Image Processing

Lecture 09: Image Compression – II

(Ch8 Image Compression and Watermarking)

Zhiguo Zhang

zhiguozhang@hit.edu.cn

Review of Last Lecture

- In the last lecture we learnt:
  - Fundamentals
  - Data Redundancies
  - Image Compression Models
  - Lossless Image Compression
    - ➢Huffman Coding

Huffman Coding

- Coding Procedures for an N-symbol source, two steps:
  - Source reduction
    - ➢List all probabilities in a descending order.
    - ➢Merge the two symbols with smallest probabilities into a new compound symbol.
    - ➢Repeat the above two steps until a reduced source with two symbols is reached.
  - Codeword assignment
    - ➢Start from the smallest source and work back to the original source.
    - ➢Each merging point corresponds to a node in binary codeword tree.

Example

Step 1: Source reduction

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

Example

Step 2: Codeword assignment

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

The average length of the code is:

$$L_{avg} = 0.4\times1 + 0.3\times2 + 0.1\times3 + 0.1\times4 + 0.06\times5 + 0.04\times5$$
$$= 2.2 \text{ bits/symbol}$$

Contents of This Lecture

- Lossless Image Compression
  - Arithmetic Coding
  - LZW Coding
  - Hybrid Coding
- Lossy Image Compression
  - Transform Coding (DFT, DCT, WHT, KLT)
  - Wavelet Coding
  - JPEG

Arithmetic Coding

- Unlike the variable-length codes, *arithmetic coding* generates non-block codes. Here, a one-to-one correspondence between source symbols and code words does not exist. Instead, an entire sequence of source symbols is assigned a single arithmetic code word.
- The code word defines an interval of real numbers between 0 and 1.
- As the number of symbols in the message increases, the interval used to represent it becomes smaller, and the number of information units (bits) required to represent the interval becomes larger.

Arithmetic Coding

- Now we explain arithmetic coding algorithm by an example:
- For an information source for encoding is "*dacab*", and the probabilities of each source symbols are: $P(a)=0.4$, $P(b)=0.2$, $P(c)=0.2$, $P(d)=0.2$.
- At the starting of the coding process, the information source is assumed to occupy the entire half open interval [0, 1), and the initial interval of each symbols is initially subdivided into four regions based on the probabilities: $a = [0, 0.4)$, $b = [0.4, 0.6)$, $c = [0.6, 0.8)$, $d = [0.8, 1)$.

Arithmetic Coding

- We define an equation:

$$\begin{cases} Start_N = Start_B + Left_C \times L_B \\ End_N = Start_B + Right_C \times L_B \end{cases}$$

where

$Start_N$ and End_N denote the start and end position of a new intervals,

$Start_B$ presents the start position of the prior interval and L_B is its length,

$Left_C$ and $Right_C$ are the current code interval's left and right border.

Arithmetic Coding

a = [0, 0.4), b = [0.4, 0.6), c = [0.6, 0.8), d = [0.8, 1)

- For the first encoded symbol of "*d*", its initial interval is [0.8, 1).
- For the second symbol "*a*", because the prior symbol "*d*" is constrained in [0.8, 1), the value of "*a*" should be in the [0, 0.4) subinterval of [0.8, 1), then

$$\begin{cases} Start_N = 0.8 + 0 \times (1.0 - 0.8) = 0.8 \\ End_N = 0.8 + 0.4 \times (1.0 - 0.8) = 0.88 \end{cases}$$

- That is, the coding interval of symbol "*da*" lying in [0.8, 0.88).

Arithmetic Coding

$a = [0, 0.4)$, $b = [0.4, 0.6)$, $c = [0.6, 0.8)$, $d = [0.8, 1)$

- For the third symbol "c", it should be in the subinterval [0.6, 0.8) of [0.8, 0.88). Then:

$$\begin{cases} Start_N = 0.8 + 0.6 \times (0.88 - 0.8) = 0.848 \\ End_N = 0.8 + 0.8 \times (0.88 - 0.8) = 0.864 \end{cases}$$

- For the fourth symbol "a":

$$\begin{cases} Start_N = 0.848 + 0 \times (0.864 - 0.848) = 0.848 \\ End_N = 0.848 + 0.4 \times (0.864 - 0.848) = 0.8544 \end{cases}$$

Arithmetic Coding

- For the fifth symbol "*b*":

$$\begin{cases} Start_N = 0.848 + 0.4 \times (0.8544 - 0.848) = 0.85056 \\ End_N = 0.848 + 0.6 \times (0.8544 - 0.848) = 0.85144 \end{cases}$$

- Now, the source information "*dacab*" has been described as a real number interval [0.85056, 0.85144), or we can say that arbitrary real number in this interval can be used to represent the source information.
- We can express the interval [0.85056, 0.85144) into binary form [0.110110011011, 0.110110011110).

Arithmetic Coding

Decimal fraction to binary fraction:

0. 85056:

0.85056*2=1.70112
0.70112*2=1.40224
0.40224*2=0.80448
0.80448*2=1.60896
0.60896*2=1.21972
0.21972*2=0.43584
0.43584*2=0.87168
0.87168*2=1.74336
0.74336*2=1.48672
0.48672*2=0.97344
0.97344*2=1.94688
0.94688*2=1.89376

0.110110011011

Decimal fraction to binary fraction:

0. 85144:

0.85144*2=1.70288
0.70228*2=1.40456
0.40456*2=0.80912
0.80912*2=1.61824
0.61824*2=1.23648
0.23648*2=0.47296
0.47296*2=0.94592
0.94592*2=1.89184
0.89184*2=1.78368
0.78368*2=1.56736
0.56736*2=1.13472
0.13472*2=0.26944

0.110110011110

Arithmetic Coding

[0.110110011011, 0.110110011110)

- We find that 0.1101100111 lies in this interval, and has the shortest length. Then we can take it as the output of the input series symbols "*dacab*".
- Since all the encoded results includes "0.", then we can delete it, and take 1101100111 as the arithmetic coding results in this example.
- Compression Ratio: $C = 40/10 = 4$.

Arithmetic Coding

- Source: *dacab*; Decoding: 1101100111

  Binary fraction to Decimal fraction:

  0.1101100111:

  0.1 → 0.5
  0.01 → 0.25
  0.0001 → 0.0625
  0.00001 → 0.03125
  0.00000001 → 0.00390625
  0.000000001 → 0.001953125
  0.0000000001 → 0.0009765625

  0.5+0.25+0.0625+0.03125+0.00390625+0.001953125+0.0009765625 = 0.8505859375

  [0.85056, 0.85144)

Arithmetic Coding

- Source: *dacab*; Decoding: 0.8505859375
- $a = [0, 0.4)$, $b = [0.4, 0.6)$, $c = [0.6, 0.8)$, $d = [0.8, 1.0)$
  - $0.8505859375 \in [0.8, 1.0) \rightarrow d$
    - Segmentation [0.8, 1.0) into 4 regions:
      $a = [0.8, 0.88)$, $b = [0.88, 0.92)$, $c = [0.92, 0.96)$, $d = [0.96, 1.0)$
  - $0.8505859375 \in [0.8, 0.88) \rightarrow a$
    - Segmentation [0.8, 0.88) into 4 regions:
      $a = [0.8, 0.832)$, $b = [0.832, 0.848)$, $c = [0.848, 0.864)$, $d = [0.864, 0.88)$
  - $0.8505859375 \in [0.848, 0.864) \rightarrow c$

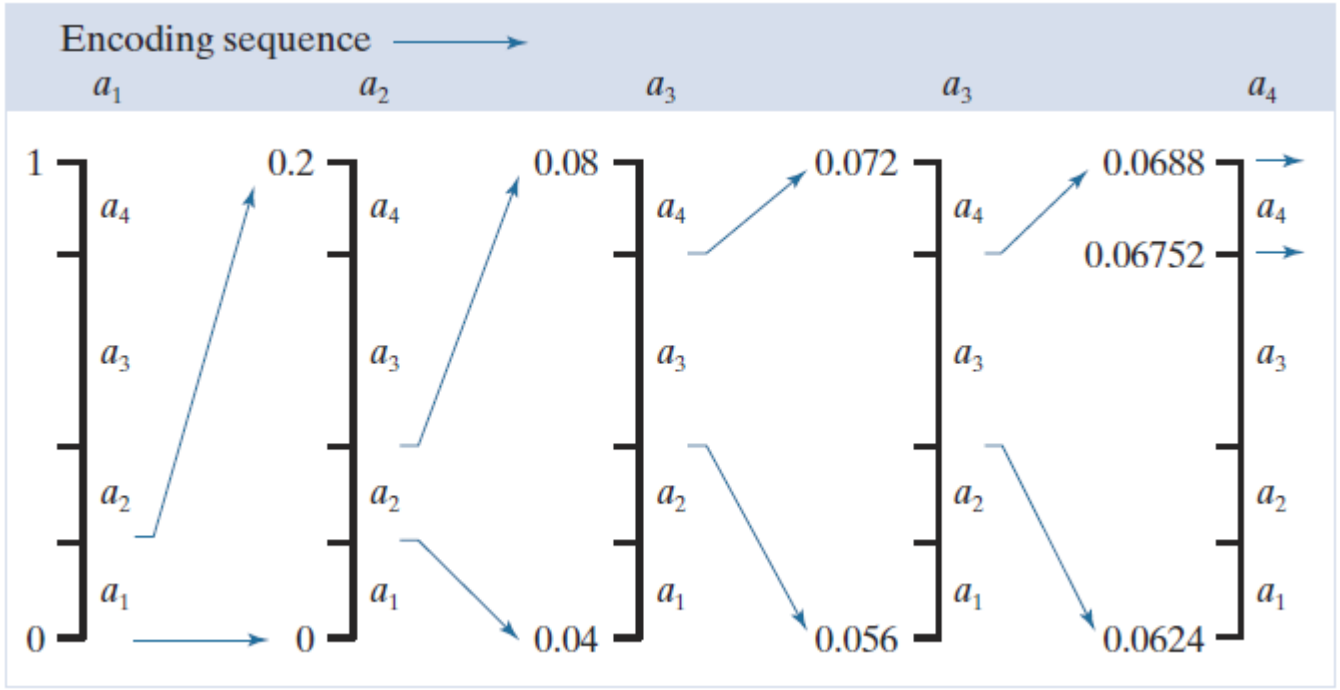
……

$$1101100111 \rightarrow 0.8505859375 \rightarrow dacab$$

Arithmetic Coding

- Another Example: Source $\underline{a_1 a_2 a_3 a_3 a_4}$

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)



LZW Coding

- Invented by Lempel and Ziv in 1977, and improved by Welch in 1984, and there are numerous variations and improvements since then.
- LZW compression has been integrated into a variety of mainstream imaging files formats:
  - graphic interchange format (GIF)
  - tagged image file format (TIFF)
  - portable document format (PDF)

LZW Coding

- Lempel-Ziv-Welch (LZW) coding
  - Assigns fixed-length code words to variable length sequences of source symbols.
  - Requires no priori knowledge of the probability of occurrence of the symbols to be encoded.
- LZW coding is conceptually very simple.
- At the onset of the coding process, a codebook or ‘dictionary’ containing the source symbols to be coded is constructed.

LZW Coding

Example: 8-bit monochrome images

1) The first 256 words of the dictionary are assigned to the gray values 0, 1, 2, … , 255.
2) As the encoder sequentially examines the image's pixels, gray-level sequences that are not in the dictionary are placed in algorithmically determined locations.
3) If the first two pixels of image are white, for instance, sequence 255-255 might be assigned to location 256, the address following the locations reserved for gray levels 0~255.
4) The next time, when 255-255 is encountered, code word 256, the address of the location containing this sequence, is used to represent them.

LZW Coding

- If a 9-bit, 512-word dictionary is employed in the coding process, the original (8+8) bits that were used to represent the two pixels are replaced by a single 9-bit code word.
- The size of the dictionary is an important system parameter.
  - If it is too small, the detection of matching gray-level sequences will be less likely.
  - If it is too large, the size of the code words will adversely affect compression performance.

LZW Example

Example: Consider the following 4 × 4, 8-bit image of a vertical edge:

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Initialization: 9 bits dictionary

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

Locations 256 through 511 initially are unused.

LZW Example

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			

- The image is encoded by processing its pixels (column 2) in a left-to-right, top-to-bottom manner.

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

LZW Example

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			

- Each successive intensity value is concatenated with a variable (column 1), called the "currently recognized sequence."
- This variable is initially null or empty.

LZW Example

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39			
	126			
	126			
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			

- The dictionary (0, …, 255, and then column 5) is searched for each concatenated sequence.
- If found, as was the case in the row 1 of the table, the currently recognized sequence (column 1) is replaced by the newly concatenated and recognized (i.e., located in the dictionary) sequence. This was done in column 1 of row 2.
- No output codes (column 3) are generated, nor is the dictionary altered (column 5).

LZW Example

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39–39
39	126			
	126			
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			
	39			
	39			
	126			
	126			

- If the concatenated sequence is *not* found, as was the case in the row 2 of the table, (1) the address of the currently recognized sequence is output as the next encoded value (column 3), (2) the concatenated but unrecognized sequence is added to the dictionary (column 5), and (3) the currently recognized sequence is initialized to the current pixel value (column 1).

LZW Example

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39–39
39	126	39	257	39–126
126	126	126	258	126–126
126	39	126	259	126–39
39	39			
39–39	126	256	260	39–39–126
126	126			
126–126	39	258	261	126–126–39
39	39			
39-39	126			
39–39–126	126	260	262	39–39–126–126
126	39			
126-39	39	259	263	126–39–39
39	126			
39-126	126	257	264	39–126–126
126		126		

- And so on…

The dictionary (0, …, 255, and then column 5) is searched for each concatenated sequence.

- **If the concatenated sequence is found,** the currently recognized sequence (column 1) is replaced by the newly concatenated and recognized (i.e., located in the dictionary) sequence. No output codes (column 3) are generated, nor is the dictionary altered (column 5).
- **If the concatenated sequence is *not* found,** (1) the address of the currently recognized sequence is output as the next encoded value (column 3), (2) the concatenated but unrecognized sequence is added to the dictionary (column 5), and (3) the currently recognized sequence is initialized to the current pixel value (column 1).

LZW Example

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39–39
39	126	39	257	39–126
126	126	126	258	126–126
126	39	126	259	126–39
39	39			
39–39	126	256	260	39–39–126
126	126			
126–126	39	258	261	126–126–39
39	39			
39-39	126			
39–39–126	126	260	262	39–39–126–126
126	39			
126-39	39	259	263	126–39–39
39	126			
39-126	126	257	264	39–126–126
126		126		

- The last two columns detail the intensity sequences that are added to the dictionary when scanning the entire 128-bit image.

LZW Example

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39–39
39	126	39	257	39–126
126	126	126	258	126–126
126	39	126	259	126–39
39	39			
39–39	126	256	260	39–39–126
126	126			
126–126	39	258	261	126–126–39
39	39			
39-39	126			
39–39–126	126	260	262	39–39–126–126
126	39			
126-39	39	259	263	126–39–39
39	126			
39-126	126	257	264	39–126–126
126		126		

- Then the image is encoded as: 39-39-126-126-256-258-260-259-257-126 (column 3).
- LZW algorithm compresses the original 16×8=128-bit image to 10×9=90 bits (10 9-bits codes).
- The resulting compression ratio is 128/90=1.42.

LZW Decoding

Decoding

- Define two variables:
  - S1: to store the input data (encoded data) one by one.
  - S2: a temporary variable.
  - They are initialized to empty.
- Each pixel is read into S1 one by one:
  - If S1 lies in the dictionary (originally 0, …, 255), then output the corresponding value, otherwise, output str(S2)-firststr(S1).
  - Add str(S2)-firststr(S1) to the dictionary entry, S2 = S1.

LZW Decoding

Decoding procedure: **39-39-126-126-256-258-260-259-257-126**

Input data (S1)	New string	Output data	S2	New dictionary entry
39		39	39	
39	str(S2)-firststr(S1)=39-39	39	39	256: 39-39
126	str(S2)-firststr(S1) =39-126	126	126	257: 39-126
126	str(S2)-firststr(S1)=126-126	126	126	258: 126-126
256	str(S2)-firststr(S1)=126-39	39-39	256	259: 126-39
258	str(S2)-firststr(S1)=39-39-126	126-126	258	260: 39-39-126
260	str(S2)-firststr(S1)=126-126-39	39-39-126	260	261: 126-126-39
259	str(S2)-firststr(S1)=39-39-126-126	126-39	259	262: 39-39-126-126
257	str(S2)-firststr(S1)=126-39-39	39-126	257	263: 126-39-39
126	str(S2)-firststr(S1)=39-126-126	126	126	264: 39-126-126

- In this table, the operations are carried out in a left-to-right, top-to-bottom manner.
- Each pixel is read into S1 one by one:
  - If S1 lies in the dictionary, then output the corresponding value, otherwise, output str(S2)-firststr(S1).
  - Add str(S2)-firststr(S1) to the dictionary entry, S2 = S1.

Decoded result: 39-39-126-126-39-39-126-126-39-39-126-126-39-39-126-126

LZW Coding

Remarks

- A unique feature of the LZW coding is that the coding dictionary or code book is created while the data are being encoded.
- Remarkably, an LZW decoder builds an identical decompression dictionary as in encoder.
- The dictionary is not saved in compressed file.
- Most practical applications require a strategy for handling dictionary overflow.
  - Simple solution: to flush or reinitialize when the dictionary becomes full and continue coding with a new initialized dictionary.
  - Complex solution: to monitor compression performance and flush the dictionary.

Hybrid Coding

- **An example of Hybrid Coding:**
  - Information source “aaaabbbccdeeeeeffffffff”, data size is 22×8=176(bits)
  - If it is coded by run-length method:
    - a4b3c2d1e5f7, data size is 6×(8+3)=66(bits)
    - The compression ratio is: 176:66=2.67
  - If it is coded by Huffman method:
    - We can get: f=01, e=11, a=10, b=001, c=0001, d=0000
    - The compression ratio is: 8:2.4=3.33

Hybrid Coding

- **An example of Hybrid Coding:**
  - Combine run-length and Huffman:
    - ➢It can be coded as "10400130001200001115017", data size is (2+3)+(3+3)+(4+3)+(4+3)+(2+3)+(2+3)=35(bits)
    - ➢The compression ratio is: 176:35=5.03

Lossy Image Compression

- Lossy image compression
  - Why lossy?
  - A simple example
- Lossy transform coding
  - Transform Coding
  - Wavelet Coding
  - Joint Photographic Expert Group (JPEG)

Why Lossy?

- In most applications related to consumer electronics, lossless compression is not necessary.
  - What we care is the subjective quality of the decoded image, not those intensity values.
  - Compromising the accuracy of the reconstructed image in exchange for increased compression.
  - If the resulting distortion can be tolerated, the increase in compression can be effective.

Why Lossy?

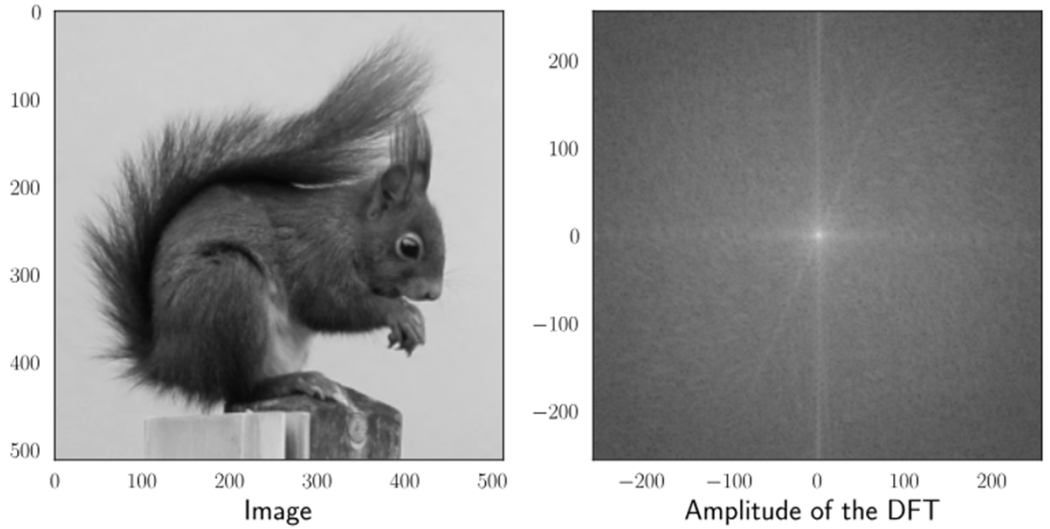
- With the relaxation, it is possible to achieve a higher compression ratio (CR).
- Many lossy encoding techniques are capable of reproducing:
  - Recognizable monochrome images from data that have been compressed by more than 100:1 (lossless coding of monochrome images, however, seldom results in more than a 3:1 reduction in data);
  - Images that are virtually indistinguishable from the originals at 10:1 to 50:1.

Transform Coding

- The coding techniques discussed before operate directly on the pixels of an image, and thus are spatial domain methods.
- Now we will look at compression techniques that are based on modifying the transform of an image (transform coding).
- A linear, reversible transform (such as the Fourier transform) is used to map the image into a set of transform coefficients, which are then quantized and coded.

Transform Coding

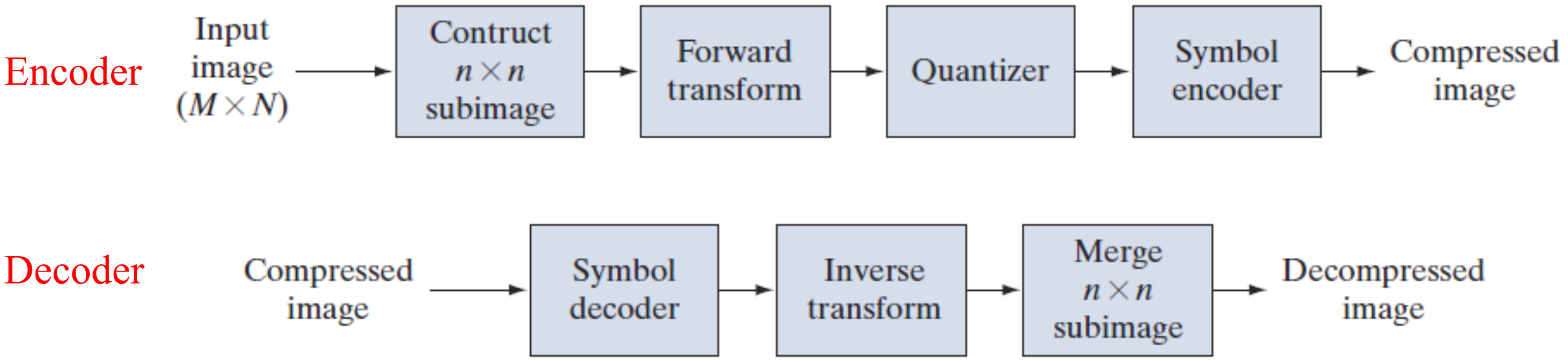
- For most natural images, a significant number of (high frequency) coefficients have small magnitudes and can be coarsely quantized with little image distortion.



- Other than the DFT, we have the Discrete Cosine Transform (DCT) and the Walsh Hadamard Transform (WHT).

Transform Coding

- A block transform coding system



Transform Coding

- For the purpose of image compression, the transform operation should:
  - decorrelate the correlations, or pack as much information as possible into the smallest number of transform coefficients;
  - having basis transform functions are independent on the input images;
  - be able to achieve fast computation.

- Typical transforms:
  - ✓ DFT (Discrete Fourier Transform)
  - ✓ DCT (Discrete Cosine Transform)
  - ✓ WHT (Walsh-Hadmard Transform)
  - ✓ KLT (Karhunen-Loève Transform)

Discrete Fourier Transform

- Discrete Fourier Transform (DFT)
  - Use $\cos\theta+\sin\theta$ as its basis functions.
  - Can be implemented using Fast Fourier Transform (FFT).
  - Not so popular in image compression because its performance is not good enough.

Discrete Fourier Transform

- Discrete Fourier Transform (DFT)

Transform:

$$F(u,v) = \frac{1}{n}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} f(j,k)\exp\left\{-\frac{2\pi i(uj+vk)}{n}\right\}$$

Inverse transform:

$$f(j,k) = \frac{1}{n}\sum_{u=0}^{n-1}\sum_{v=0}^{n-1} F(u,v)\exp\left\{\frac{2\pi i(uj+vk)}{n}\right\}$$

$i = \sqrt{-1}$ and $f(j, k)$ is the input sequence.

Discrete Cosine Transform

- Discrete Cosine Transform (DCT)
  - Use cosine function as its basis function.
  - Performance approaches KLT (optimal).
  - Fast algorithm exists.
  - Most popular in image compression application.
  - Adopted in JPEG.

2D DCT

- 2D DCT: transfer spatial domain to frequency domain
  - DC: $F(0,0)$, average of $f(x, y)$
  - AC: other coefficients

$$F(u,v) = a(u)a(v)\sum_{x=0}^{n-1}\sum_{y=0}^{n-1} f(x,y)\cos\left[\frac{(2x+1)u\pi}{2n}\right]\cos\left[\frac{(2y+1)v\pi}{2n}\right]$$

$$f(x,y) = a(u)a(v)\sum_{u=0}^{n-1}\sum_{v=0}^{n-1} F(u,v)\cos\left[\frac{(2x+1)u\pi}{2n}\right]\cos\left[\frac{(2y+1)v\pi}{2n}\right]$$

$$a(u) = \begin{cases} \sqrt{1/N} & \text{if } u = 0 \\ \sqrt{2/N} & \text{if } u > 0 \end{cases}$$

Walsh-Hadamard Transform

- Walsh-Hadamard Transform (WHT)
  - Simple basis functions: all kernel values are +1 or -1.
  - Only addition and subtraction operations are needed.
  - Popular when efficiency consideration dominates.
  - Performance moderate.

Walsh-Hadamard Transform

- The smallest Hadamard matrix is H_2: $$H_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

- A bigger Hadamard matrix can be defined as : $$H_{2n} = \frac{1}{\sqrt{2}}\begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix}$$

For example: $$H_4 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Karhunen-Loeve Transform

- Karhunen-Loeve Transform (KLT)
  - Optimal transform, better than prior transforms.
  - Basis functions are image dependent.
  - No fast algorithm exists.
  - Not so useful in image compression.
  - Usually used for comparison.

Transform Selection

- **Steps for transform-based image compression**

1) Divide the original image into 8×8 (or other size) subimages.
2) Transforme the image by DFT, WHT, or DCT.
3) Truncate 50% (or some other percentage) of the resulting coefficients. Then the compression ratio is 2:1.
4) Take the inverse transform.

Transform Selection

Original image

Left column: compressed image

Right column: error (scaled)

DFT
rms=1.28

WHT
rms=0.86

DCT
rms=0.68

It shows that DCT is superior to DFT and WHT.

Transform Selection

compressed image

error (scaled)

Left: DFT
rms=2.32

Middle: WHT
rms=1.78

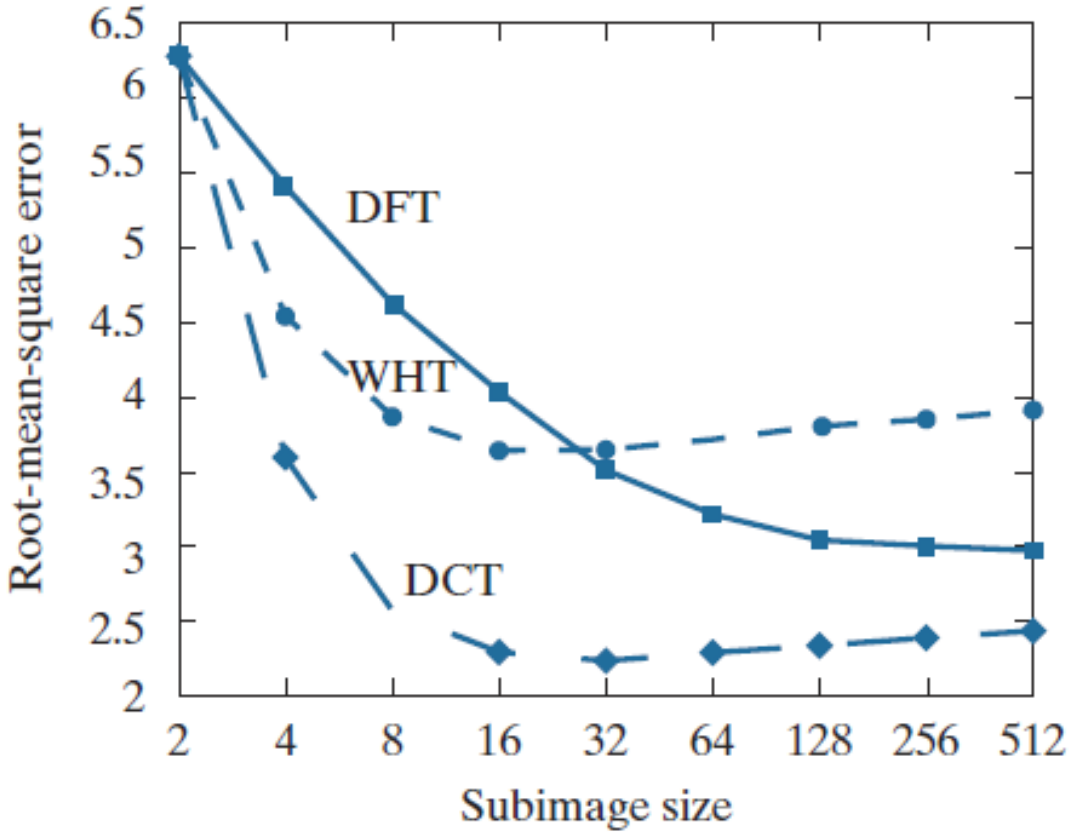
Right: DCT
rms=1.13

Transform Selection

Subimage size selection:

- The size of subimage will be significantly affecting transform coding error and computational complexity.
- In most applications, images are subdivided so that the correlation redundancy between adjacent subimages is reduced to some acceptable level.
- In general, both the level of compression and the computational complexity increase as the subimage size increases.
- The most popular sizes are 8×8, or 16×16.

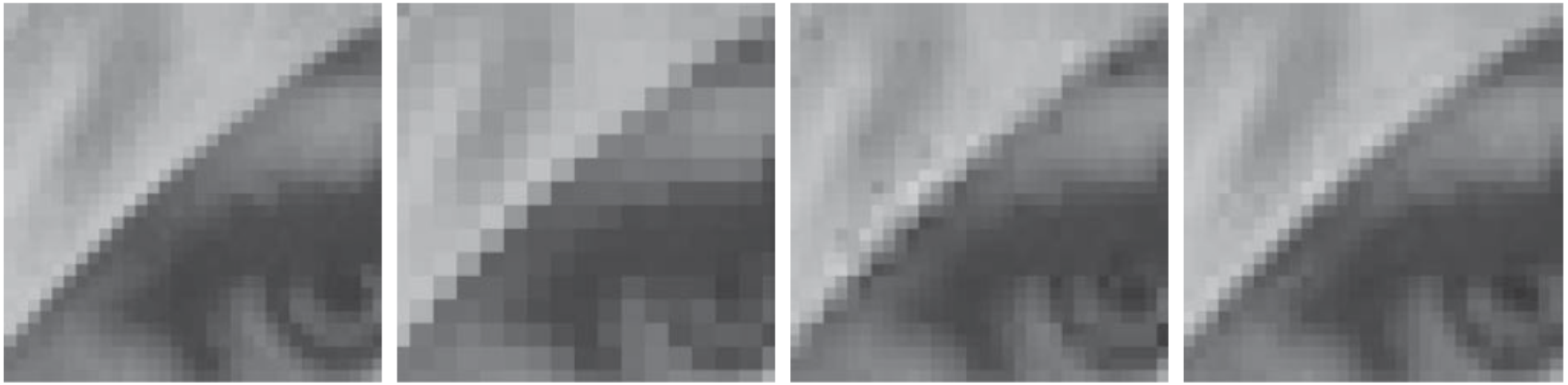
Transform Selection



Computing the transform of each subimage with different size, truncating 75% of the resulting coefficients, and taking the inverse transform.

Transform Selection

Approximations of (a) using 25% of the DCT coefficients and (b) 2 × 2 subimages, (c) 4 × 4 subimages, and (d) 8 × 8 subimages.



(a) (b) (c) (d)

Wavelet Transform

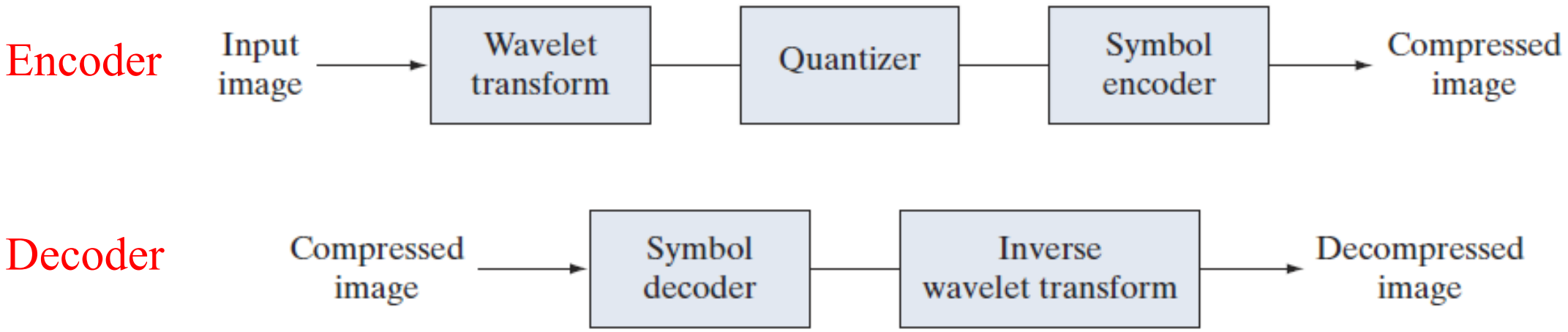
- Wavelet coding is based on the idea that the coefficients of the transform that decorrelates the pixels of an image can be coded more efficiently than the original pixels themselves.
- Wavelets can pack most of the important visual information into a small number of coefficients, the remaining coefficients can be quantized coarsely or truncated to zero with little image distortion.
- Since many of the computed coefficients carry little visual information, they can be quantized and coded to minimize intercoefficient and code redundancy.

Wavelet Transform

- Moreover, one or more of the lossless coding methods can be incorporated into the final symbol coding step.
- The main difference between the wavelet transform and the transform coding system is the omission of the transform coder's subimage processing stages.
- Because wavelet transforms are both computationally efficient and inherently local, subdivision of the original image is unnecessary.

Wavelet Transform

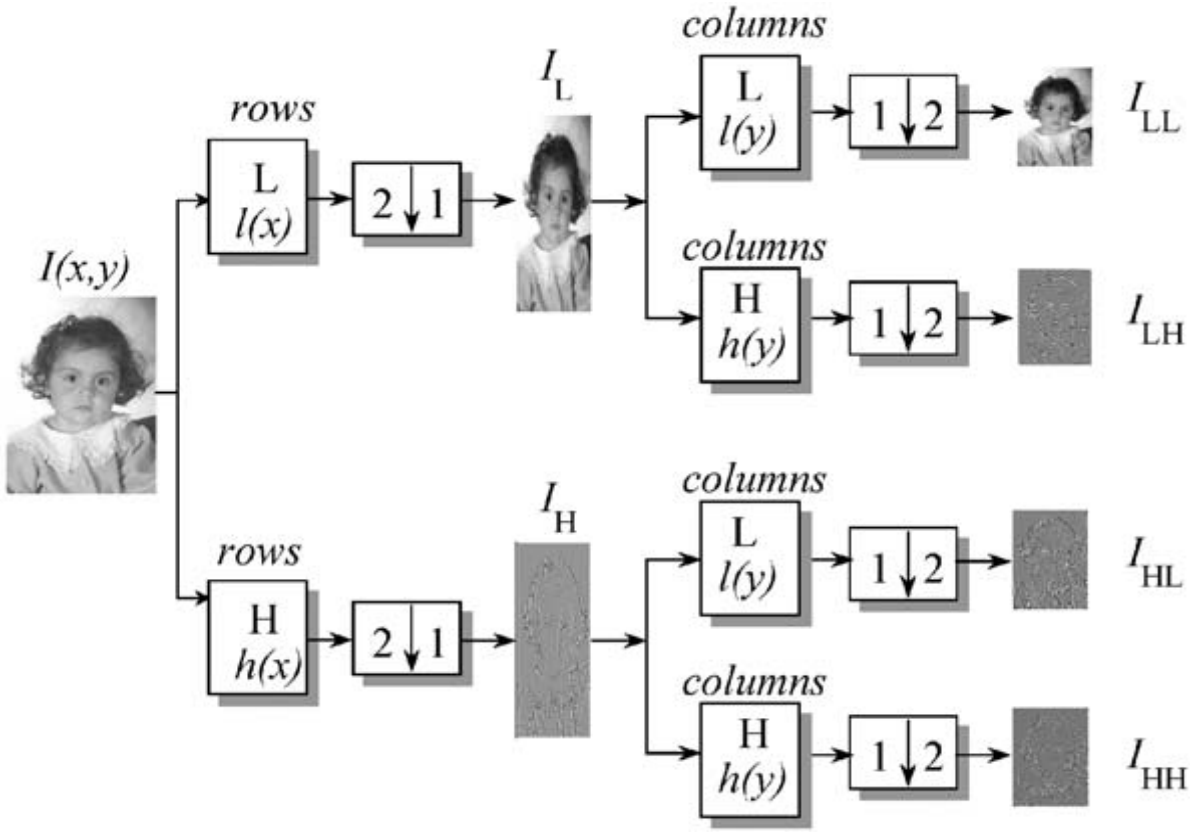
- A wavelet coding system



Wavelet Transform

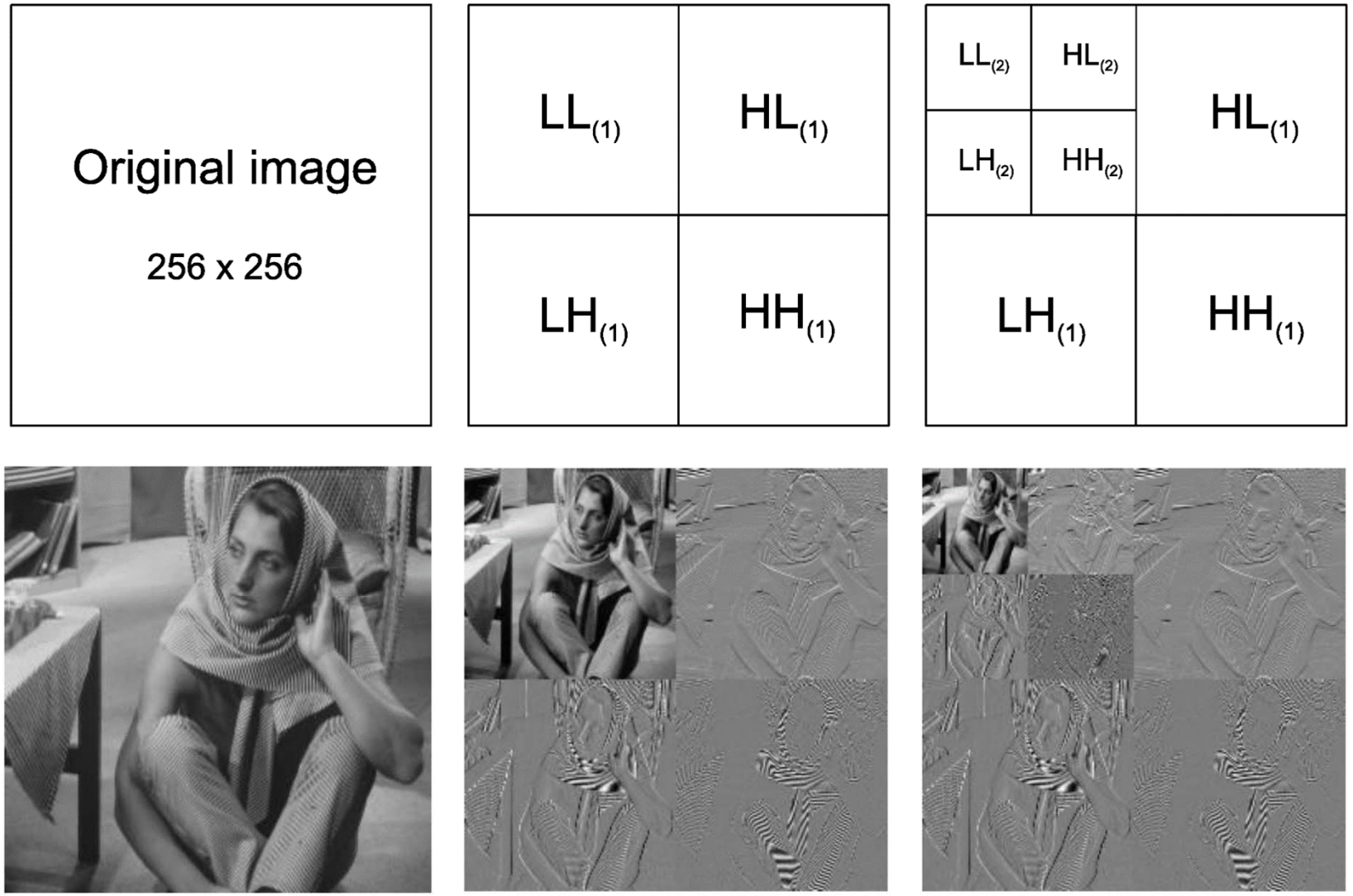
- Initially, wavelet transform just focused on the 1-D situation.
- 2-D wavelets, in the same way, can be treated as two 1-D wavelet transforms: one 1-D wavelet transforms along the row direction and the other one along the column direction.
- Generally, an one-octave transform of an image can be decomposed into four components:
  - low-pass rows, low-pass columns (LL);
  - low-pass rows, high-pass columns (LH);
  - high-pass rows, low-pass columns (HL);
  - high-pass rows, high-pass columns (HH).

Wavelet Transform



- LL: low resolution of $I(x,y)$
- HL: vertical high frequencies of $I(x,\text{i})$
- LH: horizontal high frequencies of $I(\text{i},y)$
- HH: high frequencies in both directions of $I(x,y)$

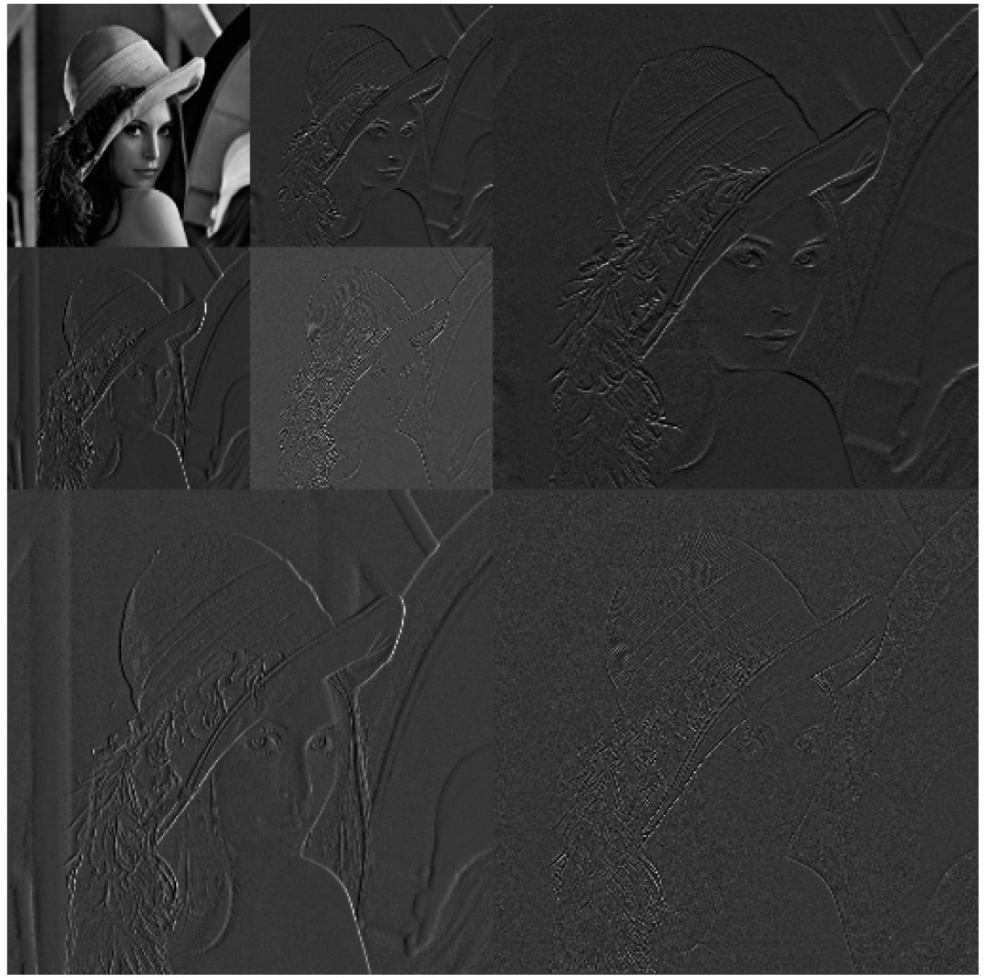
Wavelet Transform



The coefficients obtained by applying the 2-D wavelet transform on an image are called the subimages of the wavelet transform.

Wavelet Transform

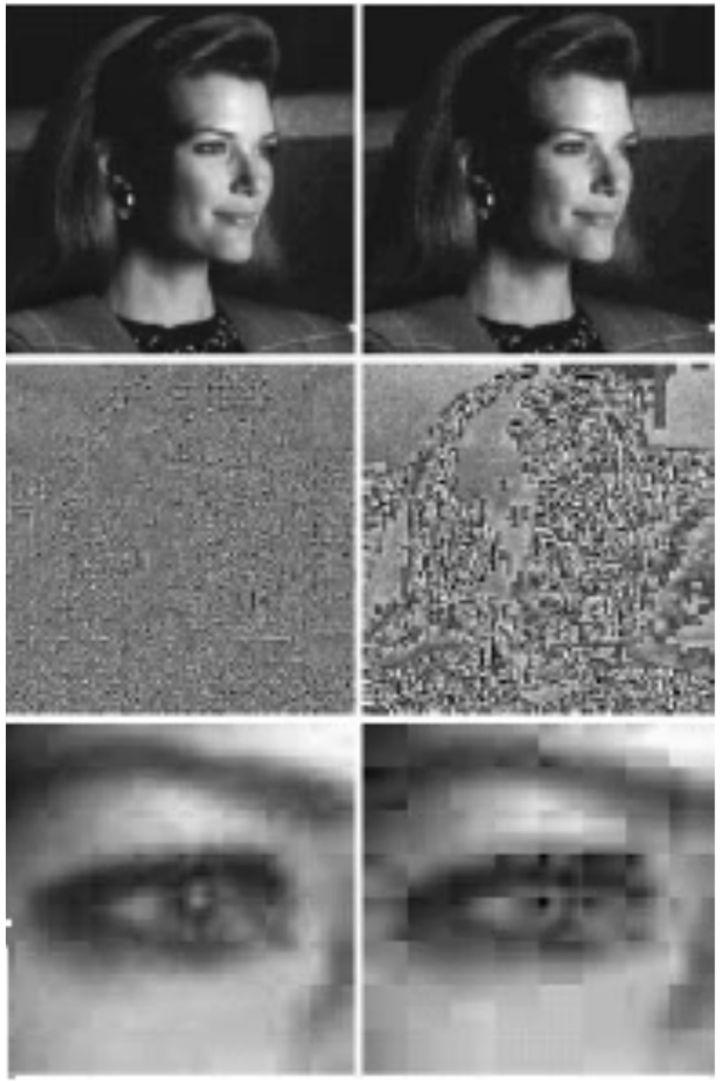
(a) First scale Haar wavelet transform



(b) Second scale Haar wavelet transform

Wavelet Transform

DCT



Comparison between DCT and Wavelet

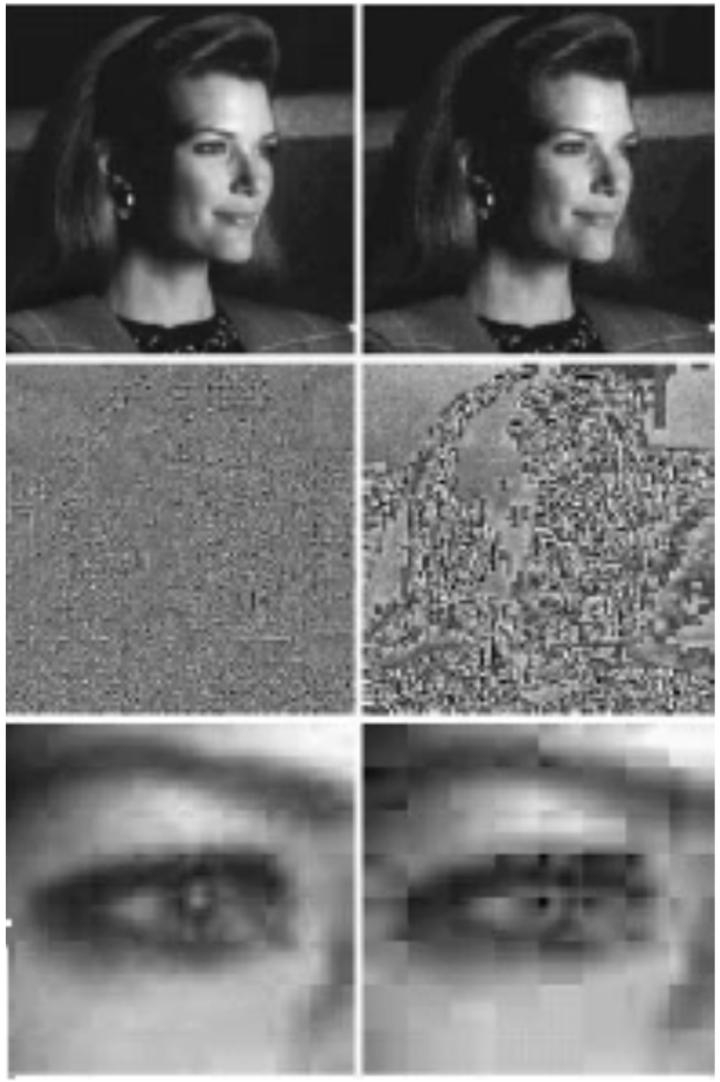
- Compression ratios are 34:1 and 67:1 for both methods.
- For DCT, the rms are 3.42 and 6.33.
- For wavelet, the rms are 2.29, 2.96.

- It reveals a noticeable decrease of error in the wavelet coding results.
- Wavelet coding also greatly increases image quality.

Wavelet

Wavelet Transform

DCT



Comparison between DCT and Wavelet

- Compression ratios are 34:1, and 67:1 for DCT, and 108:1, 167:1 for wavelet.
- For DCT, the rms are 3.42 and 6.33.
- For wavelet, the rms are 3.72 and 4.73.

- But a subjective evaluation of either image reveals wavelet is obvious better than DCT.

Wavelet

JPEG

- JPEG - Joint Photographic Experts Group
  - Joint refers to CCITT and ISO.
  - Compression standard of general continuous-tone still image.
  - Became an international standard in 1992.
- It defines three different coding systems
  - A lossy baseline coding system, which is based on the DCT and is adequate for most compression applications.
  - An extended coding system for greater compression, higher precision, or progressive reconstruction applications.
  - A lossless independent coding system for reversible compression.

JPEG 2000

- Eliminates the blocky appearance of the JPEG image standard.
  - This is because it uses a wavelet transform instead of DCT.
  - In the previous DCT version, blocks of the image (subimages) are compressed individually without reference to the adjoining blocks.
  - Using a discrete wavelet transform creates a much smoother image.
- The compression rate is much higher, while the retention rate is the same, and often, better resolution is exhibited.
  - 20%-200% better than JPEG Standard with lossy compression.

JPEG vs. JPEG 2000

discrete cosine transform based

JPEG (CR=64)

wavelet transform based

JPEG2000 (CR=64)

Summary

- In this lecture we have learnt:
  - Lossless Image Compression
    - Arithmetic Coding
    - LZW Coding
    - Hybrid Coding
  - Lossy Image Compression
    - Transform Coding (DFT, DCT, WHT, KLT)
    - Wavelet Coding
    - JPEG

Optional Homework

Check the Textbook!

- **Chapter 8: Problems 8.18, 8.20**

- Homework answers will be provided at the end of each week.