ENEE731 Project
Texture Segmentation using Gabor Filters

Naotoshi Seo, sonots@umd.edu

November 8, 2006

1 Introduction

Malik and Perona (1989) [1] presented a model of texture perception with the early stages of human visual system. Their model consists of three stages: (1) convolution of the image with a bank of even-symmetric linear filters followed by half-wave rectification to give a set of responses, (2) inhibition, localized in space, within and among the neural response profiles that results in the suppression of weak responses when there are strong responses at the same or nearby locations, and (3) texture-boundary detection by using wide odd-symmetric mechanisms. The stage (1) is referred to as the multi-channel filtering approach.

Jain and Farrokhania (1990) [2] presented a texture segmentation algorithm inspired by the multi-channel filtering theory in the early stages of human visual system. They characterized channels by a bank of Gabor filters. This is the paper which we basically followed to implement a Filtering Method based Image Segmentation system.

2 Algorithm

The texture segmentation system involves the following three steps:

- Decomposition of the input image using a filter bank,
- Feature extraction, and
- Clustering.

2.1 Filter Bank

We present the channels with a bank of two-dimensional Gabor filters. A two-dimensional Gabor function consists of a sinusoidal plane wave of some frequency and orientation, modulated by a two-dimensional Gaussian [3]. The Gabor filter in the spatial domain is given by

$$g_{\lambda\theta\psi\sigma\gamma}(x,y) = exp(-\frac{x'^2+\gamma^2y'^2}{2\sigma^2})cos(2\pi\frac{x'}{\lambda}+\psi) \tag{1}$$

where

$$x' = xcos(\theta) + ysin(\theta),$$
$$y' = ycos(\theta) - xsin(\theta).$$

In this equation, λ represents the wavelength of the cosine factor, θ represents the orientation of the normal to the parallel stripes of a Gabor function in degrees, ψ is the phase offset in degrees, and γ is the spatial aspect ratio and specifies the elliptically of the support of the Gabor function, and σ is the standard deviation of the Gaussian determines the (linear) size of the receptive field.

The parameter λ is the wavelength and $f = 1/\lambda$ is the spatial frequency of the cosine factor. The ratio σ/λ determines the spatial frequency bandwidth of simple cells and thus the number of parallel excitatory and inhibitory stripe zones which can be observed in their receptive fields. The half-response spatial frequency bandwidth b (in octaves) and the ratio σ/λ are related as follows:

$$b = log_2 \frac{\frac{\sigma}{\lambda}\pi + \sqrt{\frac{ln2}{2}}}{\frac{\sigma}{\lambda}\pi - \sqrt{\frac{ln2}{2}}}, \frac{\sigma}{\lambda} = \frac{1}{\pi}\sqrt{\frac{ln2}{2}}\frac{2^b+1}{2^b-1}. \tag{2}$$

$\psi = 0°$ and $\psi = 90°$ returns the real part and the imaginary part of Gabor filter respectively. The real part of Gabor filter is an even-symmetric filter, and the property satisfies the requirement proposed by Malik [1]. Therefore, we use the real part of Gabor.

Choice of Filter Parameters

We use orientation separation angles of 30° as recommended in [4], that is:

$$\theta : 0°, 30°, 60°, 90°, 120°, 150°$$

and following values of frequencies as recommended in [6]

$$F_L(i) = 0.25 - 2^{i-0.5}/N_c$$

$$F_H(i) = 0.25 + 2^{i-0.5}/N_c$$

where $i = 1, 2, ..., \log_2(N_c/8)$, N_c is the width of image which is a power of 2. Note that $0 < F_L(i) < 0.25$ and $0.25 \leq F_H(i) < 0.5$.

For an image with 256 columns, for example, a total of 60 filters can be used - 6 orientations and (5 + 5) frequencies. Note that in this paper we set the value of the bandwidth b of the Gabor filter to 1 octave.

2.2 Feature Extraction

Jain [2] suggested to use a nonlinear sigmoidal function,

$$\tanh(\alpha t) = \frac{1 - e^{-2\alpha t}}{1 + e^{-2\alpha t}} \tag{3}$$

which saturates the output of the filters.

Jain [2] also suggested to compute the average absolute deviation (AAD) for each filtered image. We use Gaussian smoothing function which is given by

$$g(x, y) = exp\left\{-\frac{x^2 + y^2}{2\sigma^2}\right\} \tag{4}$$

where σ is the standard deviation which determines the (linear) size of the receptive field (window size).

We choose $\sigma = 3\sigma_s$ where σ_s is the scale parameter of Gabor filter given by (2) as similar to the recommendation, $\sigma = 2\sigma_s$, by [6].

2.3 Clustering

The final step is to cluster the pixels into a number of clusters representing the texture regions. Although Jain used CLUSTER algorithm [2], we use the k-means algorithm. The algorithm of k-means is as follows:

1. Initialize centroids of K-clusters randomly.

2. Assign each sample to the nearest centroid.

3. Calculate centroids (means) of K-clusters.

4. If centroids are unchanged, done. Otherwise, go to step 2.

Furthermore, we include the spatial coordinates of the pixels as two additional features to take into account the spatial adjacency information in the clustering process as proposed by [2].

3 Experimental Results

The multi-channel image segmentation system mentioned above was implemented and tested against textured images from the Brodatz album [5].

Figure. 1 shows an original image obtained by Brodatz album [5]. Figure 2 shows the filtered images by Gabor filters. Figure 3 shows a gaussian smoothed image of Figure 2 (c). Finally, figure 4 shows the segmentation result.

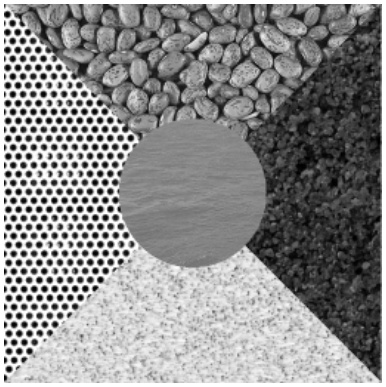

Figure 1: Five texture Brodatz image of size 256x256

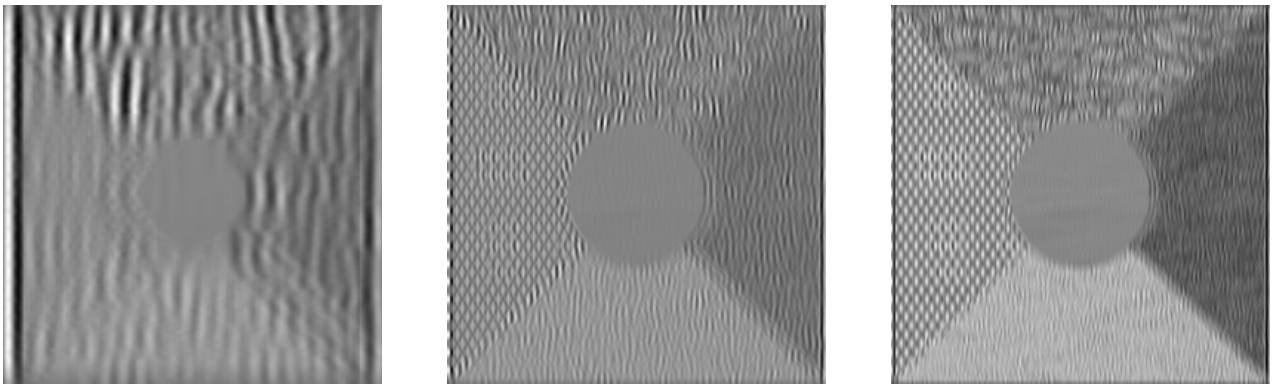

Figure 2: (a) shows a filtered image at frequency $F_L(5)$ and orientation of 0°. (b) shows a filtered image at frequency $F_L(1)$ and orientation of 0°. (c) shows a filtered image at frequency $F_H(5)$ and orientation of 0°. Note $F_L(5) < F_L(1) < F_H(5)$.

Figure 3: Smoothed Gabor response at frequency $F_L(1)$ and orientation of 0°. $\sigma = 10$.

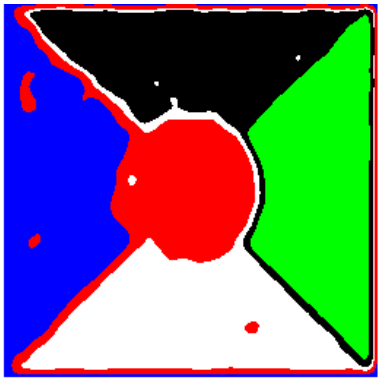

Figure 4: Segmentation result with 30° orientation separation, $\gamma = 1$, $b = 1$.

4 Discussion

K-means clustering often did not output the desired segmentation due to the random initialization, and we had to run programs several times to obtain good results. Using another criteria might resolve this problem. This method is an unsupervised technique, but we still needed to supervise the number of segments K. The nonlinear transformation at step 2 did not affect big differences, therefore, this may be able to be skipped.

The filtering methods such as Gabor filter, Gaussian filter, average filter must take into account how to process pixels at outer circumference of images. The conv2 function of matlab assumed as pixels at out of the image has 0 intensity values, and it caused bad segmentation often for pixels at outer circumference (The pixels at outer circumference will usually have lower values than other usual middle placed pixels in spite of the intensities of the original image.) This is a defect of a Filtering Method based Image Segmentation system.

This program worked reasonably fast, 9.874256 seconds, for 256x256 sized images, but took 219.196721 seconds for 512x512 sized images. The electric version of this document and source codes is available at http://note.sonots.com/index.php?SciSoftware%2FGaborTextureSegmentation

Bibliography

[1] Perona and Malik, "Preattentive texture discrimination with early vision mechanisms," *J. Opt. Soc. Am. A*, Vol. 7, No. 5, May 1990 http://mplab.ucsd.edu/~marni/Igert/Malik_Perona_1990.pdf

[2] A. K. Jain, F. Farrokhnia, "Unsupervised texture segmentation using Gabor filters," *Pattern Recognition*, vol. 24, no. 12, pp.1167-1186, 1991

[3] J.G. Daugman: "Uncertainty relations for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters", *Journal of the Optical Society of America A*, 1985, vol. 2, pp. 1160-1169.

[4] D. Clausi, M. Ed Jernigan, "Designing Gabor filters for optimal texture separability," *Pattern Recognition*, vol. 33, pp. 1835-1849, 2000.

[5] P. Drodatz, "Textures: A Photographic Album for Artists and Designers," "'Dover"', New York, 1966. http://www-dbv.informatik.uni-bonn.de/image/segmentation.html

[6] Jianguo Zhang, Tieniu Tan, Li Ma, "Invariant texture segmentation via circular gabor filter", *Proceedings of the 16th IAPR International Conference on Pattern Recognition (ICPR)*, Vol II, pp. 901-904, 2002. http://www.dcs.qmul.ac.uk/~jgzhang/ICPR_857.pdf

[7] Gabor filter Applet. http://www.cs.rug.nl/~imaging/simplecell.html

5 Appendix: Source Code

5.1 GaborTextureSegment.m (main)

```
% GaborTextureSegment: Texture Segmentation using Gabor Filters
%
%  seg = GaborTextureSegment(I,K,Sx,Sy,F,Theta,FUN,Wx,Wy,sigma)
%
%  Input and output arguments ([]'s are optional):
%   I     (matrix) of size NxM: Input Image of size NxM.
%   K     (scalar): The # of expected segments. A param for k-means.
%
%   gamma (scalar): The spatial aspect ratio, x to y.
%   Lambda (vector): The wavelengthes of the sinusoidal functions.
%   b      (scalar): The spatial frequency band-width (in octaves)
%   Theta  (vector): The orientations of the gabor filters.
%   phi    (scalar): The phase offset. 0 is real part of Gabor filter or
%    even-symmetric, pi/2 is imaginary part of Gabor filter or
%    odd-symmetric.
%   shape  (string): shape option for conv2 to apply Gabor filter to image.
%    'same' or 'valid' or 'full'. See help conv2.
%   Note: sigma (scalar): The spread of Gabor filter or the standard
%    deviation of Gaussian is automatically computed from lambda and b.
%
%   seg    (matrix) of size NxM: The segmented image. Each element has its
%     partition id.
%
% Author : Naotoshi Seo, sonots@umd.edu
% Date   : Oct, 2006
function seg = GaborTextureSegment(I, K, gamma, Lambda, b, Theta, phi, shape)
 [nRow, nCol, C] = size(I);

 % Step 1. Gabor Filter bank
 i = 0;
 for lambda = Lambda
     for theta = Theta
         i = i + 1;
         D = gabor2(I, gamma, lambda, b, theta, phi, shape);
         % Normalize into [0, 1]
         D = D - min(reshape(D, [], 1)); D = D / max(reshape(D, [], 1));
         %figure; imshow(uint8(D * 255));
         % Adjust image size to the smallest size if 'valid' (Cut off)
         if (isequal(shape, 'valid') && i >= 2)
             [nRow, nCol, C] = size(O(:, :, i-1));
             [Nr, Nc, C] = size(D);
             DNr = (Nr - nRow)/2;
             DNc = (Nc - nCol)/2;
             D = D(1+floor(DNr):Nr-ceil(DNr), 1+floor(DNc):Nc-ceil(DNc));
         end
         O(:, :, i) = D;
     end
 end
```

```
[nRow, nCol, N] = size(O);

% Step 2. Energy (Feature Extraction)
% Step 2-1. Nonlinearity
for i=1:N
    D = O(:, :, i);
    alpha = 1;
    D = tanh(double(O(:, :, i)) .* alpha); % Eq. (3). Input is [0, 1]
    % Normalize into [0, 1] although output of tanh is originally [0, 1]
    D = D - min(reshape(D, [], 1)); D = D / max(reshape(D, [], 1));
    %figure; imshow(uint8(D * 255));
    O(:, :, i) = D;
end

% Step 2-2. Smoothing
for i=1:N
    D = O(:, :, i);
    lambda = Lambda(floor((i-1)/length(Theta))+1);
    % (1) constant
    % sigma = 5;
    % (2) Use lambda. 0.5 * lambda should be near equal to gabor's sigma
    % sigma = .5 * lambda;
    % (3). Use gabor's sigma
    sigma = (1 / pi) * sqrt(log(2)/2) * (2^b+1) / (2^b-1) * lambda;
    sigma = 3 * sigma;
    D = gauss2(D, sigma, shape);  % Instead of Eq, (4) Avg filter
    % Normalize into [0, 1]
    D = D - min(reshape(D, [], 1)); D = D / max(reshape(D, [], 1));
    %figure; imshow(uint8(D * 255));
    % Adjust image size to the smallest size if 'valid' (Cut off)
    if (isequal(shape, 'valid') && i >= 2)
        [nRow, nCol, C] = size(P(:, :, i-1));
        [Nr, Nc, C] = size(D);
        DNr = (Nr - nRow)/2;
        DNc = (Nc - nCol)/2;
        D = D(1+floor(DNr):Nr-ceil(DNr), 1+floor(DNc):Nc-ceil(DNc));
    end
    P(:, :, i) = D;
end
O = P; clear P;
[nRow, nCol, N] = size(O);


% Step 3. Clustering
% Step 3-1. Adding coordinates information to involves adjacency
for i=1:nRow
    for j=1:nCol
        O(i, j, N+1) = i / nRow; % [0, 1]
        O(i, j, N+2) = j / nCol;
    end
end
```

```
 % Step 3-2. Clustering
 data = reshape(O, [], size(O, 3)); % (# of data) x (# of dimensions)
 [cluster, codebook] = kmeans_light(data, K);
 seg = reshape(cluster, nRow, nCol, 1); % 2D
end
```

5.2 gabor2.m

```
% gabor2: 2D Gabor filter
%   The Gabor filter is basically a Gaussian, modulated by a complex sinusoid
%
%  G = gabor2(I,Sx,Sy,f,theta,FUN)
%
%  Input and output arguments ([]'s are optional):
%   I     (matrix) of size NxM: Input Image of size NxM.
%   gamma (scalar): The spatial aspect ratio, x to y.
%   lambda(scalar): The wavelength of the sinusoidal function.
%   b     (scalar): The spatial frequency band-width (in octaves)
%   theta (scalar): The orientation of the gabor filter.
%   phi   (scalar): The phase offset. 0 is real part of Gabor filter or
%    even-symmetric, pi/2 is imaginary part of Gabor filter or
%    odd-symmetric.
%   Note: sigma (scalar): The spread of Gabor filter or the standard
%    deviation of Gaussian is automatically computed from lambda and b.
%  [shape] (strings): Shape for conv2. See help conv2. Default is 'same'.
%
%   GO    (matrix) of size NxM: Output images which was applied Gabor
%    filters. This is  the magnitude response.
%   [GF]  (matrix) of size (2Sx+1)x(2Sy+1): Gabor filter.
%
% Author : Naotoshi Seo, sonots@umd.edu
% Date   : Oct, 2006
function [GO, GF] = gabor2(I, gamma, lambda, b, theta, phi, shape);
if nargin < 7, shape = 'same';, end;
if isa(I, 'double') ~= 1, I = double(I); end

sigma = (1 / pi) * sqrt(log(2)/2) * (2^b+1) / (2^b-1) * lambda;
Sy = sigma * gamma;
for x = -fix(sigma):fix(sigma)
    for y = -fix(Sy):fix(Sy)
        xp = x * cos(theta) + y * sin(theta);
        yp = y * cos(theta) - x * sin(theta);
        GF(fix(Sy)+y+1,fix(sigma)+x+1) = ...
            exp(-.5*(xp^2+gamma^2*yp^2)/sigma^2) * cos(2*pi*xp/lambda+phi) ...
           ; %/ (2*pi*(sigma^2/gamma));
           % Normalize if you use different sigma (lambda or b)
    end
end

GO = conv2(I, double(GF), shape);
```

5.3 gauss2.m

```
% gauss2: 2D Gaussian filter
%
%  G = gauss2(I, sigma)
%
%  Input and output arguments ([]'s are optional):
%   I     (matrix) of size NxM: Input Image of size NxM.
%   sigma (scalar): The spread of filter or the standard
%    deviation of Gaussian.
%    Hint: Large sigma blurrs more.
%  [shape] (strings): Shape for conv2. See help conv2. Default is 'same'.
%
%   GO    (matrix) of size NxM: Output images which was applied Gabor
%    filters. This is  the magnitude response.
%   [GF]  (matrix) of size (2Wx+1)x(2Wy+1): Gaussian Filter
%
% Author : Naotoshi Seo, sonots@umd.edu
% Date   : Oct, 2006
function [GO, GF] = gauss2(I, sigma, shape);
if nargin < 4, shape = 'same';, end;
if isa(I,'double') ~= 1
    I = double(I);
end

for x = -fix(sigma):fix(sigma)
    for y = -fix(sigma):fix(sigma)
        %xPrime = x * cos(theta) + y * sin(theta);
        %yPrime = y * cos(theta) - x * sin(theta);
        GF(fix(sigma)+x+1,fix(sigma)+y+1) = exp(-.5*((x/sigma)^2+(y/sigma)^2)) ...
           ; % / (2*pi*sigma*sigma); % Nomalize
    end
end

GO = conv2(I,double(GF), shape);
```

5.4 kmeans_light.m

```
% kmeans_light: K-means clustering using euclid distance.
%
%  [dataCluster codebook] = kmeans_light(data, K, stopIter)
%
%  Input and output arguments ([]'s are optional):
%   data        (matrix) of size NxD. N is the number of data (classifiee)
%                vectors,and D is the dimension of each vector.
%   K           (scalar) The number of clusters.
%   stopIter    (scalar) The threshold [0, 1] to stop learning iterations.
%                Default is .05, and smaller makes continue interations.
%   dataCluster (matrix) of size Nx1: integers indicating the cluster indicies.
%                dataCluster(i) is the cluster id for data item i.
%   codebook    (matrix) of size KxD: set of cluster centroids, VQ codewords.
%
% See also: autolabel.m
%
% Author : Naotoshi Seo
% Date   : April, 2006
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dataCluster codebook] = kmeans_light(data, K, stopIter)
 if nargin < 3,
     stopIter = .05;
 end
 [N dim] = size(data);
 if K > N,
     error('K must be less than or equal to the # of data');
 end

 % Initial codebook
 codebook = data(randsample(N, K), :);

 improvedRatio = Inf;
 distortion = Inf;
 iter = 0;
 while true
     % Calculate euclidean distances between each sample and each codeword
     d = eucdist2(data, codebook);
     % Assign each sample to the nearest codeword (centroid)
     [dataNearClusterDist, dataCluster] = min(d, [], 2);
     % distortion. If centroids are unchanged, distortion is also unchanged.
     % smaller distortion is better
     old_distortion = distortion;
     distortion = mean(dataNearClusterDist);

     % If no more improved, break;
     improvedRatio = 1 - (distortion / old_distortion);
     fprintf('%d: improved ratio = %f\n', iter, improvedRatio); iter = iter + 1;
     if improvedRatio <= stopIter, break, end;

     % Renew codebook
```

```
    for i=1:K
        % Get the id of samples which were clusterd into cluster i.
        idx = find(dataCluster == i);
        % Calculate centroid of each cluter, and replace codebook
        codebook(i, :) = mean(data(idx, :));
    end
end

%%%% Euclidean distance matrix between row vectors in X and Y %%%%%%%
%  Input and output arguments
%   X: NxDim matrix
%   Y: PxDim matrix
%   d: distance matrix of size NxP
function d=eucdist2(X,Y);
U=~isnan(Y); Y(~U)=0;
V=~isnan(X); X(~V)=0;
d=abs(X.^2*U'+V*Y'.^2-2*X*Y');
```

5.5 demo_GaborTextureSegment.m

```
function demo_GaborTextureSegment
imfile = 'data.10.png';
I = imread(imfile); K = 5;
%I = rgb2gray(I);
%I = imresize(I, [256 256]);
[Nr, Nc, D] = size(I);
gamma = 1; b = 1; Theta = 0:pi/6:pi-pi/6; phi = 0; shape = 'valid';
%[4 8 16 ...] sqrt(2) % Jain
% Lambda = Nc./((2.^(2:log2(Nc/4))).*sqrt(2));
% J. Zhang
J = (2.^(0:log2(Nc/8)) - .5) ./ Nc;
F = [ (.25 - J) (.25 + J) ]; F = sort(F); Lambda = 1 ./ F;
seg = GaborTextureSegment(I, K, gamma, Lambda, b, Theta, phi, shape);
%imseg = uint8(seg) * floor(255 / K); % cluster id to gray scale (max 255)
[nRow, nCol] = size(seg);
color = [0 0 0; 255 255 255; 255 0 0; 0 255 0; 0 0 255]; % 5 colors reserved
imseg = zeros(nRow*nCol, 3);
for i=1:K
    idx = find(seg == i);
    imseg(idx, :) = repmat(color(i, :), [], length(idx));
end
imseg = reshape(imseg, nRow, nCol, 3);
figure; imshow(imseg);
imwrite(imseg, sprintf('seg.%s', imfile));
end
```