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.

```
% 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
```

```
% 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

b.

```
% 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
```

```
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 = im2uint8(out);
endfunction
```

b.

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

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

```
% inverse discrete wavelet transform
function retval = IPidwt2(x, s);
out = x;

[height, width] = size(x);
initrow = height / (2^(s-1));
initcol = width / (2^(s-1));

for i = 1 : s
  mid_r = initrow / 2;
  mid_c = initcol / 2;

```

```
13    % make sure we swap horizontal and vertical details
14    rowsums = [out(1:mid_r, 1:mid_c), out(mid_r+1:initrow, 1:mid_c)];
15    rowdiffs = [out(1:mid_r, mid_c+1:initcol), out(mid_r+1:initrow,
          mid_c+1:initcol)];
16
17    % Calculate and scale the result
18    plus = (rowsums+rowdiffs);
19    mins = (rowsums-rowdiffs);
20    % Combine the new values
21    combined = zeros(initrow,initcol);
22    combined(1:2:end,:) = plus;
23    combined(2:2:end,:) = mins;
24    % Replace the values in the image
25    out(1:initrow,1:initcol) = combined;
26
27    % Do the same with the columns
28    colsums = out(1:initrow,1:mid_c);
29    coldiffs = out(1:initrow,mid_c+1:initcol);
30
31    % Calculate and scale the result
32    plus = (colsums+coldiffs);
33    mins = (colsums-coldiffs);
34    % Combine the new values
35    combined(:,1:2:end) = plus;
36    combined(:,2:2:end) = mins;
37    % Replace the values in the image
38    out(1:initrow,1:initcol) = combined;
39
40    initrow *= 2;
41    initcol *= 2;
42  end
43
44  retval = im2uint8(out);
45  end
```

Exercise 3 – Image Compression

d.

```
1  % Function to compress an image using wavelet transform
2  function IPwaveletcompress(img, scale, threshold)
3
4  [width,height] = size(img);
5  wl = width / (2^s);
6  hl = height / (2^s);
7
8  % Wavelet transform
9  wtrans = IPdwt2(img,scale);
10 % TODO: scaling needed?
11
12 % Construct a matrix for the threshold
13 thresholdmat = threshold * ones(img); % TODO: evt aanpassen aan
       grijswaarde voor difference
14 % Matrix containing 1 for pixels above the threshold
15 results = img > thresholdmat;
```

```
16 % Perform the thresholding by elementwise multiplying
17 threshed = zeros(size(img));
18 threshed = img .*results;
19
20 % Copy the original image part (for dark values in the original
21 threshed(1:hl,1:wl) = wtrans(1:hl,1:wl);
22
23 % TODO: calculate the compression
24
25
26 % Convert it back
27 compressed = IPidwt2(threshed, scale);
28
29 error = compressed - img;
30 rmse = rms(error); % signal pkg
31 disp("Root mean square error:");
32 disp(rmse);
33
34 squared = compressed .* compressed;
35 errorsq = error .* error;
36 snr = squared / errorsq;
37 disp("Mean square signal to noise:");
38 disp(snr);
39
40 retval = compressed;
41
42 end
```

b.

Task distribution

ex1	design	implementation	answers questions	writing report
Klaas	50%	100%	50%	50%
Jan	50%	0%	50%	50%

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%	100%	50%	50%
Jan	50%	0%	50%	50%