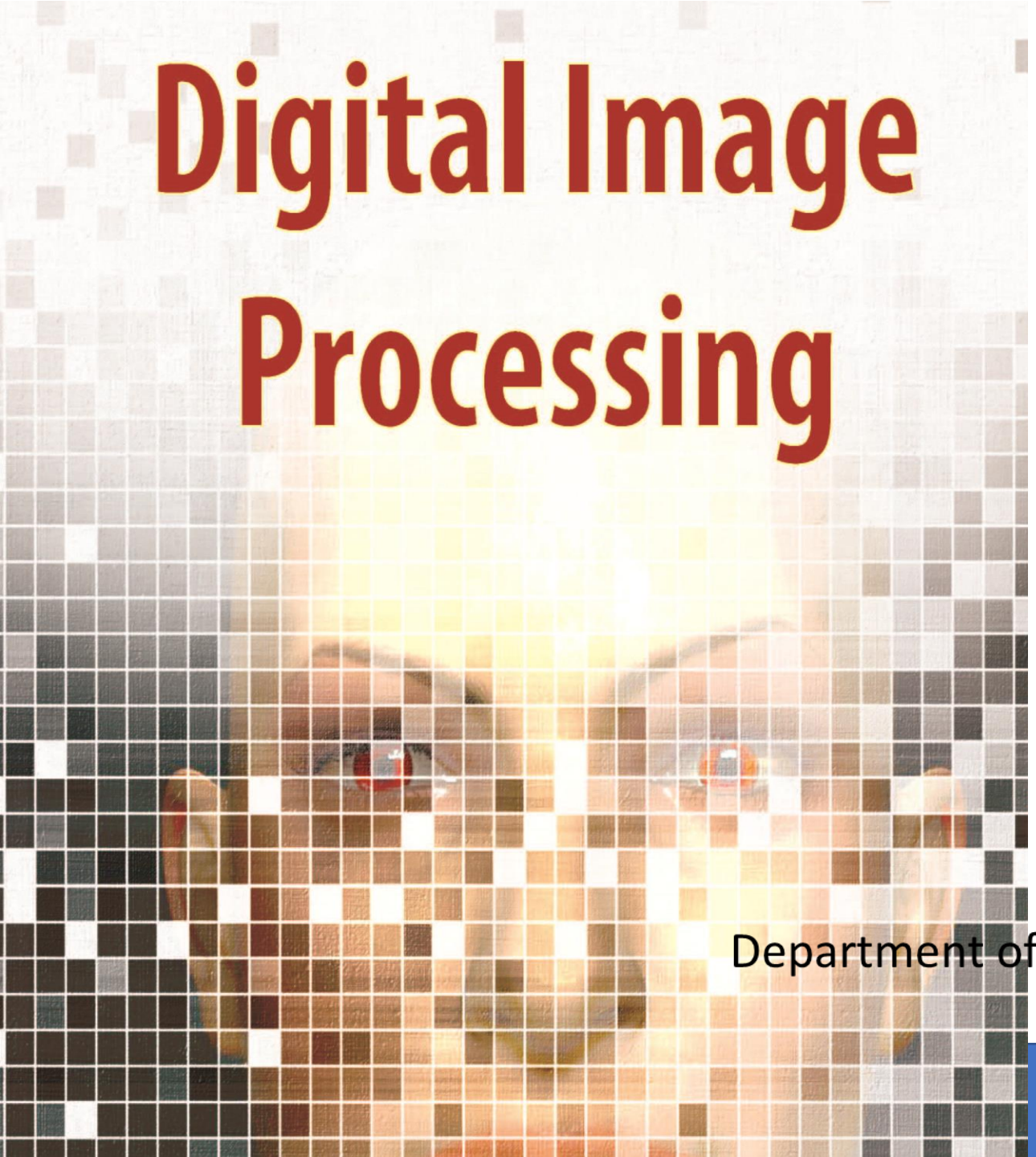


Image Compression

Prof. Sheli Sinha Chaudhuri
Department of Electronics and Telecommunication Engineering
Jadavpur University

Chapter 8
Image Compression

Contents

- Fundamentals

- Basic Compression Methods

Data Compression

The process of reducing the amount of data required to represent a given quantity of information is called data compression

Data is the means to convey information.

Let a and b denote the number of bits in two representations of the same data.
Then the relative data redundancy R=1-1/C
where C=a/b the compression ratio.

Image Compression

Compression can take place only if we have redundant information

Image compression

- Coding redundancy
  - Fewer bits to represent frequent symbols
  - Huffman coding : Lossless
- Interpixel redundancy
  - Neighboring pixels have similar values
  - Predictive coding : Lossless
- Psychovisual redundancy
  - Quantization : Lossy
  - Remove information that human visual system cannot perceive
  - Removal of high frequency data : Lossy

Examples of Redundancy

Digital Image Processing, 3rd ed.

Gonzalez & Woods

www.ImageProcessingPlace.com

Chapter 8

Image Compression

Coding Redundancy

Spatial Redundancy

Irrelevant Information

Image Compression

Lossy Image Compression

- Information is lost during compression
- Consumer TV signals
- Consumer images : Web, Digital cameras

Lossless Image Compression

- Information is preserved during compression
- Medical imaging
- Compression ratio will be different for the two cases

Image Compression

Coding Redundancy

Different coding methods yield different amount of data needed to represent the same information.

Example of Coding Redundancy : Variable Length Coding vs. Fixed Length Coding

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

L_{avg} 3 bits/symbol

L_{avg} 2.7 bits/symbol

Image Compression

Let us assume, once again, that a discrete random variable r_k in the interval $[0, 1]$ represents the gray levels of an image and that each r_k occurs with probability $p_r(r_k)$. As in Chapter 3,

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, \ldots, L-1 \qquad (8.1\text{-}3)$$

where L is the number of gray levels, n_k is the number of times that the kth gray level appears in the image, and n is the total number of pixels in the image. If the number of bits used to represent each value of r_k is $l(r_k)$, then the average number of bits required to represent each pixel is

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k). \qquad (8.1\text{-}4)$$

That is, the average length of the code words assigned to the various gray-level values is found by summing the product of the number of bits used to represent each gray level and the probability that the gray level occurs. Thus the total number of bits required to code an $M \times N$ image is MNL_{avg}.

Image Compression

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

TABLE 8.1
Example of variable-length coding.

$$L_{avg} = \sum_{k=0}^{7} l_2(r_k)p_r(r_k)$$
$$= 2(0.19) + 2(0.25) + 2(0.21) + 3(0.16) + 4(0.08) + 5(0.06) + 6(0.03) + 6(0.02)$$
$$= 2.7 \text{ bits.}$$

From Eq. (8.1-2), the resulting compression ratio C_R is 3/2.7 or 1.11. Thus approximately 10% of the data resulting from the use of code 1 is redundant. The exact level of redundancy can be determined from Eq. (8.1-1):

$$R_D = 1 - \frac{1}{1.11} = 0.099.$$

Image Compression

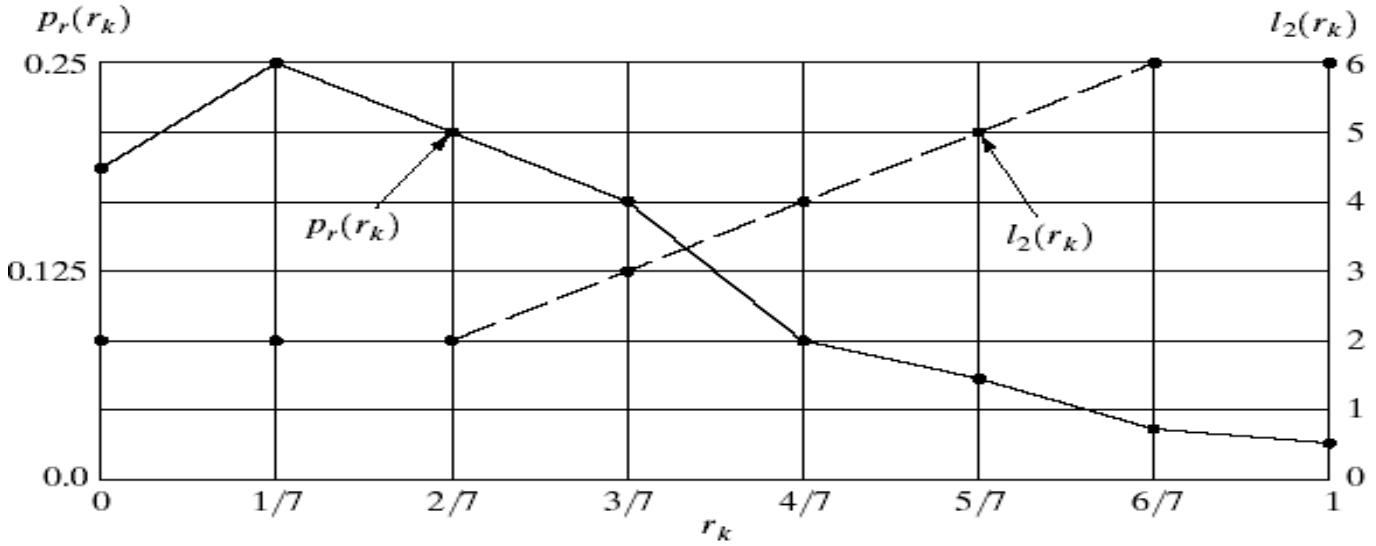


FIGURE 8.1 Graphic representation of the fundamental basis of data compression through variable-length coding.

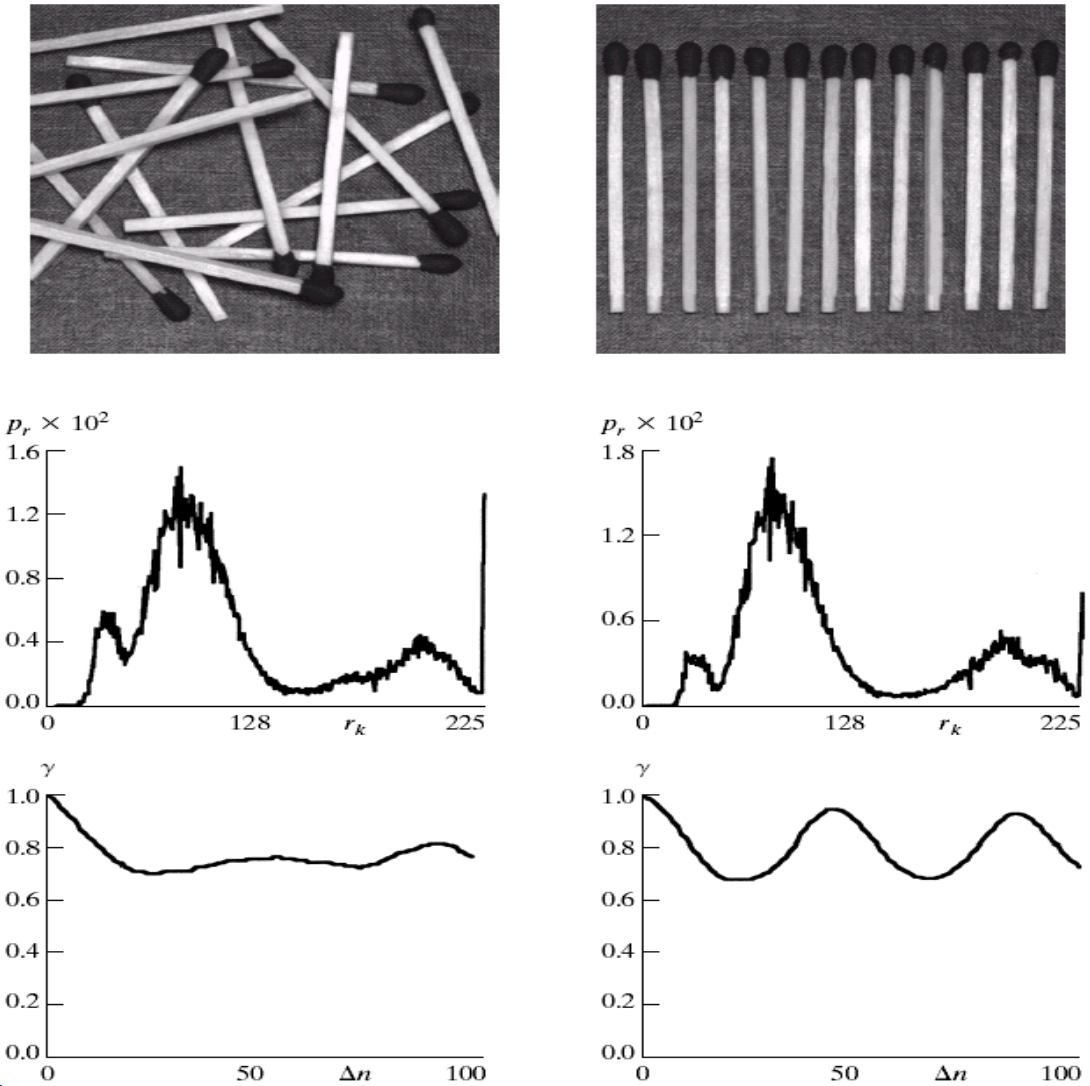
Concept: assign the longest code word to the symbol with the least probability of occurrence.

Image Compression

Spatial/Temopral Redundancy

- Internal Correlation between the pixel result from
  - Respective Autocorrelation
  - Structural Relationship
  - Geometric Relation ship
- The value of a pixel can be reasonably predicted from the values of its neighbors.
- To reduce the inter-pixel redundancies in an image the 2D array is transformed (*mapped*) into more efficient format (Frequency Domain etc.)

Image Compression



a b
c d
e f

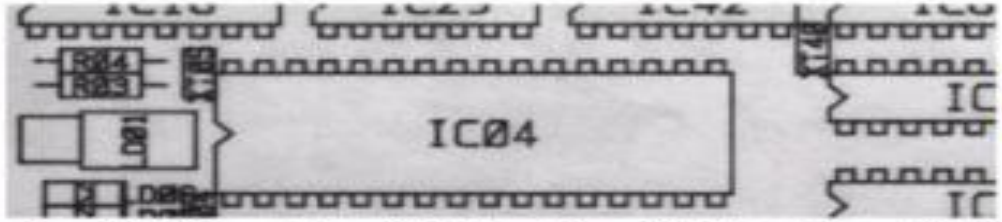
FIGURE 8.2 Two images and their gray-level histograms and normalized autocorrelation coefficients along one line.

Interpixel redundancy: Parts of an image are highly correlated.

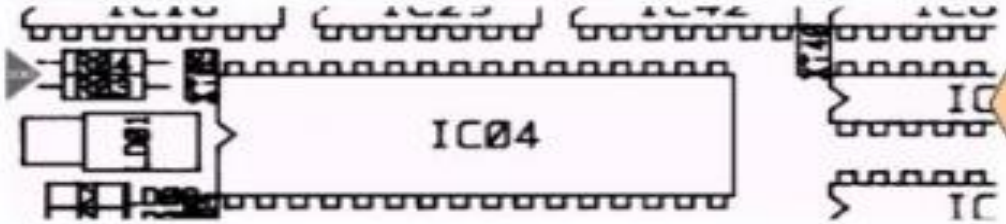
In other words,we can predict a given pixel from its neighbor.

Image Compression

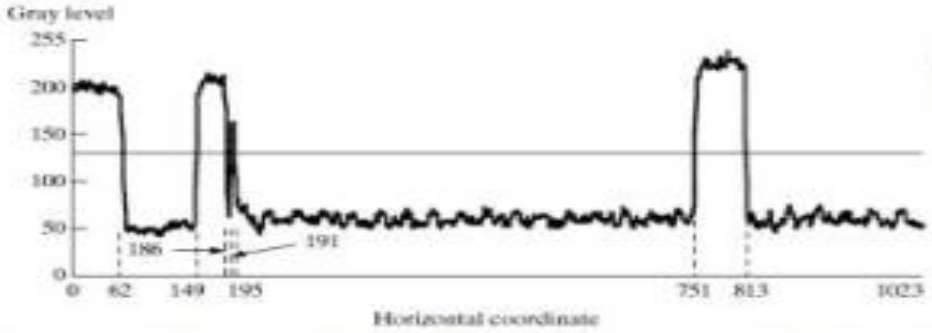
Run Length Coding



The gray scale image of size 343x1024 pixels



Binary image
= 343x1024x1 = 351232 bits



Line No. 100

Run length coding

Line 100: (1,63) (0,87) (1,37) (0,5) (1,4) (0,556) (1,62) (0,210)

Total 12166 runs, each run use 11 bits → Total = 133826 Bits

Image Compression

Irrelevant information and Psycho-Visual Redundancy

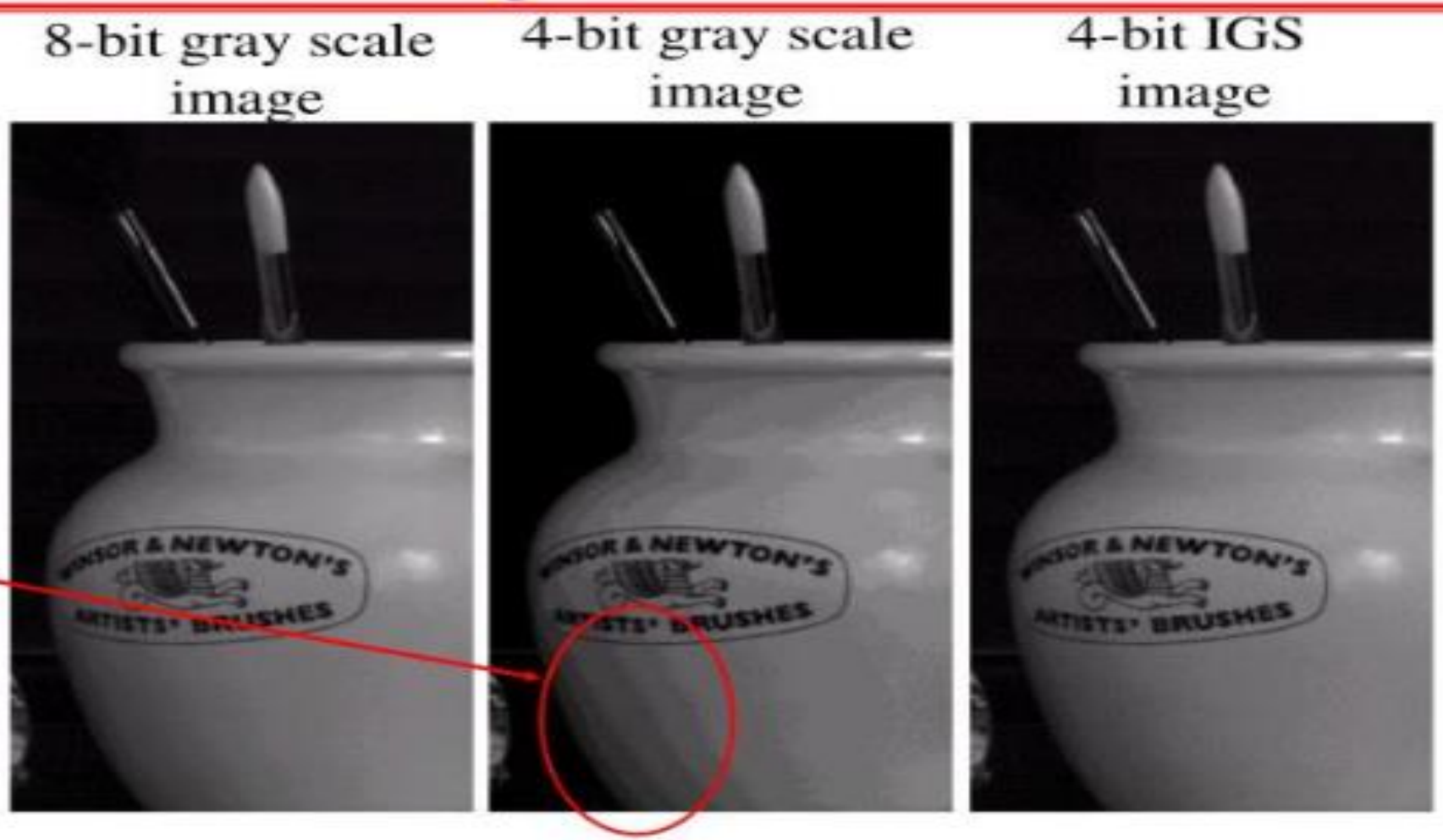
- The brightness of a region depend on other factors that the light reflection
- The perceived intensity of the eye is limited an non linear
- Certain information has less relative importance that other information in normal visual processing
- In general, observer searches for distinguishing features such as edges and textural regions.

Image Compression

Irrelevant information and Psycho-Visual Redundancy

- The brightness of a region depends on many factors like light reflection
- The perceived intensity of the eye is limited and non-linear
- Certain information has less relative information in normal visual processing
- In general, observer searches for distinguishing features such as edges and textural region

Psychovisual Redundancy



The eye does not response with equal sensitivity to all visual information.

Improved Gray Scale Quantization

Pixel	Gray level	Sum	IGS Code
i-1	N/A	0000 0000	N/A
i	0110 1100	0110 1100	0110
i+1	1000 1011	1001 0111	1001
i+2	1000 0111	1000 1110	1000
i+3	1111 0100	1111 0100	1111

Algorithm

1. Add the least significant 4 bits of the previous value of Sum to the 8-bit current pixel. If the most significant 4 bit of the pixel is 1111 then add 0000 instead. Keep the result in Sum

2. Keep only the most significant 4 bits of Sum for IGS code.

Image Compression

Fidelity Criteria

- Objective Fidelity Criteria
  - The information loss can be expressed as a function of the encoded and decoded images.
  - For image I (x,y) and its decoded approximation I'(x,y)
  - For any value of x and y, the error e(x,y) could be defined as

$$e(x,y) = I'(x,y) - I(x,y)$$

  - For the entire Image

$$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} I'(x,y) - I(x,y)$$

Image Compression

Fidelity Criteria

- The mean-square-error, e_{rms} is

$$e_{rms} = \sqrt{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[I'(x,y) - I(x,y)]^2}$$

The mean-square-error signal-to-noise ratio SNR_{ms} is

$$SNR_{ms} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} I'(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[I'(x,y) - I(x,y)]^2}$$

Image Compression

Fidelity Criteria: how good is the compression algorithm

-Objective Fidelity Criterion

- RMSE, PSNR

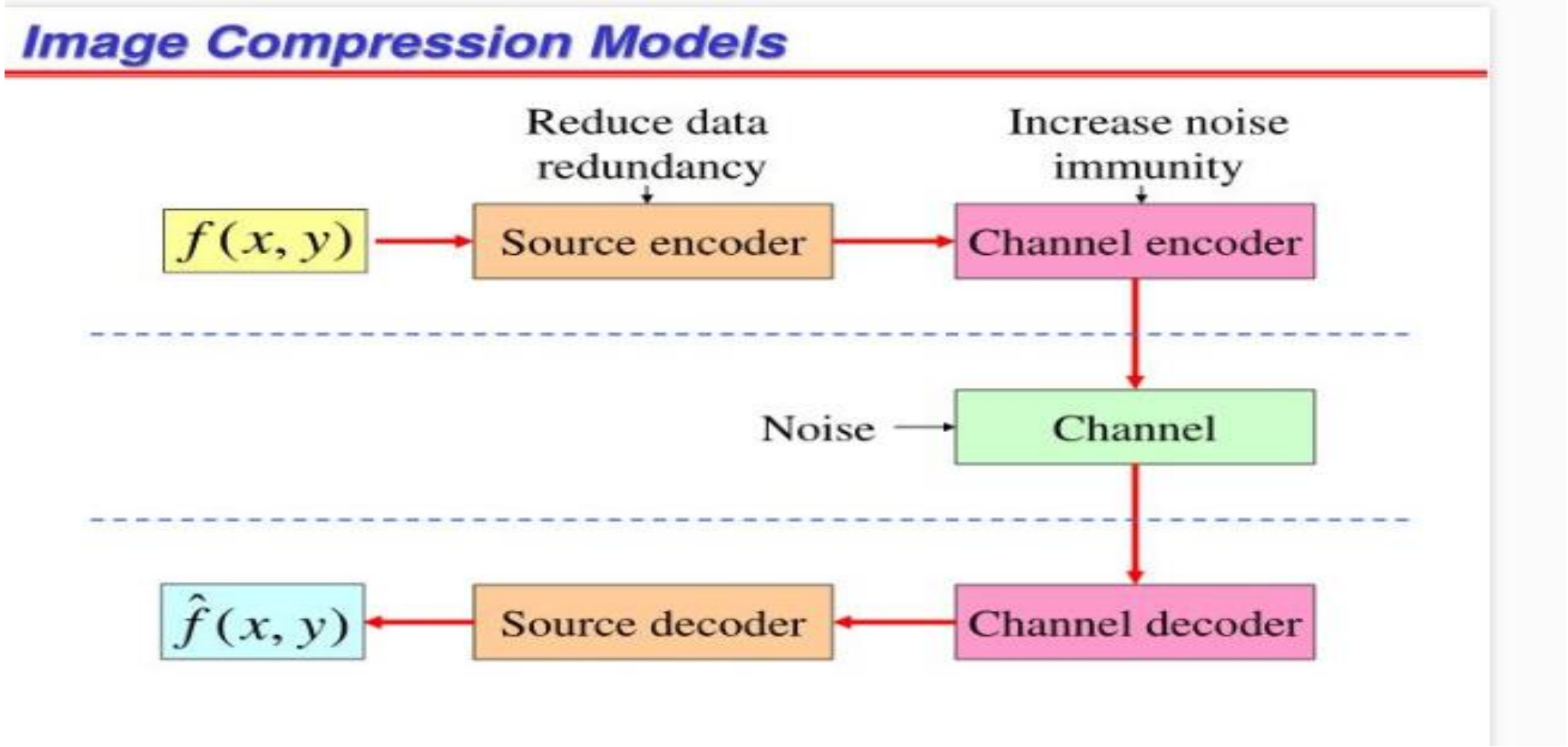
-Subjective Fidelity Criterion:

-Human Rating

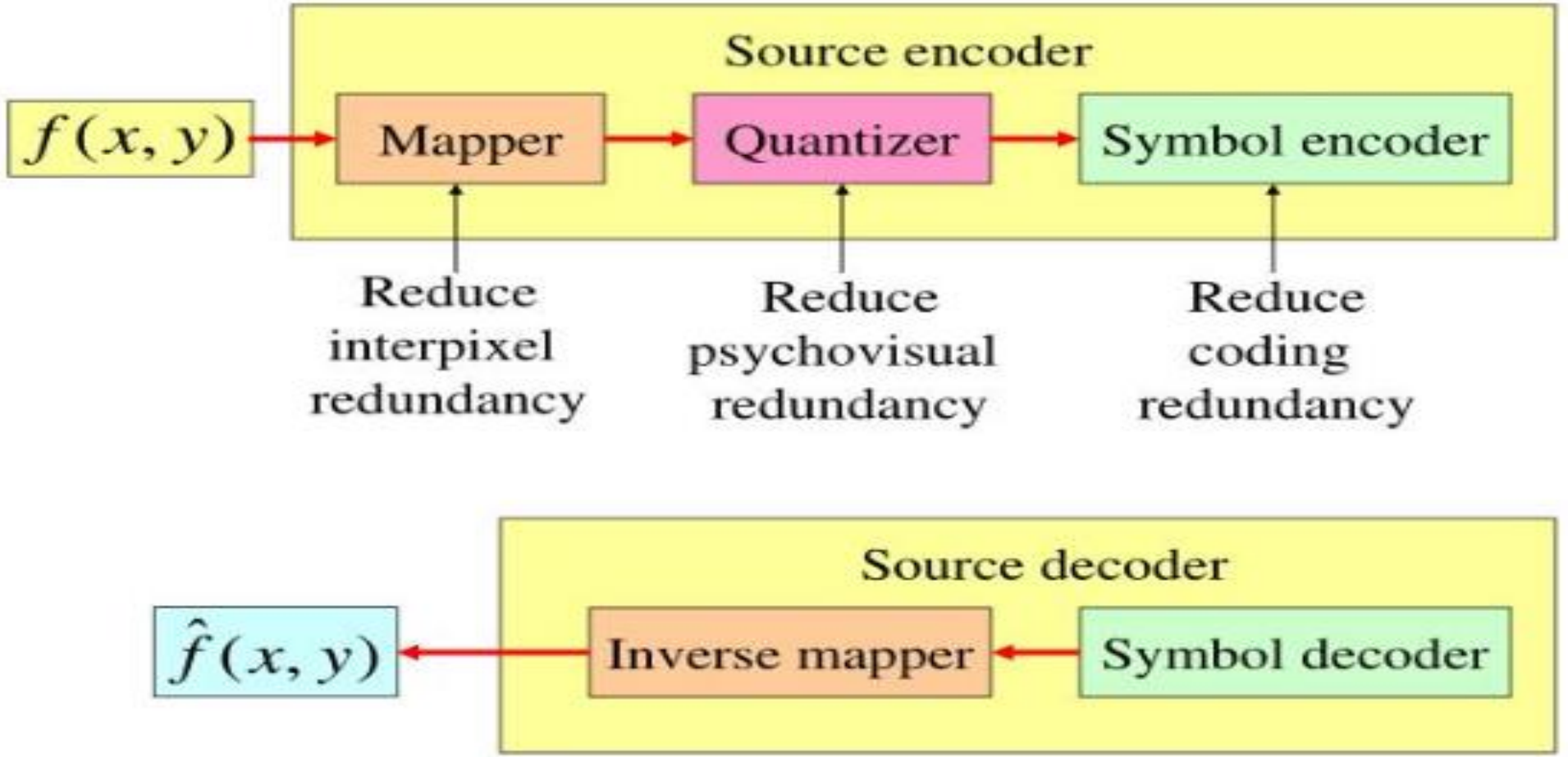
TABLE 8.3
Rating scale of the Television Allocations Study Organization. (Frendendall and Behrend.)

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.

Image Compression



Source Encoder and Decoder Models



Information Theory

Measuring information

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

Entropy or Uncertainty: Average information per symbol

$$H = -\sum_{j} P(a_j)\log(P(a_j))$$

Simple Information System

FIGURE 8.7 A simple information system.

Ensemble $(A, \mathbf{z})$
$A = \{a_j\}$
$\mathbf{z} = [P(a_1), P(a_2), \ldots, P(a_J)]^T$

$Q = [q_{kj}]$

Ensemble $(B, \mathbf{v})$
$B = \{b_k\}$
$\mathbf{v} = [P(b_1), P(b_2), \ldots, P(b_K)]^T$

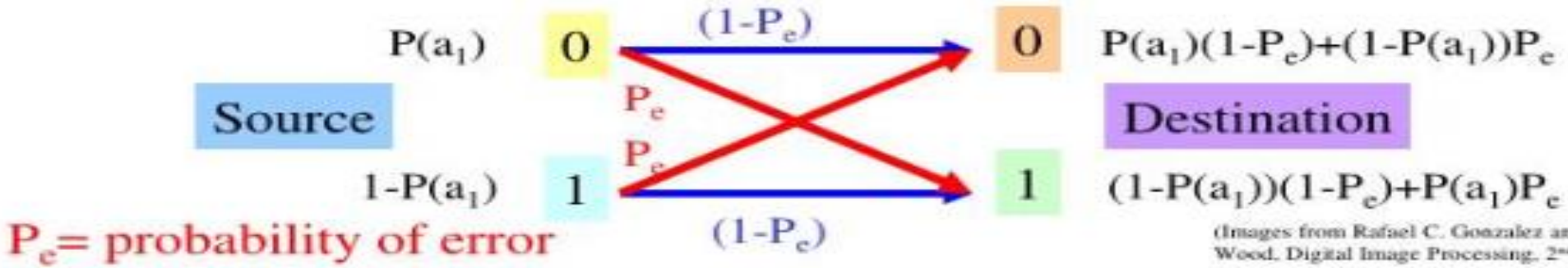
Binary Symmetric Channel

Source

$A = \{a_1, a_2\} = \{0, 1\}$
$z = [P(a_1), P(a_2)]$

Destination

$B = \{b_1, b_2\} = \{0, 1\}$
$v = [P(b_1), P(b_2)]$



P_e = probability of error

(Images from Rafael C. Gonzalez and Richard E. Wood, Digital Image Processing, 2nd Edition.

Image Compression

Binary Symmetric Channel

Source

$A = \{a_1, a_2\} = \{0, 1\}$

$\mathbf{z} = [P(a_1), P(a_2)]$

$$H(\mathbf{z}) = - P(a_1)\log_2 P(a_1) - P(a_2)\log_2 P(a_2)$$

Destination

$B = \{b_1, b_2\} = \{0, 1\}$

$\mathbf{v} = [P(b_1), P(b_2)]$

$$H(\mathbf{z}|b_1) = - P(a_1|b_1)\log_2 P(a_1|b_1) - P(a_2|b_1)\log_2 P(a_2|b_1)$$

$$H(\mathbf{z}|b_2) = - P(a_1|b_2)\log_2 P(a_1|b_2) - P(a_2|b_2)\log_2 P(a_2|b_2)$$

$$H(\mathbf{z}|\mathbf{v}) = H(\mathbf{z}|b_1) + H(\mathbf{z}|b2)$$

Mutual information

$$I(\mathbf{z},\mathbf{v}) = H(\mathbf{z}) - H(\mathbf{z}|\mathbf{v})$$

Capacity

$$C = \max_{\mathbf{z}}[I(\mathbf{z}, \mathbf{v})]$$

Binary Symmetric Channel

Let p_e = probability of error

$$\mathbf{z} = \begin{bmatrix} p(a_1) \\ 1 - p(a_1) \end{bmatrix} = \begin{bmatrix} p_{bs} \\ \bar{p}_{bs} \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} 1 - p_e & p_e \\ p_e & 1 - p_e \end{bmatrix} \begin{bmatrix} p_{bs} \\ \bar{p}_{bs} \end{bmatrix}$$

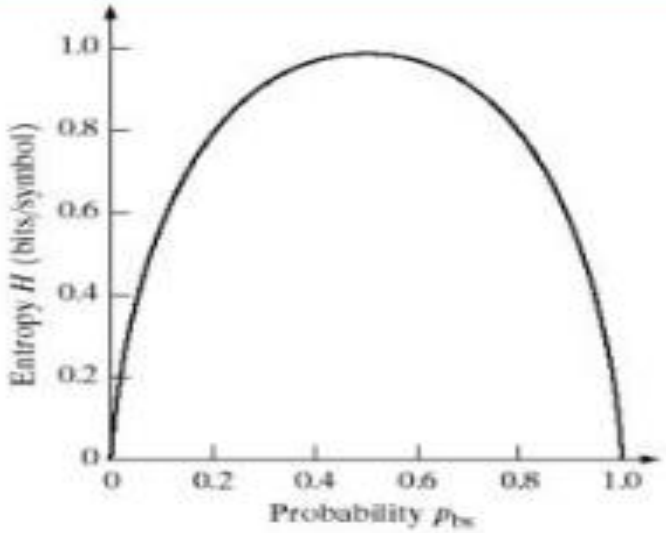
$$H(\mathbf{z}) = -p_{bs}\log_2(p_{bs}) - (1 - p_{bs})\log_2(1 - p_{bs})$$

$$H(\mathbf{z} \mid \mathbf{v}) = -p_{bs}(1 - p_e)\log_2(p_{bs}(1 - p_e)) - (1 - p_{bs})p_e \log_2((1 - p_{bs})p_e)$$
$$-(1 - p_{bs})(1 - p_e)\log_2((1 - p_{bs})(1 - p_e)) - p_{bs}p_e \log_2(p_{bs}p_e)$$

$$I(z, v) = H_{bs}(p_{bs}p_e + \bar{p}_{bs}\bar{p}_e) - H_{bs}(p_e)$$

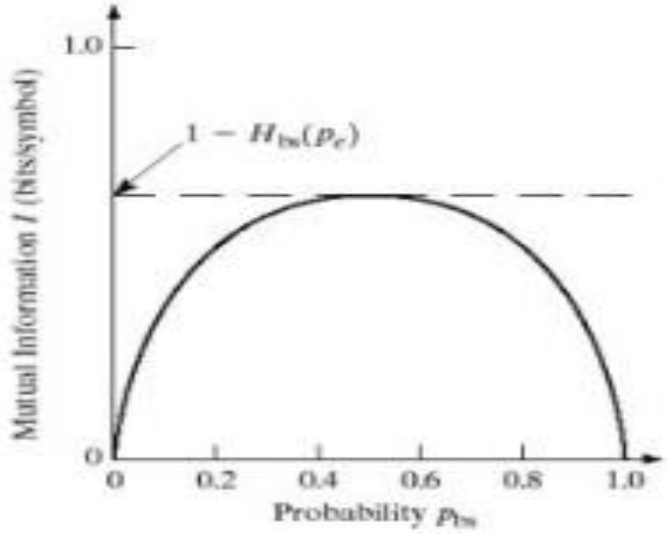
$$C = 1 - H_{bs}(p_e)$$

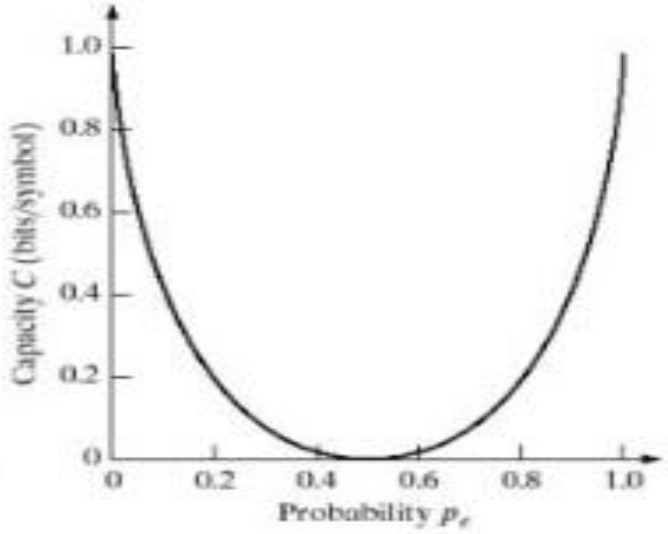
Binary Symmetric Channel



a
b c

FIGURE 8.8 Three binary information functions: (a) the binary entropy function; (b) the mutual information of a binary symmetric channel (BSC); (c) the capacity of the BSC.





(Images from Rafael C. Gonzalez and Richard E. Wood, Digital Image Processing, 2nd Edition.

Communication System Model

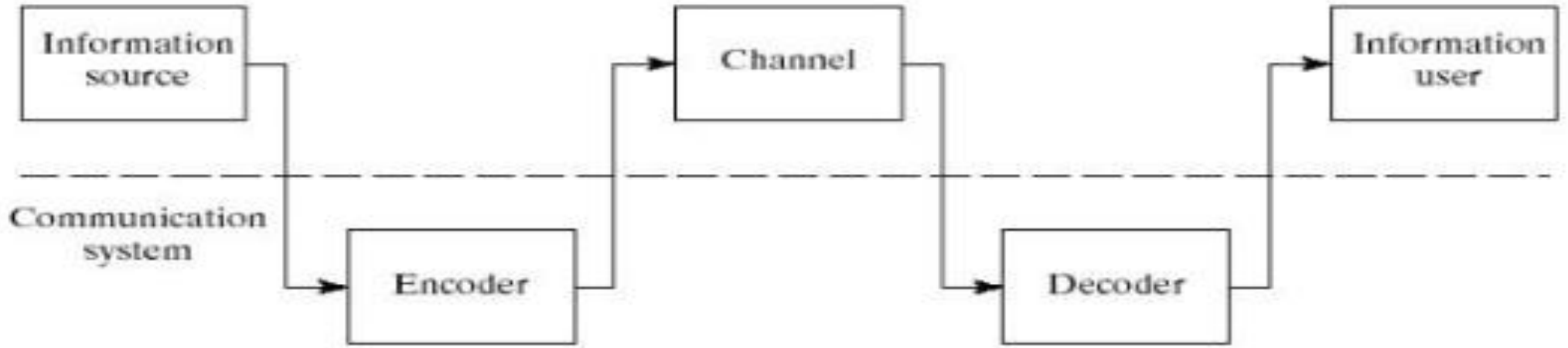


FIGURE 8.9 A communication system model.

2 Cases to be considered: Noiseless and noisy

Noiseless Coding Theorem

Problem: How to code data as compact as possible?

Shannon's first theorem: defines the minimum average code word length per source that can be achieved.

Let source be {A, z} which is zero memory source with J symbols.
(zero memory = each outcome is independent from other outcomes)

then a set of source output of n element be

$$A' = \{\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_{J^n}\}$$

Example:

$$A = \{0,1\}$$

for n = 3,

$$A' = \{000,001,010,011,100,101,110,111\}$$

Noiseless Coding Theorem (cont.)

Probability of each α_j is

$$P(\alpha_j) = P(a_{j1})P(a_{j2})\cdots P(a_{jn})$$

Entropy of source :

$$H(\mathbf{z}') = -\sum_{i=1}^{J^n} P(\alpha_i)\log(P(\alpha_i)) = nH(\mathbf{z})$$

Each code word length $l(\alpha_i)$ can be

$$\log\frac{1}{P(\alpha_i)} \le l(\alpha_i) < \log\frac{1}{P(\alpha_i)} + 1$$

Then average code word length can be

$$\sum_{i=1}^{J^n} P(\alpha_i)\log\frac{1}{P(\alpha_i)} \le \sum_{i=1}^{J^n} P(\alpha_i)l(\alpha_i) < \sum_{i=1}^{J^n} P(\alpha_i)\log\frac{1}{P(\alpha_i)} + 1$$

Image Compression

Noiseless Coding Theorem (cont.)

We get $H(\mathbf{z}') \le L'_{avg} \le H(\mathbf{z}') + 1$

from $H(\mathbf{z}') = nH(\mathbf{z})$

then

$$H(\mathbf{z}) \le \frac{L'_{avg}}{n} \le H(\mathbf{z}) + \frac{1}{n}$$

or

$$\lim_{n\to\infty}\left[\frac{L'_{avg}}{n}\right] = H(\mathbf{z})$$

The minimum average code word length per source symbol cannot lower than the entropy.

Coding efficiency

$$\eta = n\frac{H(z)}{L'_{avg}}$$

Extension Coding Example

α_i	Source Symbols	$P(\alpha_i)$ Eq. (8.3-14)	$I(\alpha_i)$ Eq. (8.3-1)	$l(\alpha_i)$ Eq. (8.3-16)	Code Word	Code Length
First Extension						
α_1	a_1	2/3	0.59	1	0	1
α_2	a_2	1/3	1.58	2	1	1
Second Extension						
α_1	a_1a_1	4/9	1.17	2	0	1
α_2	a_1a_2	2/9	2.17	3	10	2
α_3	a_2a_1	2/9	2.17	3	110	3
α_4	a_2a_2	1/9	3.17	4	111	3

$H = 0.918$

$L_{avg} = 1$

$H = 1.83$

$L_{avg} = 1.89$

$$\eta_1 = 1\frac{0.918}{1} = 0.918$$

$$\eta_2 = \frac{1.83}{1.89} = 0.97$$

Noisy Coding Theorem

Problem: How to code data as reliable as possible?

Example: Repeat each code 3 times:

Source data = {1,0,0,1,1}

Data to be sent = {111,000,000,111,111}

Shannon's second theorem: the maximum rate of coded information is

$$R = \log \frac{\varphi}{r}$$

φ = code size
r = Block length

Image Compression

Rate Distortion Function for BSC

FIGURE 8.10 The rate distortion function for a binary symmetric source.

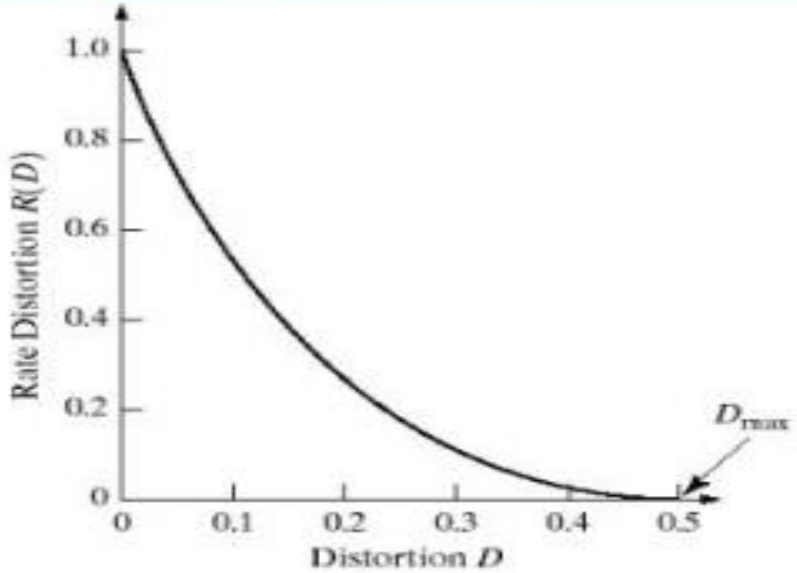


Image Compression

- There are 5 main types of Digital Image Formats:

➢TIFF

➢JPEG

➢GIF

➢PNG

➢RAW

Image Compression

- **TIFF (also known as TIF), file types ending in .tif**
- TIFF stands for Tagged Image File Format. TIFF images create very large file sizes. TIFF images are uncompressed and thus contain a lot of detailed image data (which is why the files are so big) TIFFs are also extremely flexible in terms of color (they can be grayscale, or CMYK for print, or RGB for web) and content (layers, image tags).
- TIFF is the most common file type used in photo software (such as Photoshop), as well as page layout software (such as Quark and InDesign), again because a TIFF contains a lot of image data.

Image Compression

- **JPEG (also known as JPG), file types ending in .jpg**
- JPEG stands for Joint Photographic Experts Group, which created this standard for this type of image formatting. JPEG files are images that have been compressed to store a lot of information in a small-size file. Most digital cameras store photos in JPEG format, because then you can take more photos on one camera card than you can with other formats.
- A JPEG is compressed in a way that loses some of the image detail during the compression in order to make the file small (and thus called “lossy” compression).
- JPEG files are usually used for photographs on the web, because they create a small file that is easily loaded on a web page and also looks good.
- JPEG files are bad for line drawings or logos or graphics, as the compression makes them look “bitmappy” (jagged lines instead of straight ones).

Image Compression

Main Steps in JPEG Image Compression

- Transform RGB to YIQ or YUV and subsample color.
- DCT on image blocks.
- Quantization.
- Zig-zag ordering and run-length encoding.
- Entropy coding.

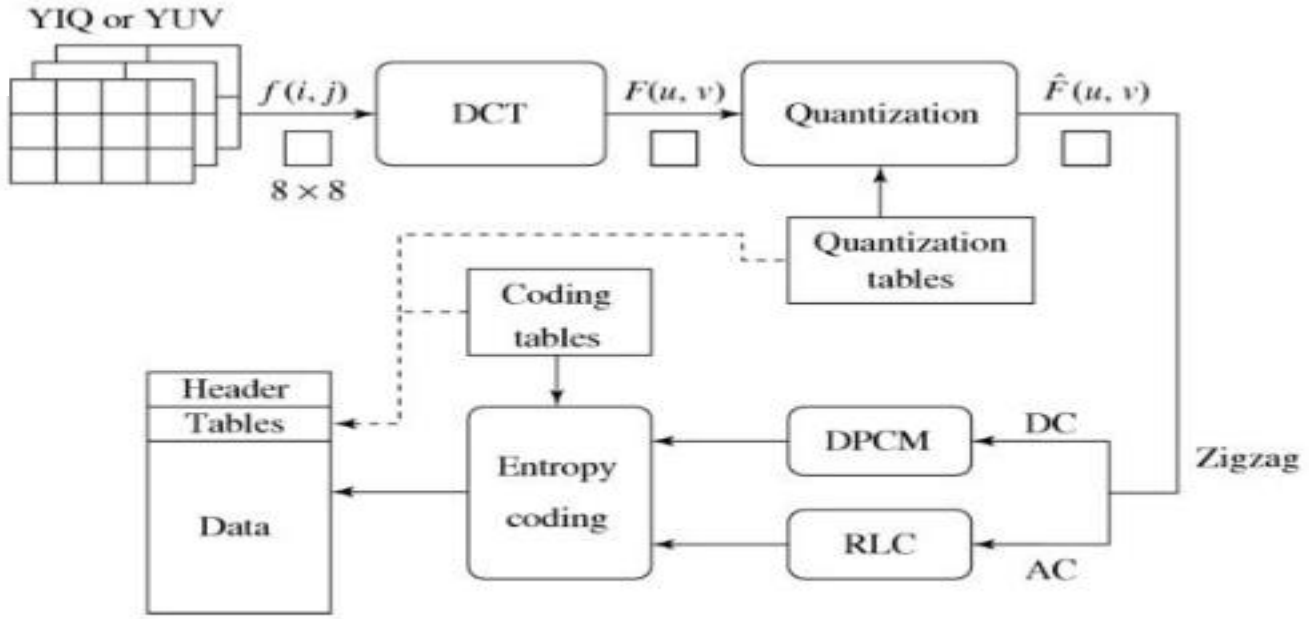


Fig. 9.1: Block diagram for JPEG encoder.

Image Compression

- **GIF, file types ending in .gif**
- GIF stands for Graphic Interchange Format. This format compresses images but, as different from JPEG, the compression is lossless (no detail is lost in the compression, but the file can’t be made as small as a JPEG).
- GIFs also have an extremely limited color range suitable for the web but not for printing. This format is never used for photography, because of the limited number of colors. GIFs can also be used for animations.

Image Compression

- **PNG, file types ending in .png**
- PNG stands for Portable Network Graphics. It was created as an open format to replace GIF, because the patent for GIF was owned by one company and nobody else wanted to pay licensing fees. It also allows for a full range of color and better compression.
- It's used almost exclusively for web images, never for print images. For photographs, PNG is not as good as JPEG, because it creates a larger file. But for images with some text, or line art, it's better, because the images look less "bitmappy."
- When you take a screenshot on your Mac, the resulting image is a PNG– probably because most screenshots are a mix of images and text.

Image Compression

- **Raw image files**
- Raw image files contain data from a digital camera (usually). The files are called raw because they haven't been processed and therefore can't be edited or printed yet. There are a lot of different raw formats–each camera company often has its own proprietary format.
- Raw files usually contain a vast amount of data that is uncompressed. Because of this, the size of a raw file is extremely large. Usually they are converted to TIFF before editing and color-correcting.
- Most of this info is courtesy of Wikipedia, which is a great place to read more about all 5 file types.

Error Free Compression

Error-free compression involves:

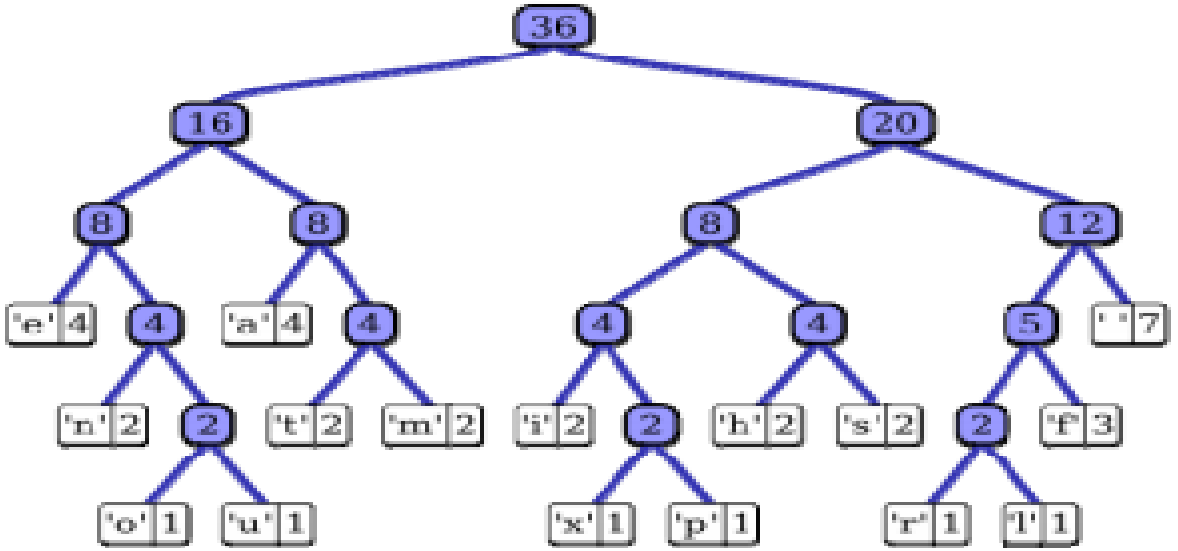
- Variable length coding
- Mapping

Variable-Length Coding (error-free compression)

- Reduce only coding redundancy (1st kind) by minimizing the L_{avg} assign shorter code words to the most probable gray levels
- The source symbols may be
  - *gray levels of an image, or*
  - *output of a gray-level mapping operation (pixel differences, run-lengths, ...*

Image Compression: Huffman Coding

Huffman coding is an entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol.



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				

Image Compression: Huffman Coding

The most popular technique for removing coding redundancy; yields the smallest possible number of code symbols per source symbol

Huffman Coding Algorithm

- Arrange the symbol probabilities p_i in a decreasing order; consider them (p_i) as leaf nodes of a tree
- While there is more than one node:
  - *merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes*
  - *Arrange the combined node according to its probability in the tree*
  - *Repeat until only two nodes are left*
- Starting from the top, arbitrarily assign 1 and 0 to each pair of branches merging into a node
- Continue sequentially from the root node to the leaf node where the symbol is located to complete the coding

Image Compression: Huffman Coding

Huffman coding

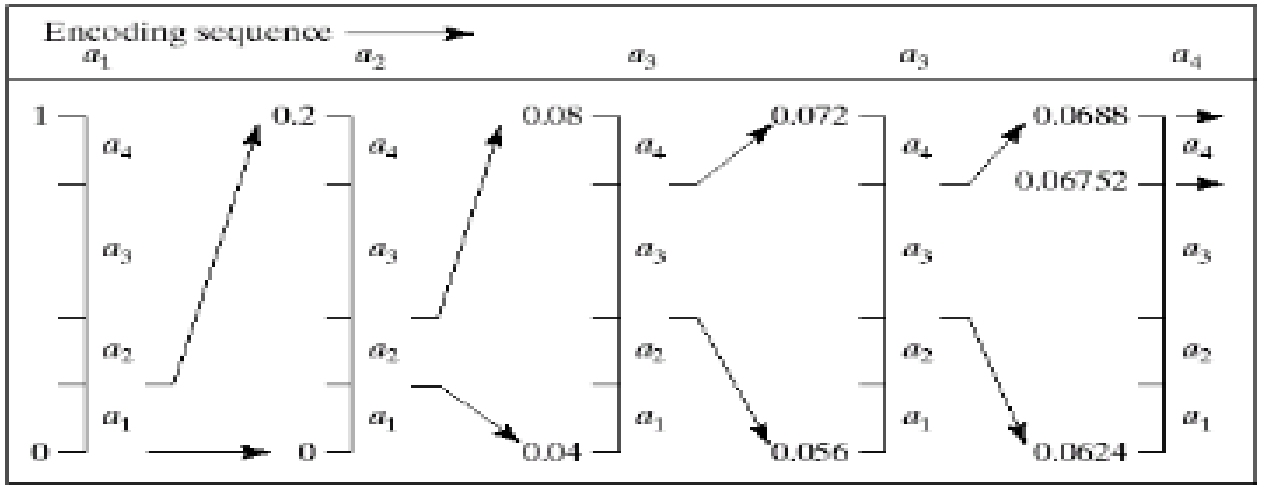
Assignment procedure

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										

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

Image Compression: Arithmetic Coding

Arithmetic coding is a form of variable-length entropy encoding. A string is converted to arithmetic encoding, usually characters are stored with fewer bits
Arithmetic coding encodes the entire message into a single number, a fraction n where ($0.0 \leq n < 1.0$).



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)

TABLE 8.6
Arithmetic coding example.

Image Compression: Arithmetic Coding

Adaptive Content Dependent Probability Estimates

- Arithmetic coders are near optimal in the sense of minimizing the average number of code symbols required to represent the symbols being coded.
- Inaccurate probability models can lead to non-optimal results.
- A simple way to improve the accuracy of the probabilities employed is to use an adaptive, context dependent probability model that update symbol probabilities as symbols are coded or become known.
- The probabilities adapt to the local statistics of the symbols being coded.
- Context dependent models provide probabilities that are based on a predefined neighbourhood of pixels, called the context.
- Both the Q-coder and MQ-coder (ISO/IEC [2000]) arithmetic coding techniques have been incorporated into the JBIG, JPEG-2000 and other compression standards.
- The Q-coder dynamically updates symbol probabilities during the interval renormalizations that are part of the arithmetic coding process

Image Compression: Arithmetic Coding

Arithmetic coding is often used when binary symbols are to be coded.

Each symbol or bit begins the coding process and its context is formed in the Context determination block

The lower blocks show three possible contexts that can be used:

1.the immediately preceding symbol,

2. a group of preceding symbols,

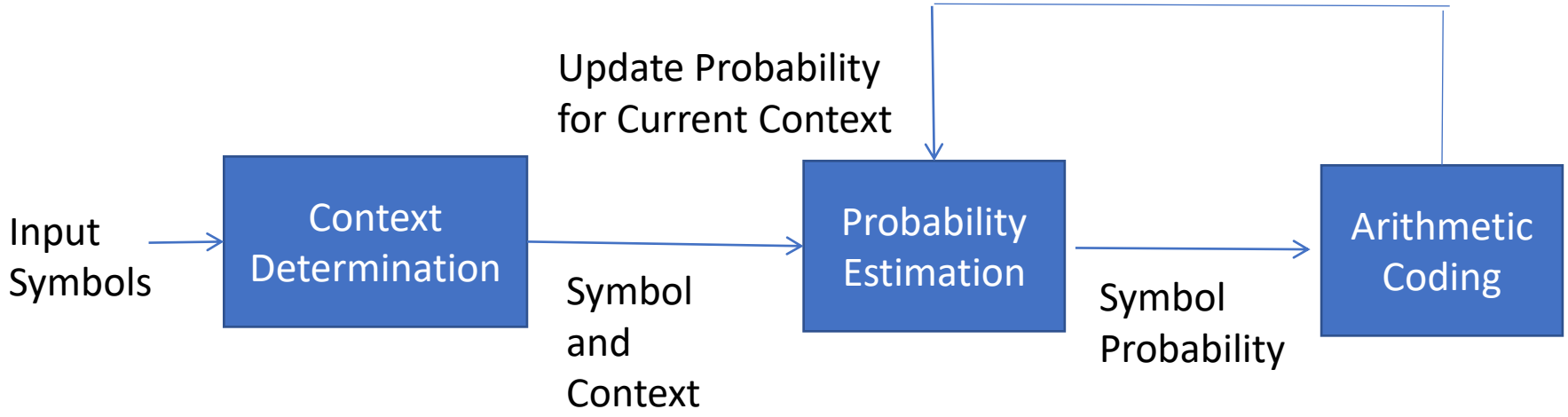
3. some number of preceding symbols plus symbols on the previous scan line.

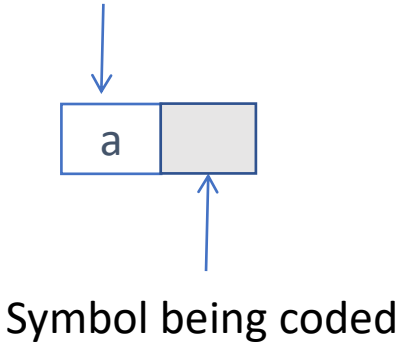
For all the cases the Probability estimation block must manage 2^1 (2), or 2^8 (256) or 2^5 (32) contexts and their associated probabilities.

The appropriate probabilities are then passed to the Arithmetic coding block as a function of the current context and drive the generation of the arithmetically coded output sequence.

The probabilities associated with the context involved in the current coding step are then updated to reflect the fact that another symbol within that context has been processed.

Image Compression: Arithmetic Coding





LZW Coding

- LZW builds a codebook (or dictionary) of symbol sequences (i.e., gray-level sequences) as it processes the image pixels.
- Each symbol sequence is encoded by its dictionary location.

Dictionary Location	Entry
0	…
1	…
…	…
255	…
256	10-120-51
…	…
511	-

For example, the sequence of gray-levels 10-120-51 (3 bytes) will be encoded by 256 (1 byte)

LZW Coding

- Initially, the first 256 entries of the dictionary are assigned to the gray levels 0,1,2,..,255 (i.e., assuming 8 bits/pixel images)

Initial Dictionary

Dictionary Location	Entry
0	0
1	1
…	…
255	255
256	-
…	…
511	-

LZW Coding

- Example:

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

Dictionary Location	Entry
0	0
1	1
.	.
255	255
256	39-39
511	-

As the encoder examines the image pixels, gray level sequences that are not in the dictionary are added to the dictionary.

- Is 39 in the dictionary……..Yes
- What about 39-39……..No
  * Add 39-39 at location 256

Requires no prior knowledge of symbol probabilities.
Assigns sequences of source symbols to fixed length code words.
There is no one-to-one correspondence between source symbols and code words.Included in GIF, TIFF and PDF file formats

So, 39-39 can be encoded by 256!

Run-Length Coding

- Run-length coding (RLC) works by counting adjacent pixels with the same gray level value called the *run-length,* which is then encoded and stored.
- RLC works best for binary, two-valued, images.
- RLC can also work with complex images that have been preprocessed by thresholding to reduce the number of gray levels to two
- RLC can be implemented in various ways, but the first step is to define the required parameters
- Horizontal RLC (counting along the rows) or vertical RLC (counting along the columns) can be used

Run-Length Coding

- In basic horizontal RLC, the number of bits used for the encoding depends on the number of pixels in a row
- If the row has 2^n pixels, then the required number of bits is *n*, so that a run that is the length of the entire row can be encoded

EXAMPLE 10.2.5:

A 256x256 image requires 8-bits, since $2^8 = 256$.

EXAMPLE 10.2.6:

A 512x512 image requires 9-bits, since $2^9 = 512$.

- The next step is to define a convention for the first RLC number in a row – does it represent a run of 0's or 1's?

Run-Length Coding

EXAMPLE 10.2.7:

The image is an 8x8 binary image, which requires 3 bits for each run-length coded word. In the actual image file are stored 1's and 0's, although upon display the 1's become 255 (white) and the 0's are 0 (black). To apply RLC to this image, using horizontal RLC:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Run-Length Coding

EXAMPLE 10.2.7 (contd):

The RLC numbers are:

First row: 8

Second row: 0, 4, 4

Third row: 1, 2, 5

Fourth row: 1, 5, 2

Fifth row: 1, 3, 2, 1, 1

Sixth row: 2, 1, 2, 2, 1

Seventh row: 0, 4, 1, 1, 2

Eighth row: 8

Note that in the second and seventh rows, the first RLC number is 0, since we are using the convention that the first number corresponds to the number of zeros in a run

Run-Length Coding

EXAMPLE 10.2.9:

Given the following 8x8, 4-bit image:

$$\begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\ 10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\ 0 & 0 & 0 & 10 & 10 & 10 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 5 & 10 & 10 & 9 & 9 & 10 \\ 5 & 5 & 5 & 4 & 4 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Run-Length Coding

EXAMPLE 10.2.9 (contd):

The corresponding gray levels pairs are as follows:

First row: 10,8

Second row: 10,5 12,3

Third row: 10,5 12,3

Fourth row: 0,3 10,3 0,2

Fifth row: 5,3 0,5

Sixth row: 5,3 10,2 9,2 10,1

Seventh row: 5,3 4,3 0,2

Eighth row: 0,8

These numbers are then stored in the RLC compressed file as:

10,8,10,5,12,3,10,5,12,3,0,3,10,3,0,2,5,3,0,5,5,3,10,2,9,2,10,1,5,3,4,3,0,2,0,8

Run-Length Coding

- The decompression process requires the number of pixels in a row, and the type of encoding used
- Standards for RLC have been defined by the International Telecommunications Union-Radio (ITU-R, previously CCIR)

These standards use horizontal RLC, but postprocess the resulting RLC with a Huffman encoding scheme

- Newer versions of this standard also utilize a two-dimensional technique where the current line is encoded based on a previous line, which helps to reduce the file size
- These encoding methods provide compression ratios of about 15 to 20 for typical documents

Lempel-Ziv-Welch Coding

- The string table is updated as the file is read, with new codes being inserted whenever a new string is encountered
- If a string is encountered that is already in the table, the corresponding code for that string is put into the compressed file
- LZW coding uses code words with more bits than the original data

Lempel-Ziv-Welch Coding

For Example:

- With 8-bit image data, an LZW coding method could employ 10-bit words
- The corresponding string table would then have $2^{10} = 1024$ entries
- This table consists of the original 256 entries, corresponding to the original 8-bit data, and allows 768 other entries for string codes
- The string codes are assigned during the compression process, but the actual string table is not stored with the compressed data
- During decompression the information in the string table is extracted from the compressed data itself

Lempel-Ziv-Welch Coding

- For the GIF (and TIFF) image file format the LZW algorithm is specified, but there has been some controversy over this, since the algorithm is patented by Unisys Corporation
- Since these image formats are widely used, other methods similar in nature to the LZW algorithm have been developed to be used with these, or similar, image file formats
- Similar versions of this algorithm include the *adaptive Lempel-Ziv*, used in the UNIX compress function, and the *Lempel-Ziv 77* algorithm used in the UNIX gzip function

Gray-Level Run Length Coding

Gray-Level Run Length Coding

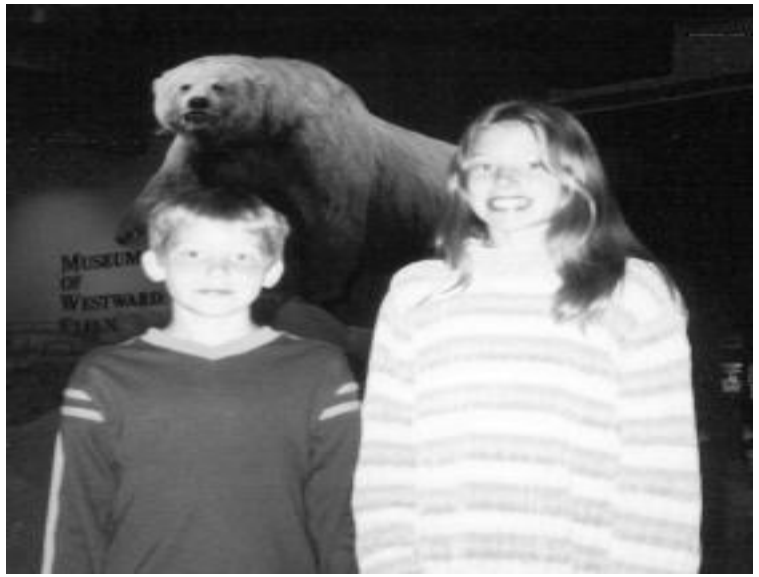
- The RLC technique can also be used for lossy image compression, by reducing the number of gray levels, and then applying standard RLC techniques

- As with the lossless techniques, preprocessing by Gray code mapping will improve the compression ratio

Lossy Bitplane Run Length Coding

Note: No compression occurs until reduction to 5 bits/pixel

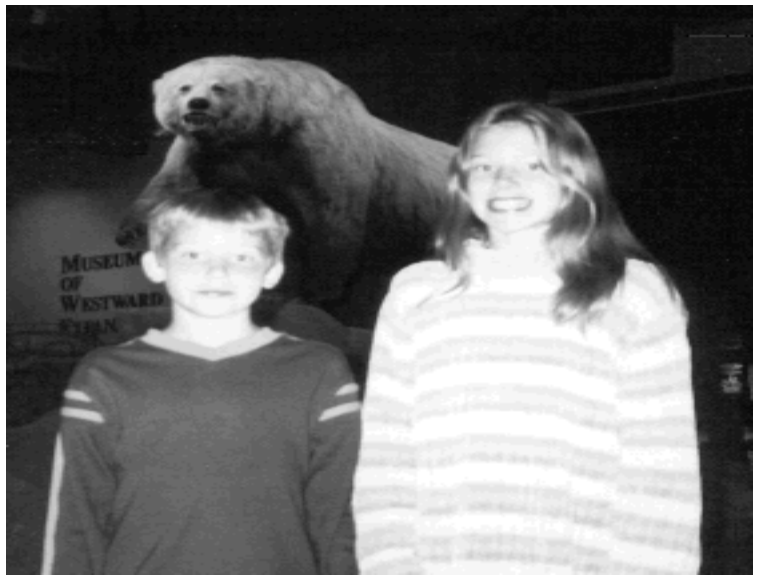
a) Original image, 8 bits/pixel, 256 gray levels



b) Image after reduction to 7 bits/pixel, 128 gray levels, compression ratio 0.55, with Gray code preprocessing 0.66

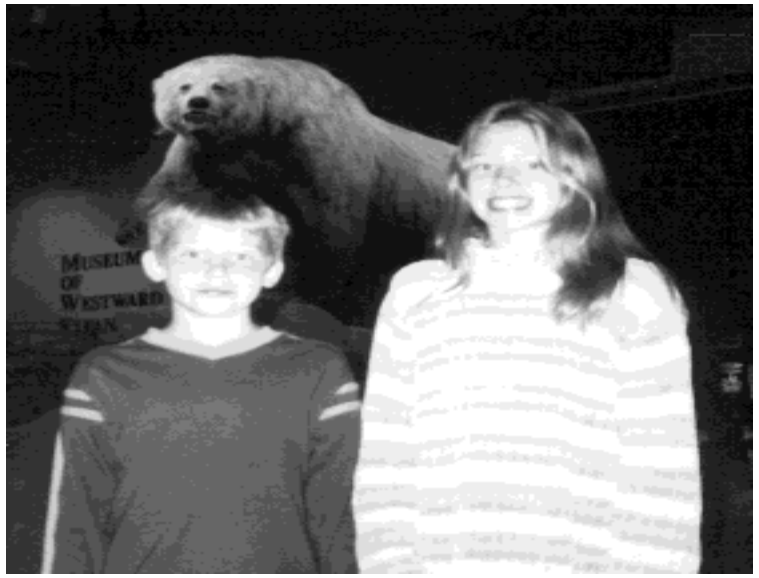
Lossy Bitplane Run Length Coding

c) Image after reduction to 6 bits/pixel, 64 gray levels, compression ratio 0.77, with Gray code preprocessing 0.97



d) Image after reduction to 5 bits/pixel, 32 gray levels, compression ratio 1.20, with Gray code preprocessing 1.60

Lossy Bitplane Run Length Coding



e) Image after reduction to 4 bits/pixel, 16 gray levels, compression ratio 2.17, with Gray code preprocessing 2.79

f) Image after reduction to 3 bits/pixel, 8 gray levels, compression ratio 4.86, with Gray code preprocessing 5.82

Lossy Bitplane Run Length Coding

g) Image after reduction to 2 bits/pixel,
4 gray levels, compression ratio 13.18,
with Gray code preprocessing 15.44

h) Image after reduction to 1 bit/pixel,
2 gray levels, compression ratio 44.46,
with Gray code preprocessing 44.46

Lossy Bitplane Run Length Coding

- A more sophisticated method is *dynamic window-based RLC*
- This algorithm relaxes the criterion of the runs being the same value and allows for the runs to fall within a gray level range, called the *dynamic window range*
- This range is dynamic because it starts out larger than the actual gray level window range, and maximum and minimum values are narrowed down to the actual range as each pixel value is encountered
- This process continues until a pixel is found out of the actual range
- The image is encoded with two values, one for the run length and one to approximate the gray level value of the run
- This approximation can simply be the average of all the gray level values in the run

Lossy Bitplane Run Length Coding

EXAMPLE 10.3.1:

Given the following pixel values in sequence:

65 67 66 64 63 68 70

and a window range of 5.

The first value is called the reference value (in this case = 65). A dynamic window range is then defined that has:

MINIMUM = reference - (window length -1) and

MAXIMUM = reference + (window length - 1)

In this case the dynamic window is: [65 - (5 - 1)] to [65 + (5 - 1]) = 61 to 69.

The next value encountered, 67, is used to adjust this range. The range based on this value alone is from 63 to 71. The new dynamic range is based on the intersection of the range from this new value with the previous range, so the new range is 63 to 69. This process continues until the value of 68 is encountered. At this point the range has been narrowed down to 63 to 67, so the 68 is out of range. This run is then encoded as:

RUN LENGTH = 5

GRAY LEVEL = (65+67+66+64+63)/5 = 65

Figure 10.3-3: Dynamic Window Range RLC

Lossy Bitplane Run Length Coding

a) Original image

b) Window length = 10
compression = 4:1

c) Error image of (b), multiplied to show detail

Lossy Bitplane Run Length Coding

- This particular algorithm also uses some preprocessing to allow for the run-length mapping to be coded so that a run can be any length and is not constrained by the length of a row

Figure 10.3-3, continued

d) Window length = 35
compression = 8:1

e) Error image of (d),
multiplied to show detail

f) Window length = 60
compression = 12.6:1

g) Error image of (f),
multiplied to show detail

Block Truncation Coding

- *Block truncation coding* (BTC) works by dividing the image into small subimages and then reducing the number of gray levels within each block
- The gray levels are reduced by a *quantizer* that adapts to local statistics
- The levels for the quantizer are chosen to minimize a specified error criteria, and then all the pixel values within each block are mapped to the quantized levels
- The necessary information to decompress the image is then encoded and stored
- The basic form of BTC divides the image into N * N blocks and codes each block using a two-level quantizer

Block Truncation Coding

- The two levels are selected so that the mean and variance of the gray levels within the block are preserved
- Each pixel value within the block is then compared with a threshold, typically the block mean, and then is assigned to one of the two levels
- If it is above the mean it is assigned the high level code, if it is below the mean, it is assigned the low level code

Block Truncation Coding

- If we call the high value *H* and the low value *L*, we can find these values via the following equations:

$$H = m_b + \sigma_b \sqrt{\frac{n^2 - q}{q}}$$

$$L = m_b - \sigma_b \sqrt{\frac{q}{n^2 - q}}$$

Where the block size is $n \times n$

$b =$ *the current block*

$m_b =$ *the block mean* $= \frac{1}{n^2} \sum_{I(r,c) \in b} I(r,c)$

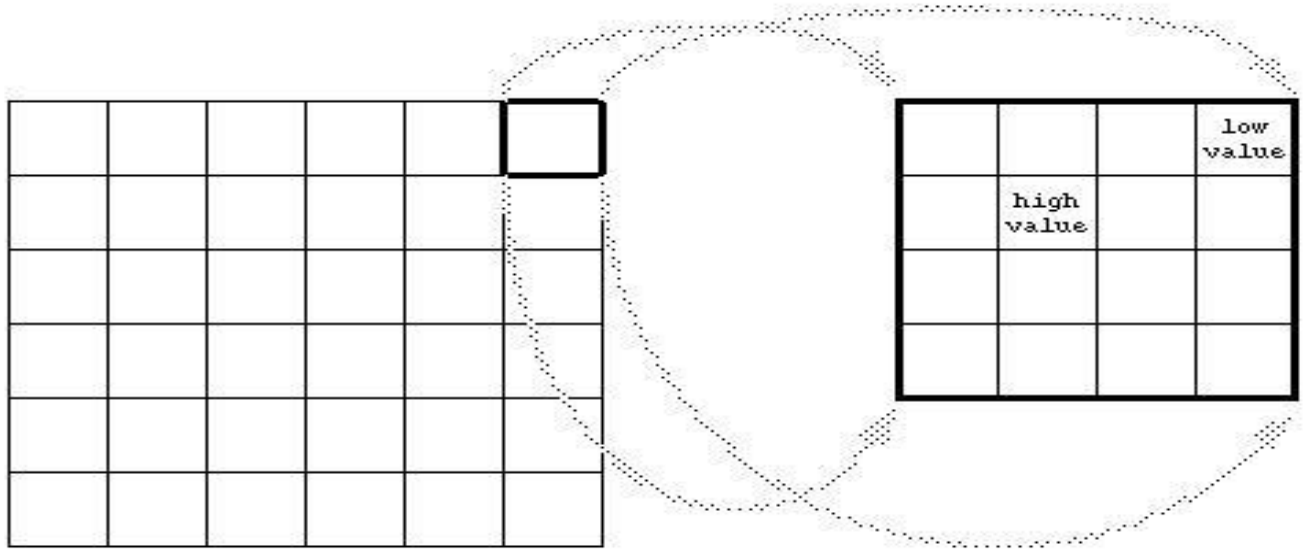
$\sigma_b =$ *the block variance* $= \sqrt{\frac{1}{n^2} \sum_{I(r,c) \in b} [I(r,c)]^2 - m_b^2}$

$q =$ *the number of values in the block* $\geq m_b$

- If *n = 4*, then after the *H* and *L* values are found, the 4x4 block is encoded with four bytes
- Two bytes to store the two levels, *H* and *L*, and two bytes to store a bit string of 1's and 0's corresponding to the high and low codes for that particular block

Block Truncation Coding

Figure 10.3-4: Basic Block Truncation Coding



Block Truncation Coding

EXAMPLE 10.3.2:

Given the following 4x4 subimage, apply basic BTC and find the resulting values.

$$\begin{bmatrix} 12 & 16 & 15 & 17 \\ 13 & 16 & 17 & 17 \\ 4 & 4 & 35 & 35 \\ 42 & 42 & 12 & 12 \end{bmatrix}$$

$$m_b = \frac{1}{n^2} \sum_{I(r,c) \in b} I(r,c) = \frac{1}{16}[12+16+15+17+13+16+17+17+4+4+35+35+42+42+12+12]$$

$$= 19.3125$$

$$\sigma_b = \sqrt{\frac{1}{n^2} \sum_{I(r,c) \in b} [I(r,c)]^2 - m_b^2}$$

$$= \sqrt{\frac{1}{16}\left[12^2+16^2+15^2+17^2+13^2+16^2+17^2+17^2+4^2+4^2+35^2+35^2+42^2+42^2+12^2+12^2\right] - (19.3125)^2}$$

$$\approx 11.85$$

Block Truncation Coding

EXAMPLE 10.3.2 (contd):

There are 4 pixels values greater than the mean, so q = 4.

$$H = m_b + \sigma_b \sqrt{\frac{n^2 - q}{q}} = 19.3125 + 11.85\sqrt{\frac{16-4}{4}} \approx 40$$

$$L = m_b - \sigma_b \sqrt{\frac{q}{n^2 - q}} = 19.3125 - 11.85\sqrt{\frac{4}{16-4}} \approx 13$$

Now, find the bit string by using 0 for values less than the mean and 1 for values greater than the mean:

$$\begin{bmatrix} 12 & 16 & 15 & 17 \\ 13 & 16 & 17 & 17 \\ 4 & 4 & 35 & 35 \\ 42 & 42 & 12 & 12 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \Rightarrow (0000000000111100_2)$$

The high value, H, and the low value, L, will be stored along with the bit string. The subimage, when decompressed will be:

$$\begin{bmatrix} 13 & 13 & 13 & 13 \\ 13 & 13 & 13 & 13 \\ 13 & 13 & 40 & 40 \\ 40 & 40 & 13 & 13 \end{bmatrix}$$

Block Truncation Coding

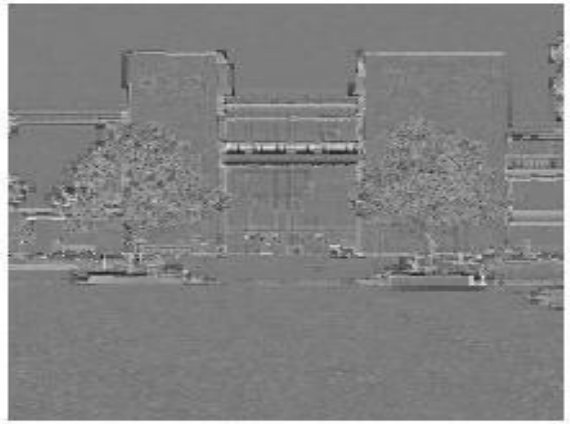
- This algorithm tends to produce images with blocky effects
- These artifacts can be smoothed by applying enhancement techniques such as median and average (lowpass) filters

Block Truncation Coding

Figure 10.3-5: Block Truncation Coding (BTC)

a) Original image

b) Block truncation coded image, compression = 4:1



c) Error image of (b), histogram stretched to show detail

Block Truncation Coding

Figure 10.3-5, continued

d) Image (b) post-processed with a 3x3 median filter

e) Image (b) post-processed with a 3x3 averaging filter

Transform Coding

- *Transform coding,* is a form of block coding done in the transform domain
- The image is divided into blocks, or subimages, and the transform is calculated for each block
- Any of the previously defined transforms can be used, frequency (e.g. Fourier) or sequency (e.g. Walsh/Hadamard), but it has been determined that the discrete cosine transform (DCT) is optimal for most images
- The newer JPEG2000 algorithms uses the wavelet transform, which has been found to provide even better compression

Transform Coding

- After the transform has been calculated, the transform coefficients are quantized and coded
- This method is effective because the frequency/sequency transform of images is very efficient at putting most of the information into relatively few coefficients, so many of the high frequency coefficients can be quantized to 0 (eliminated completely)
- This type of transform is a special type of mapping that uses spatial frequency concepts as a basis for the mapping
- The main reason for mapping the original data into another mathematical space is to pack the information (or energy) into as few coefficients as possible

Transform Coding

- The simplest form of transform coding is achieved by *filtering* by eliminating some of the high frequency coefficients
- However, this will not provide much compression, since the transform data is typically *floating point* and thus 4 or 8 bytes per pixel (compared to the original pixel data at 1 byte per pixel), so quantization and coding is applied to the reduced data
- Quantization includes a process called *bit allocation*, which determines the number of bits to be used to code each coefficient based on its importance
- Typically, more bits are used for lower frequency components where the energy is concentrated for most images, resulting in a *variable bit rate* or *nonuniform quantization* and better resolution

Transform Coding

EXAMPLE 10.3.6:

We have decided to use transform coding with a DCT on an image by dividing it into 4x4 blocks. The selected bit allocation can be represented by the following mask:

$$\begin{bmatrix} 8 & 6 & 4 & 1 \\ 6 & 4 & 1 & 0 \\ 4 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Where the numbers in the mask are the number of bits used to represent the corresponding transform coefficients. The upper left corner corresponds to the zero-frequency coefficient, DC or average value, and the frequency increases to the right and down. This allows the lower frequencies less quantization (more resolution), since they have more bits to represent them. Because the number of bits varies with different coefficients, we have a *variable bit rate*. Additionally, more bits means finer quantization and fewer bits means coarser quantization resulting in *nonuniform quantization*

Transform Coding

- Then a quantization scheme, such as Lloyd-Max quantization is applied
- As the zero-frequency coefficient for real images contains a large portion of the energy in the image and is always positive, it is typically treated differently than the higher frequency coefficients
- Often this term is not quantized at all, or the differential between blocks is encoded
- After they have been quantized, the coefficients can be coded using, for example, a Huffman or arithmetic coding method

Transform Coding

- Two particular types of transform coding have been widely explored:

1. *Zonal coding*
2. *Threshold coding*

- These two vary in the method they use for selecting the transform coefficients to retain (using ideal filters for transform coding selects the coefficients based on their location in the transform domain)

Zonal Coding

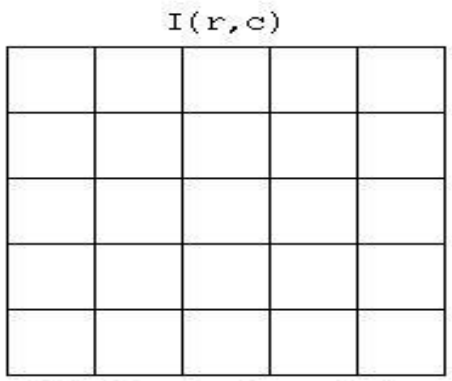
1. *Zonal coding*

- It involves selecting specific coefficients based on maximal variance
- A zonal mask is determined for the entire image by finding the variance for each frequency component
- This variance is calculated by using each subimage within the image as a separate sample and then finding the variance within this group of subimages

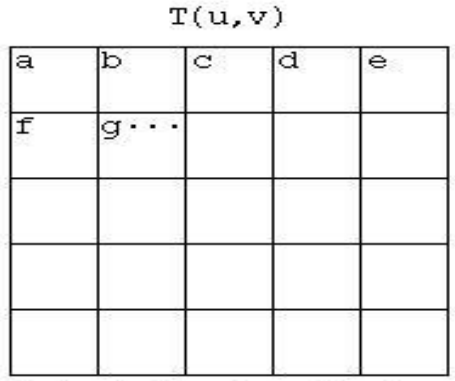
Zonal Coding

- The *zonal mask* is a bitmap of 1's and 0', where the 1's correspond to the coefficients to retain, and the 0's to the ones to eliminate
- As the zonal mask applies to the *entire image,* only one mask is required

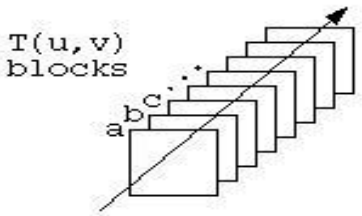
Figure 10.3-17: Zonal Coding



a) Divide the image into blocks



b) Apply the transform to each block



c) Treating each transform block from T(u,v) as a separate sample, calculate the variance for each frequency component. Retain only the components with variance above a specified threshold.

1	1	1	1	0
1	1	1	0	0
1	1	0	0	0
1	0	0	0	0
0	0	0	0	0

d) Generate zonal masks;
1 = retain
0 = eliminate
A typical mask is shown.

Threshold Coding

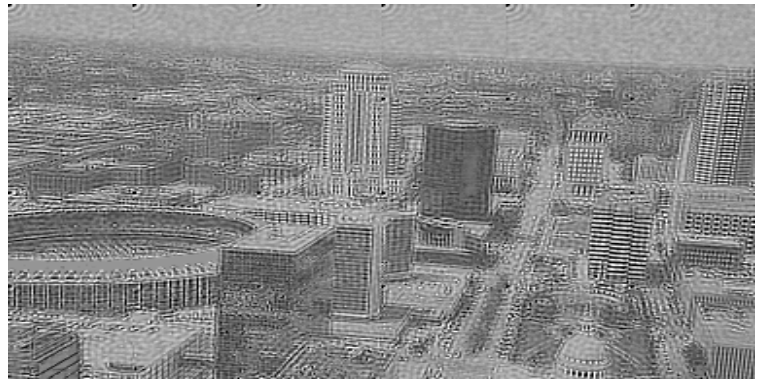
- It selects the transform coefficients based on specific value
- A different threshold mask is required for each block, which increases file size as well as algorithmic complexity
- In practice, the zonal mask is often predetermined because the low frequency terms tend to contain the most information, and hence exhibit the most variance
- In this case we select a fixed mask of a given shape and desired compression ratio, which streamlines the compression process
- In practice, the zonal mask is often predetermined because the low frequency terms tend to contain the most information, and hence exhibit the most variance
- In this case we select a fixed mask of a given shape and desired compression ratio, which streamlines the compression process

Zonal Compression with DCT and Walsh Transforms

A block size of 64x64 was used, a circular zonal mask, and DC coefficients were not quantized

a) Original image, a view of St. Louis, Missouri, from the Gateway Arch

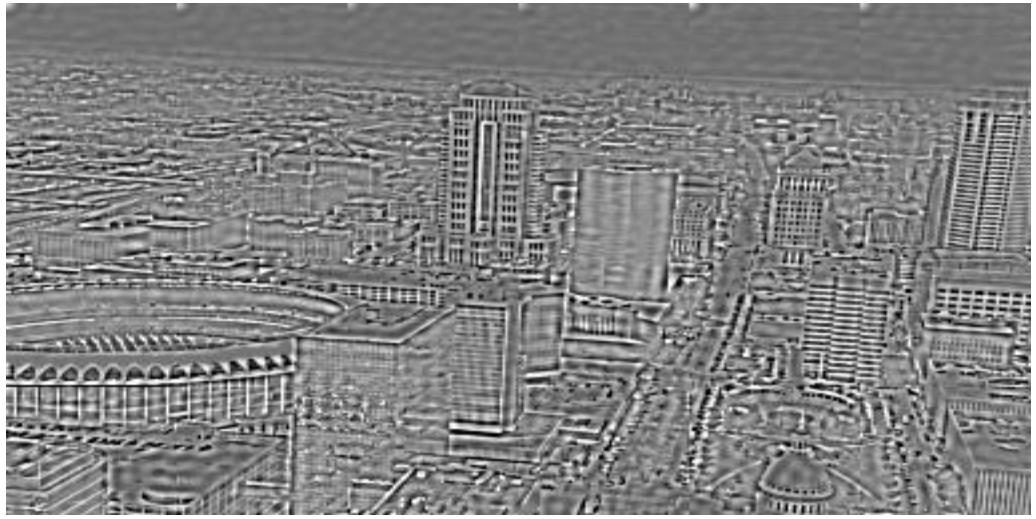
b) Results from using the DCT with a compression ratio = 4.27



c) Error image comparing the original and (b), histogram stretched to show detail

Zonal Compression with DCT and Walsh Transforms

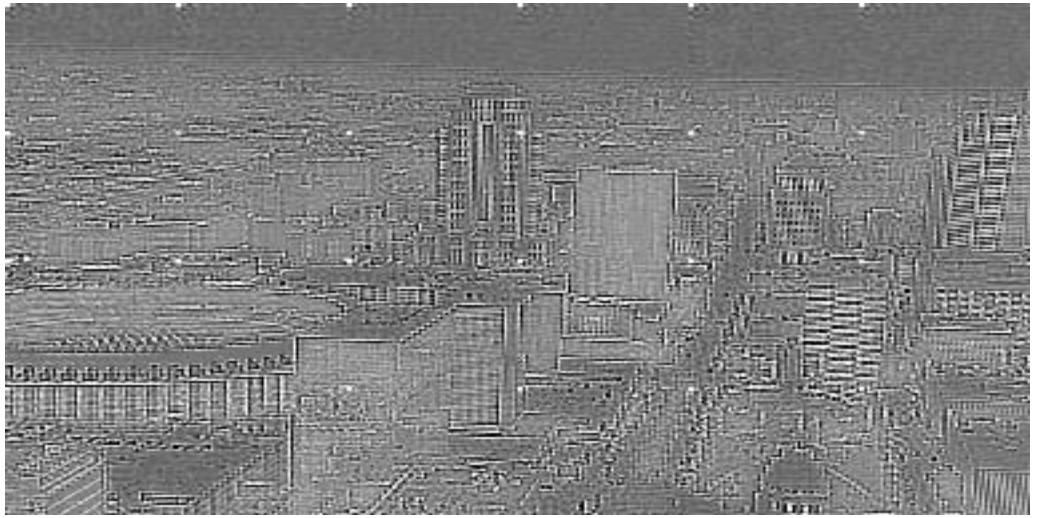
d) Results from using the DCT with a compression ratio = 14.94



e) Error image comparing the original and (d), histogram stretched to show detail,

Zonal Compression with DCT and Walsh Transforms

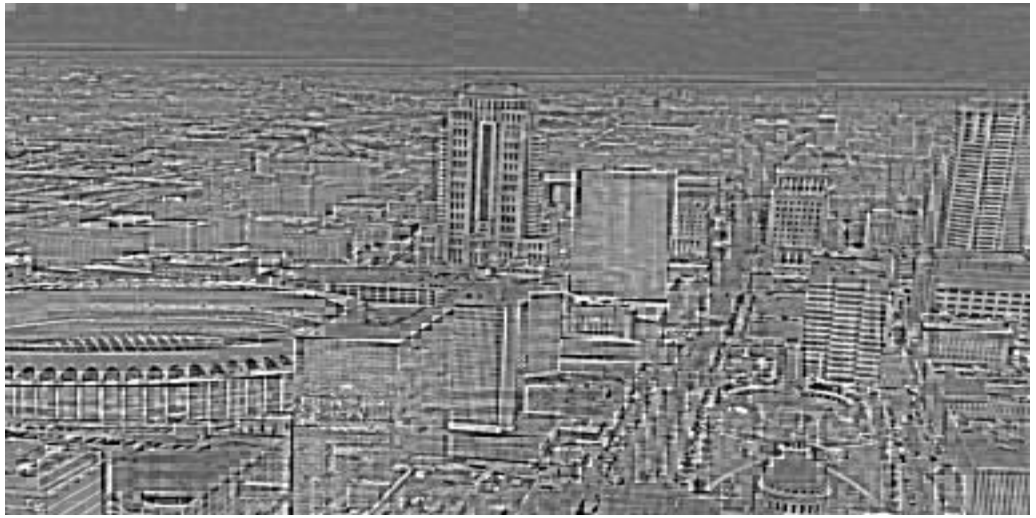
f) Results from using the Walsh Transform (WHT) with a compression ratio = 4.27



g) Error image comparing the original and (f), histogram stretched to show detail

Zonal Compression with DCT and Walsh Transforms

h) Results from using the WHT with a compression ratio = 14.94



i) Error image comparing the original and (h), histogram stretched to show detail

Zonal Compression with DCT and Walsh Transforms

- One of the most commonly used image compression standards is primarily a form of transform coding
- The Joint Photographic Expert Group (JPEG) under the auspices of the International Standards Organization (ISO) devised a family of image compression methods for still images
- The original JPEG standard uses the DCT and 8x8 pixel blocks as the basis for compression
- Before computing the DCT, the pixel values are level shifted so that they are centered at zero
- EXAMPLE 10.3.7:

  A typical 8-bit image has a range of gray levels of 0 to 255. Level shifting this range to be centered at zero involves subtracting 128 from each pixel value, so the resulting range is from -128 to 127

Zonal Compression with DCT and Walsh Transforms

- After level shifting, the DCT is computed
- Next, the DCT coefficients are quantized by dividing by the values in a quantization table and then truncated
- For color signals JPEG transforms the RGB components into the *YCrCb* color space, and subsamples the two color difference signals (*Cr* and *Cb*), since we perceive more detail in the luminance (brightness) than in the color information
- Once the coefficients are quantized, they are coded using a Huffman code
- The zero-frequency coefficient (DC term) is differentially encoded relative to the previous block

Zonal Compression with DCT and Walsh Transforms

Figure 10.3-19 Original JPEG DCT Coefficient Quantization Tables. a) luminance (brightness) quantization table corresponding to the Y component of the color transform, b) chrominance (color) quantization table corresponding to the Cr and Cb components of the color transform. The zero frequency (DC) term is in the upper left corner, at (1,1), and the frequencies increase to the right and down. As the quantization coefficient increases, the quantization gets coarser. For many high frequency terms the quantization coefficient is large enough that, after dividing and truncating, they are zero.

a)

Mask	1	2	3	4	5	6	7	8
1	16	11	10	16	24	40	51	61
2	12	12	14	19	26	58	60	55
3	14	13	16	24	40	57	69	56
4	14	17	22	29	51	87	80	62
5	18	22	37	56	68	109	103	77
6	24	35	55	64	81	104	113	92
7	49	64	78	87	103	121	120	101
8	72	92	95	98	112	100	103	99

b)

Mask	1	2	3	4	5	6	7	8
1	17	18	24	47	99	99	99	99
2	18	21	26	66	99	99	99	99
3	24	26	56	99	99	99	99	99
4	47	66	99	99	99	99	99	99
5	99	99	99	99	99	99	99	99
6	99	99	99	99	99	99	99	99
7	99	99	99	99	99	99	99	99
8	99	99	99	99	99	99	99	99

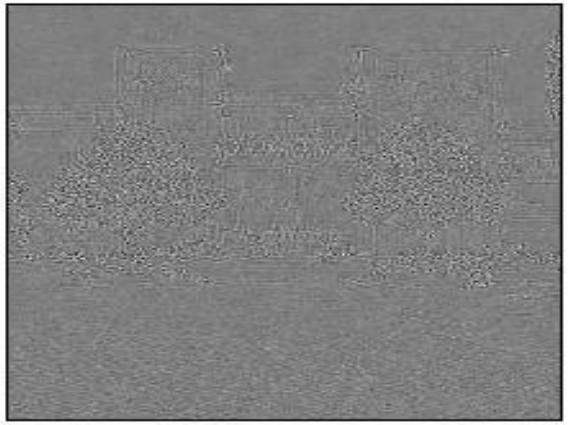
These quantization tables were experimentally determined by JPEG to take advantage of the human visual system's response to spatial frequency which peaks around 4 or 5 cycles per degree

Figure 10.3-20: The Original DCT-based JPEG Algorithm

Zonal Compression with DCT and Walsh Transforms

a) Original image

b) JPEG compression = 10:1

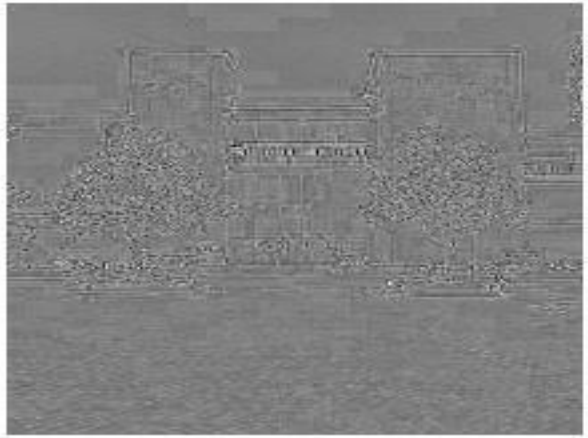


c) Error image for (b), multiplied by 8 to show detail

Zonal Compression with DCT and Walsh Transforms

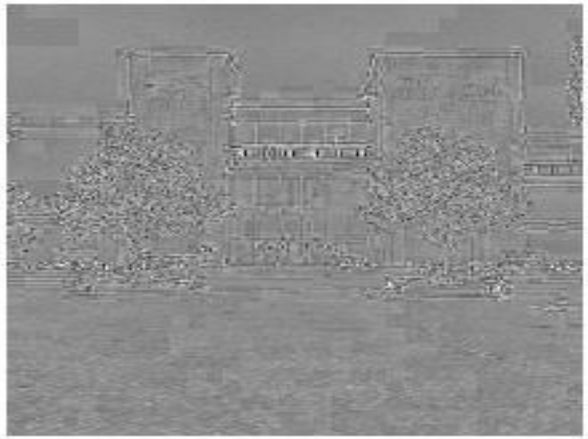
Figure 10.3-20, continued

d) JPEG compression = 20:1



e) Error image for (d), multiplied by 8 to show detail

f) JPEG compression = 30:1



g) Error image for (f), multiplied by 8 to show detail

The Original DCT-based JPEG Algorithm Applied to a Color Image

a) The original image

b) Compression ratio = 34.34

The Original DCT-based JPEG Algorithm Applied to a Color Image

c) Compression ratio = 57.62

d) Compression ratio = 79.95

The Original DCT-based JPEG Algorithm Applied to a Color Image

e) Compression ratio = 131.03

f) Compression ratio = 201.39

Image Compression

Lossless VS Lossy Coding

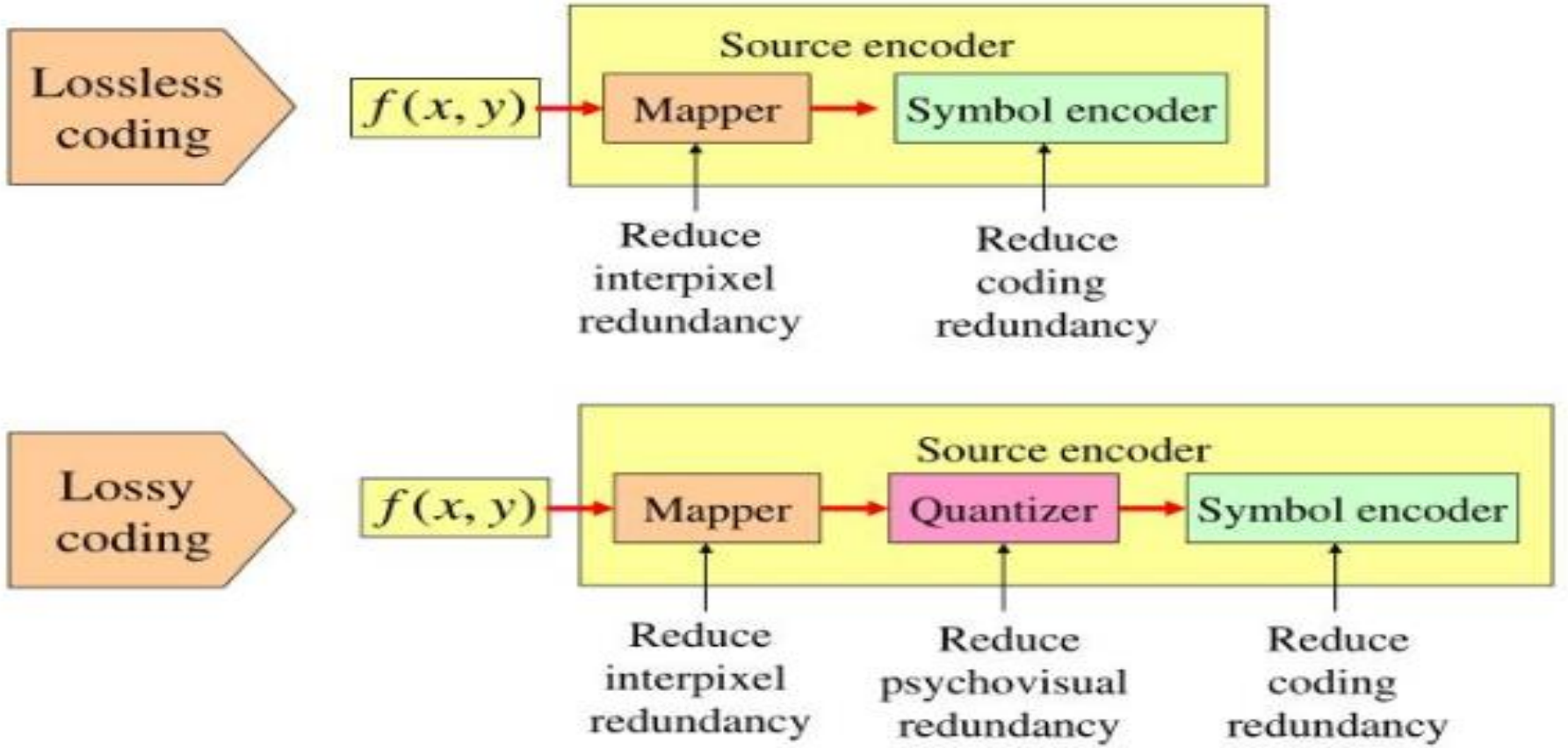


Image Compression

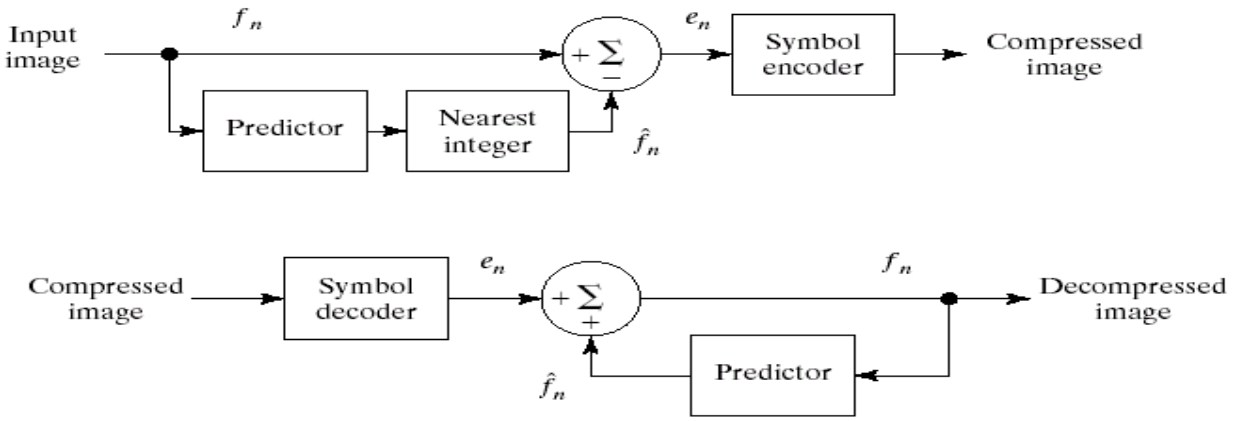
Lossless Predictive coding

The encoder expects a discrete sample of a signal f(n).

1. A predictor is applied and its output is rounded to the nearest integer. $\hat{f}(n)$
2. The error is estimated as $e(n) = f(n) - \hat{f}(n)$
3. The compressed stream consist of first sample and the errors, encoded using variable length coding

a
b

FIGURE 8.19 A lossless predictive coding model: (a) encoder; (b) decoder.



The decoder uses the predictor and the error stream to reconstructs the original signal f(n).

1. The predictor is initialized using the first sample.
2. The received error is added to predictor result.

$$f(n) = \hat{f}(n) + e(n)$$

Image Compression

Lossless Predictive coding

Linear predictors usually have the form:

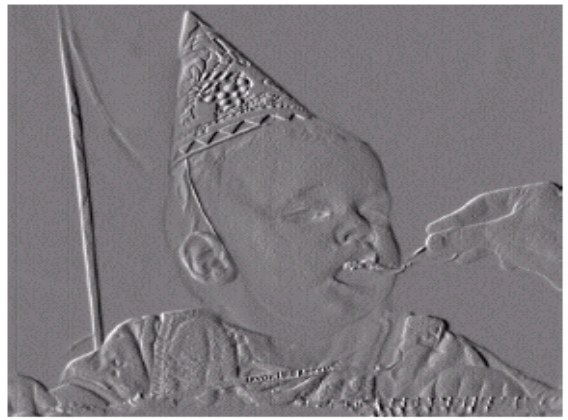
$$\hat{f}(n) = round\left[\sum_{i=0}^{m} a_i f(n-i)\right]$$

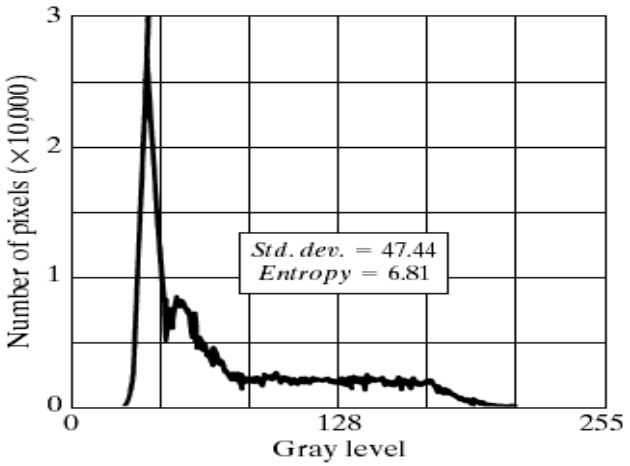
Original Image (view of the earth).
The prediction error and its histogram.

1. The error is small in uniform regions
2. Large close to edges and sharp changes in pixel intensities

a
b c

FIGURE 8.20
(a) The prediction error image resulting from Eq. (8.4-9).
(b) Gray-level histogram of the original image.
(c) Histogram of the prediction error.





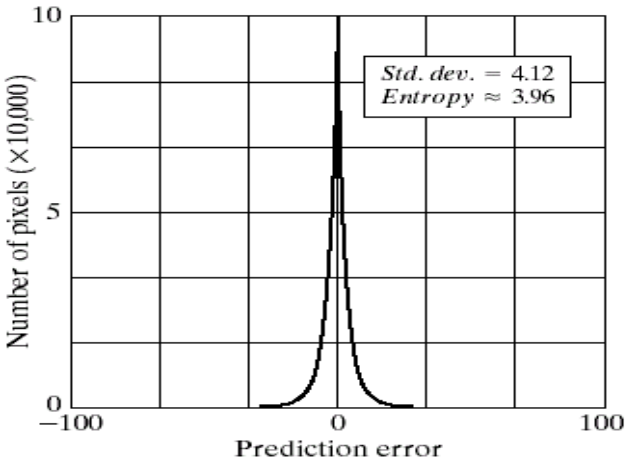
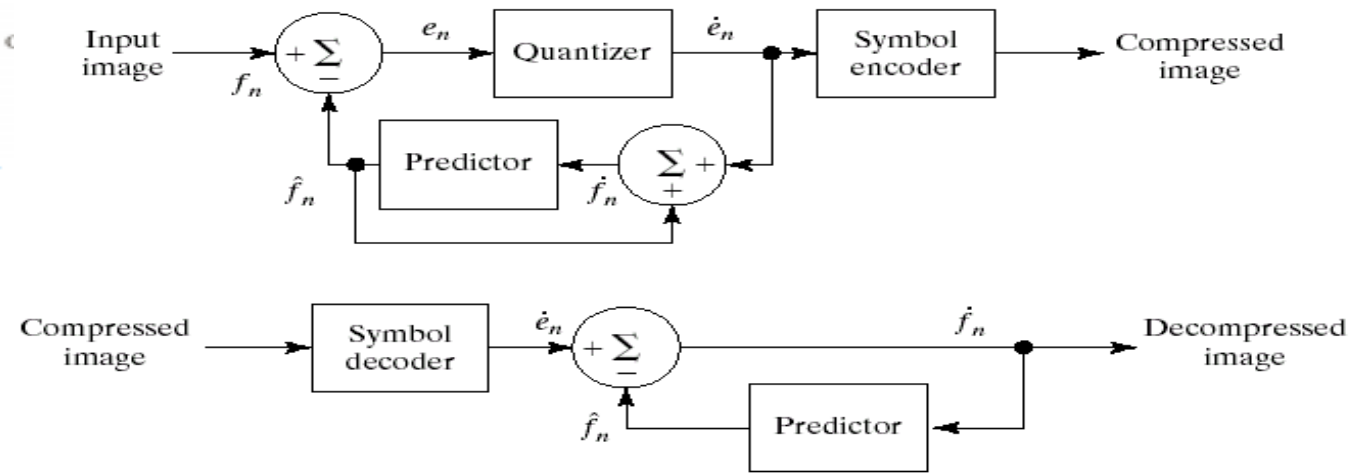


Image Compression

Lossy Predictive coding

The encoder expects a discrete samples of a signal f(n).

1. A predictor is applied and its output is rounded to the nearest integer, $\hat{f}(n)$
2. The error is mapped into limited rage of values (quantized) $\dot{e}(n)$
3. The compressed stream consist of first sample and the mapped errors, encoded using variable length coding



a
b

FIGURE 8.21 A lossy predictive coding model: (a) encoder and (b) decoder.

Image Compression

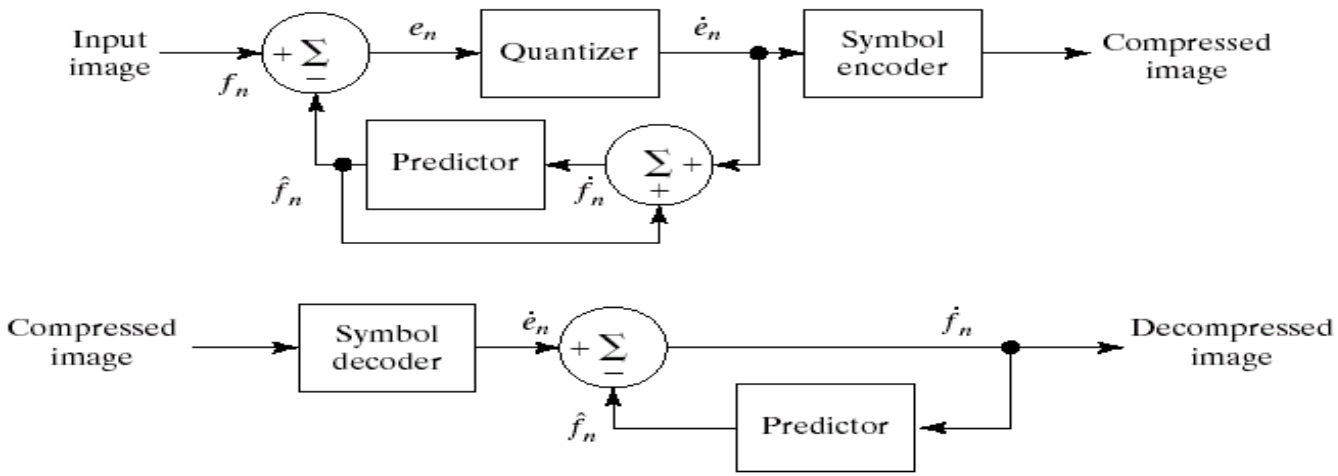
Lossy Predictive coding

The decoder uses error stream to reconstructs an approximation of the original signal, $\dot{f}(n)$

1. The predictor is initialized using first sample.
2. The received error is added to predictor result.

$$\dot{f}(n) = \dot{e}(n) + \hat{f}(n)$$

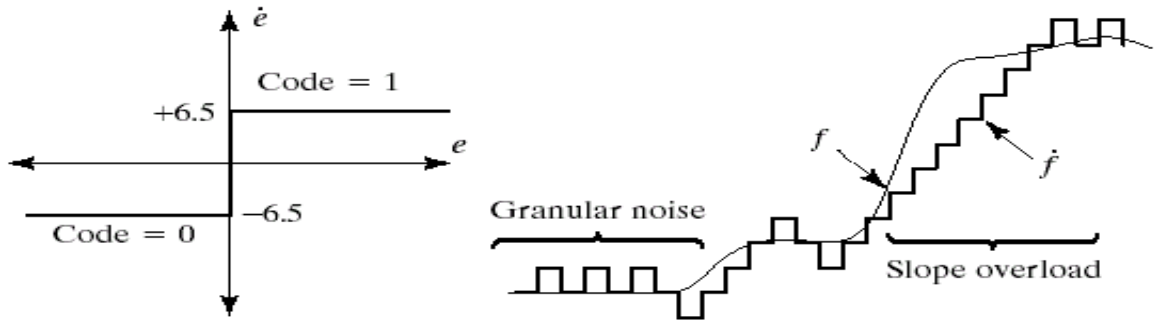
$$\dot{e}(n) = \begin{cases} +\xi & e(n) > 0 \\ -\xi & otherwise \end{cases}$$



a
b

FIGURE 8.21 A lossy predictive coding model: (a) encoder and (b) decoder.

Image Compression



a b
c

FIGURE 8.22 An example of delta modulation.

Input		Encoder				Decoder		Error
n	f	$\hat{f}$	e	$\dot{e}$	$\dot{f}$	$\hat{f}$	$\dot{f}$	$[f - \dot{f}]$
0	14	—	—	—	14.0	—	14.0	0.0
1	15	14.0	1.0	6.5	20.5	14.0	20.5	–5.5
2	14	20.5	–6.5	–6.5	14.0	20.5	14.0	0.0
3	15	14.0	1.0	6.5	20.5	14.0	20.5	–5.5
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
14	29	20.5	8.5	6.5	27.0	20.5	27.0	2.0
15	37	27.0	10.0	6.5	33.5	27.0	33.5	3.5
16	47	33.5	13.5	6.5	40.0	33.5	40.0	7.0
17	62	40.0	22.0	6.5	46.5	40.0	46.5	15.5
18	75	46.5	28.5	6.5	53.0	46.5	53.0	22.0
19	77	53.0	24.0	6.5	59.6	53.0	59.6	17.5
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Image Compression

a b
c d

FIGURE 8.24 A comparison of four linear prediction techniques.

Prediction Error

The following images show the prediction error of the predictor

$$\hat{f}(x,y) = 0.97 f(x,y-1)$$

$$\hat{f}(x,y) = 0.5 f(x,y-1) + 0.5 f(x-1,y)$$

$$\hat{f}(x,y) - 0.75 f(x,y-1) + 0.75 f(x-1,y) - 0.5 f(x-1,y-1)$$

$$\hat{f}(x,y) = \begin{cases} 0.97 f(x,y-1) & \Delta h \le \Delta v \\ 0.97 f(x-1,y) & otherwise \end{cases}$$

$$\Delta h = | f(x-1,y) - f(x-1,y-1) |$$

$$\Delta v = | f(x,y-1) - f(x-1,y-1) |$$

Image Compression

Optimal Predictors

What are the parameters of a linear predictor that minimize error

$$E\{e^2(n)\} = E\left\{\left[f(n) - \hat{f}(n)\right]^2\right\}$$

While taking into account

$$\dot{f}(n) = \dot{e}(n) + \hat{f}(n) \cong e(n) + \hat{f}(n) = f(n)$$

Using the definition of linear predictor

$$E\{e^2(n)\} = E\left\{\left[f(n) - \sum_{i=1}^{m} \alpha_i f(n-1)\right]^2\right\}$$

We assume that f(n) has a mean zero and variance σ^2

$$\alpha = R^{-1} r$$

Image Compression

And R^{-1} is the mxn autocorrelation matrix

$$R = \begin{bmatrix} E\{f(n-1)f(n-1)\} & E\{f(n-1)f(n-2)\} & \dots & E\{f(n-1)f(n-m)\} \\ E\{f(n-2)f(n-1)\} & E\{f(n-2)f(n-2)\} & \dots & E\{f(n-2)f(n-m)\} \\ \vdots & \vdots & \vdots & \vdots \\ E\{f(n-m)f(n-1)\} & E\{f(n-m)f(n-1)\} & \dots & E\{f(n-m)f(n-1)\} \end{bmatrix}$$

$$r = \begin{bmatrix} E\{f(n)f(n-1)\} \\ \vdots \\ E\{f(n-1)f(n-m)\} \end{bmatrix}$$

$$a = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix}$$

Image Compression

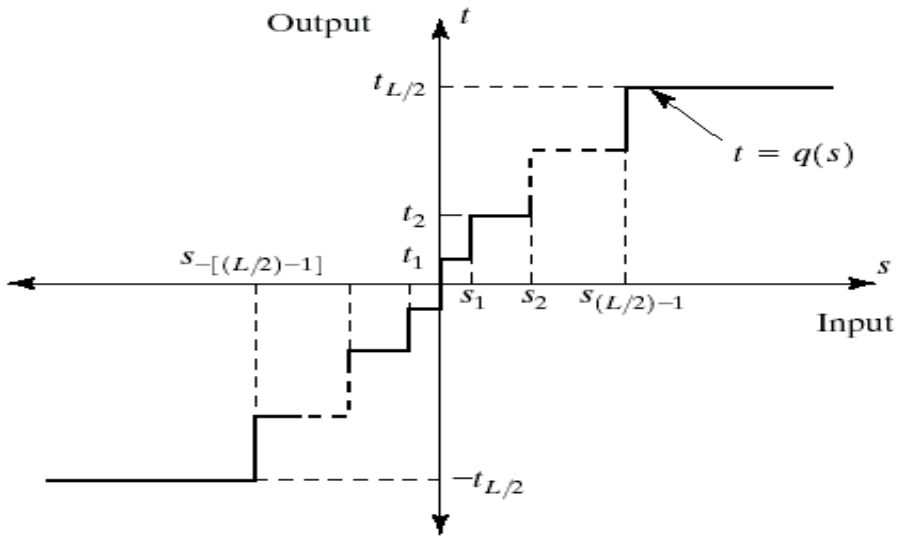


FIGURE 8.25 A typical quantization function.

TABLE 8.10 Lloyd-Max quantizers for a Laplacian probability density function of unit variance.

Levels	2		4		8	
i	s_i	t_i	s_i	t_i	s_i	t_i
1	∞	0.707	1.102	0.395	0.504	0.222
2			∞	1.810	1.181	0.785
3					2.285	1.576
4					∞	2.994
θ	1.414		1.087		0.731	

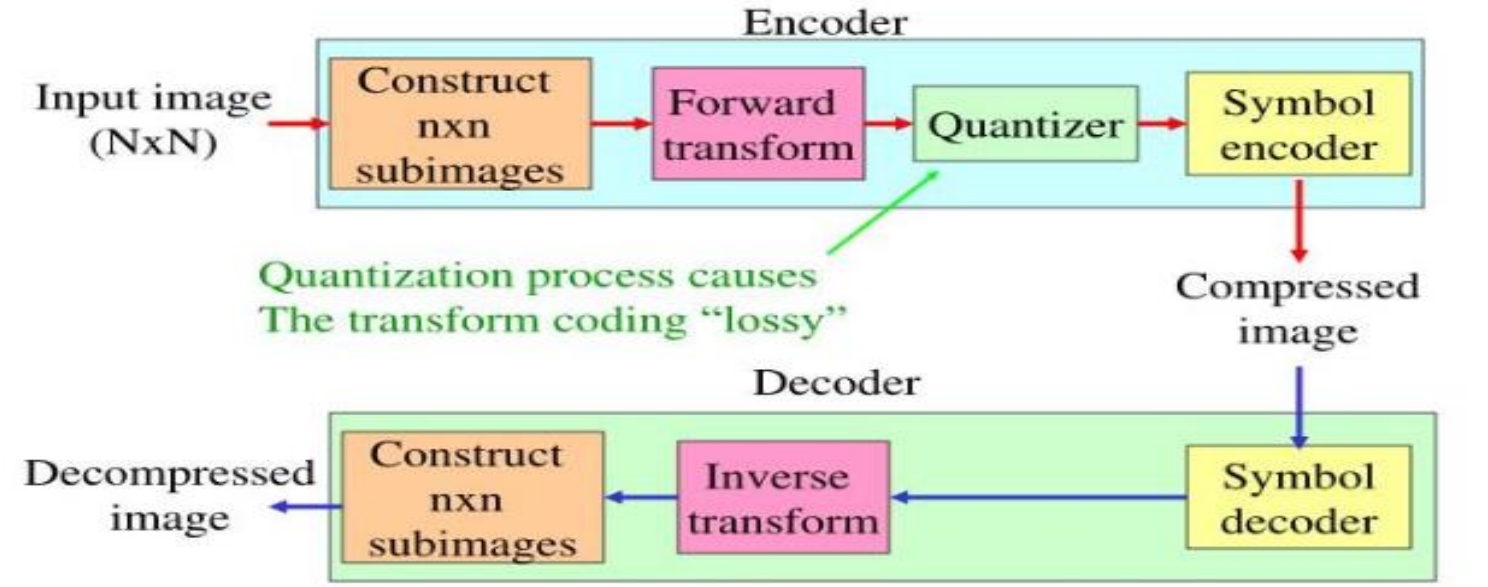
Image Compression

Predictor	Lloyd-Max Quantizer			Adaptive Quantizer		
	2-level	4-level	8-level	2-level	4-level	8-level
Eq. (8.5-16)	30.88	6.86	4.08	7.49	3.22	1.55
Eq. (8.5-17)	14.59	6.94	4.09	7.53	2.49	1.12
Eq. (8.5-18)	9.90	4.30	2.31	4.61	1.70	0.76
Eq. (8.5-19)	38.18	9.25	3.36	11.46	2.56	1.14
Compression	8.00:1	4.00:1	2.70:1	7.11:1	3.77:1	2.56:1

TABLE 8.11 Lossy DPCM root-mean-square error summary.

Image Compression

Transform Coding (for fixed resolution transforms)



Examples of transformations used for image compression: DFT and DCT

3 Parameters that effect transform coding performance:

1. Type of transformation
2. Size of subimage
3. Quantization algorithm

Image Compression

2D Discrete Transformation

Forward transform:

$$T(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\,g(x,y,u,v)$$

where $g(x,y,u,v)$ = forward transformation kernel or basis function

$T(u,v)$ is called the transform coefficient image.

Inverse transform:

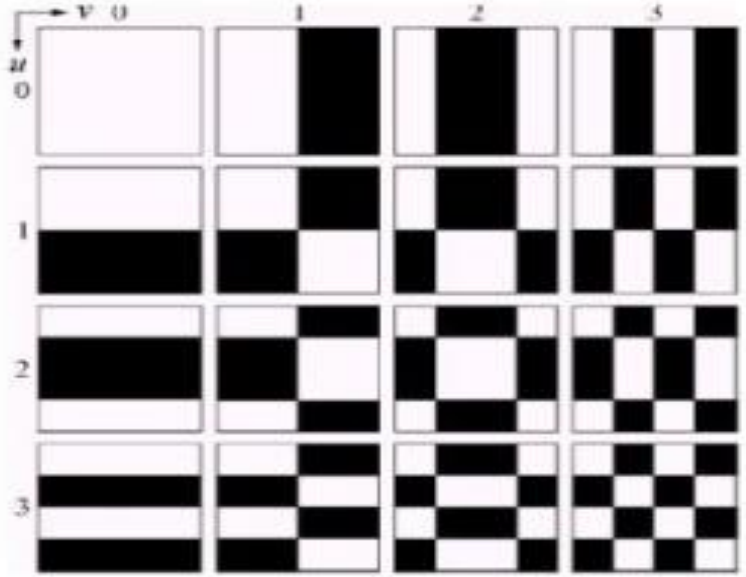
$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} T(u,v)\,h(x,y,u,v)$$

where $h(x,y,u,v)$ = inverse transformation kernel or inverse basis function

Image Compression

Transform Example: Walsh-Hadamard Basis Functions

$$g(x,y,u,v) = h(u,v,x,y) = \frac{1}{N}(-1)^{\sum_{i=0}^{m-1}\left[b_i(x)p_i(u)+b_i(y)p_i(v)\right]}$$



N = 4

$$N = 2^m$$

$b_k(z)$ = the k^{th} bit of z

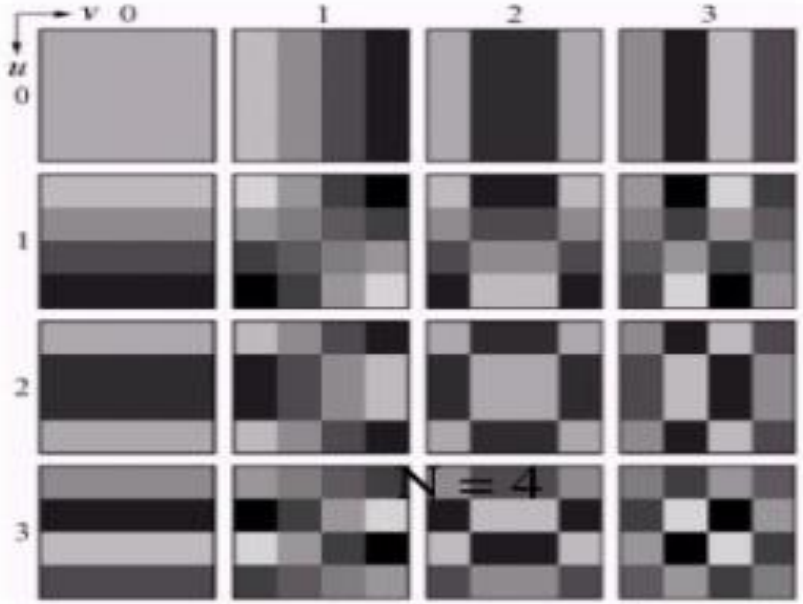
$$p_0(u) = b_{m-1}(u)$$
$$p_1(u) = b_{m-1}(u) + b_{m-2}(u)$$
$$p_2(u) = b_{m-2}(u) + b_{m-3}(u)$$
$$\cdots$$
$$p_{m-1}(u) = b_1(u) + b_0(u)$$

Advantage: simple, easy to implement
Disadvantage: not good packing ability

Image Compression

Transform Example: Discrete Cosine Basis Functions

$$g(x,y,u,v) = h(u,v,x,y) = \alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$



N = 4

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } u = 0 \\ \sqrt{\frac{2}{N}} & \text{for } u = 1,\ldots,N-1 \end{cases}$$

DCT is one of the most frequently used transform for image compression. For example, DCT is used in JPG files.

Advantage: good packing ability, modulate computational complexity

Image Compression

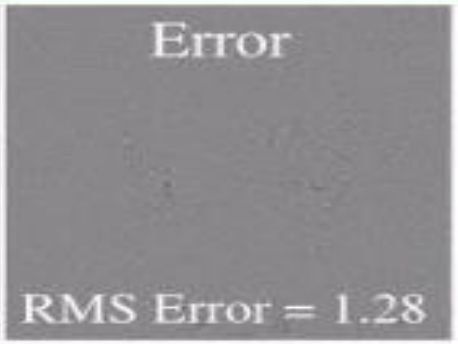
Transform Coding Examples

Original image
512x512 pixels

Subimage size:
8x8 pixels = 64 pixels

Quatization by truncating
50% of coefficients (only
32 max cofficients are kept.)

(Images from Rafael C. Gonzalez and Richard E. Wood, Digital Image Processing, 2nd Edition.



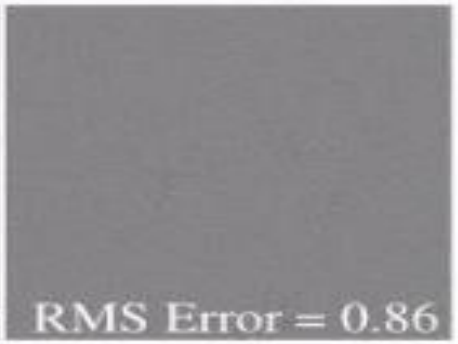


Image Compression

DCT vs DFT Coding

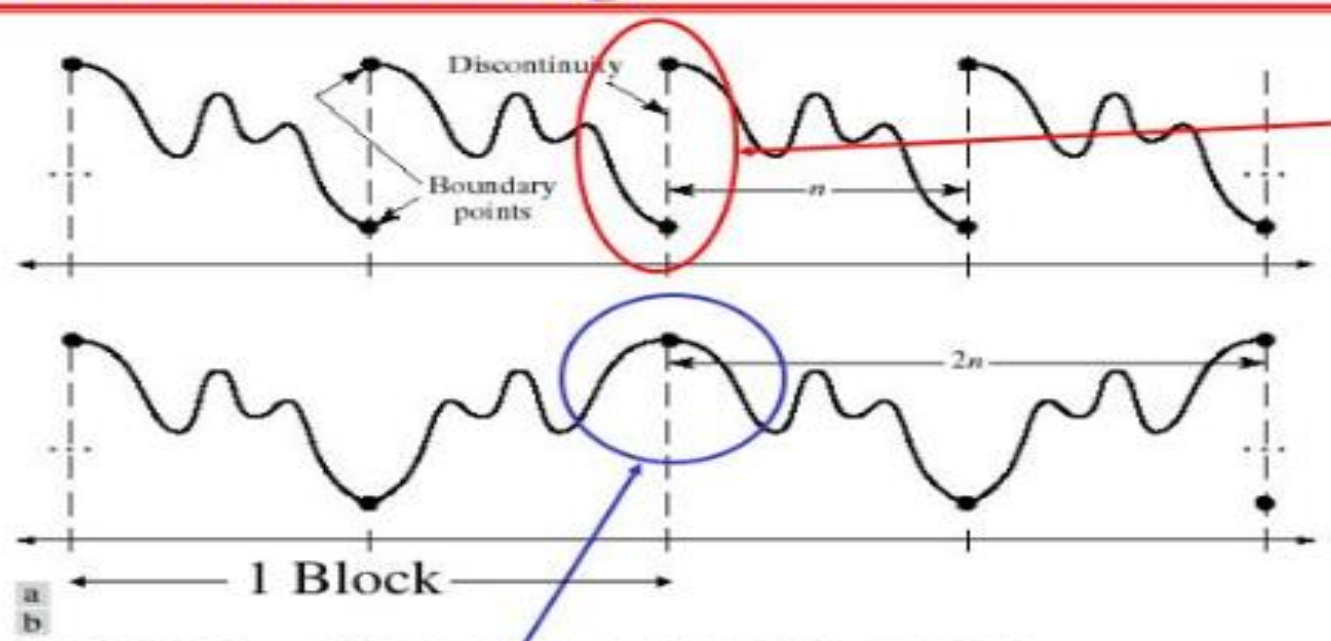


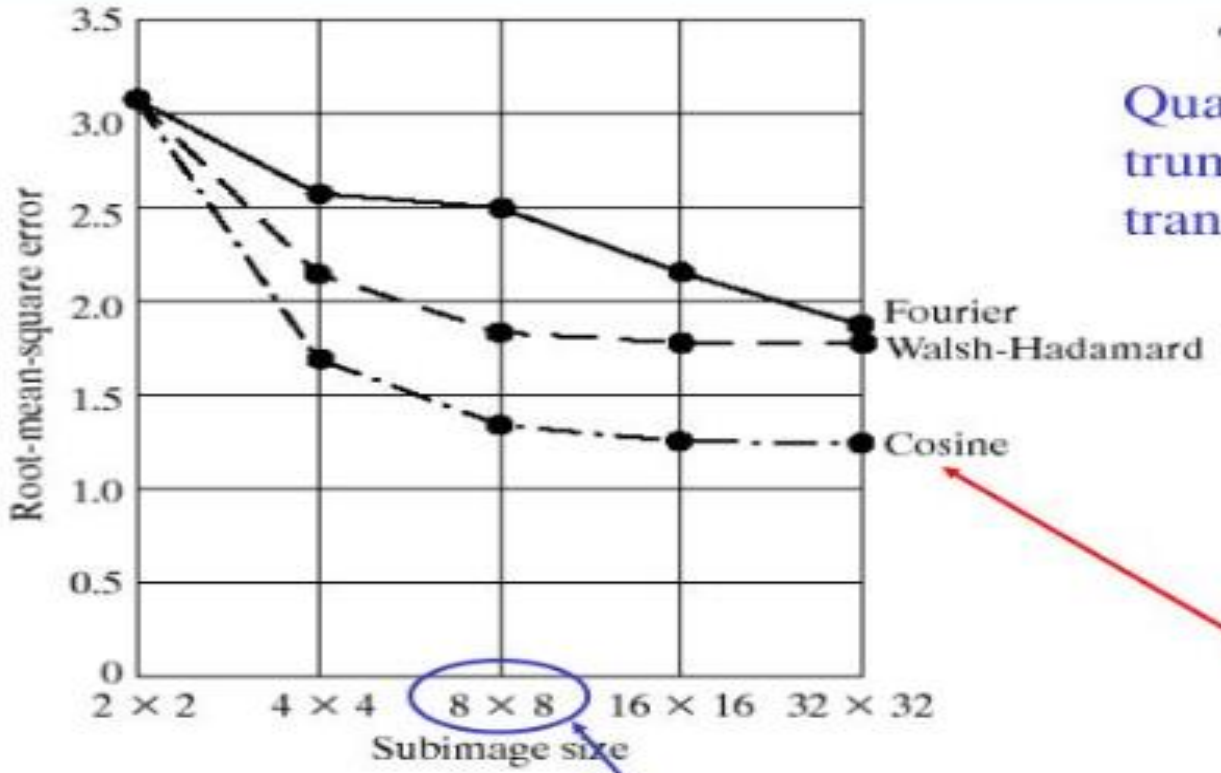
FIGURE 8.32 The periodicity implicit in the 1-D (a) DFT and (b) DCT.

DFT coefficients have abrupt changes at boundaries of blocks

Advantage of DCT over DFT is that the DCT coefficients are more continuous at boundaries of blocks.

Image Compression

Subimage Size and Transform Coding Performance



This experiment: Quatization is made by truncating 75% of transform coefficients

DCT is the best

Size 8x8 is enough

Image Compression

Subimage Size and Transform Coding Performance

Reconstructed by using 25% of coefficients ($C_R = 4:1$) with 8x8 sub-images

DCT Coefficients

Zoomed detail Original

Zoomed detail Subimage size: 2x2 pixels

Zoomed detail Subimage size: 4x4 pixels

Zoomed detail Subimage size: 8x8 pixels

Image Compression

Quantization Process: Bit Allocation

To assign different numbers of bits to represent transform coefficients based on importance of each coefficient:

- More importance coefficeitns → assign a large number of bits

- Less importance coefficients → assign a small number of bits or not assign at all

2 Popular bit allocation methods

1. Zonal coding : allocate bits based on the basis of maximum variance, using fixed mask for all subimages

2. Threshold coding : allocate bits based on maximum magnitudes of coefficients

Image Compression

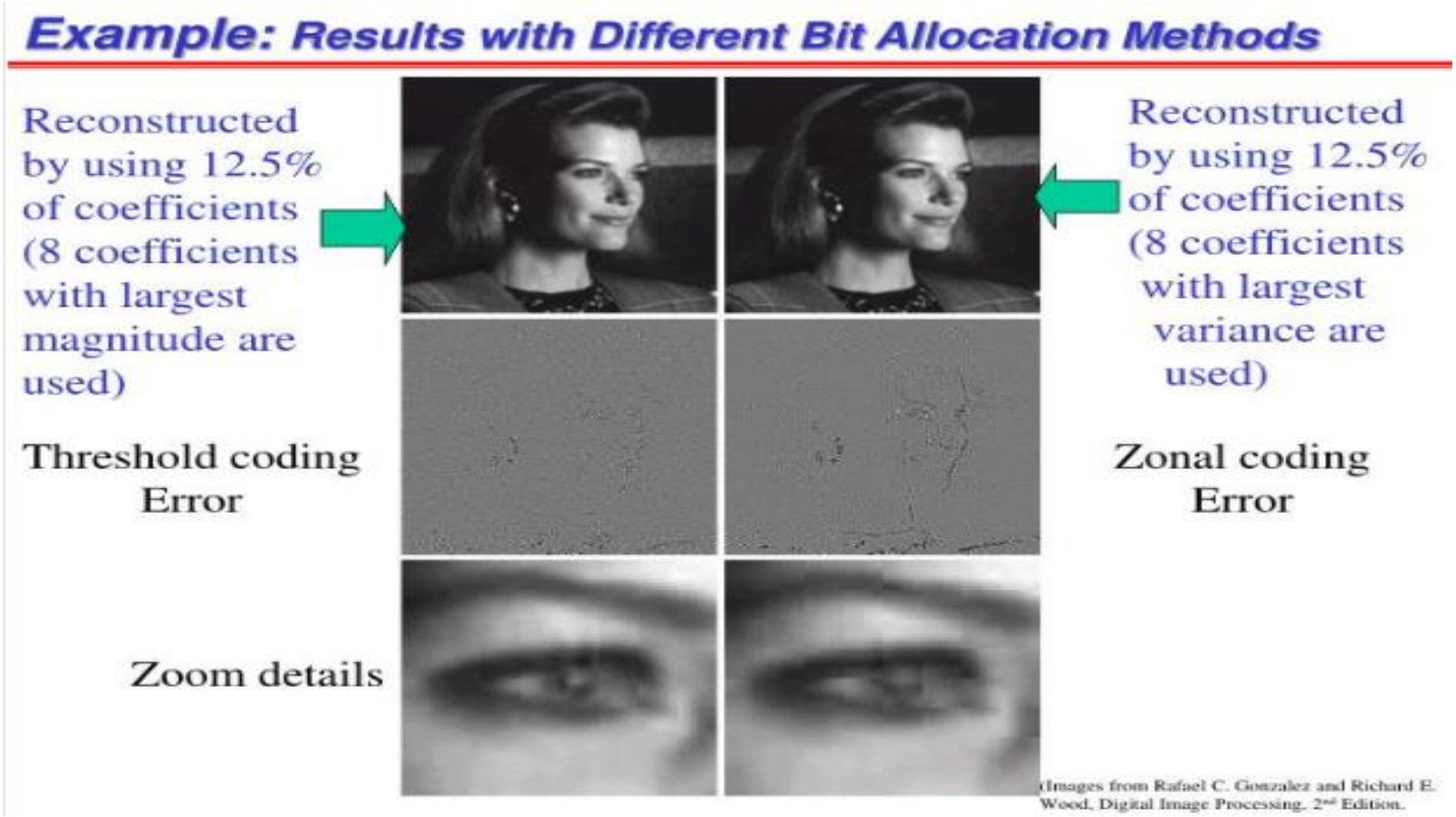


Image Compression

a b
c d

FIGURE 8.36 A typical (a) zonal mask, (b) zonal bit allocation, (c) threshold mask, and (d) thresholded coefficient ordering sequence. Shading highlights the coefficients that are retained.

1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

8	7	6	4	3	2	1	0
7	6	5	4	3	2	1	0
6	5	4	3	3	1	1	0
4	4	3	3	2	1	0	0
3	3	3	2	1	1	0	0
2	2	1	1	1	0	0	0
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

1	1	0	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Image Compression

Thresholding Coding Quantization

3 Popular Thresholding Methods

Method 1: Global thresholding : Use a single global threshold value for all subimages

Method 2: N-largest coding: Keep only N largest coefficients

Method 3: Normalized thresholding: each subimage is normalized by a normalization matrix before rounding

Bit allocation

$$\hat{T}(u,v) = round\left[\frac{T(u,v)}{Z(u,v)}\right]$$

Restoration before decompressing

$$\tilde{T}(u,v) = \hat{T}(u,v)Z(u,v)$$

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Example of Normalization Matrix $Z(u,v)$

(Images from Rafael C. Gonzalez and Richard E. Wood, Digital Image Processing, 2nd Edition.

Image Compression

DCT Coding Example

($C_R = 38:1$)

($C_R = 67:1$)

Error image
RMS Error = 3.42

Method:
- Normalized Thresholding,
- Subimage size: 8x8 pixels

Zoom details

Blocking Artifact at Subimage boundaries

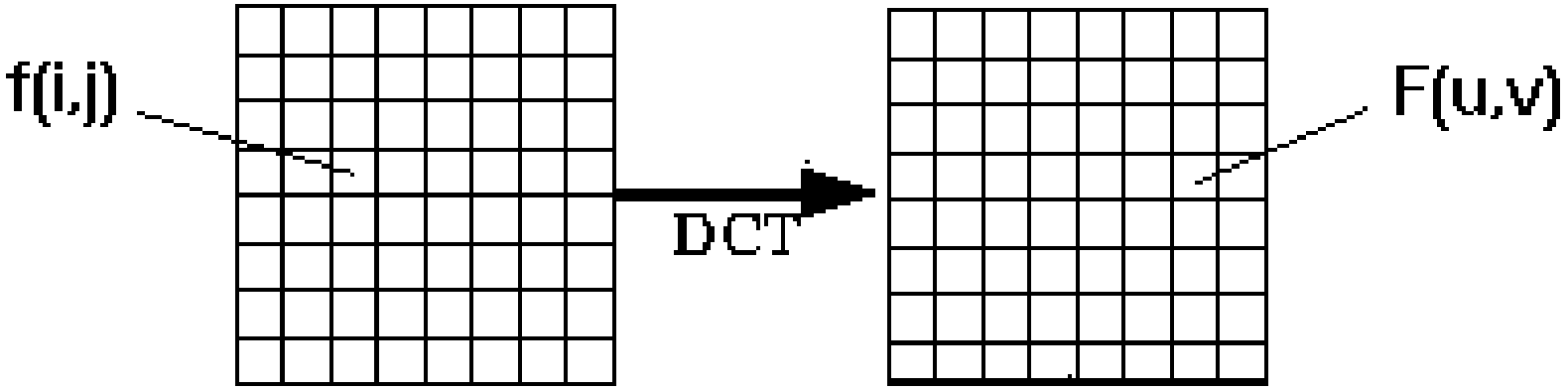
(Images from Rafael C. Gonzalez and Richard E. Wood, Digital Image Processing, 2nd Edition.

Relation between DCT and Fourier Transform

DCT (Discrete Cosine Transform) is a simplified version of the Fourier Transform

- It is the real portion of Fourier Transform.
- It computationally easy.
- It is effective for multimedia signal processing than Fourier transform as this only deals with the real term.

DCT Calculation



$$1-D\ DCT\ Calculation: F(u) = a(u)\left(\frac{2}{N}\right)^{0.5}\sum_{i=0}^{N-1}\text{Acos}\left[\frac{\pi.u}{2N}(2i+1)\right]f(i)$$

$$2-D\ DCT\ Calculation: F(u,v) = a(u)a(v)\left(\frac{2}{M}\right)^{0.5}\left(\frac{2}{N}\right)^{0.5}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\cos\left[\frac{\pi u}{2N}(2i+1)\right]\cos\left[\frac{\pi v}{2M}(2j+1)\right]f(i,j)$$

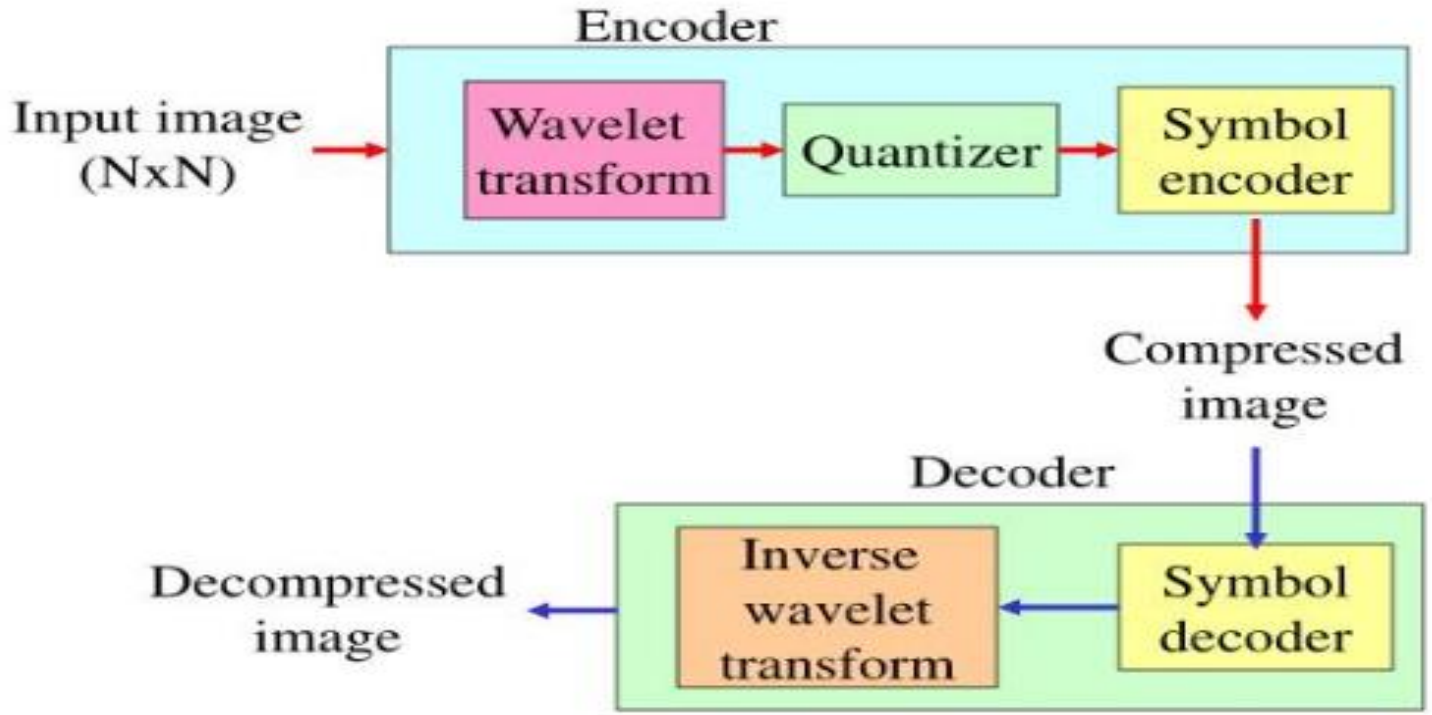
$$where\quad a(u) = \begin{matrix}\frac{1}{\sqrt{2}} & for\ u=0\\ 1 & for\ otherwise\end{matrix}\quad and\quad a(v) = \begin{matrix}\frac{1}{\sqrt{2}} & for\ v=0\\ 1 & for\ otherwise\end{matrix}$$

DCT Calculation

- The input image is divided into $N \times M$ blocks.
- $f(i,j)$ is the intensity of the i^{th} row and j^{th} column pixel in one of the $N \times M$ blocks.
- $F(u,v)$ is the coefficient of the DCT matrix.
- The values of the $N \times M$ block are shifted from a positive range to one centered on zero before computing the DCT. Each entry in the original block of an 8-bit image falls in the range $[0,255]$. The midpoint of the range 128 is subtracted from each entry to produce a data range centered on zero, so that the modified range is $[-128,127]$. This step reduces the dynamic range requirements for the subsequent DCT processing.
- After the intensity value is centered around zero, DCT matrix was calculated.

Image Compression

Wavelet Transform Coding: Multiresolution approach



Unlike DFT and DCT, Wavelet transform is a multiresolution transform.

Image Compression

a b
c d
e f

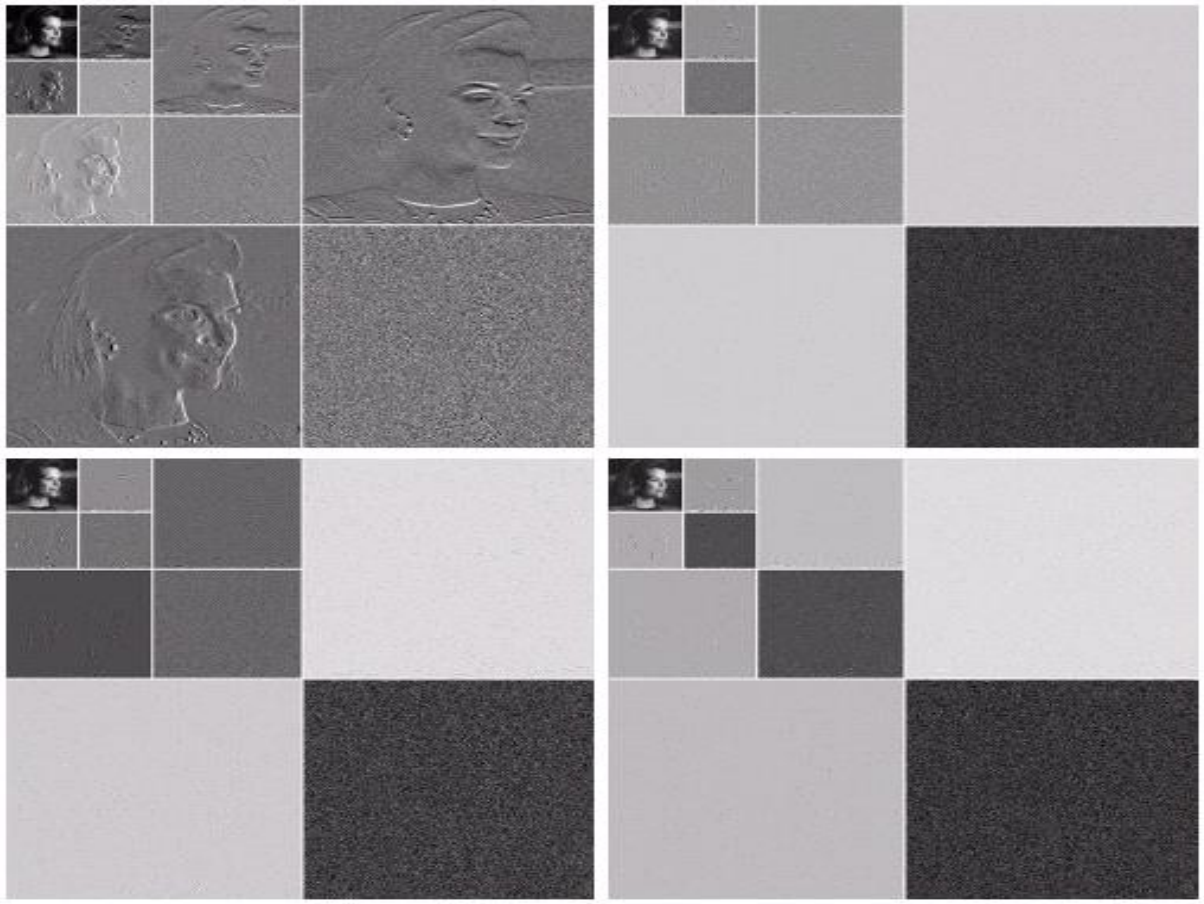
FIGURE 8.40 (a), (c), and (e) Wavelet coding results comparable to the transform-based results in Figs. 8.38(a), (c), and (e); (b), (d), and (f) similar results for Figs. 8.38(b), (d), and (f).

Image Compression

a b
c d
e f

FIGURE 8.41 (a), (c), and (e) Wavelet coding results with a compression ratio of 108 to 1; (b), (d), and (f) similar results for a compression of 167 to 1.

Image Compression



a b
c d

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.

Image Compression

Wavelet	Filter Taps (Scaling + Wavelet)	Zeroed Coefficients
Haar (see Ex. 7.10)	2 + 2	46%
Daubechies (see Fig. 7.6)	8 + 8	51%
Symlet (see Fig. 7.24)	8 + 8	51%
Biorthogonal (see Fig. 7.37)	17 + 11	55%

TABLE 8.12 Wavelet transform filter taps and zeroed coefficients when truncating the transforms in Fig. 8.42 below 1.5.

Scales and Filter Bank Iterations	Approximation Coefficient Image	Truncated Coefficients (%)	Reconstruction Error (rms)
1	256 × 256	75%	1.93
2	128 × 128	93%	2.69
3	64 × 64	97%	3.12
4	32 × 32	98%	3.25
5	16 × 16	98%	3.27

TABLE 8.13 Decomposition level impact on wavelet coding the 512 × 512 image of Fig. 8.23.

Image Compression

FIGURE 8.43 The impact of dead zone interval selection on wavelet coding.

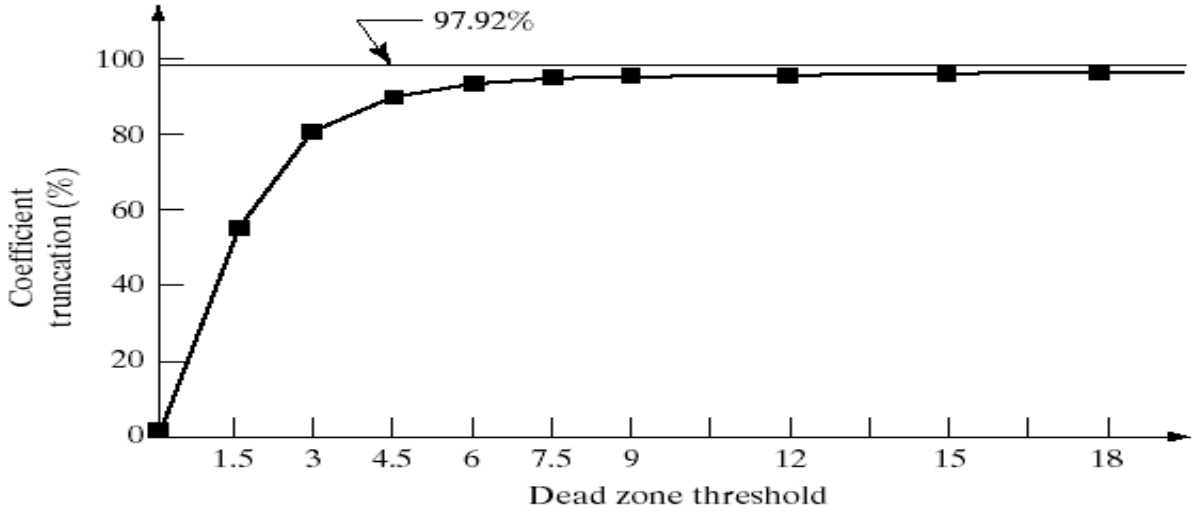


Image Compression

TABLE 8.17 JPEG coefficient coding categories.

Range	DC Difference Category	AC Category
0	0	N/A
−1, 1	1	1
−3, −2, 2, 3	2	2
−7, . . . , −4, 4, . . . , 7	3	3
−15, . . . , −8, 8, . . . , 15	4	4
−31, . . . , −16, 16, . . . , 31	5	5
−63, . . . , −32, 32, . . . , 63	6	6
−127, . . . , −64, 64, . . . , 127	7	7
−255, . . . , −128, 128, . . . , 255	8	8
−511, . . . , −256, 256, . . . , 511	9	9
−1023, . . . , −512, 512, . . . , 1023	A	A
−2047, . . . , −1024, 1024, . . . , 2047	B	B
−4095, . . . , −2048, 2048, . . . , 4095	C	C
−8191, . . . , −4096, 4096, . . . , 8191	D	D
−16383, . . . , −8192, 8192, . . . , 16383	E	E
−32767, . . . , −16384, 16384, . . . , 32767	F	N/A

Image Compression

TABLE 8.18
JPEG default DC code (luminance).

Category	Base Code	Length	Category	Base Code	Length
0	010	3	6	1110	10
1	011	4	7	11110	12
2	100	5	8	111110	14
3	00	5	9	1111110	16
4	101	7	A	11111110	18
5	110	8	B	111111110	20

Image Compression

Run/ Category	Base Code	Length	Run/ Category	Base Code	Length
0/0	**1010 (= EOB)**	**4**			
0/1	00	3	8/1	11111010	9
0/2	01	4	8/2	111111111000000	17
0/3	100	6	8/3	1111111110110111	19
0/4	1011	8	8/4	1111111110111000	20
0/5	11010	10	8/5	1111111110111001	21
0/6	111000	12	8/6	1111111110111010	22
0/7	1111000	14	8/7	1111111110111011	23
0/8	1111110110	18	8/8	1111111110111100	24
0/9	1111111110000010	25	8/9	1111111110111101	25
0/A	1111111110000011	26	8/A	1111111110111110	26
1/1	1100	5	9/1	111111000	10
1/2	111001	8	9/2	1111111110111111	18
1/3	1111001	10	9/3	1111111111000000	19
1/4	111110110	13	9/4	1111111111000001	20
1/5	11111110110	16	9/5	1111111111000010	21
1/6	1111111110000100	22	9/6	1111111111000011	22
1/7	1111111110000101	23	9/7	1111111111000100	23
1/8	1111111110000110	24	9/8	1111111111000101	24
1/9	1111111110000111	25	9/9	1111111111000110	25
1/A	1111111110001000	26	9/A	1111111111000111	26
2/1	11011	6	A/1	111111001	10
2/2	11111000	10	A/2	1111111111001000	18
2/3	1111110111	13	A/3	1111111111001001	19
2/4	1111111110001001	20	A/4	1111111111001010	20
2/5	1111111110001010	21	A/5	1111111111001011	21
2/6	1111111110001011	22	A/6	1111111111001100	22
2/7	1111111110001100	23	A/7	1111111111001101	23

TABLE 8.19
JPEG default AC code (luminance) *(continues on next page)*.

Image Compression

Table 8.19 (Contd.)

2/8	1111111110001101	24	A/8	1111111111001110	24
2/9	1111111110001110	25	A/9	1111111111001111	25
2/A	1111111110001111	26	A/A	1111111111010000	26
3/1	111010	7	B/1	111111010	10
3/2	111110111	11	B/2	1111111111010001	18
3/3	11111110111	14	B/3	1111111111010010	19
3/4	1111111110010000	20	B/4	1111111111010011	20
3/5	1111111110010001	21	B/5	1111111111010100	21
3/6	1111111110010010	22	B/6	1111111111010101	22
3/7	1111111110010011	23	B/7	1111111111010110	23
3/8	1111111110010100	24	B/8	1111111111010111	24
3/9	1111111110010101	25	B/9	1111111111011000	25
3/A	1111111110010110	26	B/A	1111111111011001	26
4/1	111011	7	C/1	1111111010	11
4/2	1111111000	12	C/2	1111111111011010	18
4/3	1111111110010111	19	C/3	1111111111011011	19
4/4	1111111110011000	20	C/4	1111111111011100	20
4/5	1111111110011001	21	C/5	1111111111011101	21
4/6	1111111110011010	22	C/6	1111111111011110	22
4/7	1111111110011011	23	C/7	1111111111011111	23
4/8	1111111110011100	24	C/8	1111111111100000	24
4/9	1111111110011101	25	C/9	1111111111100001	25
4/A	1111111110011110	26	C/A	1111111111100010	26

Image Compression

Filter Tap	Highpass Wavelet Coefficient	Lowpass Scaling Coefficient
0	−1.115087052456994	0.6029490182363579
±1	0.5912717631142470	0.2668641184428723
±2	0.05754352622849957	−0.07822326652898785
±3	−0.09127176311424948	−0.01686411844287495
±4	0	0.02674875741080976

TABLE 8.20 Impulse responses of the low and highpass analysis filters for an irreversible 9-7 wavelet transform.

Image Compression

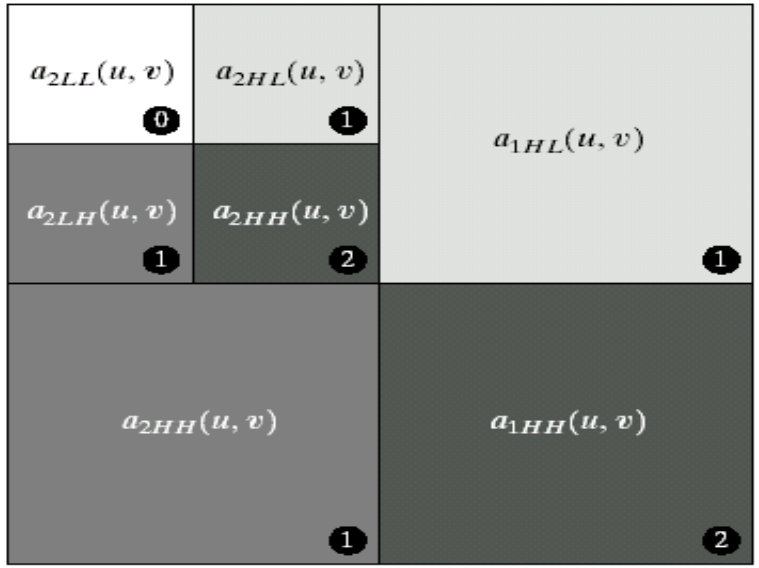
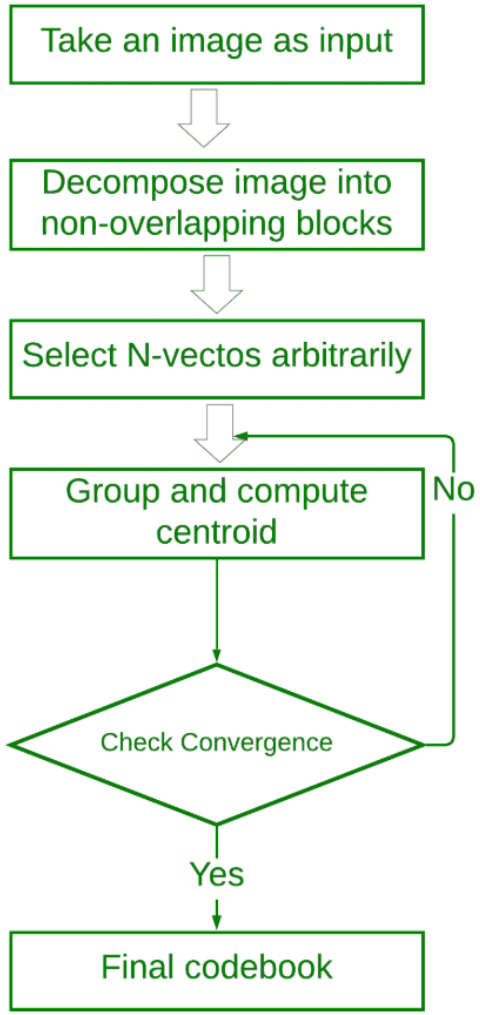


FIGURE 8.46 JPEG 2000 two-scale wavelet transform tile-component coefficient notation and analysis gain.

Image Compression-Vector Quantization

- Vector quantization (VQ) is a classical quantization technique from signal processing that allows the modeling of probability density functions by the distribution of prototype vectors.
- The vector quantization method that has gained the most popularity was proposed by Linde, Buzo, and Gray. The algorithm was given the acronym LBG algorithm as a result.
- The LBG algorithm can be considered as generalized Lloyd scalar algorithm. Consequently, it is also known as the generalized Lloyd algorithm (GLA).

Flowchart of LBG Algorithm



LBG Algorithm

- The image is divided into non-overlapping equal sized blocks and then processed the image using these blocks. Let the size of every block is $n \times n$.
- Vector quantization process converts the blocks into one dimensional vector. The size of every image vector is $n \times n$.
- Initially, Nc image vectors were chosen at random to serve as codewords in a codebook of size Nc. The size of every codeword is similar to image vector.
- Based on the Euclidian distance between the image vector and the codewords in the codebook, map the image vectors $\{I_1, I_2, \ldots, I_M\}$. The codeword $\{C_1, C_2, \ldots, C_{Nc}\}$ with the lowest Euclidian distance is the one to which the image vector belongs.

$$FlagCluster_{mi} = \begin{matrix} TRUE & if\ difference(C_i, I_m) < difference\left(C_j, I_m\right)_{j \in 1 \ldots N_c | j \neq 1} \\ FALSE & otherwise \end{matrix}$$

The m^{th} row and i^{th} column of the $(M \times N_C)$ Boolean matrix *FlagCluster* are denoted as $FlagCluster_{mi}$. This position will be true when m^{th} image vector belongs to i^{th} cluster as the Euclidian distance between m^{th} image vector and i^{th} codeword is the minimum among all other codewords.

- The codewords are updated by computing the centroid of the new clusters.

$$C_i' = \frac{\sum_{m=1}^{M} I_m \times FlagCluster_{mi}}{Count\ of\ image\ vector\ in\ i^{th} cluster.}$$

- The algorithm will terminate when the overall mean square error between image vector and their respective codeword of present iteration is smaller than the mean square error of next iteration. Otherwise, the algorithm repeats the process from step 2.

Image and Image Vectors

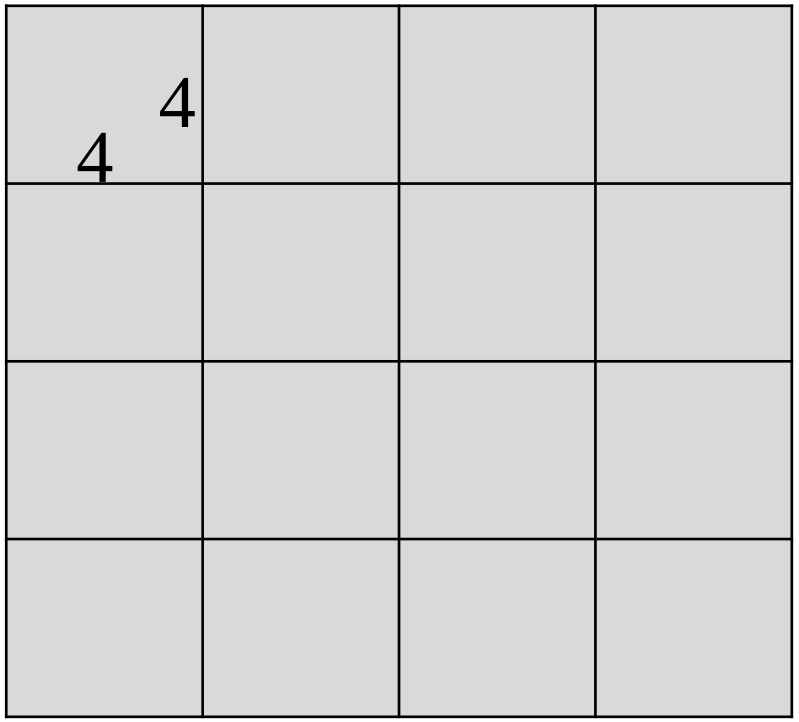


Image divided into blocks

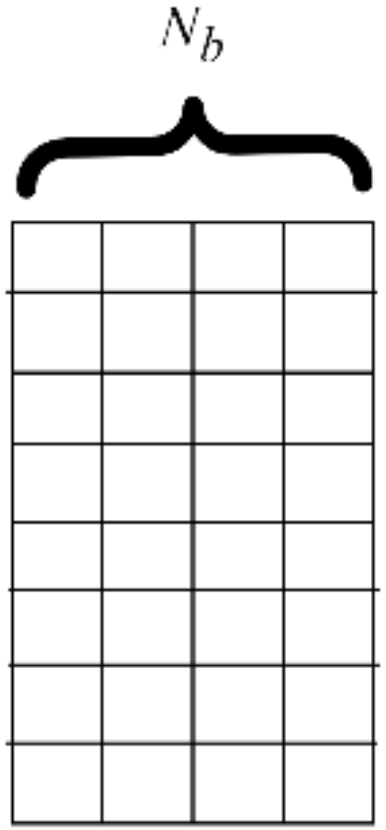
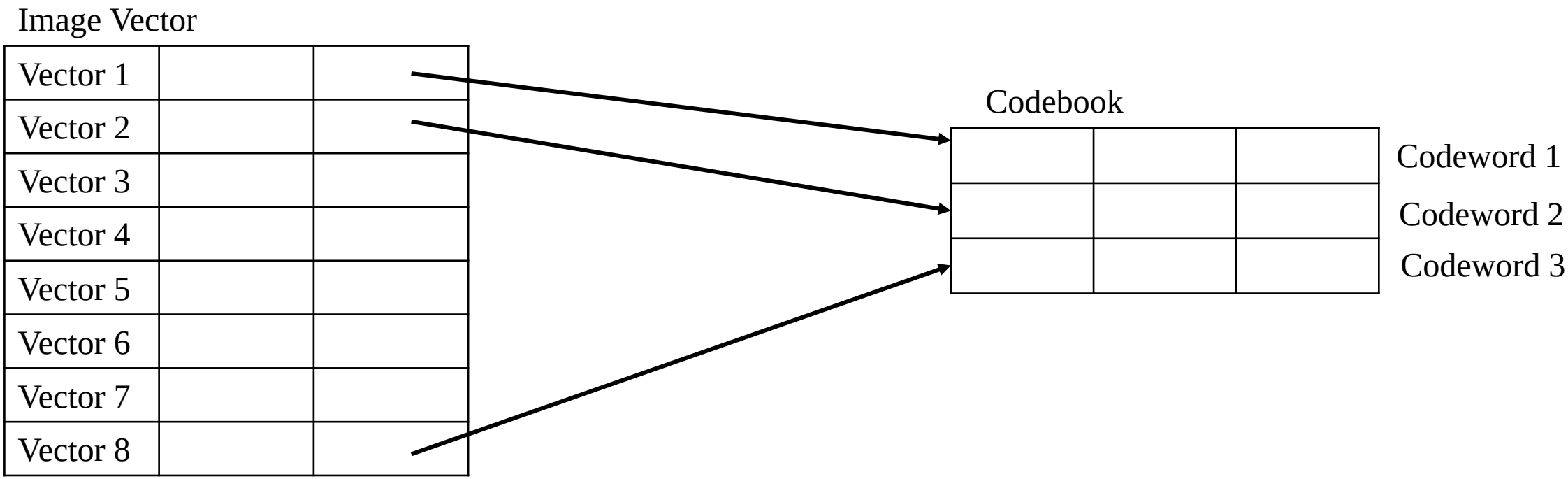


Image converted into vectors

$$N_b = 4 \times 4 = 16$$

Generation of codebook



Assign the image vector to codewords.

$$flag_{mi} = \begin{matrix} true \\ false \end{matrix} \quad \begin{matrix} if\ |C_i - I_m| < |C_{j \in 1,..,N_c : j \neq i} - I_m| \\ otherwise \end{matrix}$$

Calculate the new centroid

$$C_i' = \frac{\sum_{m=1}^{M} I_m * flag_{mi}}{Number\ of\ vectors\ in\ the\ cluster}$$

End of Lecture on Image Compression

Thank You