Image Compression

Recap

- The 2-D DFT - Some Observations
- Separability of Fourier Transform
- IDFT in terms of DFT
- Fast Fourier Transform (FFT)
  - FFT Process in 1-D
  - Special Properties of W_M
  - FFT even-odd approach
  - FFT "Butterfly" Method
  - FFT – time complexity
  - Can we speed it up??
  - FFT Algorithm

Lecture Objectives

- Introduction to Image Compression
- Types of Data Redundancy
  - Coding redundancy
  - Spatial and Temporal Redundancy
  - Irrelevant Information
- Measuring Image Information
- Fidelity Criteria
- General Image Compression Model
  - Encoding/Compression Process
  - Decoding/Decompression Process
- Lossless Compression
  - Huffman Coding

Key Stages in DIP

Image Acquisition → Image Enhancement → Image Restoration → **Image Compression** → Morphological Processing → Image Segmentation → Object Recognition → Representation & Description

Original Lena Image (12KB size)

Lena Image, Compressed (85% less information, 1.8KB)

Lena Image, Highly Compressed (96% less information, 0.56KB)

Introduction to Image Compression

Illustrating Example

8.9 MB

68.34 KB

Illustrating Example

- Let a 2-hour, **standard definition** (SD) movie using $720 \times 480 \times 24$ bit pixel arrays *to represent* *<u>each frame</u>* and the Frame rate = 30 fps.

- What is the size of the movie file?

- It requires **≈223 GB** memory to store this movie file !!!.

- We need **25 DVDs** each of 8.5 GB (single-sided, double-layer) to save this movie.

- Hence, ***we need to compress each frame***.

- Compression must be even higher for **high definition(HD)**, where image resolution is $1920 \times 1080 \times 24$ bits/frame.

Illustrating Examples

- Image: 6.0 megapixels camera, 3000×2000 px/inch
  - 13804 MB per image → 13.48 1GB/image

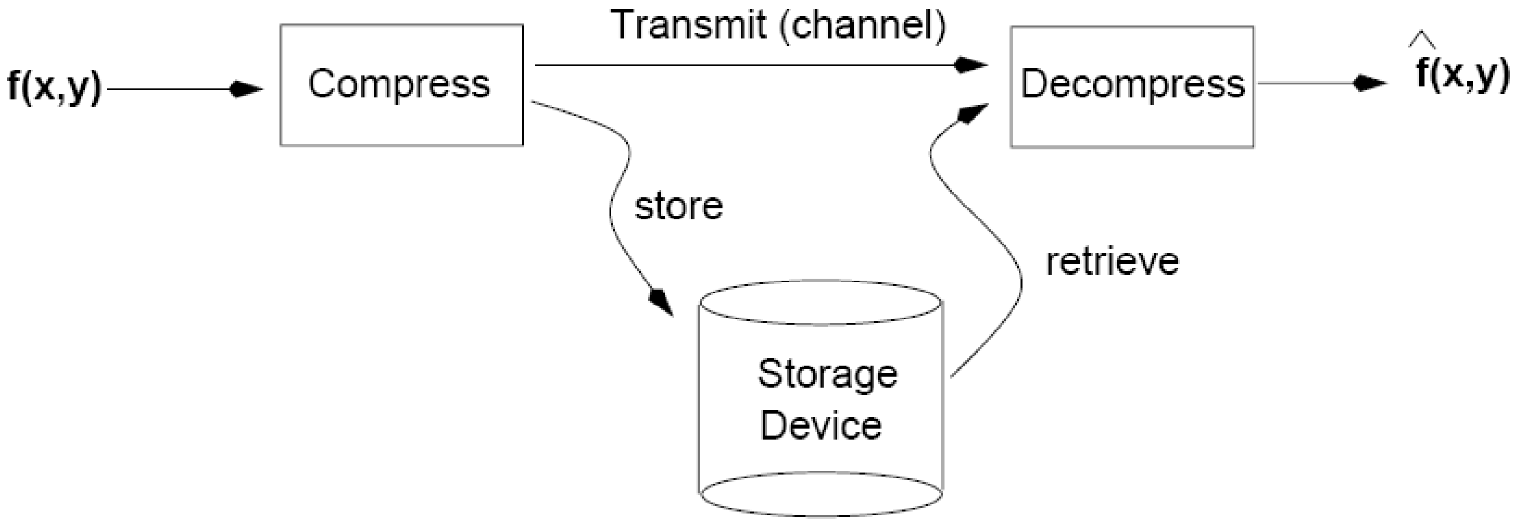
    One megapixel refers to one million pixels/inch, which are small squares of information that combine to make up an image. So, if a camera has a resolution of six megapixels, it would be able to capture images with about six million pixels of information per inch.

- Video: DVD Disc 4.7 GB
  - video 720×480×24/frame, RGB, 30 frames/sec → 31.1MB/sec
  - audio 16bits × 44.1KHz stereo → 176.4KB/sec
    - 1.5 min per DVD disc

- Send video from cellphone
  - 352×240×24, RGB, 15 frames / sec
    - ❑ 3.8 MB/sec

Objective of Image Compression

- The goal of image compression is to reduce the amount of data required to represent a digital image.



Approaches

- Lossless
  - Information preserving
  - Low compression ratios

- Lossy
  - Not information preserving
  - High compression ratios

- Trade-off: image quality V.S. compression ratio

Data ≠ Information

- Data and information **are not** synonymous terms!
- Data is the means by which information is conveyed.
- Data compression aims to **reduce** the *amount of data required to represent a given quantity of information* while **preserving** as much information as possible.

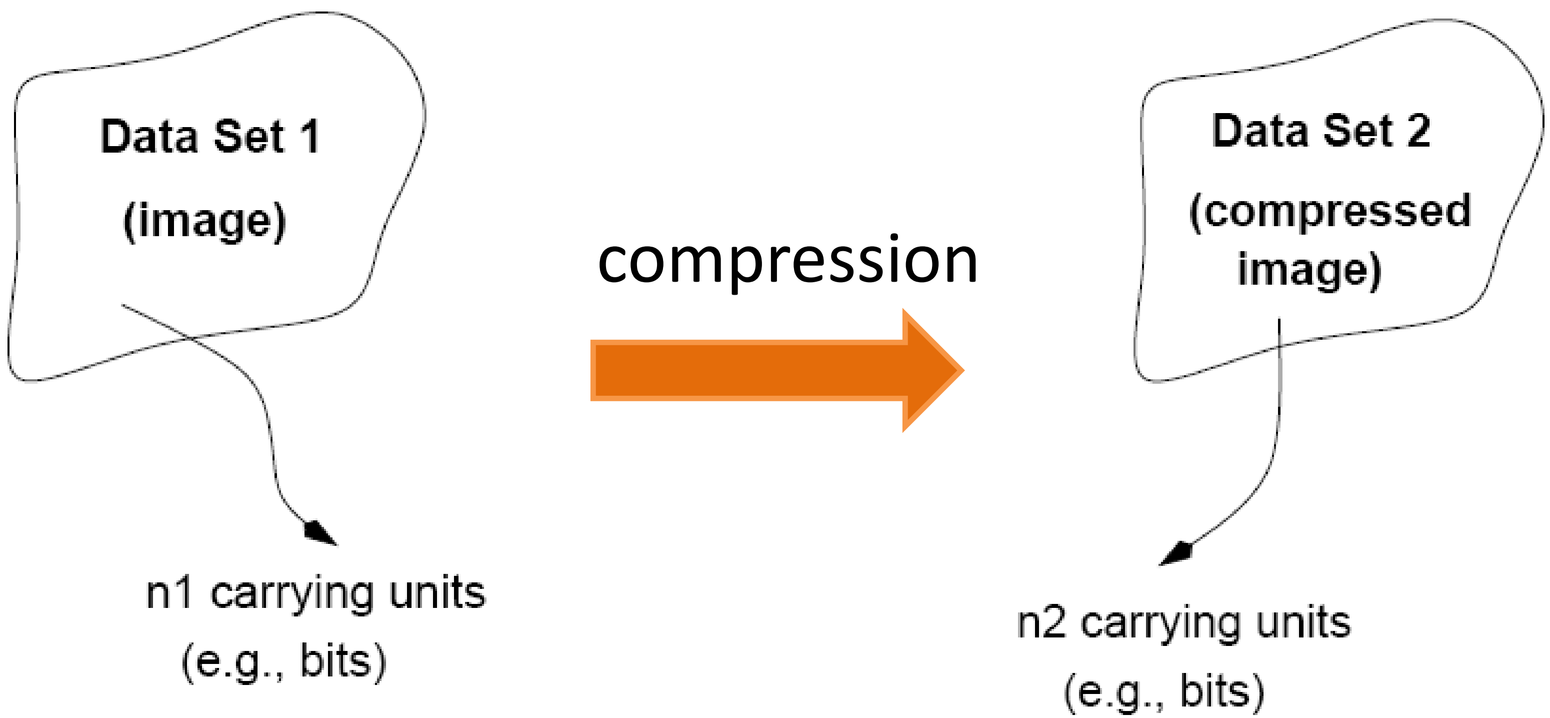
Data V.S. Information

- The **same amount** of information can be represented by various amount of data.

  Examples:

  - Your girlfriend, Helen, will meet you at *Common Man Coffee Roasters*, 22 Martin Rd, #01-00, Singapore 239058 at 5 minutes past 4:00 pm tomorrow afternoon.
  - Your girlfriend will meet you at *Common Man Coffee Roasters* at 5 minutes past 4:00 pm tomorrow afternoon.
  - Helen will meet you tomorrow at the regular café at the regular time.
  - Helen will meet you tomorrow.

Definitions: Compression Ratio



Compression ratio: $$C_R = \frac{n_1}{n_2}$$

Definitions: Data Redundancy

- **Relative data redundancy**:

  Given the compression ratio $\mathbf{C_R}$, $R_D = 1 - \frac{1}{C_R}$

- **Example:** let a dataset-1 *before the compression* has 10 bits of data, and *after the compression* it has only 1 bit of data.

$$\text{If } C_R = \frac{10}{1}, \text{ then } R_D = 1 - \frac{1}{10} = 0.9$$

(90% of the data in dataset 1 is redundant)

$$C_R = \frac{n_1}{n_2}$$

$$\text{if } n_2 = n_1, \text{ then } C_R{=}1,\ R_D{=}0$$

$$\text{if } n_2 \ll n_1, \text{ then } C_R \to \infty,\ R_D \to 1$$

Types of Data Redundancy

Types of Data Redundancy

- Compression attempts to reduce one or more of these redundancy types:
  - **Coding Redundancy**
  - **Spatial/Temporal Redundancy**
  - **Irrelevant Information**

Coding Redundancy

- Code: a list of symbols (*letters, numbers, bits* etc).
- Code word: a sequence of symbols used to represent a piece of information or an event (e.g., *gray levels/intensity levels*).
- Code word length: number/length of symbols in each code word.

Example: (binary code, symbols: 0,1, length: 3)

0: 000	4: 100
1: 001	5: 101
2: 010	6: 110
3: 011	7: 111

Coding Redundancy

- **Recall:** *Intensity as random variables*.

$$p_r(r_k) = \frac{n_k}{MN} \qquad k = 0,1,2,\ldots,L-1$$

Where,

- L is the number of intensity values.
- n_k is the number of times that the k^{th} intensity appears in the image.
- r_k represent the intensities of an $M \times N$ image.
- $P_r(r_k)$ is the probability of occurrence of pixels with intensity r_k in the image.

- The *average number of bits* required to represent *each pixel* is:

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

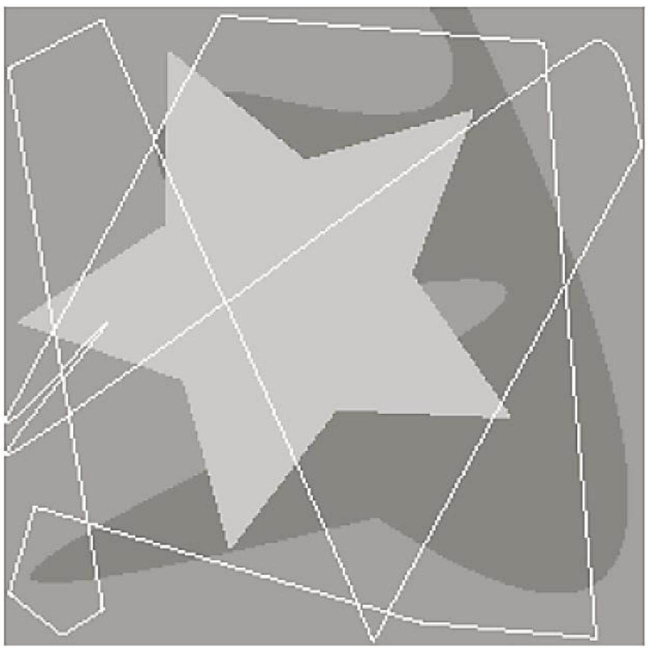
- The *total number of bits* required to represent *M x N image* is: MNL_{avg}

Where,

- $l(r_k)$ is the number of bits used to represent r_k.

Coding Redundancy - Example

- **Case 1**: $l(r_k)$ = fixed length coding



256×256

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$
$r_{87} = 87$	0.25	01010111	8
$r_{128} = 128$	0.47	10000000	8
$r_{186} = 186$	0.25	11000100	8
$r_{255} = 255$	0.03	11111111	8
r_k for $k \neq 87, 128, 186, 255$	0	—	8

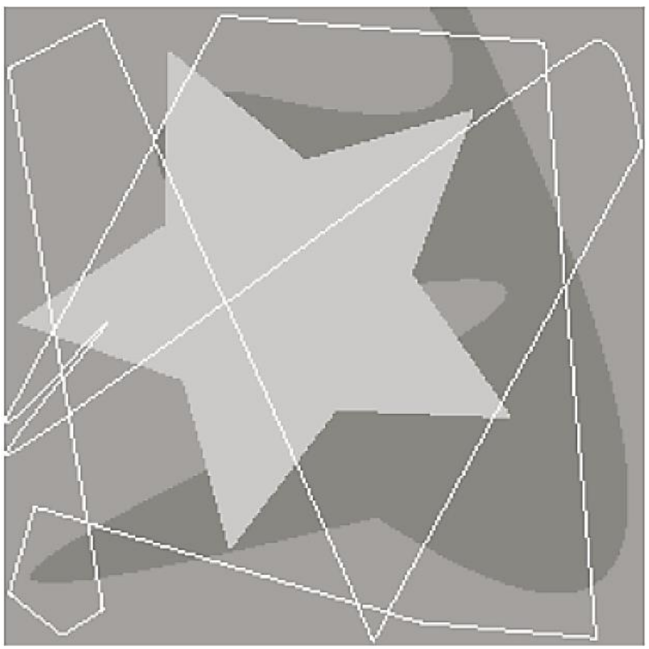
$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

$\mathbf{L_{avg}} = 0.25 \times 8 + 0.47 \times 8 + 0.25 \times 8 + 0.03 \times 8 = \mathbf{8\ bits}$

Total Number of bits: $256 \times 256 \times 8 = \mathbf{64\ KB}$

Coding Redundancy - Example

- **Case 2:** $l(r_k)$ = variable length coding



$256 \times 256 \times 8$

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

011001000
2130

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

L_{avg}=0.25×2+0.47×1+0.25×3+0.03×3=1.81 bits

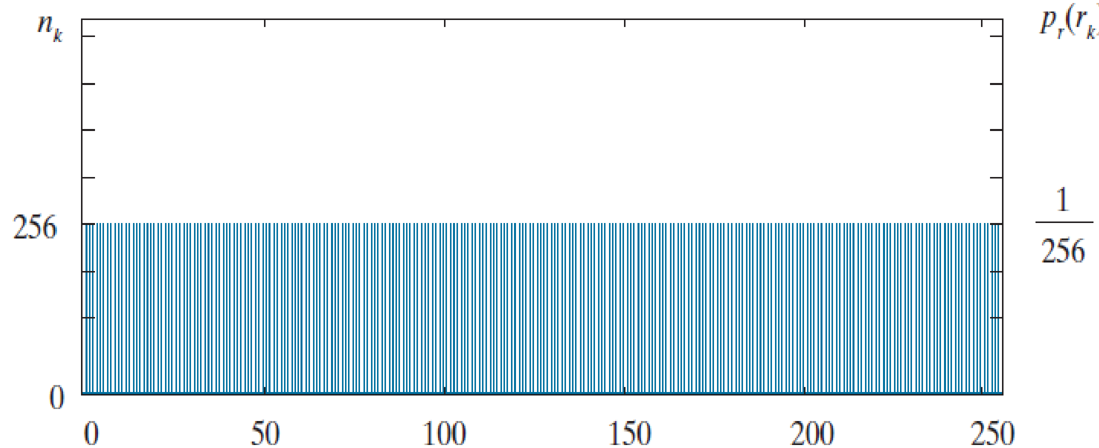
Total Number of bits: 256×256×1.81=14.4 KB

C_R=8/1.81=4.42
R_D=1-1/4.42=0.774

Spatial and Temporal Redundancy (mappings)

- **In majority of cases, any pixel value can be reasonably predicted by its neighbors (i.e., if correlated).**

256×256×8



The intensity histogram

cannot be compressed by variable-length coding

11111
3,7 OR 7,3 ??

- All 250 intensities are equally probable.
- Pixels are independent of one another in the *vertical direction*.
- Pixels are completely dependent (correlated) on one another in the *horizontal direction*.

Spatial and Temporal Redundancy

- **Run-length pairs:**
  - Specifies the *start of a new intensity* and the *number of consecutive pixels that have that intensity.*

    $=$ **(8-bit intensity value per pixel $\times$ 256 pixels) + number of pixels per line + number of lines**

    $= (8 \times 256) + 8 + 8$

    $= 2064$ **bits**

- $\mathbf{C_R}=256\times256\times8/2064$

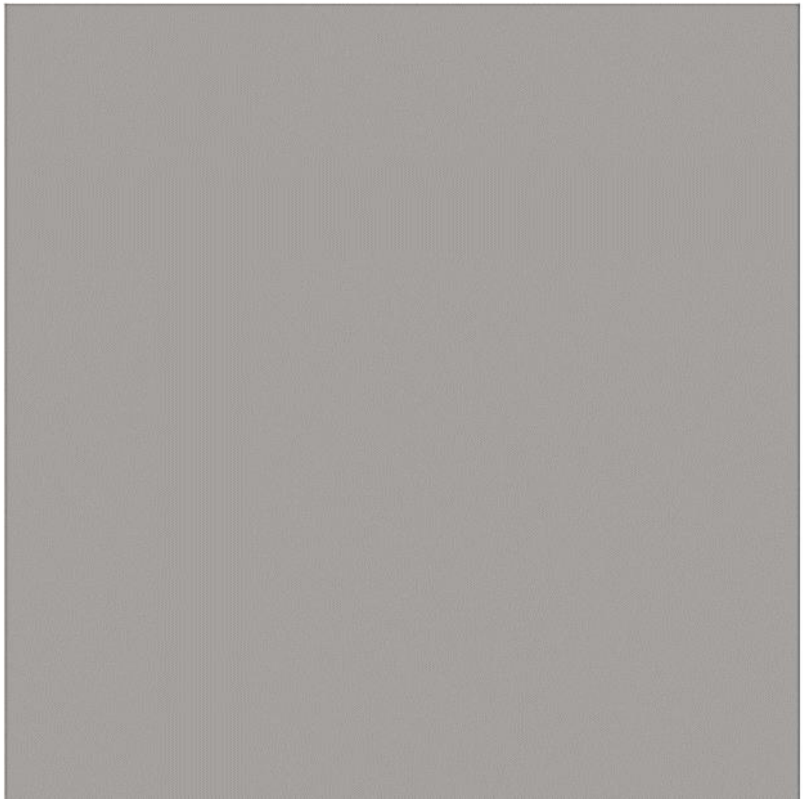
  $= 254$

- $\mathbf{R_D}=1\text{-}1/128=0.996$

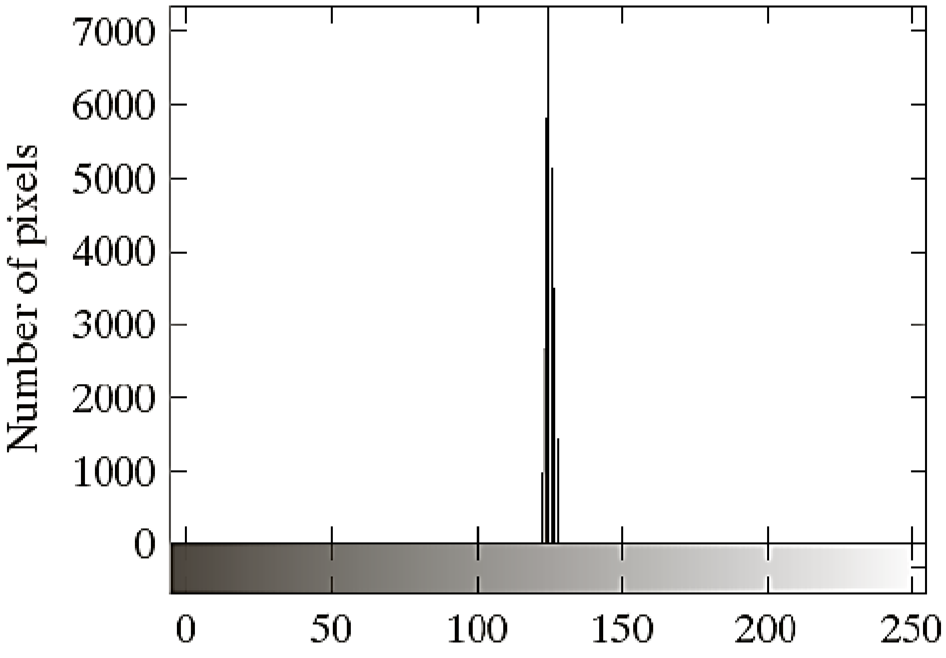
Irrelevant Information (quantization)

- **The human eye does not respond with equal sensitivity to all visual information.**
- It is more sensitive to the lower frequencies than to the higher frequencies in the visual spectrum.
- **Idea:** discard data that is perceptually insignificant!

Irrelevant Information



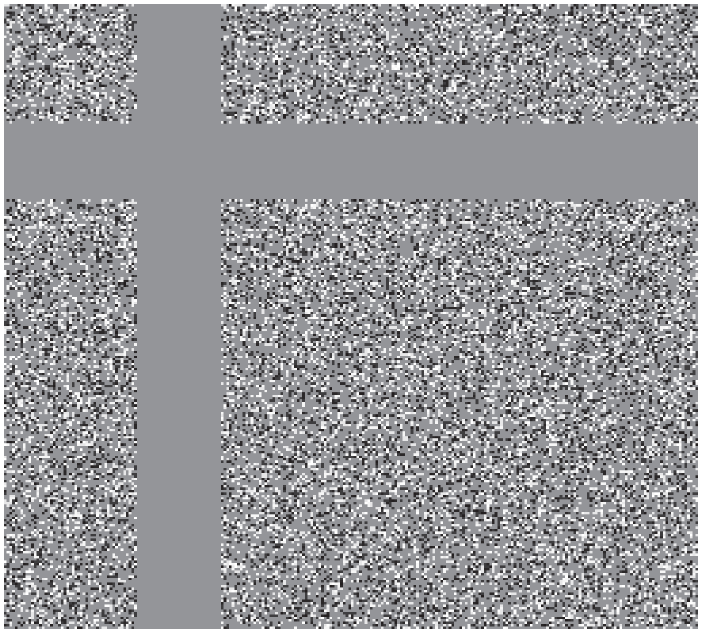
256×256×8



Intensities **125** through **131** are present, but the human visual system averages these intensities, perceives only the average value.

Irrelevant Information

Original image



Histogram equalized version

The histogram equalized version of the original image makes the intensity changes visible *and* reveals two previously undetected regions of constant intensity—one oriented vertically, and the other horizontally.

Irrelevant Information

- This image can be represented by its *average intensity* alone
  - a single **8-bit** value.
  - The original **256 * 256 * 8** bit intensity array is represented by:

(#Rows, #Cols, Average intensity)

- $C_R = (256 \times 256 \times 8)/(8+8+8) = \mathbf{21845}$

- $R_D = 1 - 1/21845 = 0.99995422$

Measuring Image Information

Measuring Image Information

- What is the minimum amount of data that is actually needed to describe an image completely, without losing information?

- We assume that information generation is a probabilistic process.

- Idea: **associate information with probability !**

- A random event E with probability $P(E)$ contains $I(E)$ units of information.

$$I(E) = \log \frac{1}{P(E)} = -\log P(E)$$

How much information does a pixel contain?

- Suppose that the *gray level* values are generated by a random variable $\mathbf{r_k}$.
- The *amount of information in pixel* is given by :

$$I(r_k)=-\log\left(P_r(r_k)\right)$$

How much information does an image contain?

- Average information content of an image is given by:

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

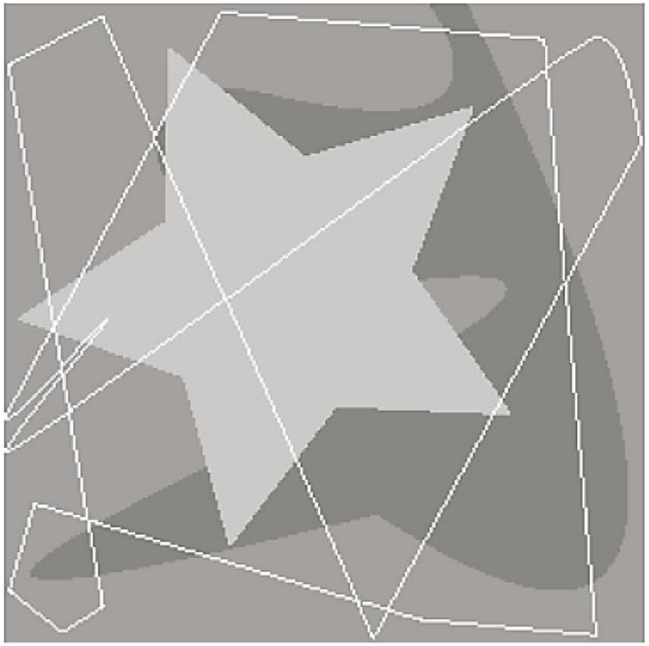
- **Since $I(r_k)=-\log (P_r(r_k))$**, we have:

$$H = -\sum_{k=0}^{L-1} P_r(r_k) log(P_r(r_k))$$ units/pixel

- **H** is called the *Entropy* of the image.

It is not possible to code the *intensity values* of an image with fewer than ***H*** bits/pixel.

Entropy - Example



r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	–	8	–	0

$$H = -\sum_{k=0}^{L-1} P_r(r_k) log(P_r(r_k))$$

$$\begin{aligned} H &= -[0.25\log_2 0.25 + 0.47\log_2 0.47 + 0.25\log_2 0.25 + 0.03\log_2 0.03] \\ &= -[0.25(-2) + 0.47(-1.09) + 0.25(-2) + 0.03(-5.06)] \\ &\approx 1.6614 \text{ bits/pixel} \end{aligned}$$

It is not possible to code the *intensity values* of this image with fewer than ***1.6614*** bits/pixel.

Redundancy Revisited

$$R = L_{avg} - H$$

Where,

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$ and $$H = -\sum_{k=0}^{L-1} P_r(r_k) log(P_r(r_k))$$

- **Note:** if $\mathbf{L_{avg} = H}$, then $\mathrm{R=0}$ (no redundancy)

Fidelity Criteria

Fidelity Criteria

- Fidelity criteria provides a means of *quantifying the nature/amount of the loss.*

- The **error** (ex:- root-mean-squared error) between *original image* $\boldsymbol{f(x,y)}$ and the compressed-decompressed *approximation* of it $\boldsymbol{\hat{f}(x,y)}$ is given as:

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

- How close is $\boldsymbol{f(x,y)}$ to $\boldsymbol{\hat{f}(x,y)}$?

- Criteria
  - **Subjective**: based on human observers
  - **Objective**: based on mathematical model

Subjective Fidelity Criteria

Value	Rating	Description
1	Excellent	An image of extremely high quality, as good as you could desire.
2	Fine	An image of high quality, providing enjoyable viewing. Interference is not objectionable.
3	Passable	An image of acceptable quality. Interference is not objectionable.
4	Marginal	An image of poor quality; you wish you could improve it. Interference is somewhat objectionable.
5	Inferior	A very poor image, but you could watch it. Objectionable interference is definitely present.
6	Unusable	An image so bad that you could not watch it.

Objective Fidelity Criteria

- The ***total error*** between the two images $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$ and $\hat{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{y})$ is:

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

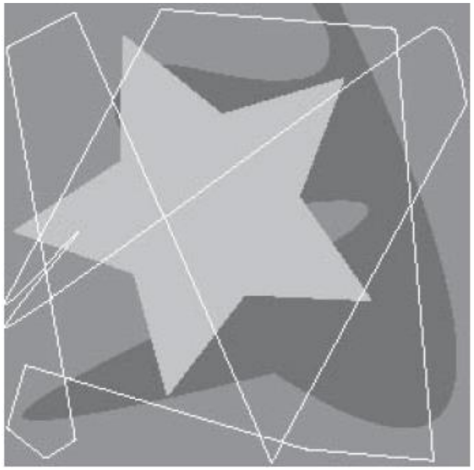
- The ***root-mean-squared-error*** **(e_{rms})** between $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$ and $\hat{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{y})$ is:

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

- The ***mean-squared signal-to-noise ratio*** **(SNR_{rms})** of the output image is:

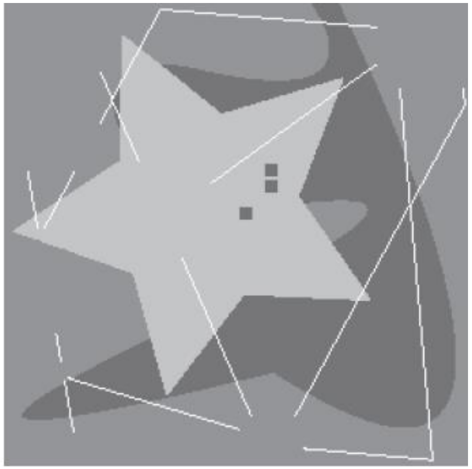
$$\text{SNR}_{\text{ms}}=\frac{\displaystyle\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\displaystyle\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y)-f(x,y)\right]^2}$$

Subjective Vs. Objective Fidelity Criteria



$e_{rms} = 5.17$

$e_{rms} = 15.67$

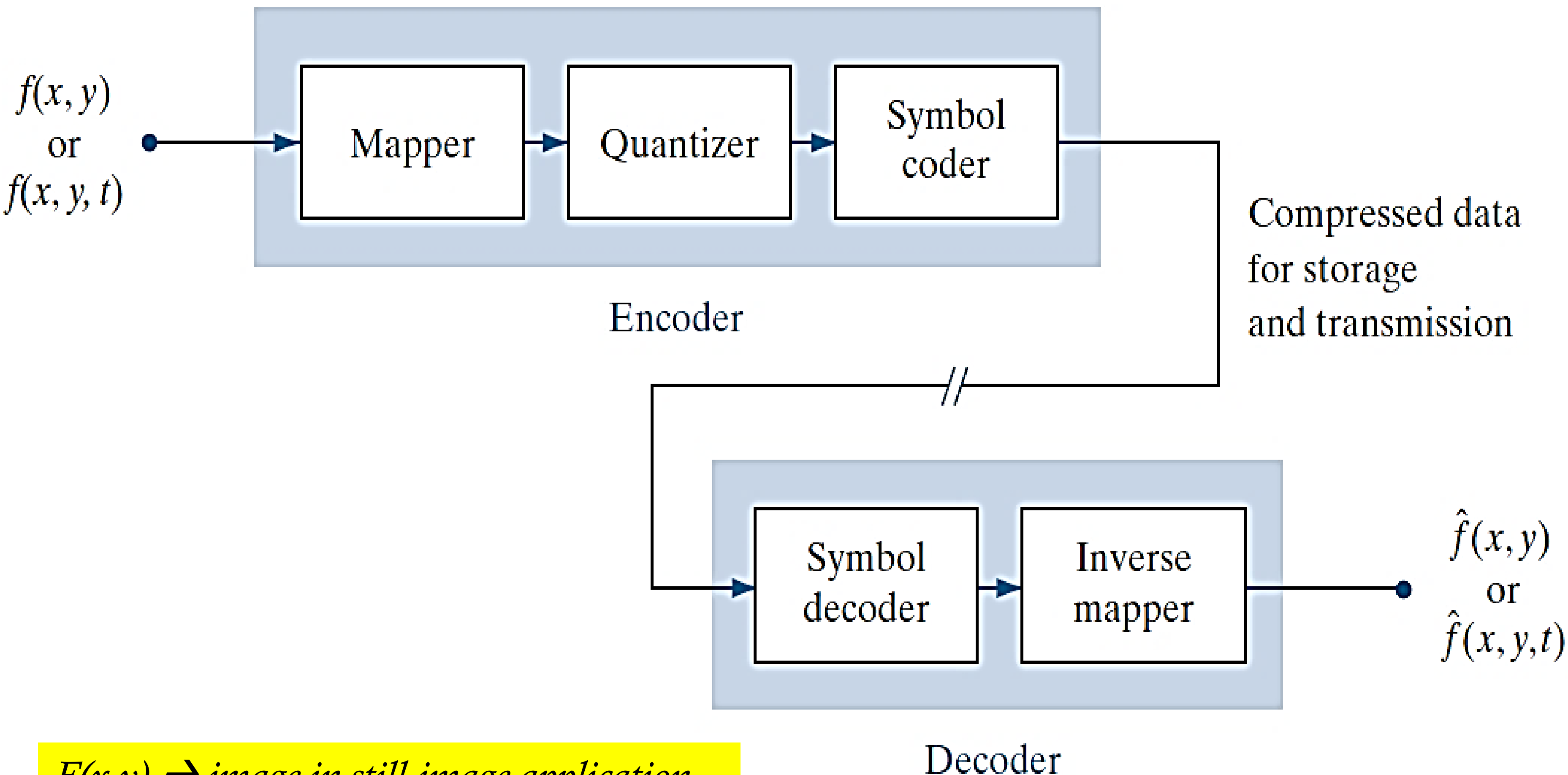


$e_{rms} = 14.17$

e_{rms} should be **less** and **SNR**$_{rms}$ should be **high**

General Image Compression Model

General Image Compression Model



F(x,y) → image in still-image application

F(x,y,t) → image in video application where 't' specifies time

General Image Compression Model

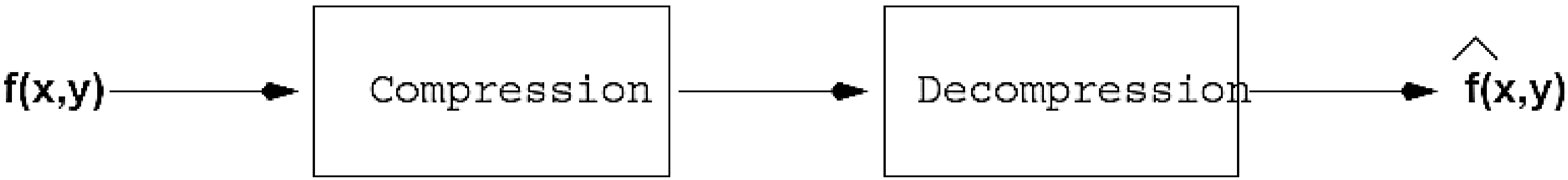
- **Mapper**
  - Transforms input into a (usually nonvisual) format designed to reduce spatial and temporal redundancy.
  - This operation generally is reversible, and may or may not directly reduce the amount of data required to represent the image.

- **Quantizer**
  - Reduces the accuracy of the mapper’s output in accordance with a pre-established fidelity criterion.
  - The goal is to keep irrelevant information out of the compressed representation.
  - This operation is irreversible.
  - It must be omitted when error-free compression is desired.

- **Symbol coder**
  - Generates a fixed-length or variable-length code to represent the quantizer output, and maps the output in accordance with the code.
  - This operation is reversible.

Lossless Compression

Lossless Compression



$$e(x,y) = \hat{f}(x,y) - f(x,y) = 0$$

Binary Codes

Binary code: Maps each character of an alphabet Σ to a binary string.

Example: **Σ = a-z** and various punctuation (size **32** overall, say).

- Use the 32, **5-bit binary strings** to encode this **Σ**. (a fixed-length coding)

  i.e. 00000 to 11111 , so we need $\mathbf{32 \times 5 = 160}$ **bits**

- **Can we do better**? **Yes**, if some characters of **Σ** are much more frequent than others, use a variable-length code.

Ambiguity in Variable-length Encoding

Suppose **Σ ={A,B,C,D}**.

- Suppose, the variable-length encoding used is {0,01,10,1}.
- What does **001** represent?

A) AB

B) CD

C) AAD

D) None

Ambiguity in Variable-length Encoding

Suppose **Σ ={A,B,C,D}**.

- Suppose, the variable-length encoding used is {0,01,10,1}.
- What does **001** represent?

A) AB $\longrightarrow$ Leads to 001

B) CD

C) AAD $\longrightarrow$ Also leads to 001

D) None

Prefix-Free Codes

- **Problem:** With variable-length codes, it is not clear where one character ends & the next one begins.

  Example: **001** string in the previous slide

- **Solution:** *Prefix-free codes* - make sure that for every pair (i, j)∈Σ, neither of the encoding f(i), f(j) is a prefix of the other.

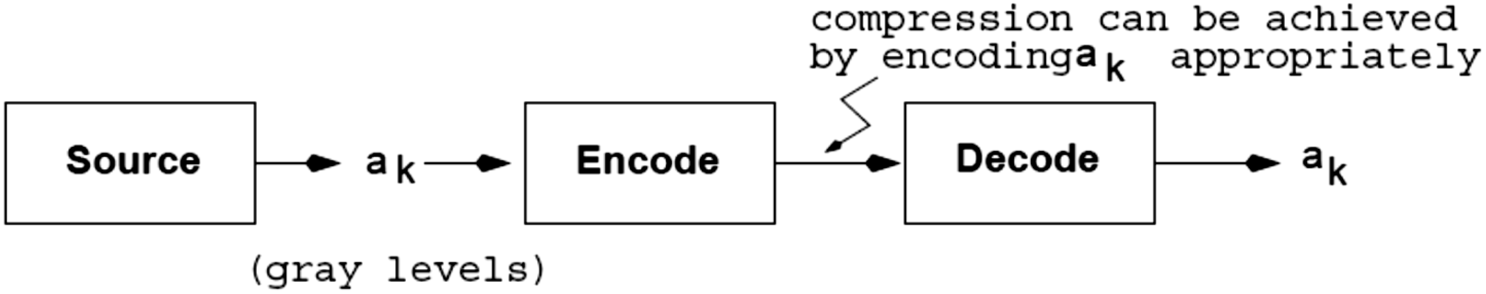
  Example: **{0,10,110,111}**

Huffman Coding
(Reducing Coding Redundancy)

- A variable-length coding technique.

- Symbols are encoded one at a time!
  - There is a one-to-one correspondence between source symbols and code words

- Optimal code (i.e., it yields the smallest possible number of code symbols per source symbol).



Huffman Coding

- <u>**Create series of source reductions**</u>

  Step-1: Order the probabilities of the symbols under consideration.

  Step-2: Combine the lowest two probabilities symbols.

  Step-3: Repeat *Step-1* and *Step-2* until only two probabilities remain.

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				

Huffman Coding

- **Code each reduced source**

Step-1: Starting with the *highest reduced source* and working back to the *original source*, assign the unique codes to the symbols in the increasing order of code word length.

Step-2: Repeat *Step-1* for each reduced source until the original source is reached.

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								

Huffman Coding

Symbols (like intensity levels)	Probabilities (sorted)	Source Reduction (do till two values are left) (Maintain in sorted order here as well)			
		1	2	3	4
a2	0.4 1	0.4	0.4	0.4	0.6 0
a6	0.3 00	0.3	0.3	0.3 00	0.4 1
a1	0.1 011	0.1	0.2 010	0.3 01	
a4	0.1 0100	0.1 0100	0.1 011		
a3	0.06 01010	0.1 0101			
a5	0.04 01011				

Huffman Coding

- The **average length** of Huffman code for the previous example is:

$$\begin{aligned} L_{\text{avg}} &= (0.4)(1)+(0.3)(2)+(0.1)(3)+(0.1)(4)+(0.06)(5)+(0.04)(5) \\ &= 2.2 \text{ bits/pixel} \end{aligned}$$

- The Huffman coding table *must be known* to the decoder to retrieve (decompress) the original data.

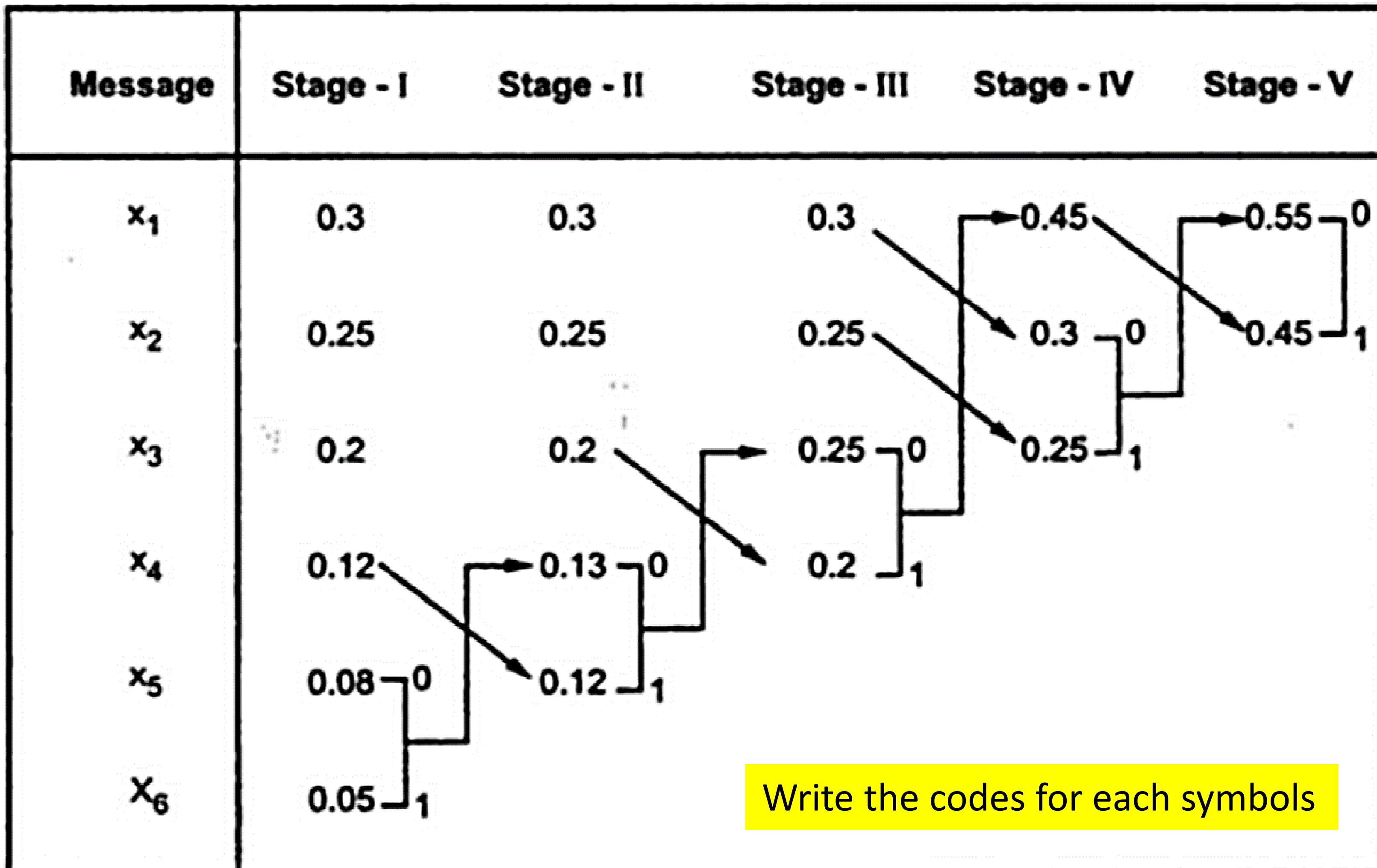
- Any string of Huffman encoded symbols can be decoded by examining the individual symbols of the string in a *left-to-right manner*.

Example: encoded string is 010100111100

decoded string is *a*3 *a*1 *a*2 *a*2 *a*6

Symbol	Code
a_2	1
a_6	00
a_1	011
a_4	0100
a_3	01010
a_5	01011

Huffman Coding



Huffman Coding/Decoding

- **Coding/decoding** can be implemented using a predefined look-up table.

- Decoding can be done unambiguously.

Next Lecture

- The image degradation/restoration model

- Noise models
  - Important noise probability density functions
  - Periodic noise
  - Estimating noise parameters

- Restoration using spatial filters
  - Mean filters
  - Order-static filters
  - Adaptive filters