Multiresolution Representation and Wavelet Transform

Yao Wang
Tandon School of Engineering, New York University
Lecture Outline

• Introduction
• Multi-resolution representation of images: Gaussian and Laplacian pyramids
• Wavelet transform through Iterated Filterbank Implementation
• Applications for Multiresolution Representations
Gaussian pyramid (Approximation Pyramid)

Laplacian pyramid (Prediction Residual Pyramid)

From [Gonzalez2008]
Multi-Resolution Representation (aka Pyramid Representation)

Prefilter before downsampling

From [Gonzalez2008]
Averaging and Interpolation Filters

• Approximation filters:
  – Gaussian filters
  – Averaging filters
  – Binomial filter: \[1 4 6 4 1]/16\]
  – (used in the original paper [Burt-Adelson1993a], can be implemented with shifts and add only)

• Interpolation filters (on zero-filled signals)
  – Bilinear \([1/2, 1, 1/2]\)
  – Binomial filter: \([1 4 6 4 1]/8\)
  – (used in the original paper [Burt-Adelson1993a])
Figure 3.33  The Gaussian pyramid shown as a signal processing diagram: The (a) analysis and (b) re-synthesis stages are shown as using similar computations. The white circles indicate zero values inserted by the $\uparrow 2$ upsampling operation. Notice how the reconstruction filter coefficients are twice the analysis coefficients. The computation is shown as flowing down the page, regardless of whether we are going from coarse to fine or vice versa.

From [Szeliski2012]
Pseudo Code to Generate a Pyramid

• Simultaneously creating a Gaussian and a Laplacian Pyramid
• Ex: 3 level, using h[] for pre-filtering, g[] for interpolation

Gimg3=Inimg;
Gimg2=downsize(Gimg3, h);
Uimg3=upresize(Gimg2, g);
Limg3=Gimg3-Uimg3;
Gimg1=downsize(Gimg2, h);
Uimg2=upsize(Gimg1,g);
Limg2=Gimg2-Uimg2;

• Gaussian pyramid: Gimg1, Gimg2, Gimg3
• Laplacian pyramid: GIimg1, Limg2, Limg3
Sample Python Code for Pyramid Generation and Display

```python
# -*- coding: utf-8 -*-

Created on Sun Feb 05 12:03:45 2017

@author: Dawn Knight

import numpy as np
import matplotlib.pyplot as plt
import cv2
from skimage.transform import pyramid_gaussian
from skimage.transform import pyramid_laplacian

img = cv2.imread('lena.jpg')
rows, cols, dim = img.shape
pyramid = tuple(pyramid_gaussian(img, downscale=2))
pyramid_L = tuple(pyramid_laplacian(img, downscale=2))

# create a space to put pyr img in
composite_img = np.zeros((rows, cols + cols // 2, 3), dtype=np.double)
composite_img[:, :, :] = pyramid[0]
composite_img_L = np.zeros((rows, cols + cols // 2, 3), dtype=np.double)
composite_img_L[:, :, :] = pyramid_L[0]

i_row = 0
for p, q in zip(pyramid[1:], pyramid_L[1:]):
    n_rows, n_cols = p.shape[:]
    composite_img[i_row:i_row + n_rows, cols:cols + n_cols] = p
    composite_img_L[i_row:i_row + n_rows, cols:cols + n_cols] = q
    i_row += n_rows

cv2.imshow('pyramid_laplacian', composite_img_L)
fig, ax = plt.subplots()
plt.suptitle('pyramid Gaussian')
ax.imshow(composite_img[:, :, ::-1])
plt.show()
```
Use of Pyramid Representations

- Feature extraction across scales (SIFT)
- Enable object search (e.g. faces) of different sizes
- Speed up computations: motion estimation
- Image blending
Figure 3.41  Laplacian pyramid blending (Burt and Adelson 1983b) © 1983 ACM: (a) original image of apple, (b) original image of orange, (c) regular splice, (d) pyramid blend.

From [Szeliski2012]

Figure 3.42  Laplacian pyramid blending details (Burt and Adelson 1983b) © 1983 ACM. The first three rows show the high, medium, and low frequency parts of the Laplacian pyramid (taken from levels 0, 2, and 4). The left and middle columns show the original apple and orange images weighted by the smooth interpolation functions, while the right column shows the averaged contributions.
Pyramid is a redundant representation

- A pyramid (either Gaussian or Laplacian) includes an image of the original size plus additional smaller images
- How many samples in all levels?
- Original image (level J-0) \( N \times N \)
- Next level (level J-1): \( N/2 \times N/2 \)
- Level J-l (l=0 to J): \( N/(2^l) \times N/(2^l) \)
- Total # samples = \( N^2 \sum_{l=0}^{J} (1/4^l) = N^2(1-(1/4)^{(J-1)})/(1-1/4) \approx 4/3 \ N^2 \)

- However, with Laplacian pyramid, many samples are close to zero except at the top level.
- Wavelet transform is similar to Laplacian pyramid but does not incur additional pixels (non-redundant representation).
Wavelet vs. Pyramid vs. Subband Decomposition

- Pyramid is a redundant transform (more samples than original)
- Wavelet is a non-redundant multi-resolution representation
- There are many ways to interpret wavelet transform. Here we describe the generation of discrete wavelet transform using the tree-structured subband decomposition (aka iterated filterbank) approach
  - 1D 2-band decomposition
  - 1D tree-structured subband decomposition (discrete wavelet transform)
  - Harr wavelet as an example
  - Extension to 2D by separable processing
Two Band Filterbank

**FIGURE 7.4** (a) A two-band filter bank for one-dimensional subband coding and decoding, and (b) its spectrum splitting properties.

From [Gonzalez2008]
What does the filter bank do?

- **h0**: Lowpass filter (0-¼)
  - $y_0$: a low-passed and then down-sampled version of $x$ (Sampling theorem tells us we can down-sample after bandlimiting)

- **h1**: Highpass filter (1/4-1/2)
  - $y_1$: a high-passed and then down-sampled version of $x$ (Sampling theorem also works in this case)

- **g0**: interpolation filter for low-pass subsignal
  - $q$: reconstructed low-pass filtered signal $s$

- **g1**: interpolation filter for high-pass subsignal
  - $r$: reconstructed high-pass filtered signal $t$

- Can reach perfect reconstruction even if these filters are not ideal low-pass/high-pass filters!
  - When the filters $h_0, h_1, g_0, g_1$ are designed appropriately,
  - $X^\wedge = X$ (perfect reconstruction filterbank)
DTFT of signals after downsampling and upsampling (Optional)

Down-sampling by factor of 2:

\[ x_d(m) = x(2m) \iff X_d(u) = \frac{1}{2} \left( X\left(\frac{u}{2}\right) + X\left(\frac{u}{2} - \frac{1}{2}\right) \right) \]

Up-sampling by factor of 2:

\[ x_u(m) = \begin{cases} 
  x\left(\frac{m}{2}\right), & m = \text{even} \\
  0, & \text{otherwise}
\end{cases} \iff X_u(u) = X(2u) \]

Conceptual proof by doing sampling on continuous signal under 2 different sampling rates.
Perfect Reconstruction Conditions (Optional)

\[
x(n) * h_0(n) \Leftrightarrow X(u)H_0(u)
\]
\[
x_i(n) = \text{down}(x(n) * h_0(n)) \Leftrightarrow X_i(u) = (X(u/2)H_0(u/2) + X(u/2 - 1/2)H_0(u/2 - 1/2))/2
\]
\[
\text{up}(x_i(n)) \Leftrightarrow X_i(2u) = (X(u)H_0(u) + X(u-1)H_0(u-1))/2
\]
\[
\tilde{x}(n) = \text{up}(x_i(n)) * g_0(n) + \text{up}(x_i(n)) * g_1(n)
\]
\[
\Leftrightarrow \tilde{X}(u) = (X(u)H_0(u)G_0(u) + X(u-1)H_0(u-1)G_0(u))/2
\]
\[
+ (X(u)H_1(u)G_1(u) + X(u-1)H_1(u-1)G_1(u))/2
\]
\[
= X(u)(H_0(u)G_0(u) + H_1(u)G_1(u))/2
\]
\[
+ X(u-1)(H_0(u-1)G_0(u) + H_1(u-1)G_1(u))/2
\]

To guarantee \( \tilde{X}(u) = X(u) \), we need
\[
H_0(u)G_0(u) + H_1(u)G_1(u) = 2
\]
\[
H_0(u-1)G_0(u) + H_1(u-1)G_1(u) = 0 \quad \text{(To remove aliasing component!)}
\]
Perfect reconstruction condition:
\[ H_0(u)G_0(u) + H_1(u)G_1(u) = 2 \]
\[ H_0(u-1)G_0(u) + H_1(u-1)G_1(u) = 0 \]

The second equation (aliasing cancelation) can be guaranteed by requiring
\[ G_0(u) = H_1(u-1) \iff g_0(n) = (-1)^n h_1(n) \]
\[ G_1(u) = -H_0(u-1) \iff g_1(n) = (-1)^{n+1} h_0(n) \]
(Quadrature Mirror Condition)

To guarantee perfect reconstruction, the filters must satisfy the biorthogonality condition:
\[ < h_i(2n-k), g_j(k) >= \delta(i-j)\delta(n) \]

One has freedom to design both \( g_0(n), g_1(n) \), which can have different length.

A more strict condition requires orthonality between \( g_0(n), g_1(n) \):
\[ < g_i(n), g_j(n+2m) >= \delta(i-j)\delta(m) \]
which yields
\[ g_1(n) = (-1)^n g_0(L-1-n) \]
\[ h_0(n) = g_0(L-1-n) \]
\[ h_1(n) = g_1(L-1-n) = (-1)^n g_0(n) = (-1)^n h_0(L-1-n) \]

One only has freedom to design \( g_0(n) \). filter length \( L \) must be even and all filters have same length.
Haar Filter (Simplest Orthogonal Wavelet Filter)

$h_0$ : averaging, $[1,1]/\sqrt{2}$; $h_1$ : difference, $[1,-1]/\sqrt{2}$;

g_0 = [1,1]/\sqrt{2}; g_1 = [-1,1]/\sqrt{2}$

Input sequence : $[x_1,x_2,x_3,x_4,...]$

Analysis (Assuming samples outside the boundaries are 0, remember to flip the filter when doing convolution)

$s = x * h_0 = [s_0,s_1,s_2,s_3,s_4,...]$, $s_0 = (x_1+0)/\sqrt{2}$, $s_1 = (x_2+x_1)/\sqrt{2}$, $s_2 = (x_3+x_2)/\sqrt{2}$, $s_3 = (x_4+x_3)/\sqrt{2}$...

$y_0 = s \downarrow 2 = [s_1,s_3,...]$

t = x * h_1 = [t_0,t_1,t_2,t_3,t_4,...]$, $t_0 = [x_1-0]/\sqrt{2}$, $t_1 = [x_2-x_1]/\sqrt{2}$, $t_2 = [x_3-x_2]/\sqrt{2}$, $t_3 = [x_4-x_3]/\sqrt{2}$...

$y_1 = t \downarrow 2 = [t_1,t_3,...]$

Synthesis :

$u = y_0 \uparrow 2 = [0,s_1,0,s_3,...]$

$q = u * g_0 = [q_1,q_2,q_3,q_4,...]$, $q_1 = (s_1+0)/\sqrt{2} = (x_1+x_2)/2$, $q_2 = (0+s_1)/\sqrt{2} = (x_1+x_2)/2$, $q_3 = (s_3+0)/\sqrt{2} = (x_3+x_4)/2$

$v = y_1 \uparrow 2 = [0,t_1,0,t_3,...]$

$r = v * g_1 = [r_1,r_2,r_3,r_4,...]$, $r_1 = (-t_1+0)/\sqrt{2} = (x_1-x_2)/2$, $r_2 = (-0+t_1)/\sqrt{2} = (-x_1+x_2)/2$, $r_3 = (-t_3+0)/\sqrt{2} = (x_3-x_4)/2$, $r_4 = (-0+t_3)/\sqrt{2} = (x_4-x_3)/2$,

$\hat{x} = q + r = [q_1+r_1,q_2+r_2,...] = [x_1,x_2,x_3,...]$

Note with Haar wavelet, the lowpass subband essentially takes the average of every two samples, $L=(x_1+x_2)/\sqrt{2}$, and the highpass subband takes the difference of every two samples, $H=(x_1-x_2)/\sqrt{2}$.

For synthesis, you take the sum of the lowpass and high pass signal to recover first sample $A=(L+H)/\sqrt{2}$, and you take the difference to recover the second sample $B=(L-H)/\sqrt{2}$. 
MATLAB example

```matlab
>> [ca,cd]=dwt(y,'db4');
>> z=idwt(ca,cd,'db4');
>> wy=[ca,cd];
>> subplot(3,1,1),plot(y), title('Original sequence');
>> subplot(3,1,2),plot(wy), title('Wavelet transform: left=low band, right=high band');
>> subplot(3,1,3),plot(z), title('Reconstructed sequence');
```
Iterated Filter Bank

3. Iterated filter bank. The lowpass branch gets split repeatedly to get a discrete-time wavelet transform.

From [Vetterli01]
>> [ca,cd]=dwt(y,'db4');
[caa,cad]=dwt(ca,'db4');
[caaa,caad]=dwt(caa,'db4');
wy1=[ca,cd];
wy2=[caa,cad,cd];
>> wy3=[caaa,caad,cad,cd];
>> subplot(4,1,1),plot(y),title('Original Signal');
>> subplot(4,1,2),plot(wy1),title('1-level Wavelet Transform');
>> subplot(4,1,3),plot(wy2),title('2-level Wavelet Transform');
>> subplot(4,1,4),plot(wy3),title('3-level Wavelet Transform');
Discrete Wavelet Transform = Iterated Filter Bank

From [Gonzalez2008]
Temporal-Frequency Domain Partition

**Figure 7.21** Time-frequency tilings for (a) sampled data, (b) FFT, and (c) FWT basis functions.

From [Gonzalez2008]
Wavelet Transform vs. Fourier Transform

• Fourier transform:
  – Basis functions cover the entire signal range, varying in frequency only

• Wavelet transform
  – Basis functions vary in frequency (called “scale”) as well as spatial extend
    • High frequency basis covers a smaller area
    • Low frequency basis covers a larger area
    • Non-uniform partition of frequency range and spatial range
    • More appropriate for non-stationary signals
How to Apply Filterbank to Images?

Applying the 1D decomposition along rows of an image first, and then columns!

From [Gonzalez2008]
With Harr filter, you can work on every 2x2 blocks in an image, [A,B;C,D]. LL=(A+B+C+D)/2; LH=(A+B-C-D)/2; HL=(A-B+C-D)/2; HH=(A+D-B-C)/2. For synthesis, A=(LL+LH+HH+HH)/2; B=((LL+LH)-(HL+HH))/2; C=((LL+HL)-(LH+HH))/2; D=((LL+HH)-(LH+HL))/2;
Wavelet Transform for Images

From [Usevitch01]
FIGURE 7.8 (a) A discrete wavelet transform using Haar basis functions. Its local histogram variations are also shown; (b)–(d) Several different approximations (64 × 64, 128 × 128, and 256 × 256) that can be obtained from (a).

From [Gonzalez2008]
Common Wavelet Filters

- Haar: simplest, orthogonal, not very good
- Daubechies 8/8: orthogonal
- Daubechies 9/7: bi-orthogonal, most commonly used if numerical reconstruction errors are acceptable
- LeGall 5/3: bi-orthogonal, integer operation, can be implemented with integer operations only, used for lossless image coding
- Differ in energy compaction capability
### Table 3. Daubechies 9/7 Analysis and Synthesis Filter Coefficients.

<table>
<thead>
<tr>
<th>i</th>
<th>Low-Pass Filter $h_L(i)$</th>
<th>High-Pass Filter $h_H(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6029490182363579</td>
<td>1.115087052456994</td>
</tr>
<tr>
<td>±1</td>
<td>0.2668641184428723</td>
<td>-0.5912717631142470</td>
</tr>
<tr>
<td>±2</td>
<td>-0.07822326652898785</td>
<td>-0.05754352622849957</td>
</tr>
<tr>
<td>±3</td>
<td>-0.01686411844287495</td>
<td>0.09127176311424948</td>
</tr>
<tr>
<td>±4</td>
<td>0.02674875741080976</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4. Le Gall 5/3 Analysis and Synthesis Filter Coefficients.

<table>
<thead>
<tr>
<th>i</th>
<th>Low-Pass Filter $h_L(i)$</th>
<th>High-Pass Filter $h_H(i)$</th>
<th>Low-Pass Filter $g_L(i)$</th>
<th>High-Pass Filter $g_H(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.115087052456994</td>
<td>0.6029490182363579</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>±1</td>
<td>0.5912717631142470</td>
<td>-0.2668641184428723</td>
<td>-1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>±2</td>
<td>-0.05754352622849957</td>
<td>-0.07822326652898785</td>
<td>-1/8</td>
<td></td>
</tr>
<tr>
<td>±3</td>
<td>-0.09127176311424948</td>
<td>0.01686411844287495</td>
<td></td>
<td></td>
</tr>
<tr>
<td>±4</td>
<td>0.02674875741080976</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison of Different Filters

FIGURE 8.42 Wavelet transforms of Fig. 8.23 with respect to (a) Haar wavelets, (b) Daubechies wavelets, (c) symlets, and (d) Cohen-Daubechies-Feauveau biorthogonal wavelets.

From [Gonzalez2008]
Impact of Filters and Decomposition Levels on Energy Compaction

- Coefficients with magnitude $< 1.5$ are set to zero.

From [Gonzalez2008]
Wavelet Based Image Compression
(Basic Idea)

• Wavelet transform is a good representation to use for image compression because many coefficients can be truncated to zeros.

• Three steps:
  – Apply wavelet transform to an image
  – Quantize wavelet coefficients in all subbands (e.g. uniform quantization)
    • \( Q(f) = \text{floor} \left( \frac{(f-\text{mean}+QS/2)/QS}{QS} \right) \times QS + \text{mean} \)
  – Represent the quantized coefficients using binary bits (entropy coding)
  – Wavelet based coders differ mainly in entropy coding.
MATLAB Tools for 2D Wavelet: 1 Level

- \([CA,CH,CV,CD] = \text{DWT2}(X,'wname', 'mode',\text{MODE}),\)
- \([CA,CH,CV,CD] = \text{DWT2}(X,\text{Lo}_D,\text{Hi}_D, 'mode',\text{MODE})\)
- \(X = \text{IDWT2}(CA,CH,CV,CD,'wname', 'mode',\text{MODE}),\)
- \(X = \text{IDWT}(CA,CD,\text{Lo}_R,\text{Hi}_R, 'mode',\text{MODE})\)
- Available wavelet names 'wname' are:
  - Daubechies: 'db1' or 'haar', 'db2', ... , 'db45'
  - Coiflets: 'coif1', ... , 'coif5'
  - Symlets: 'sym2', ... , 'sym8', ... , 'sym45'
  - Discrete Meyer wavelet: 'dmey'
  - Biorthogonal: ...
- Use following to find actual filters:
  - \([\text{LO}_D,\text{HI}_D,\text{LO}_R,\text{HI}_R] = \text{WFILTERS('wname')}\)
- Modes of boundary treatment:
  - 'sym': symmetric-padding (half-point): boundary value symmetric replication - default mode.
  - 'zpd': zero padding
  - 'ppd': periodic-padding
- Let \(SX = \text{size}(X)\) and \(LF = \text{the length of filters}\); then \(\text{size}(CA) = \text{size}(CH) = \text{size}(CV) = \text{size}(CD) = \) \(SA\) where \(SA = \text{CEIL}(SX/2)\), if \(\text{mode}='ppd'\). \(SA = \text{FLOOR}((SX+LF-1)/2)\) for other modes.
- Show examples in class
MATLAB Tools for Wavelet: Multi-level

- Wavedec2( ), waverec2( ): multi-level
- \([C,S] = \text{WAVEDEC2}(X,N,'wname')\)
Python tool for Wavelet (1/3)

- Python package for wavelet: Pywavelet (version 5.0.1)
- Install command: pip install PyWavelets

1D single level dwt:
- \((cA, cD) = \text{pywt.dwt}(data, \text{wavelet}, \text{mode} = \text{'mode_type'})\)
- \(\text{data} = \text{pywt.idwt}(cA, cD, \text{wavelet}, \text{mode} = \text{'mode_type'})\)

2D single level dwt:
- \((cA, (cH, cV, cD)) = \text{pywt.dwt2}(data, \text{wavelet}, \text{mode} = \text{'mode_type'})\)
- \(\text{data} = \text{pywt.idwt2}((cA, (cH, cV, cD)), \text{wavelet}, \text{mode} = \text{'mode_type'})\)

2D multi-level dwt:
- \([cAn, (cHn, cVn, cDn), \ldots (cH1, cV1, cD1)] = \text{pywt.wavedec2}(data, \text{wavelet}, \text{mode} = \text{'mode_type'}, \text{level}=\text{None})\)
- \(\text{data} = \text{pywt.waverec2}([cAn, (cHn, cVn, cDn), \ldots (cH1, cV1, cD1)], \text{wavelet}, \text{mode} = \text{'mode_type'})\)
Python tool for Wavelet (2/3)

- Currently the built-in wavelet families in `pywt` are:
  - Haar (haar)
  - Daubechies (db)
  - Symlets (sym)
  - Coiflets (coif)
  - Biorthogonal (bior)
  - Reverse biorthogonal (rbio)
  - “Discrete” FIR approximation of Meyer wavelet (dmey)
  - Gaussian wavelets (gaus)
  - Mexican hat wavelet (mexh)
  - Morlet wavelet (morl)
  - Complex Gaussian wavelets (cgau)
  - Shannon wavelets (shan)
  - Frequency B-Spline wavelets (fbsp)
  - Complex Morlet wavelets (cmor)
**Python tool for Wavelet (3/3)**

- **Built-in wavelet mode in pywt**

<table>
<thead>
<tr>
<th>PyWavelets</th>
<th>Matlab</th>
</tr>
</thead>
<tbody>
<tr>
<td>symmetric</td>
<td>sym, symh</td>
</tr>
<tr>
<td>reflect</td>
<td>symw</td>
</tr>
<tr>
<td>smooth</td>
<td>spd, sp1</td>
</tr>
<tr>
<td>constant</td>
<td>sp0</td>
</tr>
<tr>
<td>zero</td>
<td>zpd</td>
</tr>
<tr>
<td>periodic</td>
<td>ppd</td>
</tr>
<tr>
<td>periodization</td>
<td>per</td>
</tr>
<tr>
<td>N/A</td>
<td>asym, asymh</td>
</tr>
<tr>
<td>N/A</td>
<td>asymw</td>
</tr>
</tbody>
</table>
Non-separable Wavelet Transforms

- Separable implementation leads to 3 high-freq subband at each scale
  - Horizontal, vertical, cross (checkerboard pattern)
  - Cross band mixes different directions
- Steerable pyramid [Simoncelli1992]
  - No mixing of directions
  - Four high-freq subbands: 0, 45, 90, 135
  - Enable better image enhancement and feature extraction
  - Necessarily redundant
Figure 3.40  Steerable shiftable multiscale transforms (Simoncelli, Freeman, Adelson et al. 1992) © 1992 IEEE: (a) radial multi-scale frequency domain decomposition; (b) original image; (c) a set of four steerable filters; (d) the radial multi-scale wavelet decomposition.

From [Szeliski2012]

Wavelet Domain Image Denoising

- Apply wavelet transform to an image
- Modify the coefficients based on signal and noise statistics
  - If noise is Gaussian N(0,\(\sigma_n\)), true signal coeff is Laplacian with STD \(\sigma\)
  - Soft-thresholding

\[
\hat{w}(y) = \text{soft} \left( y, \frac{\sqrt{2\sigma_n^2}}{\sigma} \right)
\]

- Inverse wavelet transform
- Remove noise yet not blurring the edges!
- Other more sophisticated approaches
- How to estimate signal and noise statistics?
1. For the given image below, manually compute a 3-level approximation pyramid and corresponding prediction residual pyramid. Use 2x2 averaging for the approximation filter, and use bilinear for the interpolation filter (for pixels on the boundary, you can just use nearest neighbor).

\[ F = \begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16
\end{bmatrix} \]

2. From the residual pyramid generated in Prob. 1, reconstruct the original image.

3. For the same image above, manually compute the wavelet transform (with 3-level) using the Haar analysis filters. Use separable processing, i.e., at each level, do row transform first, followed by column transform. Comment on the differences between the residual pyramids generated in Prob. 1 with the wavelet transform generated here, in terms of number of samples and signal energy in different levels/bands.

4. Reconstruct the image from the wavelet transform in Prob. 3 using Haar synthesis filters, show the reconstructed image at all levels. Do you get back the original image?

5. Quantize all the wavelet coefficients created in Prob. 4 by a stepsize of 2. Then reconstruct the 4x4 image from the quantized wavelet coefficients using Haar synthesis filter. Compare the results of Prof. 5.

6. Using MATLAB freqz( ) function to derive the frequency response of the low-pass and high-pass filters used in the following wavelet transforms: Haar, Daubechies 9/7, and LeGall 5/3. Plot the magnitude response of each and comment on their pros and cons.
Computer Exercises (optional)

1. Learn how to use the following MATLAB functions through online help: dwt, idwt, wavedec, waverec.

2. Write a function that i) applies 3 level wavelet transform to an image using a specified wavelet transform; ii) quantize all transform coefficients with a uniform quantizer with a given quantization stepsize (QS); iii) Reconstruct the image (which we will call the quantized image) from the quantized transform coefficients; iv) Count the number of non-zero coefficients after quantization and compute the PSNR of the quantized image against the original image; v) Show the original and quantized image. The function should have the original image, the filter name, and the QS as input, and the number of non-zero coefficients and PSNR, and quantized image as output, as follows:

   \[
   \text{[NonZeroNum,PSNR,outimg]=WaveletQuant(inimg,'wname',QS);}
   \]

   Test your program within a main program that read in an image, extract the grayscale version, and applies your function to the grayscale image.

   A note on quantization: for the lowest band, please assume the coefficient values have a mean value of 128. For all other bands, assume the coefficient values have a mean value of 0. Your quantizer should be centered around the mean value. That is

   \[
   Q(f)=\text{floor}(\ (f-\text{mean}+\text{QS}/2)/\text{QS})\ *\text{QS}+\text{mean}.
   \]

3. Write a main program that applies the above function to an image using the Haar wavelet and a series of QS including 1, 4, 16, 32, and record the NonZeroNumber and PSNR corresponding to different QS. It then applies the above function to the same image with another more complicated wavelet filter (e.g. ‘db4’) with the same set of QS. Plot in the same figure, the PSNR vs. NonZeroNumber curves, obtained by the two different wavelet filters. You should include this figure in the report, and explain the pros and cons of different filters. Which filter is likely to yield higher coding efficiency (i.e. produced better quality at the same bit rate, or reduces low bit rate to achieve the same quality)? Note that you may assume that the number of bits needed to code the quantized wavelet coefficients is proportional to the number of non-zero coefficients. Therefore, each of the two curves represent the achievable rate-quality performance by a wavelet-based image coder using the corresponding filter.
The following assignments are all OPTIONAL.

1. Write a program that can generate 1-level 1D wavelet transform of a finite length 1D sequence using a given pair of wavelet analysis filters. Your program should have a syntax 
   \[ \text{[CA,CD]} = \text{MYDWT}(X, \text{Lo}_D, \text{Hi}_D) \]
   You can call the \text{conv()} function of MATLAB. For simplicity, you could choose the “same” option for boundary treatment. This way, each the resulting subsignal should be half length of your original signal (make your original sequence has even length). Test your program on any 1D sequence (manually generated, for example, or 1 row of an image) and different wavelet filters. You can generate different wavelet filters (e.g. Haar and db4) using “\text{wfilters()}” function.

2. Write a program that can reconstruct a 1D sequence from its 1-level 1D wavelet transform using a given pair of wavelet synthesis filters. Your program should have a syntax 
   \[ X = \text{MYIDWT}(CA, CD, \text{Lo}_R, \text{Hi}_R) \]
   Apply this program to the subband signals generated in Prob. 1 and you should get back the original sequence approximately. Note that your program may not generate exact reconstruction at boundaries because of simplified boundary treatment.

3. Write a program that can generate 1-level 2D wavelet transform of an image by using your function \text{MYDWT()} or the \text{dwt()} function of MATLAB, if your program does not work well. Basically, you need to apply \text{dwt()} to rows and columns separately, and you need to organize your data structure properly. You should save the four subbands in a single image (all in floating point) so that the LL band is in the top-left, HL band is in the top-right, etc. Your program would have a syntax \[ \text{WIMG= MYDWT2(IMG, Lo}_D, \text{Hi}_D) \]. Use your program to generate the wavelet transform of a gray scale image (or a cropped to a smaller size) using two wavelet filters: Haar and db4. Display the resulting transform images and comment on their differences.
4. Write a program that can reconstruct an image from its 1-level 2D wavelet transform image using your function MYIDWT() or the idwt( ) function of MATLAB. Basically, you need to apply idwt( ) to rows and columns of the wavelet transform image separately. Apply your program to the results from Prob. 3.

5. Quantize the wavelet coefficients you obtained in Prob. 3 using a uniform quantizer with a user-given step size, and then reconstruct the image from quantized coefficients using the program in Prob. 4. Show the reconstructed images with two different quantization step sizes, 4 and 16. If you cannot get your programs working for Prob. 3 and 4, you could use the dwt2() and idwt2() functions instead.

7. Develop MATLAB codes that implement 2-level 2D wavelet transform and reconstruction. Basically you can apply the 1-level program you have developed on the LL-band of 1-level transform to produce 2-level transform. Show the decomposed images and reconstructed images at different stages.
References

- Optional