

# Digital Image Processing

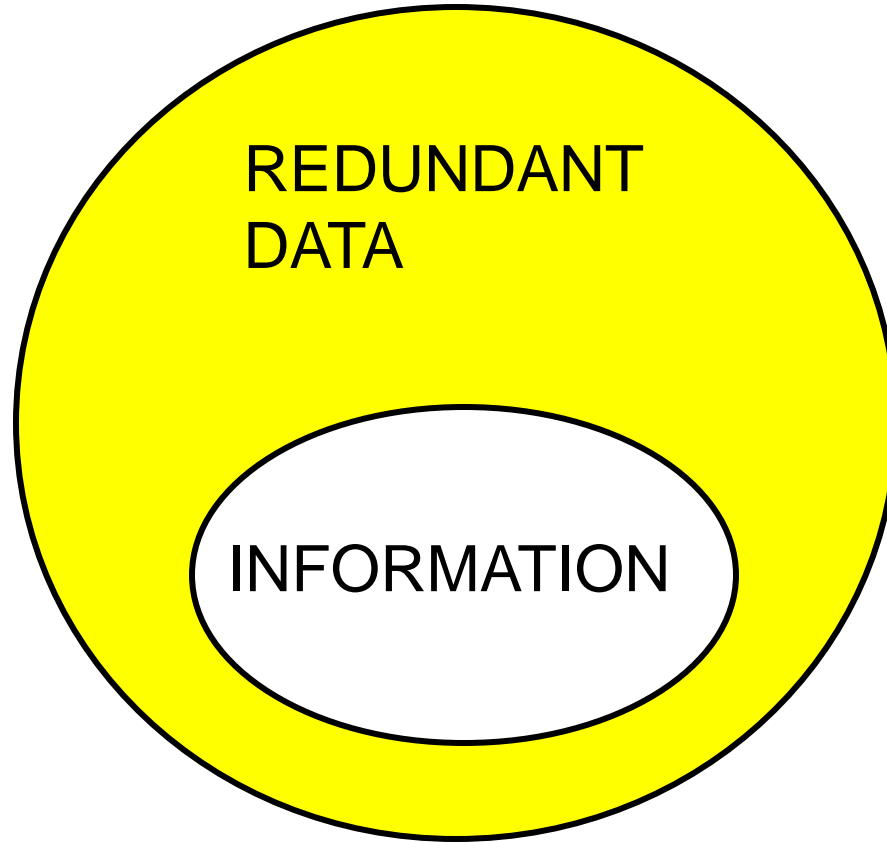
## **Lecture # 10** **Image Compression**

# IMAGE COMPRESSION

# IMAGE COMPRESSION

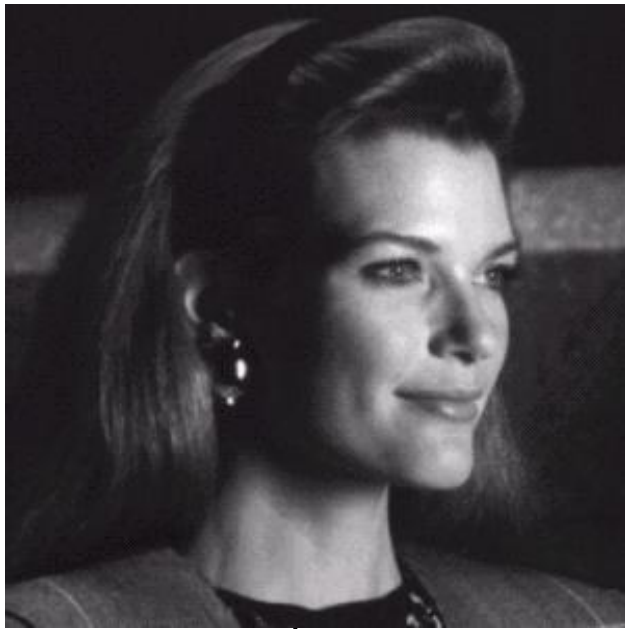
- Addresses the problem of reducing the amount of data required to represent a digital image
- The underlying basis of the reduction process is the removal of redundant data

# Information vs Data



DATA = INFORMATION + REDUNDANT DATA

# IMAGE COMPRESSION: CODING AND DECODING

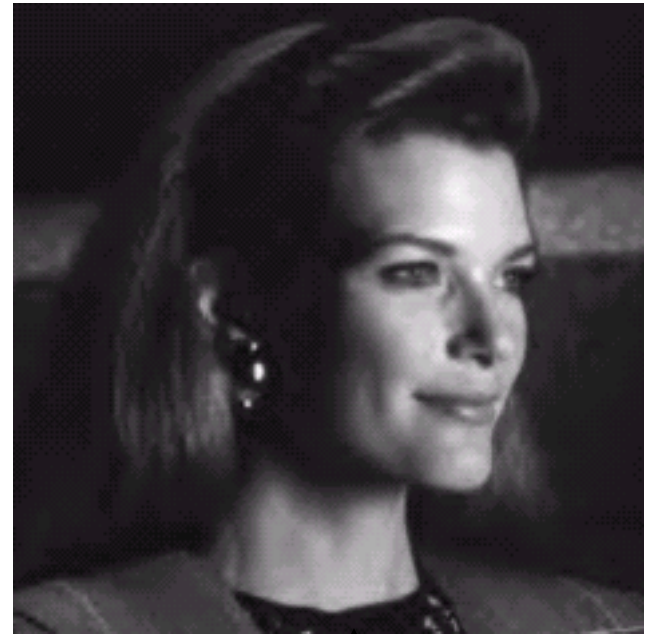


original image  
262144 Bytes

**image  
encoder**

compressed bit stream  
00111000001001101...  
(2428 Bytes)

**image  
decoder**



compression ratio (CR) = 108:1

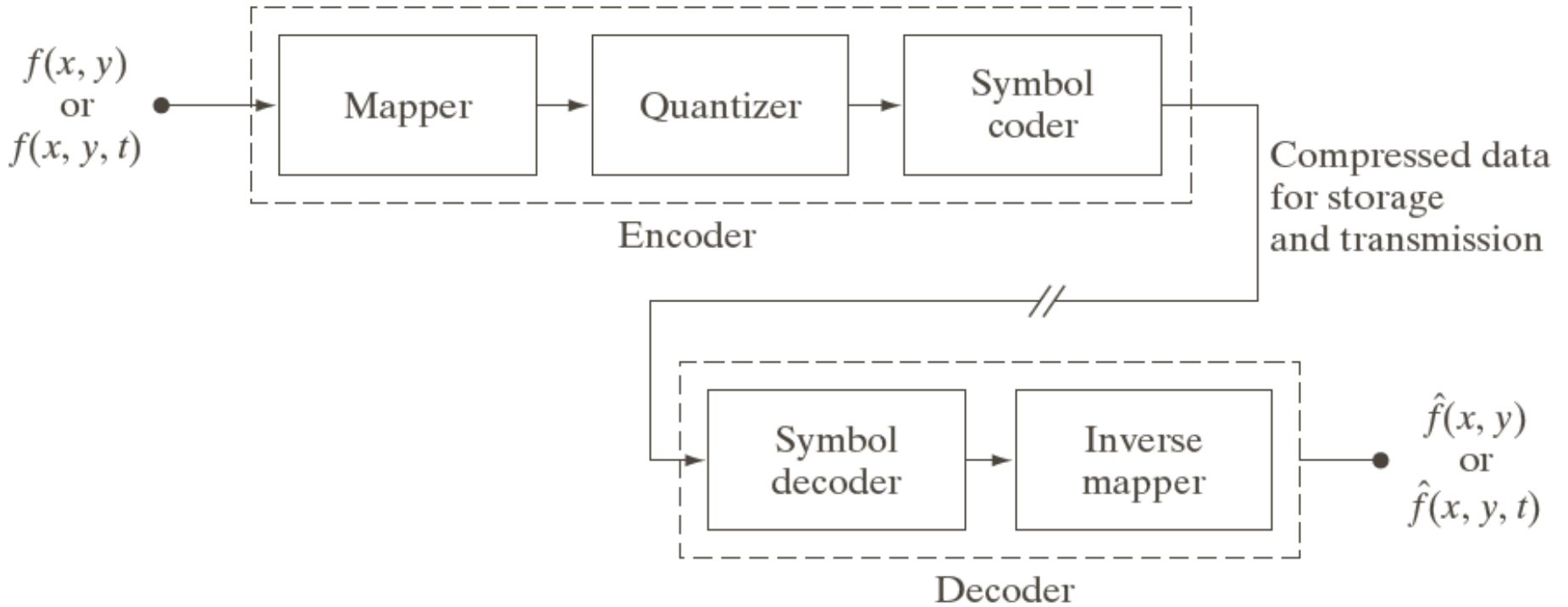
# LOSSY VS LOSSLESS COMPRESSION

- Lossless (Information preserving)
  - Images can be compressed and restored without any loss of information.
  - Application: Medical images
- Lossy
  - Perfect recovery is not possible but provides a large data compression.
  - Example: TV signals, teleconferencing

# FUNDAMENTALS

- Raw image: A set of  $n_1$  bits
- Compressed image: A set of  $n_2$  bits.
- Compression ratio: 
$$C_R = \frac{n_1}{n_2}$$
- Relative Data Redundancy of first set: 
$$R_D = 1 - \frac{1}{C_R}$$
- Example:  $n_1 = 100\text{KB}$  and  $n_2 = 10\text{Kb}$ , then  $CR = 10$ , and  $RD = 90\%$
- Special cases:
  - $n_1 \gg n_2 \rightarrow CR \approx \infty, RD \approx 1$
  - $n_1 \approx n_2 \rightarrow CR \approx 1, RD \approx 0$

# IMAGE COMPRESSION MODEL



**FIGURE 8.5**  
Functional block diagram of a general image compression system.

# DATA REDUNDANCY

- Three basic data redundancies:
  - Coding redundancy
  - Spatial and Temporal redundancy
  - Irrelevant Information



a b c

**FIGURE 8.1** Computer generated  $256 \times 256 \times 8$  bit images with (a) coding redundancy, (b) spatial redundancy, and (c) irrelevant information. (Each was designed to demonstrate one principal redundancy but may exhibit others as well.)

---

# CODING REDUNDANCY

- Type of coding (# of bits for each gray level)
- Image histogram:
  - $rk$ : Represents the gray levels of an image
  - $pr(rk)$ : Probability of occurrence of  $rk$

$$p_r(r_k) = \frac{n_k}{n} \quad k = 0, 1, 2, \dots, L-1$$

- $l(rk)$ : Number of bits used to represent each  $rk$  (after compression)
- $L_{avg}$ : Average # of bits required to represent each pixel:

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

# CODING REDUNDANCY

- It makes sense to assign fewer bits to those  $r_k$  for which  $p_r(r_k)$  are large in order to reduce the sum.
- This achieves data compression and results in a variable length code.
- More probable gray levels will have fewer # of bits.
- Basic type is variable length coding

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

# VARIABLE LENGTH CODING

$r_k$	$p_r(r_k)$	code1	$l_1(r_k)$	code2	$l_2(r_k)$
$r_0=0$	<b>0.19</b>	<b>000</b>	<b>3</b>	<b>11</b>	<b>2</b>
$r_1=1/7$	<b>0.25</b>	<b>001</b>	<b>3</b>	<b>01</b>	<b>2</b>
$r_2=2/7$	<b>0.21</b>	<b>010</b>	<b>3</b>	<b>10</b>	<b>2</b>
$r_3=3/7$	<b>0.16</b>	<b>011</b>	<b>3</b>	<b>001</b>	<b>3</b>
$r_4=4/7$	<b>0.08</b>	<b>100</b>	<b>3</b>	<b>0001</b>	<b>4</b>
$r_5=5/7$	<b>0.06</b>	<b>101</b>	<b>3</b>	<b>00001</b>	<b>5</b>
$r_6=6/7$	<b>0.03</b>	<b>110</b>	<b>3</b>	<b>000001</b>	<b>6</b>
$r_7=1$	<b>0.02</b>	<b>111</b>	<b>3</b>	<b>000000</b>	<b>6</b>

# VARIABLE LENGTH CODING

- Computing  $L_{avg}$

$$L_{avg} = \sum_{k=0}^7 l_2(r_k) p_r(r_k)$$

$$= 2(0.19) + 2(0.05) + 2(0.21) + 3(0.16) + 4(0.08) + 5(0.06) + 6(0.03) + 6(0.02)$$

$$= 2.7 \text{ bits}$$

- $C_R = 3/2.7 = 1.11$

- $R_D = 1 - 1/1.11 = 0.099 = 9.9\%$

# LOSSLESS GRAY-SCALE IMAGE COMPRESSION

# VARIABLE WORD LENGTH CODING: EXAMPLE

✚ A 4x4 4bits/pixel original image is given by

## Default Code Book

- 0: 0000
- 1: 0001
- 2: 0010
- 3: 0011
- 4: 0100
- 5: 0101
- 6: 0110
- 7: 0111
- 8: 1000
- 9: 1001
- 10: 1010
- 11: 1011
- 12: 1100
- 13: 1101
- 14: 1110
- 15: 1111

2	8	6	6
6	8	8	8
8	8	10	10
9	10	10	14

↓ encode

0010	1000	0110	0110
0110	1000	1000	1000
1000	1000	1010	1010
1001	1010	1010	1110

Bit rate = 4bits/pixel

Total # of bits used to represent the image:

$$4 \times 16 = 64 \text{ bits}$$

# VARIABLE WORD LENGTH CODING: EXAMPLE

✚ Encode the original image with a **CODE BOOK** given left

## Huffman Code Book

0: 0000000  
 1: 0000001  
 2: 0001  
 3: 0000010  
 4: 0000011  
 5: 0000100  
 6: 01  
 7: 0000101  
 8: 10  
 9: 00100  
 10: 11  
 11: 0000110  
 12: 0000111  
 13: 001010  
 14: 0011  
 15: 001011

2	8	6	6
6	8	8	8
8	8	10	10
9	10	10	14

encode

0001	10	01	01
01	10	10	10
10	10	11	11
00100	11	11	0011

Total # of bits used to represent the image:

$$4+2+2+2+2+2+2+2+2+2+2+2+5+2+2+4 = 39 \text{ bits}$$

$$\text{Bit rate} = 39/16 = 2.4375 \text{ bits/pixel}$$

$$\text{CR} = 64/39 = 1.6410$$



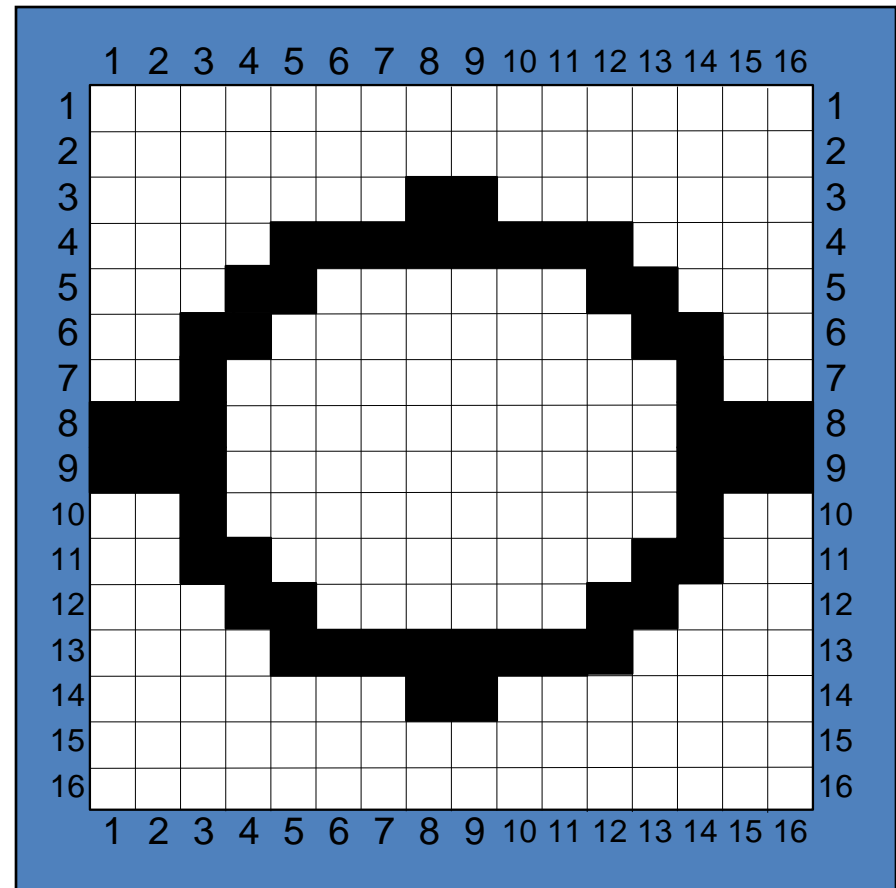
# RUN LENGTH CODING

## Decoding Example

A binary image is encoded using run length code row by row, where "0" represents white, and "1" represents black. The code is given by

Row 1: "0", 16  
Row 2: "0", 16  
Row 3: "0", 7, 2, 7  
Row 4: "0", 4, 8, 4  
Row 5: "0", 3, 2, 6, 3, 2  
Row 6: "0", 2, 2, 8, 2, 2  
Row 7: "0", 2, 1, 10, 1, 2  
Row 8: "1", 3, 10, 3  
Row 9: "1", 3, 10, 3  
Row 10: "0", 2, 1, 10, 1, 2  
Row 11: "0", 2, 2, 8, 2, 2  
Row 12: "0", 3, 2, 6, 3, 2  
Row 13: "0", 4, 8, 4  
Row 14: "0", 7, 2, 7  
Row 15: "0", 16  
Row 16: "0", 16

decode



# Entropy

Entropy calculation for a two symbol alphabet.

Example 1:

A	$p_A=0.5$
B	$p_B=0.5$

$$\begin{aligned} H(A, B) &= -p_A \log_2 p_A - p_B \log_2 p_B = \\ &= -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1 \end{aligned}$$

*It requires one bit per symbol on the average to represent the data.*

Example 2:

A	$p_A=0.8$
B	$p_B=0.2$

$$\begin{aligned} H(A, B) &= -p_A \log_2 p_A - p_B \log_2 p_B = \\ &= -0.8 \log_2 0.8 - 0.2 \log_2 0.2 \cong 0.7219 \end{aligned}$$

*It requires less than one bit per symbol on the average to represent the data.*

*How can we code this ?*

# HUFFMAN CODING

- ✚ Uses frequencies (Probability) of symbols in a string to build a variable rate prefix code.
- ✚ Each symbol is mapped to a binary string.
- ✚ More frequent symbols have shorter codes.
- ✚ No code is a prefix of another. (Uniquely decodable)

# HUFFMAN CODING

- Each symbol is assigned a variable-length code, depending on its frequency. The higher its frequency, the shorter the codeword
- Number of bits for each codeword is an integral number
- A prefix code
- A variable-length code
- Huffman code is the **optimal** prefix and variable-length code, **given** the symbols' probabilities of occurrence
- Codewords are generated by building a Huffman Tree

# HUFFMAN CODING

- **Prefix code** - No codeword is a prefix of any other codeword.

A = 0; B = 10; C = 110; D = 111

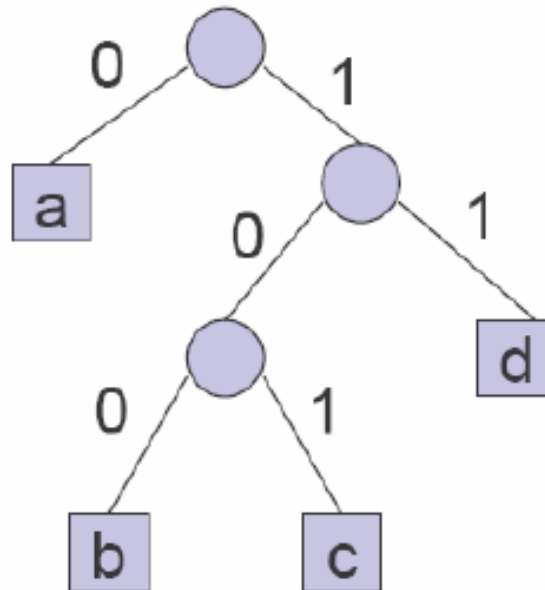
- **Uniquely decodable code** - Has only one possible source string producing it.
  - Unambiguously decoded
  - Examples:
    - Prefix code - the end of a codeword is immediately recognized without ambiguity: 010011001110 → 0 | 10 | 0 | 110 | 0 | 111 | 0
    - Fixed-length code

# HUFFMAN CODING

## Example:

- We have four symbols a, b, c, and d.

a 0  
b 100  
c 101  
d 11



# HUFFMAN CODING

✚ A source string: **aabddcaa**

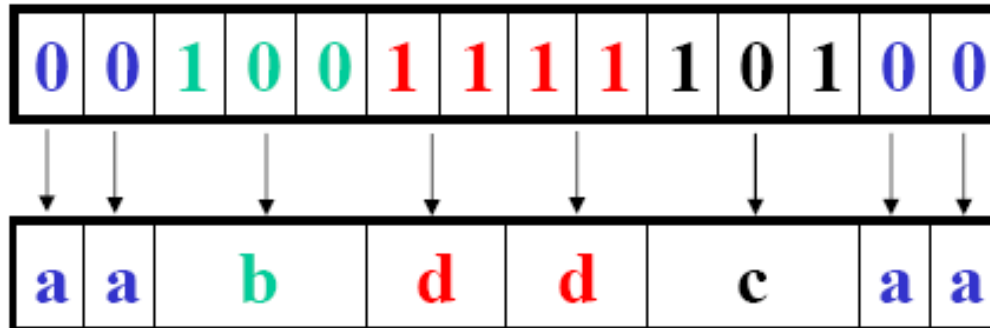
■ Fixed Length Coding: 16 bits (ordinary coding)

■ **00 00 01 11 11 10 00 00**

✚ Variable length coding: 14 bits (Huffman coding)

■ **0 0 100 11 11 101 0 0**

✚ Uniquely Decodable:



# HUFFMAN CODING

- Step-1
  - Arrange probability in decreasing order and consider them as tree leaves
- Step-2
  - Merge two nodes with smallest prob. to a new node and sum up prob.
  - Arbitrarily assign 1 and 0 to each pair of merging branch
- Step-3
  - Repeat until no more than one node left.
  - Read out codeword sequentially from root to leaf

Original source		Source reduction			
Symbol	Probability	1	2	3	4
$a_2$	0.4	0.4	0.4	0.4	0.6 0.4
$a_6$	0.3	0.3	0.3	0.3	
$a_1$	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				

**FIGURE 8.7**  
Huffman source reductions.

# HUFFMAN CODING

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								

**FIGURE 8.8**  
Huffman code  
assignment  
procedure.

# OBJECTIVE CRITERIA

- The error  $e(x,y)$  can be defined as

$$e(x, y) = \hat{f}(x, y) - f(x, y)$$

- Total error between the two images

$$\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]$$

- Root-mean-square-error

$$e_{rms} = \left[ \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]^2 \right]^{1/2}$$

- Mean-square signal-to-noise ratio of the output image

$$SNR_{ms} = \frac{\left[ \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y)]^2 \right]}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]^2}$$

# Readings from Book (3<sup>rd</sup> Edn.)

- Image Compression (Chapter-8)



# Acknowledgements

- ◆ "Digital Image Processing", Rafael C. Gonzalez & Richard E. Woods, Addison-Wesley, 2002