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. 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.

```
% 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/sqrt(2);

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

for i = 1 : j
  % 1D DWT along the rows
  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;

  % 1D DWT along the columns
  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;

  % save the parts as in the figures in the book
  % 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;
```

```
end
```

```
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;
end
```

b.

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

%% lab 3 ex 2abcd
y = IPdwt2(x, 3);
imwrite(y, 'unscaled.tif');
imwrite(im2uint8(IPdwt2scale(x, 3)), 'scaled.tif');
imwrite(im2uint8(IPidwt2(y,3)), 'output.tif');
% the difference
sum(sum(im2uint8(x - IPidwt2(y,3))))
```

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 implemention, 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 iplementation first applies a 1D inverse DWT to the columns and the one to the rows.

```
% inverse discrete wavelet transform
function retval = IPidwt2(x, s);
out = x;
coef = 1/sqrt(2);

[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;

  % make sure we swap horizontal and vertical details
  rowsums = [out(1:mid_r, 1:mid_c), out(mid_r+1:initrow, 1:mid_c)];
  rowdiffs = [out(1:mid_r, mid_c+1:initcol), out(mid_r+1:initrow,
      mid_c+1:initcol)];

  % Calculate and scale the result
  plus = (rowsums+rowdiffs);
  mins = (rowsums-rowdiffs);
  % Combine the new values
  combined = zeros(initrow,initcol);
  combined(1:2:end,:) = plus * coef;
  combined(2:2:end,:) = mins * coef;
  % Replace the values in the image
  out(1:initrow,1:initcol) = combined;

  % Do the same with the columns
  colsums = out(1:initrow,1:mid_c);
  coldiffs = out(1:initrow,mid_c+1:initcol);

  % Calculate and scale the result
  plus = (colsums+coldiffs);
  mins = (colsums-coldiffs);
  % Combine the new values
  combined(:,1:2:end) = plus * coef;
  combined(:,2:2:end) = mins * coef;
  % Replace the values in the image
  out(1:initrow,1:initcol) = combined;

  initrow *= 2;
  initcol *= 2;
end

retval = out;
end
```

Exercise 3 – Image Compression

a.

```
% 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);
% TODO: scaling needed?

% Construct a matrix for the threshold
thresholdmat = threshold * ones(size(img)); % TODO: evt aanpassen aan grijswaarde voor difference
% 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);

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:1\n",snr);

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


retval = compressed;

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%