

1. In this problem we are required to create an edge detector and train it using the various edge patches we have obtained from the Berkeley Segmentation Database [1]. We begin by creating oriented first and second order derivative of the Gaussian 2D filter for different scales ($\sigma = 1, 2$) and orientations measures ($\theta = \pi/4, \pi/2, 3\pi/4$). In addition to these 12 Gaussian filters we also create another filter that is a square matrix of all 1s. Next we convolve each of training image patches (with edges and non-edges) obtained from [1] to generate responses. These responses are stored in a 13 dimensional feature vector.

```

1 clear all;
2 close all;
3 clc;
4
5 basedir = '/home/subh/courses/cap5415/ps3/edge_examples_subset/';
6 etrain = 'EDGE.TRAIN/';
7 netrain = 'NOT.EDGE.TRAIN/';
8
9 dirinfo = [basedir etrain];
10 fmetrain = buildFeatureMatrix (dirinfo, '*.png');
11
12 dirinfo = [basedir netrain];
13 fmnetrain = buildFeatureMatrix (dirinfo, '*.png');
14
15 pts = [fmetrain; fmnetrain];
16 labels = [fmetrain(:,13); -fmnetrain(:,13)];
17 save('PtsLblsWts.mat','pts','labels');
```

Listing 1: Matlab script for Problem 1

```

1 function fmatrix = buildFeatureMatrix (dirinfo, files)
2 % This function creates a feature matrix for all files in the training
3 % directory. number of theta and sigma could be increased/decreased to
4 % increase the size of the feature matrix
5 pi = 3.1415;
6 theta = [pi/4 pi/2 3*(pi/4)];
7 sigma = [1 2];
8
9 flidx = dir([dirinfo files]);
10 nImg = length(flidx);
11
12 for icnt = 1:nImg
13     flname = [dirinfo flidx(icnt).name];
14     fvector = computeFeatureVector(flname, theta, sigma);
15     fmatrix(icnt,:) = fvector(:);
16 end
17 return;
```

Listing 2: Matlab script for Building Feature Matrices from the feature vectors

```

1 function fvector = computeFeatureVector(flname, theta, sigma)
2 % This function computes a feature vector of 12 dimensions for an image
3 % patch. For each scale value and orientation value, the response of the
4 % given image patch to both first and second directional derivative of
5 % Gaussian filters is computed. flname is the absolute path of the file
6 % containing the image patch.
7
8 % img is 13 x 13 matrix
9 img = imread (flname);
10 img = im2double (img);
11 fvector = ones(1,2* length(theta) * length(sigma) +1);
12 % Counting features
13 lcnt = 1;
14 % Create feature vector for each image
15 for jcnt = 1:length(theta) % for 3 theta (orientation)
16     for kcnt = 1:length(sigma) % for 2 sigma (scales)
17         [x y] = meshgrid(-6: 6); % Create a 13 x 13 template
18
19         var = sigma(kcnt)^2;
20         tcos = cos(theta(jcnt));
```

```

21     tsin = sin(theta(jcnt));
22     gfexp = exp(-(x.^2 + y.^2)/(2*var));
23
24     % Computing the first derivative of Gaussian wrt theta
25     % f1 = (gfexp./var).*(x.*tcos + y.*tsin);
26     f1 = 2.*gfexp.*(x.*tcos + y.*tsin);
27     f1 = f1 /sum(abs(f1(:)));
28
29     % Computing the second derivative of Gaussian wrt theta
30     f2 = (gfexp./var).*((2*(x.^2) - var) + 2*(x.*y)*tcos*tsin + (2*(y.^2) - var));
31     f2 = f2/ sum(abs(f2(:)));
32
33     % Convolve filters f1 and f2 with each image patch and store it in a
34     % feature vector
35     feat = conv2 (img, f1, 'valid');
36     fvector(lcnt) = abs(feat(:));
37     lcnt = lcnt + 1;
38     feat = conv2 (img, f2, 'valid');
39     fvector(lcnt) = abs(feat(:));
40     lcnt = lcnt + 1;
41 end
42 end
43 return;

```

Listing 3: Matlab script for Computing Feature Vector from the edge and non edge Patches

- Here we use the inputs from the previous problem to test our edge detector. The objective of this exercise is to run the edge detection program over a set of image patches that have edge and non-edge pixels. We evaluate how clearly our edge detection program identifies an image and how often the program detects a non-edge portion in an image patch as a patch containing edge. We use a logistic regression function to find out how every filter fares in determining edges in image patches. Based on each of these filter's performance in detecting an edge, they are assigned weights by the logistic regression function. A Receiver Operating Characteristic curve is then plotted to visually interpret the rate of detection of an edge in an image patch against that of misinterpretation of a non-edge pixel as an edge pixel in an image patch.

```

1  clear all;
2  close all;
3  clc;
4
5  basedir = '/home/subh/courses/cap5415/ps3/edge_examples_subset/';
6  etest = 'EDGE_TEST/';
7  netest = 'NOT_EDGE_TEST/';
8  % Obtain Data for learning
9  load('PtsLbLsWts.mat');
10
11 % Determining the weights by performing logistic regression
12 weights = logisticRegression(pts, labels);
13 save('PtsLbLsWts.mat', 'weights');
14
15 % Determine feature vector from the test images that have edges
16 dirinfo = [basedir etest];
17 fmetest = buildFeatureMatrix (dirinfo, '*.png');
18
19 % For non edges
20 dirinfo = [basedir netest];
21 fmnetest = buildFeatureMatrix (dirinfo, '*.png');
22
23 % These would be inputs to the function produceROCInputs which would return
24 % the probability of detection and probability of false alarm for each
25 % image patch.
26 icnt = 1;
27 for thd = 0.001: 0.005: 0.99
28     pd(icnt) = produceROCInputs(fmetest, weights/1000, thd);
29     pfa(icnt) = produceROCInputs(fmnetest, weights/1000, thd);
30     icnt = icnt + 1;
31 end
32
33 plot(pfa, pd);
34 title('Receiver Operating Characteristic Curve');

```

```

35 ylabel('Probability of Detection');
36 xlabel('Probability of False Alarm');

```

Listing 4: Matlab script for Problem 2

```

1 function w = logisticRegression(pts, labels)
2 % LOGISTIC_REGRESSION -
3 % initial value of the weights
4 w0 = [1; 0; 5; 1; 0; 5; 0; 0; 0; 0; 1; 1; 1];
5 w = w0;
6 step = 0.125;
7 max_num_iter=1000;
8 stop_flag = 0;
9 citer=1;
10
11 while (~stop_flag)
12     g = calculateGradientLogLoss(pts, labels, w);
13     % This is required to find out if L is converging over iterations,
14     % number of iterations and step-sizes are modified according to this
15     % value.
16     L = calculateLogLoss(pts, labels, w)
17     fprintf(1, 'L :%e\n', L);
18     w = w - step*g;
19     if(citer >= max_num_iter)
20         stop_flag=1;
21     end
22     citer=citer+1;
23 end

```

Listing 5: Matlab script for Performing Logistic Regression

```

1 function L=calculate_log_loss(pts, labels, theta)
2 % LOG_LOSS -
3 L = sum(log(1+exp(-labels.*(pts*theta))));
4 return;

```

Listing 6: Matlab script for Calculating Logloss function used in Logistic Regression

```

1 function g=calculate_gradient_log_loss(pts, labels, theta)
2 % CALCULATE_GRADIENT_LOG_LOSS
3 resp = labels.*(pts*theta);
4 p_y = 1./(1+exp(-resp));
5 g = ((-labels.*(1-p_y))*pts)';

```

Listing 7: Matlab script for Modified Gradient Logloss function

```

1 function [p] = produceROCInputs(pts, w, thd)
2 % This function calculates the probability of each feature vector returning
3 % an edge for an image patch. pts contains 13 dimensional feature vector, w
4 % contains weights (13 dimensional). labels is a 1 dimensional
5 % vector containing if the image patch is actually an edge or not
6
7 % Get probabilities of all feature vector for being an edge or not, this
8 % is a 1-dimensional vector conatinng the same number of rows as pts
9 pli = 1./ (1+exp(-(pts*w)));
10 p = 0;
11 % Find out the probability for each class
12 for icnt = 1:length(pli)
13     if (pli(icnt) >= thd)
14         p = p + 1;
15     end
16 end
17 p = double(p/length(pli));
18 return;

```

Listing 8: Matlab script for Generating ROC curve

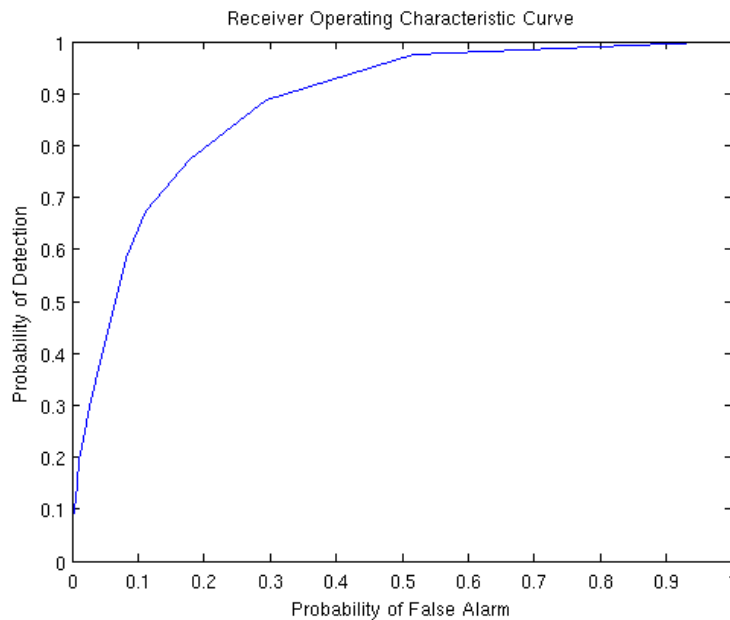


Figure 1: Receiver Operating Characteristic Curve

3. After running our edge detector on both patches containing edges and non-edges, we would test how the edgedetector responds to a real image. We obtain our test images from the [1]. Firstly, we determine the responses of a test image with the set of filters we had created for problem 1 by convolving the image with each of these filters except for the last filter in the set(which is a 13×13 square matrix). This is because the edge detector algorithm does not have any responses with the bias term. The responses are then used with their filters' corresponding weights to compute the probability of encountering an edge pixel in the image. If this probability is greater than a certain threshold(we determine this after several trial and errors), we mark the corresponding pixel as an edge pixel. The threshold is determined out to be 0.55. The rest of the pixels are marked as white background. Finally we compare our results against that done manually by humans.

```

1 clear all;
2 close all;
3 clc;
4
5 basedir = '/home/subh/courses/cap5415/ps3/';
6 imgfiles = 'BSDS300/images/test/';
7 % Obtain Data for learning
8 load('PtsLbIsWts.mat');
9
10 % Pick an image randomly from the dataset
11 files = dir([imgfiles '*.jpg']);
12 flidx = find(~[files.isdir]);
13 nImg = length(flidx);
14 idx = nImg - round(nImg*rand(1));
15 flname = files(flidx(idx)).name;
16
17 edgeimg = detectEdge([basedir imgfiles flname], weights, 0.55);
18 imwrite(edgeimg,['edge-' flname '.png'], 'png');
19 img = imread([imgfiles flname]);
20 imshow(uint8(img), []); figure; imshow(uint8(edgeimg), []);

```

Listing 9: Matlab script for Problem 3

```

1 function edgeImg = detectEdge (flname, weights, thd)
2     weights = weights/1000;
3     pi = 3.1415;
4     theta = [pi/4 pi/2 3*(pi/4)];

```

```

5  sigma = [1 2];
6
7  % load the image
8  img = im2double(rgb2gray(imread(flname)));
9  [width height] = size(img);
10
11  lcnt = 1;
12  for jcnt = 1:length(theta) % for 3 theta (orientation)
13      for kcnt = 1:length(sigma) % for 2 sigma (scales)
14          [x y] = meshgrid(-6: 6); % Create a 13 x 13 template
15
16          var = sigma(kcnt)^2;
17          tcos = cos(theta(jcnt));
18          tsin = sin(theta(jcnt));
19          gfexp = 2*exp(-(x.^2 + y.^2));
20
21          % Computing the first derivative of Gaussian wrt theta
22          f1 = (gfexp./var).*(x.*tcos + y.*tsin);
23          f1 = f1/sum(abs(f1(:)));
24
25          % Computing the second derivative of Gaussian wrt theta
26          f2 = (gfexp./var).*((2*(x.^2) - var) + 2*(x.*y)*tcos*tsin + (2*(y.^2) - var));
27          f2 = f2/sum(abs(f2(:)));
28
29          % Convolve filters f1 and f2 with each image patch and store it in a
30          % feature vector
31          feat = conv2 (img, f1, 'same');
32          resp(:,lcnt) = abs(feat(:));
33          lcnt = lcnt + 1;
34          feat = conv2 (img, f2, 'same');
35          resp(:,lcnt) = abs(feat(:));
36          lcnt = lcnt + 1;
37      end
38  end
39  %feat = conv2 (img, ones(13,13), 'same');
40  %resp(:,lcnt) = abs(feat(:));
41
42  [fsize nfilter] = size(resp);
43
44  % Responses generated, now get the edge
45  % Create a blank template for the output
46  tplt = 255*ones(size(img(:)));
47  % Compare likelihood of each filter coming up with an edge across a threshold
48  Sx = zeros(fsize,1);
49  for jcnt =1: nfilter
50      x = resp(:,jcnt);
51      Sx = Sx + weights(jcnt).*x;
52  end
53  p = 1./(1+ exp(-Sx));
54  for icnt = 1:fsize
55      if (p(icnt,1) >= thd)
56          tplt(icnt,1) = 0;
57      end
58  end
59  edgeImg = reshape(tplt, [width height]);
60  return

```

Listing 10: Matlab script for Detecting Edge inside an image

```

1  function CompareResults(filename, outputname)
2      seg = readSeg(filename);
3      bmap = seg2bmap(seg);
4      bmap = xor(bmap, ones(size(bmap)));
5      figure;imshow(uint8(bmap), []);
6      imwrite(bmap, outputname);
7  return;

```

Listing 11: Matlab script for Displaying a Human Segmented Image

4. We need to derive the derivative of log-likelihood of the following expression

$$P[t = k|x] = \frac{e^{-\theta_k \cdot x}}{\sum_{j=1}^M e^{-\theta_j \cdot x}}$$

The likelihood of all x_i in x , being correctly labeled k_i in k is given by:

$$\prod_{i=1}^N P[t_i = k_i|x_i] = \prod_{i=1}^N \frac{e^{-\theta_{k_i} \cdot x_i}}{\sum_{j=1}^M e^{-\theta_j \cdot x_i}}$$

Therefore the Log Likelihood is given by:

$$L = \log \left(\prod_{i=1}^N P[t_i = k_i|x_i] \right) = \sum_{i=1}^N \log \left(\frac{e^{-\theta_{k_i} \cdot x_i}}{\sum_{j=1}^M e^{-\theta_j \cdot x_i}} \right) \quad (1)$$

The derivative of the Log-likelihood wrt $\theta_{j,k}$ (which is the k^{th} entry in θ_j) could be obtained by partially differentiating 1 wrt $\theta_{j,k}$

$$\begin{aligned} \frac{\partial L}{\partial \theta_{j,k}} &= \sum_{i=1}^N \frac{\partial}{\partial \theta_{j,k}} \left[\log(e^{-\theta_{k_i} \cdot x_i}) - \log\left(\sum_{j=1}^M e^{-\theta_j \cdot x_i}\right) \right] \\ &= - \sum_{i=1}^N \frac{\partial}{\partial \theta_{j,k}} \log\left(\sum_{j=1}^M e^{-\theta_j \cdot x_i}\right) \\ &= x_k \sum_{i=1}^N \frac{e^{-\theta_j \cdot x_i}}{\sum_{j=1}^M e^{-\theta_j \cdot x_i}} \quad (x_k \text{ being the } k^{th} \text{ value of } x) \end{aligned}$$

References

- [1] D. Martin and C. Fowlkes and D. Tal and J. Malik, *Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistic* Proc. 8th Int'l Conf. Computer Vision, 2001, July, Vol 2 pp 416–423.

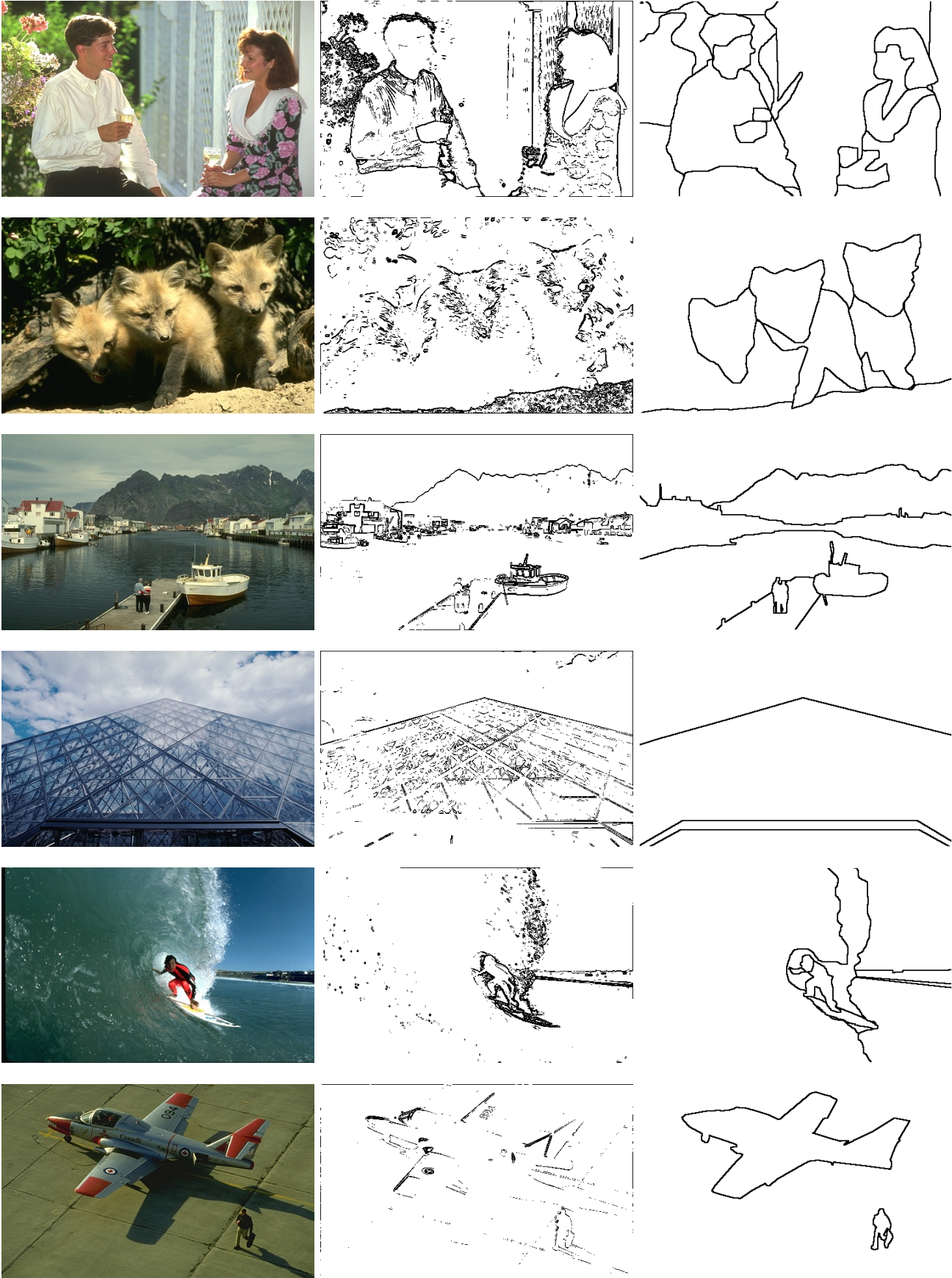


Figure 2: Comparison of Edge Detection Results: Leftmost column is the original image, Rightmost show the result obtained by Human Detection and the central column show the results obtained by the edge detector implemented by us