MIT CSAIL

6.869 Advances in Computer Vision

Spring 2011

Problem Set 5: 1-d BP and Texture Synthesis

Posted: Wednesday, March 2, 2011	**Due:** Wednesday, March 9, 2011

You should submit a hard copy of your work in class, and upload your code (and all files needed to run it, images, etc) to stellar.
Your report should include images and plots showing your results, as well as pieces of your code that you find relevant.

Problem 5.1 *Belief Propagation*

Many vision problems consist of measuring local evidence, then propagating it across space. Belief propagation is often useful for such tasks. This homework problem was presented as a belief propagation example by Yair Weiss in a NIPS paper [1].

A task of early vision is to make a figure/ground assignment – which side of a contour is the foreground object, and which side is the background? A good cue for that assessment is convexity. Contours typically encircle the object, rather than form holes within it, so the foreground side is often on the inside of a contour's curve.

Locally, a complex contour may bend both ways and only a global assessment of convexity can tell us the right answer. We define a Markov chain of points along a contour in an image (we assume this contour has already been detected). The hidden states are the side of the foreground assignment for the contour (+1 means to the right as you traverse the contour, incrementing the node index; -1 is to the left). The local evidence at each node is based on the local curvature, defined by the angle θ_j based on the local three adjacent points on the curve (nodes $j-1$, j, and $j+1$). Let $\theta_j = 0$ correspond to a straight line, and $\theta_j = \frac{\pi}{2}$ correspond to a 90° right bend, and $\theta_j = -\frac{\pi}{2}$ correspond to a 90° left bend.

Let the local evidence for a positive or negative curvature curve be: $\phi(x_j, y_j) = \begin{pmatrix} \frac{\pi+\theta_j}{2\pi} \\ \frac{\pi-\theta_j}{2\pi} \end{pmatrix}$
This favors figure/ground evidence in proportion to the acuteness of the local angle of bending.

The hidden state compatibility requires that hidden states have the same value as that of the neighboring node: $\psi(x_j, x_{j+1}) = \begin{pmatrix} 1.0 & 0 \\ 0 & 1.0 \end{pmatrix}$

The joint probability of figure/ground estimates conditioned on the observed curve is given

by the product of the local evidence and the node capabilities:

$$P(\vec{x}|\vec{y}) = \prod_j \phi(x_j, y_j)\psi(x_j, x_{j+1}) \tag{1}$$

On the class website, you will find the file `curves.zip` which contains three curves stored in `mat` files with the x and y coordinates of each curve. In general we can use computer vision algorithms (some of which you have already encountered) for extracting such contours from images, however here we produced them manually using the supplied `DrawContour` function.

1. Load and plot the supplied curves in the 2-d plane. Notice that the coordinate system origin for those curves is the upper left corner of the image.
2. Indicate the local direction of figure (draw a small arrow to the foreground side at every node) based on local evidence alone, before running belief propagation.
3. Show the final estimated direction of figure after running belief propagation.
4. **[Optional]** Use the `DrawContour` function to draw contours for your own images (or from scratch), and estimate their direction of figure. Can you think of cases where this algorithm will fail?

Problem 5.2 *Texture synthesis*

In this problem you will implement the Efros and Leung algorithm for texture synthesis discussed in Section 9.3 of Forsyth and Ponce. In addition to reading the textbook you may also find it helpful to visit Efros' texture synthesis website:
http://www.cs.berkeley.edu/~efros/research/synthesis.html,

in which many of the implementation details described below can be found. As discussed in class, the Efros and Leung algorithm synthesizes a new texture by performing an exhaustive search of a source texture for each synthesized pixel in the target image, in which sum-of-squared differences (SSD) is used to associate similar image patches in the source image with that of the target. The algorithm is initialized by randomly selecting a 3×3 patch from the source texture and placing it in the center of the target texture. The boundaries of this patch are then recursively filled until all pixels in the target image have been considered. Implement the Efros and Leung algorithm as the following MATLAB function:

```
synthIm = SynthTexture(sample, w, s)
```

where `sample` is the source texture image, `w` is the width of the search window, and `s=[ht, wt]` specifies the height and width of the target image `synthIm`. As described above, this algorithm will create a new target texture image, initialized with a 3x3 patch from the source image. It will then grow this patch to fill the entire image. As discussed in the textbook, when growing the image un-filled pixels along the boundary of the block of synthesized values are considered at each iteration of the algorithm. A useful technique for recovering the location of these pixels in MATLAB is using *dilation*, a morphological operation that expands image regions.

Use MATLAB's `imdilate` and `find` routines to recover the un-filled pixel locations along the boundary of the synthesized block in the target image.

In addition to the above function we ask you to write a subroutine that for a given pixel in the target image, returns a list of possible candidate matches in the source texture along with their corresponding SSD errors. This function should have the following syntax:

```
[bestMatches, errors] = FindMatches(template, sample, G)
```

where `bestMatches` is the list of possible candidate matches with corresponding SSD errors specified by `errors`. `template` is the $w \times w$ image template associated with a pixel of the target image, `sample` is the source texture image, and `G` is a 2D Gaussian mask discussed below. This routine is called by `SynthTexture` and a pixel value is randomly selected from `bestMatches` to synthesize a pixel of the target image. To form `bestMatches` accept all pixel locations whose SSD error values are less than the minimum SSD value times $(1 + \epsilon)$. To avoid randomly selecting a match with unusually large error, also check that the error of the randomly selected match is below a threshold δ. Efros and Leung use threshold values of $\epsilon = 0.1$ and $\delta = 0.3$.

Note that `template` can have values that have not yet been filled in by the image growing routine. Mask the template image such that these values are not considered when computing SSD. Efros and Leung suggest using the following image mask:

```
Mask = G .* validMask
```

where `validMask` is a square mask of width w that is 1 where the template is filled, 0 otherwise and `G` is a 2D zero-mean Gaussian with variance $\sigma = w/6.4$ sampled on a $w \times w$ grid centered about its mean. `G` can be pre-computed using MATLAB's `fspecial` routine. The purpose of the Gaussian is to down-weight pixels that are farther from the center of the template. Also, make sure to normalize the mask such that its elements sum to 1.

Test and run your implementation using the grayscale source texture image `rings.jpg` available on the class website, with window widths of $w = 5, 7, 13$, $s = [100, 100]$ and an initial starting seed of $(x, y) = (4, 32)$. Explain the algorithm's performance with respect to window size. For a given window size, if you re-run the algorithm with the same starting seed do you get the same result? Why or why not? Is this true for all window sizes?

Include in your report the synthesized textures that correspond to each window size along with answers to the above questions.

References

[1] Yair Weiss. Interpreting images by propagating bayesian beliefs. In *Advances in Neural Information Processing Systems 9*, pages 908–915, 1996. `http://www.cs.huji.ac.il/~yweiss/nips96.pdf`.