

Image Processing

lab 3

Klaas Kliffen Jan Kramer

December 10, 2015

Exercise 1 – 1-D wavelet transforms

- a. The algorithm given in the assignment can be represented as a filter bank based on the Haar scaling and wavelet vectors. Normally the whole input would be convolved with these vectors. However since the two-point sums and differences are taken this is convolution is combined with downscaling. Our implementation of a j -scale DWT is based on this algorithm by applying this “filter bank” j times. Otherwise its implementation is rather trivial.

```
1 % function to perform 1D Haar wavelet transform
2 function retval = IPdwt(x,s)
3     sqrt2 = sqrt(2);
4     out = x;
5
6     initl = length(out);
7
8     for i = 1 : s
9         % Get the odd and even elements
10        odds = out(1:2:initl);
11        evens = out(2:2:initl);
12        % Calculate the means and details
13        sums = (odds + evens);
14        diffs = (odds - evens);
15        % Put the new values
16        out(1:initl) = [sums, diffs] / sqrt2;
17        initl /= 2;
18    end
19
20    retval = out;
21 end
```

- b. For the inverse wavelet transform, the initial length is set to the end of the length of the last sum vector. Then for each step the sum and difference vectors are retrieved from the input vector. Two new component vectors are created for the values. These need to be scaled again to retrieve the right results. The component vectors are then interleaved and replace their original values in the input vector. This is repeated until the original scale of the transformation is reached.

```
1 % function to perform 1D Haar wavelet transform
```

```

2  function retval = IPidwt(x,s)
3      sqrt2 = sqrt(2);
4      out = x;
5
6      % Determine the initial length
7      initl = length(x) / (2^(s-1));
8
9      for i = 1 : s
10         % Retrieve the sums and the differences
11         sums = out(1:initl/2);
12         diffs = out(initl/2+1:initl);
13         % Calculate and scale the result
14         plus = (sums+diffs)/sqrt2;
15         mins = (sums-diffs)/sqrt2;
16         % Combine the new values
17         combined = zeros(initl,1);
18         combined(1:2:end) = plus;
19         combined(2:2:end) = mins;
20         % Store them in the output matrix
21         out(1:initl) = combined;
22         % Increase the length or the next iteration
23         initl *= 2;
24     end
25
26     retval = out;
27 end

```

Exercise 2 – 2-D wavelet transforms

- a. According to the Section 7.5 in the book the extension of 1D DWT to 2D is simple, because of the separable scaling and wavelet functions. It also mentions that the 2D DWT can be computed by first doing a 1D DWT of the columns and then doing a 1D DWT of the rows. Note that one could also calculate the DWT first for rows and then for columns. Our implementation uses this fact. In each iteration j the algorithm of exercise 1 is applied to each row and then to each column to get the approximation, the horizontal detail, the vertical detail and the diagonal detail.

```
1  % function to perform 2D Haar wavelet transform
2  function retval = IPdwt2(x, j)
3  % note that x should be double instead of uint, because
4  % the result can get negative
5  out = x;
6  coef = 1/sqrt(2);
7
8  initrow = size(out, 1);
9  initcol = size(out, 2);
10
11  for i = 1 : j
12      % 1D DWT along the rows
13      odds_c = out(1:initrow, 1:2:initcol);
14      evens_c = out(1:initrow, 2:2:initcol);
15      sums = (odds_c + evens_c);
16      diffs = (odds_c - evens_c);
17      out(1:initrow, 1:initcol) = [sums, diffs] * coef;
18
19      % 1D DWT along the columns
20      odds_r = out(1:2:initrow, 1:initcol);
21      evens_r = out(2:2:initrow, 1:initcol);
22      sums = (odds_r + evens_r) * coef;
23      diffs = (odds_r - evens_r) * coef;
24
25      mid_r = initrow / 2;
26      mid_c = initcol / 2;
27
28      % save the parts as in the figures in the book
29      % approximation image
30      out(1:mid_r, 1:mid_c) = sums(:, 1:mid_c);
31      % vertical detail
32      out(mid_r+1:initrow, 1:mid_c) = sums(:, mid_c+1:initcol);
33      % horizontal detail
34      out(1:mid_r, mid_c+1:initcol) = diffs(:, 1:mid_c);
35      % detail detail
36      out(mid_r+1:initrow, mid_c+1:initcol) = diffs(:, mid_c+1:initcol);
37
38      initrow = mid_r;
39      initcol = mid_c;
40  end
41
42  retval = out;
43
```

44 **end**

b. TODO: text

```
1 function retval = IPdwt2scale(x, j)
2 % calculate the dwt with a shifted image around 0 (assumes doubles)
3 out = x - 0.5;
4 out = IPdwt2(out, j);
5 out = out + 0.5;
6
7 initrow = size(out, 1);
8 initcol = size(out, 2);
9
10 % iterate through the levels in the image
11 for i = 1 : j
12     mid_r = initrow / 2;
13     mid_c = initcol / 2;
14
15     % contrast stretch the horizontal details
16     w = out(1:mid_r, mid_c+1:initcol);
17     out(1:mid_r, mid_c+1:initcol) = (w - min(min(w))) * (1 / (max(max(w)
18         )) - min(min(w))));
19
20     % contrast stretch the vertical details
21     w = out(mid_r+1:initrow, 1:mid_c);
22     out(mid_r+1:initrow, 1:mid_c) = (w - min(min(w))) * (1 / (max(max(w)
23         )) - min(min(w))));
24
25     % contrast stretch the diagonal details
26     w = out(mid_r+1:initrow, mid_c+1:initcol);
27     out(mid_r+1:initrow, mid_c+1:initcol) = (w - min(min(w))) * (1 / (
28         max(max(w)) - min(min(w))));
29
30     initrow = mid_r;
31     initcol = mid_c;
32 end
33
34 % contrast stretch the approximation image
35 w = out(1:initrow, 1:initcol);
36 out(1:initrow, 1:initcol) = (w - min(min(w))) * (1 / (max(max(w)) -
37     min(min(w))));
38
39 retval = out;
40 end
```

c. TODO: text

```
1
2 x = im2double(imread(' ../images/vase.tif'));
3
4 %% lab 3 ex 2abcd
5 y = IPdwt2(x, 3);
6 <<<<<< HEAD
7 imwrite(y, 'unscaled.png');
```

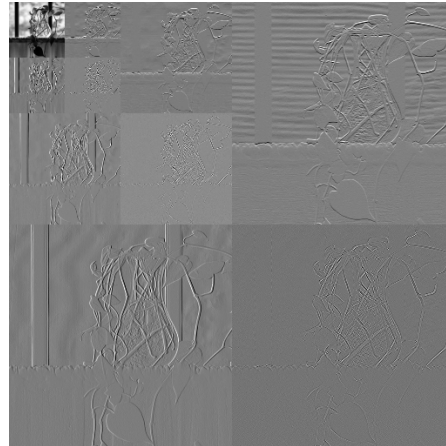
```

8  imwrite(im2uint8(IPdwt2scale(x, 3)), 'scaled.png');
9  imwrite(im2uint8(IPidwt2(y,3)), 'output.png');
10 =====
11 imwrite(y, 'unscaled.tif');
12 imwrite(im2uint8(IPdwt2scale(x, 3)), 'scaled.tif');
13 imwrite(im2uint8(IPidwt2(y,3)), 'output.tif');
14 % the difference
15 sum(sum(im2uint8(x - IPidwt2(y,3))))
16 >>>>>> 601721b5002403fcf64310be05990e1282626034

```



Original image



3 scale wavelet transform

Figure 1: 3 scale wavelet transform of the original image

- d. According to Section 7.5 the 2D inverse DWT can also be computed by using a 1D inverse DWT function. So similarly on how our 2D DWT implementation is based on our 1D DWT implementation, the 2D inverse DWT is also based on the 1D DWT implementation. Note however that the order of applying 1D DWT first to rows and then columns in the 2D DWT, has to be inverted in the 2D inverse DWT. Hence our implementation first applies a 1D inverse DWT to the columns and the one to the rows.

```

1  % inverse discrete wavelet transform
2  function retval = IPidwt2(x, s);
3  out = x;
4  coef = 1/sqrt(2);
5
6  [height, width] = size(x);
7  initrow = height / (2^(s-1));
8  initcol = width / (2^(s-1));
9
10 for i = 1 : s
11     mid_r = initrow / 2;
12     mid_c = initcol / 2;
13
14     % make sure we swap horizontal and vertical details
15     rowsums = [out(1:mid_r, 1:mid_c), out(mid_r+1:initrow, 1:mid_c)];

```

```

16     rowdiffs = [out(1:mid_r, mid_c+1:initcol), out(mid_r+1:initrow,
17               mid_c+1:initcol)];
18
19     % Calculate and scale the result
20     plus = (rowsums+rowdiffs);
21     mins = (rowsums-rowdiffs);
22     % Combine the new values
23     combined = zeros(initrow,initcol);
24     combined(1:2:end,:) = plus * coef;
25     combined(2:2:end,:) = mins * coef;
26     % Replace the values in the image
27     out(1:initrow,1:initcol) = combined;
28
29     % Do the same with the columns
30     colsums = out(1:initrow,1:mid_c);
31     coldiffs = out(1:initrow,mid_c+1:initcol);
32
33     % Calculate and scale the result
34     plus = (colsums+coldiffs);
35     mins = (colsums-coldiffs);
36     % Combine the new values
37     combined(:,1:2:end) = plus * coef;
38     combined(:,2:2:end) = mins * coef;
39     % Replace the values in the image
40     out(1:initrow,1:initcol) = combined;
41
42     initrow *= 2;
43     initcol *= 2;
44 end
45
46 retval = out;
47 end

```



Original image



Restoration of the image from figure 1

Figure 2: 3 scale wavelet transform restoration

Exercise 3 – Image Compression

- a. First the wavelet transform is performed on the input image. A threshold matrix consisting solely of the threshold value is used to find all pixels larger than the threshold in the transform. The thresholded image is then retrieved by pointwise multiplication of the results of comparing the threshold matrix with the wavelet transform. Since the approximation of the original matrix is also passed by the threshold, it needs to be reconstructed. This is done by copying the values from the wavelet transform back to the thresholded image. The compressed image is then retrieved by applying the inverse wavelet transform.

The root mean square error and the mean square signal to noise ratio are calculated from the given formulas. For the compression ratio the built-in function of entropy is used to calculate the entropy of the compressed and original image. The entropy is a value for how complex an image is. Compressing the image lowers the complexity. Dividing the original entropy by the compressed entropy yields a compression ratio.

```
1  % Function to compress an image using wavelet transform
2  function retval = IPwaveletcompress(img, scale, threshold)
3
4  [width,height] = size(img);
5  wl = width / (2^scale);
6  hl = height / (2^scale);
7
8  % Wavelet transform
9  wtrans = IPdwt2(img,scale);
10
11 % Construct a matrix for the threshold
12 thresholdmat = threshold * ones(size(img));
13 % Matrix containing 1 for pixels above the threshold
14 results = abs(wtrans) > thresholdmat;
15 % Perform the thresholding by elementwise multiplying
16 threshed = zeros(size(img));
17 threshed = wtrans .*results;
18
19 % Copy the original image part (for dark values in the original
20 threshed(1:hl,1:wl) = wtrans(1:hl,1:wl);
21
22 % Convert it back
23 compressed = IPidwt2(threshed, scale);
24
25 printf("Scale: %d Threshold: %f\n", scale, threshold);
26 error = compressed - img;
27 errorsq = error .* error;
28 rmse = sqrt(mean(mean(errorsq)));
29 printf("Root mean square error: %f\n",rmse);
30
31 squared = compressed .* compressed;
32 snr = sum(sum(squared)) / sum(sum(errorsq));
33 printf("Mean square signal to noise: %f\n",snr);
34
35 % Calculatute the compression
36 orig = entropy(im2uint8(img));
37 comp = entropy(im2uint8(threshed));
```

```

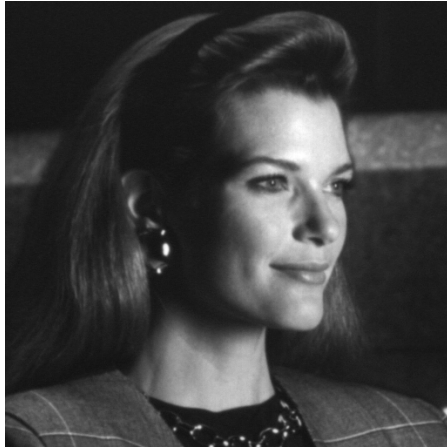
38 printf("Compress ratio: %f:1\n\n", orig/comp);
39
40
41 retval = compressed;
42
43 end

```

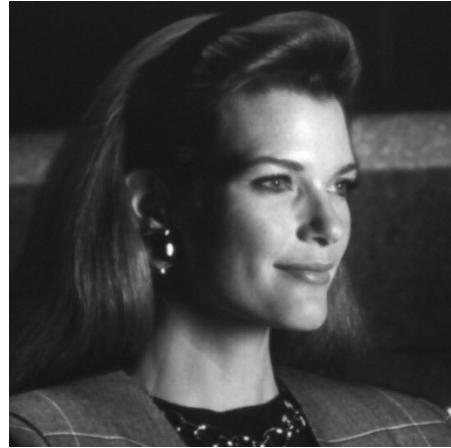
- b. The quantitative properties for several different scales and thresholds can be seen in table 1. Globally the signal to noise ratio decreases and the errors and compression ratio increase while increasing scale and threshold. Although for this image the increasing the scale past 9 was not possible, due to its size being a square of 512 pixels. It would seem that increasing it further would not compress the image any further past a compression ratio of 37. Increasing the threshold will yield in higher compression ratios. Although the signal to noise ratio decreases exponentiall, while the errors increase almost linearly.

Scale	Threshold	ϵ_{rms}	SNR	Compression ratio
1	0.02	0.0086	1277.34	2.59:1
3	0.02	0.0157	385.47	17.42:1
3	0.05	0.0249	152.09	24.87:1
3	0.10	0.0322	90.33	29.14:1
5	0.02	0.0220	194.71	34.00:1
7	0.02	0.0272	127.44	36.40:1
7	0.05	0.0575	27.627	113.18:1
7	0.10	0.1035	7.837	460.20:1
9	0.02	0.0321	91.02	36.55:1

Table 1: Quantitative compression quality for different scales and threshold



Original image



Scale: 1 Threshold: 0.02

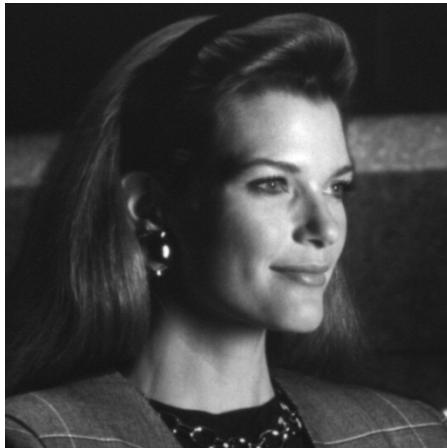


Scale: 5 Threshold: 0.02



Scale: 9 Threshold: 0.02

Figure 3: Increasing wavelet compression scale on an image



Original image



Scale: 3 Threshold: 0.02

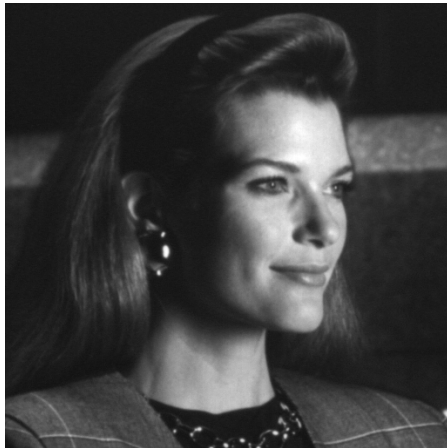


Scale: 3 Threshold: 0.05



Scale: 3 Threshold: 0.10

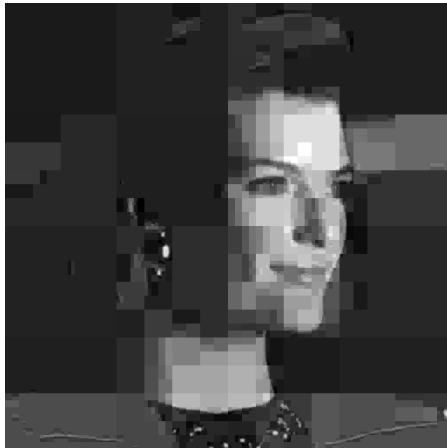
Figure 4: Increasing wavelet compression threshold on scale 3 on an image



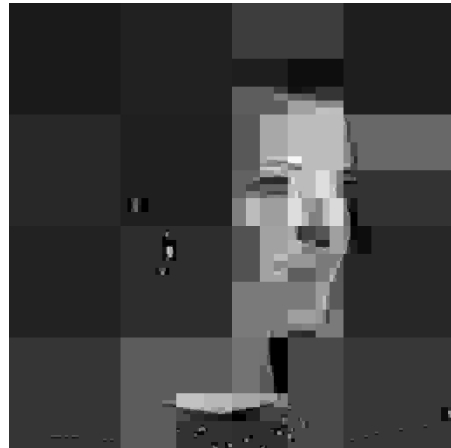
Original image



Scale: 7 Threshold: 0.02



Scale: 7 Threshold: 0.05



Scale: 7 Threshold: 0.10

Figure 5: Increasing wavelet compression threshold on scale 7 on an image

Task distribution

ex1	design	implementation	answers questions	writing report
Klaas	60%	90%	n.a.	75%
Jan	40%	10%	n.a.	25%

ex2	design	implementation	answers questions	writing report
Klaas	50%	50%	50%	50%
Jan	50%	50%	50%	50%

ex3	design	implementation	answers questions	writing report
Klaas	50%	75%	50%	75%
Jan	50%	25%	50%	25%