facepts1.mat` and `facepts2.mat`. Compute and display the forward (1 to 2) and backward (2 to 1) warp fields, and the corresponding warped images. (Hint: you can make debugging easier by replacing the face images by a grid picture of the same resolution, which will better reveal any problem.)

(c) **Morphing**. Morphing interpolates seamlessly between two images by combining a warp of the initial and final images with linear interpolation of color values. Morphing performs spatial warping so that the provided features are displaced along time from their initial position to their final position. You will compute a 5-image morphing animation where t varies from 0 to 1 in steps of 0.2 (that is, the two inputs plus 3 intermediate images). For each intermediate image, first compute the interpolated position of each feature. Perform a spatial warp as above from the initial image to these intermediate positions. This will give you a first image. Similarly, perform a warp of the final image to these intermediate positions, whcih gives you a second image. Then simply blend the two images according to the value of time (where the influence of the initial image varies from 0 to 1 with as time progresses). Display the morphing sequence.