Image Processing
lab 3

Klaas Kliffen Jan Kramer

December 10, 2015

Exercise 1 – 1-D wavelet transforms

a. Assuming the length of the input vector is a power of two, it is possible to iterate over the input vector applying the algoritm described by example 7.8. The current length is initialized to the length of the vector. For each step, this length is halved. Since the input vector of a step is split in half, a part with sums and a part with differences, the next step will only use the sums, as specified by the algoritm. First two vector are created by taking the odd and even element of the input vector until the current length of the scale. Two result vectors are calculated for the sums and the differences. The difference vector is positioned after the sum vector and written to the input vector. The values of the sum and difference vector needs normalization, which is a factor of the square root of 2.

```
% function to perform 1D Haar wavelet transform
function retval = IPdwt(x,s)
sqrt2 = sqrt(2);
out = x;

initl = length(out);

for i = 1 : s
  % Get the odd and even elements
  odds = out(1:2:initl);
  evens = out(2:2:initl);
   % Calculate the means and details
  sums = (odds + evens);
  diffs = (odds - evens);
  % Put the new values
  out(1:initl) = [sums, diffs] / sqrt2;
  initl /= 2;
end

retval = out;
end
```

b. For the inverse wavelet transform, the inital 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 orignal scale of the transformation is reached.

```
% function to perform 1D Haar wavelet transform
function retval = IPidwt(x,s)
sqrt2 = sqrt(2);
out = x;

% Determine the initial length
initl = length(x) / (2^(s-1));

for i = 1 : s
  % Retrieve the sums and the differences
  sums = out(1:initl/2);
  diffs = out(initl/2+1:initl);
  % Calculate and scale the result
  plus = (sums+diffs)/sqrt2;
  mins = (sums-diffs)/sqrt2;
  % Combine the new values
  combined = zeros(initl,1);
  combined(1:2:end) = plus;
  combined(2:2:end) = mins;
  % Store them in the output matrix
  out(1:initl) = combined;
  % Increase the length or the next iteration
  initl *= 2;
end

retval = out;
end
```

Exercise 2 – 2-D wavelet transforms

a. TODO: text

```
% function to perform 2D Haar wavelet transform
function retval = IPdwt2(x, j)
% note that x should be double instead of uint, because
% the result can get negative
out = x;
coef = 1/2;

initrow = size(out, 1);
initcol = size(out, 2);

for i = 1 : j
  odds_c = out(1:initrow, 1:2:initcol);
  evens_c = out(1:initrow, 2:2:initcol);
  sums = (odds_c + evens_c);
  diffs = (odds_c - evens_c);
  out(1:initrow, 1:initcol) = [sums, diffs] * coef;

  odds_r = out(1:2:initrow, 1:initcol);
  evens_r = out(2:2:initrow, 1:initcol);
  sums = (odds_r + evens_r) * coef;
  diffs = (odds_r - evens_r) * coef;

  %
  mid_r = initrow / 2;
  mid_c = initcol / 2;
  % approximation image
  out(1:mid_r, 1:mid_c) = sums(:, 1:mid_c);
  % vertical detail
  out(mid_r+1:initrow, 1:mid_c) = sums(:, mid_c+1:initcol);
  % horizontal detail
  out(1:mid_r, mid_c+1:initcol) = diffs(:, 1:mid_c);
  % detail detail
  out(mid_r+1:initrow, mid_c+1:initcol) = diffs(:, mid_c+1:initcol);

  initrow = mid_r;
  initcol = mid_c;
end

retval = out;

endfunction
```

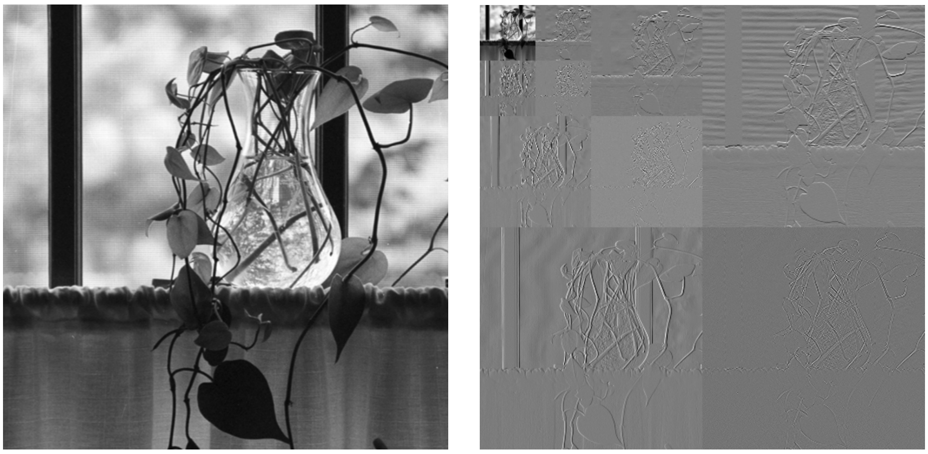
b. TODO: text

```
function retval = IPdwt2scale(x, j)
% calculate the dwt with a shifted image around 0 (assumes doubles)
out = x - 0.5;
out = IPdwt2(out, j);
out = out + 0.5;

```

```
initrow = size(out, 1);
initcol = size(out, 2);

% iterate trough the levels in the image
for i = 1 : j
  mid_r = initrow / 2;
  mid_c = initcol / 2;

  % contrast stretch the horizontal details
  w = out(1:mid_r, mid_c+1:initcol);
  out(1:mid_r, mid_c+1:initcol) = (w - min(min(w))) * (1 / (max(max(w)) - min(min(w))));

  % contrast stretch the vertical details
  w = out(mid_r+1:initrow, 1:mid_c);
  out(mid_r+1:initrow, 1:mid_c) = (w - min(min(w))) * (1 / (max(max(w)) - min(min(w))));

  % contrast stretch the diagonal details
  w = out(mid_r+1:initrow, mid_c+1:initcol);
  out(mid_r+1:initrow, mid_c+1:initcol) = (w - min(min(w))) * (1 / (max(max(w)) - min(min(w))));

  initrow = mid_r;
  initcol = mid_c;
end

% contrast stretch the approximation image
w = out(1:initrow, 1:initcol);
out(1:initrow, 1:initcol) = (w - min(min(w))) * (1 / (max(max(w)) - min(min(w))));

retval = out;
endfunction
```

c. TODO: text

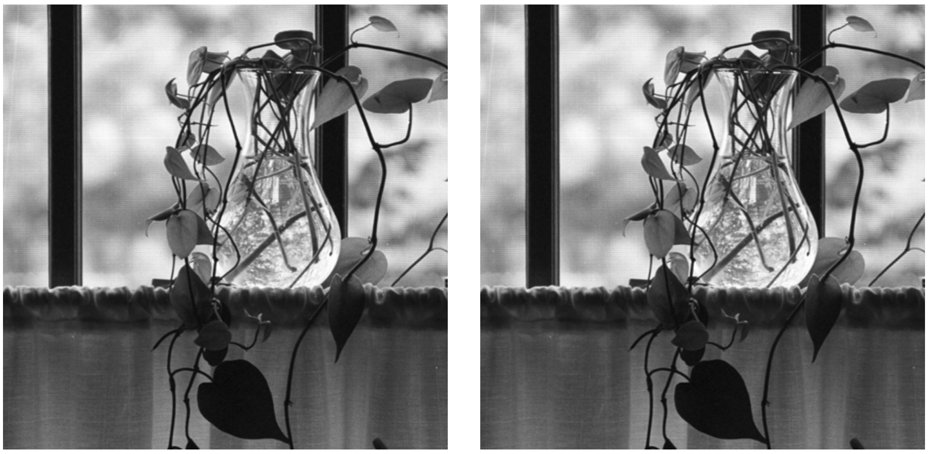
```

x = im2double(imread('../images/vase.tif'));

%% lab 2 ex 2abc
y = IPdwt2(x, 3);
imwrite(y, 'unscaled.png');
imwrite(im2uint8(IPdwt2scale(x, 3)), 'scaled.png');
imwrite(im2uint8(IPidwt2(y,3)), 'output.png');
```



Original image 3 scale wavelet transform

Figure 1: 3 scale wavelet transform of the orignal image

d. TODO: text



Original image Restoration of the image form 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.

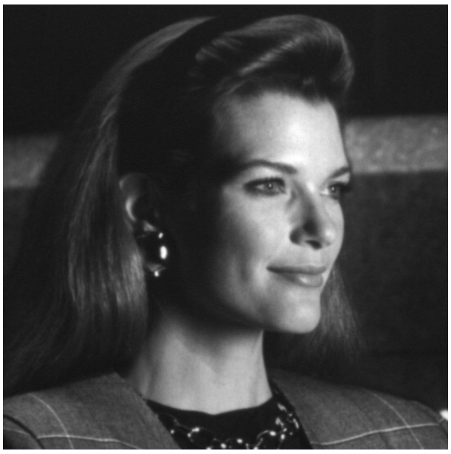
The root mean square error and the mean square signal to noise ratio are calculated from the given formulas. For the compression ratio the the build-in function of entropy is used to calculate the entropy of the compressed and original image. The entropy is a value for how complex an image is. Compressing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

```
% Function to compress an image using wavelet transform
function retval = IPwaveletcompress(img, scale, threshold)

[width,height] = size(img);
wl = width / (2^scale);
hl = height / (2^scale);

% Wavelet transform
wtrans = IPdwt2(img,scale);

% Construct a matrix for the threshold
thresholdmat = threshold * ones(size(img));
% Matrix containing 1 for pixels above the threshold
results = abs(wtrans) > thresholdmat;
% Perform the thresholding by elementwise multiplying
threshed = zeros(size(img));
threshed = wtrans .*results;

% Copy the original image part (for dark values in the original
threshed(1:hl,1:wl) = wtrans(1:hl,1:wl);

% Convert it back
compressed = IPidwt2(threshed, scale);

printf("Scale: %d Threshold: %f\n", scale, threshold);
error = compressed - img;
errorsq = error .* error;
rmse = sqrt(mean(mean(errorsq)));
printf("Root mean square error: %f\n",rmse);

squared = compressed .* compressed;
snr = sum(sum(squared)) / sum(sum(errorsq));
printf("Mean square signal to noise: %f\n",snr);

% Calculatute the compression
orig = entropy(im2uint8(img));
comp = entropy(im2uint8(threshed));
printf("Compress ratio: %f:1\n\n",orig/comp);


retval = compressed;

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



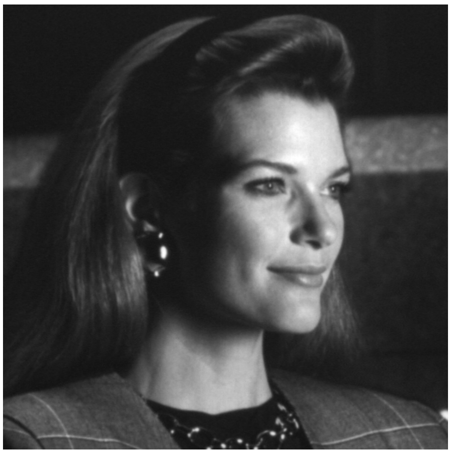
Original image

Scale: 1 Threshold: 0.02

Scale: 5 Threshold: 0.02

Scale: 9 Threshold: 0.02

Figure 3: Increasing wavelet compression scale on an image



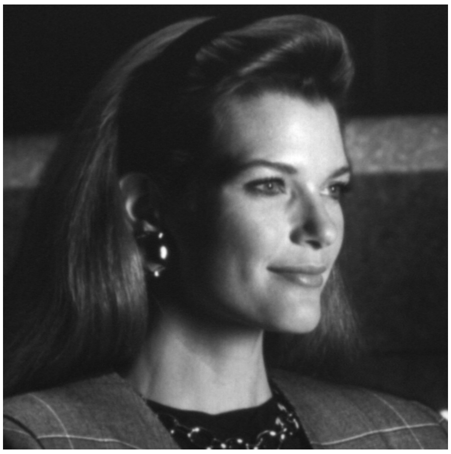
Original image

Scale: 3 Threshold: 0.02

Scale: 3 Threshold: 0.05

Scale: 3 Threshold: 0.10

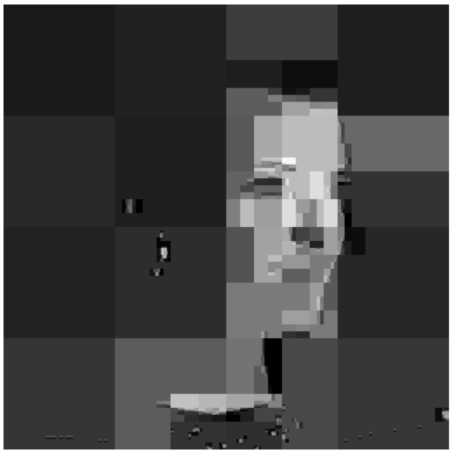
Figure 4: Increasing wavelet compression threshold on scale 3 on an image



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	40%	20%	50%	25%
Jan	60%	80%	50%	75%

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