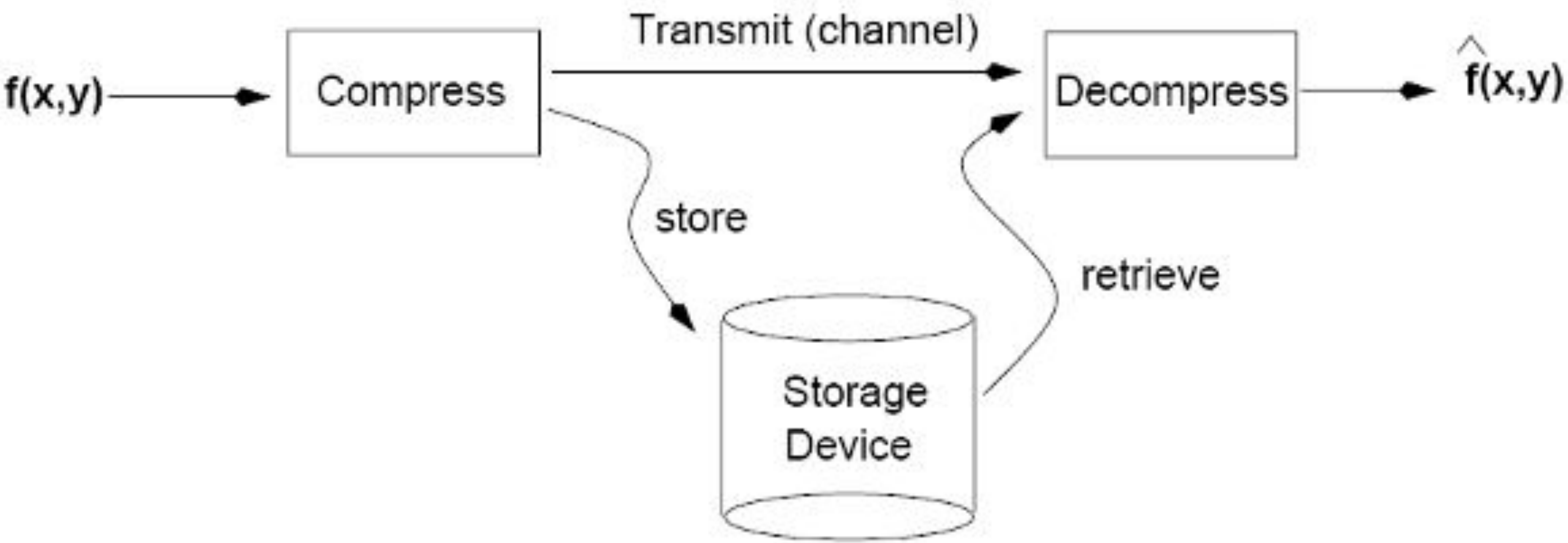
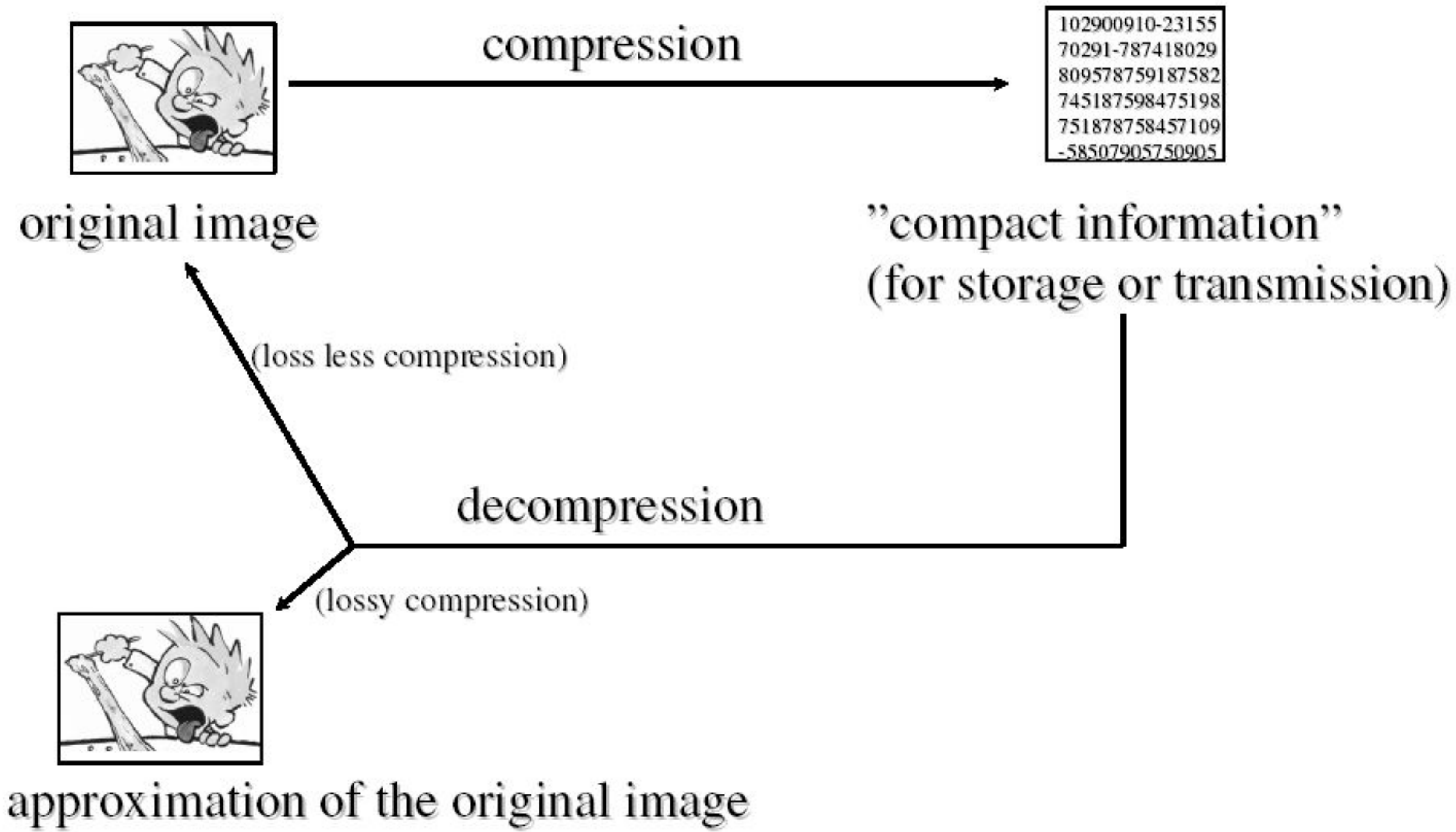
Image Compression

- Image Compression is the art and science of reducing amount data required to represent an image.
- Digital images require huge amounts of space for storage and large bandwidths for transmission.
  - A 640 x 480 color image requires close to 1MB of space.
- The goal of image compression is to reduce the amount of data required to represent a digital image.
  - Reduce storage requirements and increase transmission rates.


f(x,y)
Compress
Transmit (channel)
Decompress
$\hat{f}(x,y)$
store
retrieve
Storage
Device


compression
102900910-23155
70291-787418029
809578759187582
745187598475198
751878758457109
-58507905750905
original image
"compact information"
(for storage or transmission)
(loss less compression)
decompression
(lossy compression)
approximation of the original image

Lossless

- Information preserving
- Low compression ratios

Lossy

- Not information preserving
- High compression ratios

Data and 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.

- The same amount of information can be represented by various amount of data, e.g.:

Ex1: *Your wife, Helen, will meet you at Logan Airport in Boston at 5 minutes past 6:00 pm tomorrow night*

Ex2: *Your wife will meet you at Logan Airport at 5 minutes past 6:00 pm tomorrow night*

Ex3: *Helen will meet you at Logan at 6:00 pm tomorrow night*

Data redundancy and compression ratio.

Relative data redundancy R

$R = 1 - 1/C$ where C commonly called the compression ratio is defined as

$C = b/b'$ where b and b' denote the number of bits in two representations of the same information

If $C = 10$, for instance, means larger representation has 10 bits of data for every 1 bit of data in the smaller representation

Corresponding relative data redundancy of the larger representation is 0.9 indicating 90% of its data is redundant

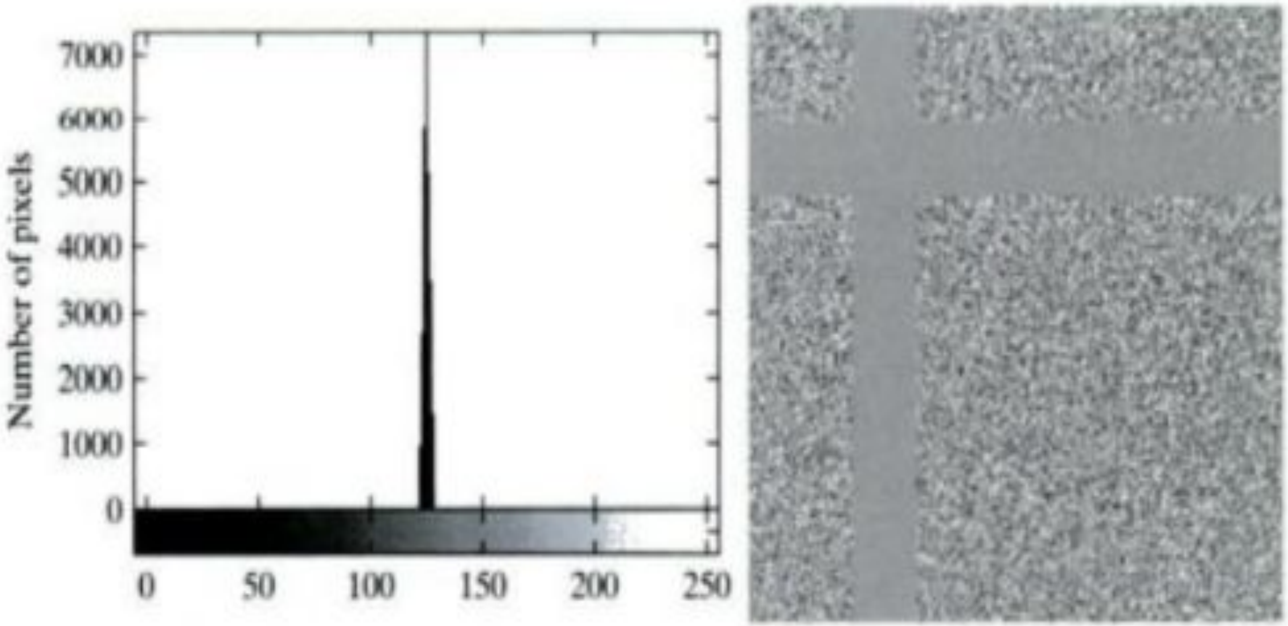
Types of data redundancy

1. Coding redundancy
2. Spatial redundancy
3. Irrelevant information

a b c

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

Irrelevant Information



a b

FIGURE 8.3
(a) Histogram of the image in Fig. 8.1(c) and (b) a histogram equalized version of the image.

1) Coding Redundancy: A code is a system of symbols (letters, numbers, bits) used to represent a body of information or set of events.

Each piece of information or event is assigned a sequence of code symbols, called a code word.

Number of symbols in each code word is its length

2) Spatial and temporal redundancy: Pixels of most 2-D intensity arrays are correlated spatially (i.e. each pixel is similar to or dependent on neighboring pixels.In a video sequence, temporally correlated pixels also duplicate information

3) Irrelevant information: Most 2-D intensity arrays contain information that is ignored by human visual system and/or extraneous to the intended use of the image. It is redundant in the sense that it is not used

Coding Redundancy

Let $0 \le r_k \le 1$: gray levels (discrete random variable)

$p_r(r_k)$: probability of occurrence of r_k

n_k : number of pixels that r_k appears in the image

n : total number of pixels in an image

L : number of gray levels

$l(r_k)$: number of bits used to represent r_k

L_{avg} : average length of code words assigned to the grey levels

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k) \quad \text{where} \quad p_r(r_k) = \frac{n_k}{n}, k = 0,1,\ldots\ldots, L-1$$

Hence, total number of bits required to code an $M \times N$ image is MNL_{avg}

For a natural m-bit coding $L_{avg} = m$.

Examples of variable length encoding

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

$$L_{avg} = 0.25(2) + 0.47(1) + 0.25(3) + 0.03(3) = 1.81 \text{ bits}$$

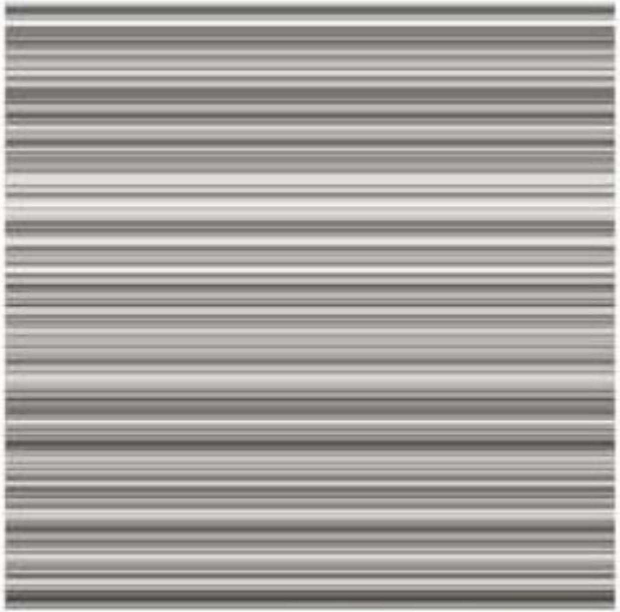
The total number of bits needed to represent the entire image is $MNL_{avg} = 256 \times 256 \times 1.81$ or 118,621. From Eqs. (8.1-2) and (8.1-1), the resulting compression and corresponding relative redundancy are

$$C = \frac{256 \times 256 \times 8}{118{,}621} = \frac{8}{1.81} \approx 4.42$$

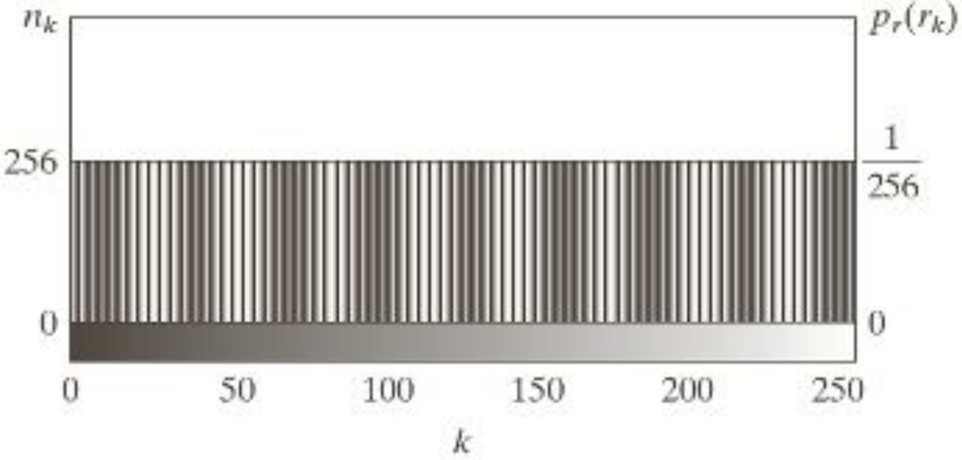
and

$$R = 1 - \frac{1}{4.42} = 0.774$$

Spatial redundancy



Spatial Redundancy



Run-length coding (RLC)
(interpixel redundancy)

- Used to reduce the size of a repeating string of characters (i.e., runs):

1 1 1 1 1 0 0 0 0 0 0 1 (1,5) (0, 6) (1, 1)

a a a b b b b b b c c (a,3) (b, 6) (c, 2)

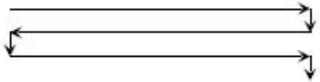
- Encodes a run of symbols into two bytes: **(symbol, count)**
- Can compress any type of data but cannot achieve high

Run-length coding

Every code word is made upp of a pair **(g,l)** where **g** is the graylevel and **l** is the number of pixels with that graylevel (length, or ”run”).

Ex 56 56 56 82 82 82 83 80
56 56 56 56 56 80 80 80

creates the runlength code (56,3) (82,3) (83,1) (80,4) (56,5)



-The code is calculated row by row.
-Very efficient coding for binary data.
-Important to know position, and the image dimensions must be stored with the coded image.

Measuring Image Information

A random event E with probability P(E) contains:

$I(E) = \log(1/P(E)) = -\log(P(E))$ units of information

If the base 2 is selected, the unit of information is the bit.

Note: $I(E)=0$ when $P(E)=1$

How much information does a pixel contain?

Suppose that gray level values are generated by a random variable, then rk contains:

$$I(r_k) = -\, log(P(r_k))$$

units of information!

Entropy of the image is defined as:

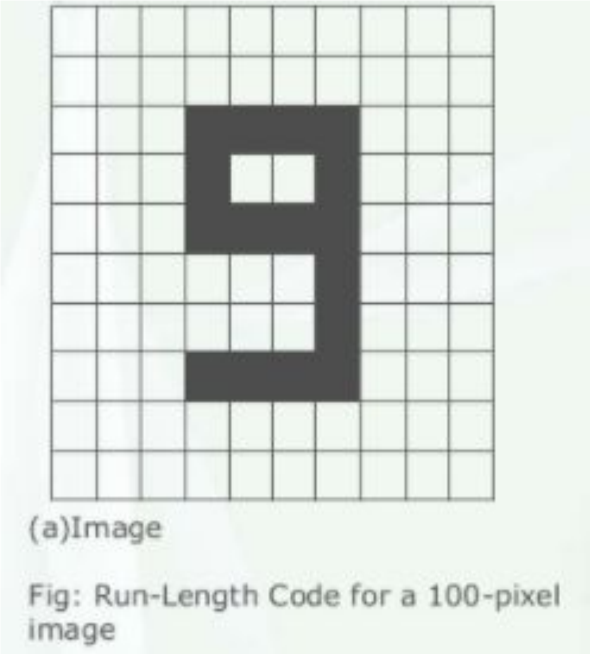
$$\tilde{H} = -\sum_{k=0}^{L-1} p_r(r_k) \log_2 p_r(r_k)$$

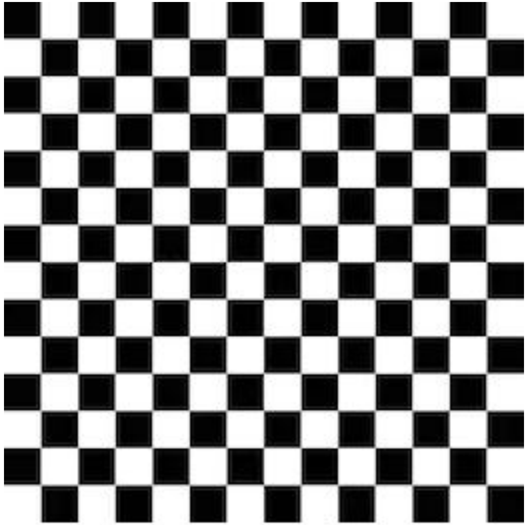
$$
\begin{aligned}
\tilde{H} &= -[0.25 \log_2 0.25 + 0.47 \log_2 0.47 + 0.25 \log_2 0.25 + 0.03 \log_2 0.03] \\
&\approx -[0.25(-2) + 0.47(-1.09) + 0.25(-2) + 0.03(-5.06)] \\
&\approx 1.6614 \text{ bits/pixel}
\end{aligned}
$$

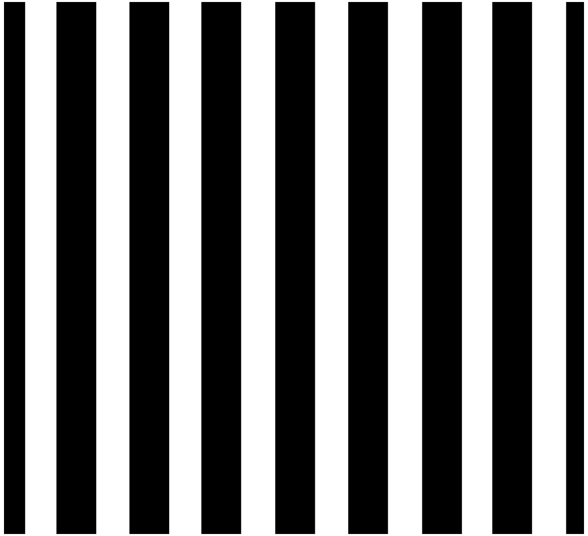
Can variable length coding procedure be used to compress a histogram equalized image?

Can such image contain spatial or temporal redundancies?

Can run length encoding be beneficial for every image?







Example:

Consider the simple 4×8, 8-bit image:

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

(a) Compute the entropy of the image.

(a) The entropy of the image is estimated using Eq. (8.1-7) to be

$$\begin{aligned}
\tilde{H} &= -\sum_{k=0}^{255} p_r(r_k)\log_2 p_r(r_k) \\
&= -\left[\frac{12}{32}\log_2\frac{12}{32}+\frac{4}{32}\log_2\frac{4}{32}+\frac{4}{32}\log_2\frac{4}{32}+\frac{12}{32}\log_2\frac{12}{32}\right] \\
&= -[-0.5306-0.375-0.375-0.5306] \\
&= 1.811 \text{ bits/pixel.}
\end{aligned}$$

Fidelity Criteria

Quantify the nature and extent of information loss.

Objective fidelity criteria:

Level of information loss can be expressed as a function of the original (input) and compressed-decompressed (output) image.

Given an $M \times N$ image $f(x,y)$ (original image), its compressed-then-decompressed image: $\hat{f}(x,y)$, then the error between corresponding values are given as:

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

Total error is given by:

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

Normally the objective fidelity criterion parameters are as follows:

e_{rms} (root-mean-square error):

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

SNR_{ms} (mean-square signal-to-noise ratio):

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

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.

TABLE 8.2
Rating scale of the Television Allocations Study Organization. (Frendendall and Behrend.)

a b c

FIGURE 8.4 Three approximations of the image in Fig. 8.1(a).

Consider an 8-pixel line of intensity data, $\{108, 139, 135, 244, 172, 173, 56, 99\}$. If it is uniformly quantized with 4-bit accuracy, compute the rms error and rms signal-to-noise ratios for the quantized data.

$f(x,y)$		$\hat{f}(x,y)$
Base 10	Base 2	Base 2
108	01101100	0110
139	10001011	1000

Table P8.3

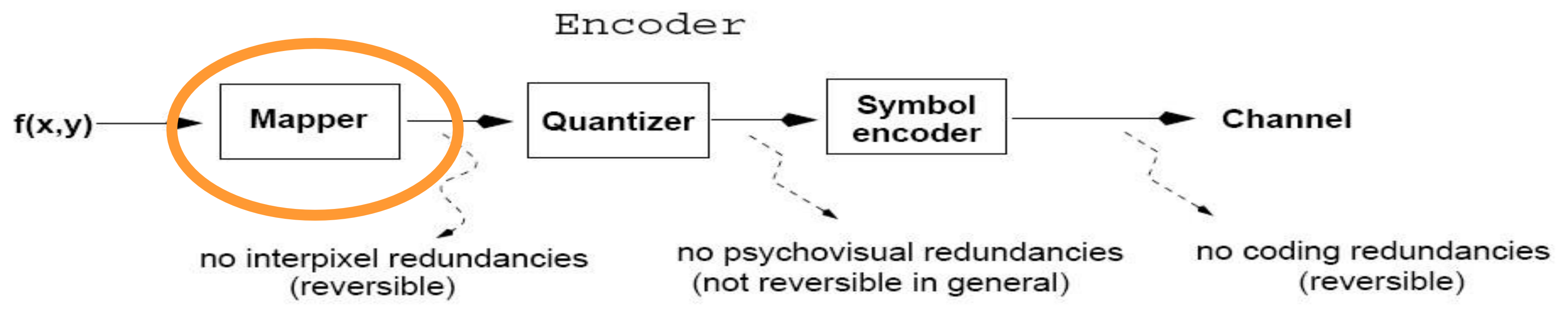
$f(x,y)$		$\hat{f}(x,y)$		$16\hat{f}(x,y)-f(x,y)$
Base 10	Base 2	Base 2	Base 10	Base 10
108	01101100	0110	6	-12
139	10001011	1000	8	-11
135	10000111	1000	8	-7
244	11110100	1111	15	-4
172	10101100	1010	10	-12
173	10101101	1010	10	-13
56	00111000	0011	3	-8
99	01100011	0110	6	-3

Using Eq. (8.1-10), the rms error is

$$
\begin{aligned}
e_{rms} &= \sqrt{\frac{1}{8}\sum_{x=0}^{0}\sum_{y=0}^{7}\left[16\hat{f}(x,y)-f(x,y)\right]^2} \\
&= \sqrt{\frac{1}{8}\left[(-12)^2+(-11)^2+(-7)^2+(-4)^2+(-12)^2+(-13)^2+(-8)^2+(-3)^2\right]} \\
&= \sqrt{\frac{1}{8}(716)} \\
&= 9.46
\end{aligned}
$$

$$
\begin{aligned}
SNR_{ms} &= \frac{\sum_{x=0}^{0}\sum_{y=0}^{7}\left[16\hat{f}\left(x,y\right)\right]^{2}}{\sum_{x=0}^{0}\sum_{y=0}^{7}\left[16\hat{f}\left(x,y\right)-f\left(x,y\right)\right]^{2}} \\
&= \frac{96^2+128^2+128^2+240^2+160^2+160^2+48^2+96^2}{716} \\
&= \frac{162304}{716} \\
&\simeq 227.
\end{aligned}
$$

Image Compression Model (cont'd)



Mapper: transforms input data in a way that facilitates reduction of interpixel redundancies (spatial and temporal redundancy)

- Example - Run length coding

Image Compression Model (cont'd)

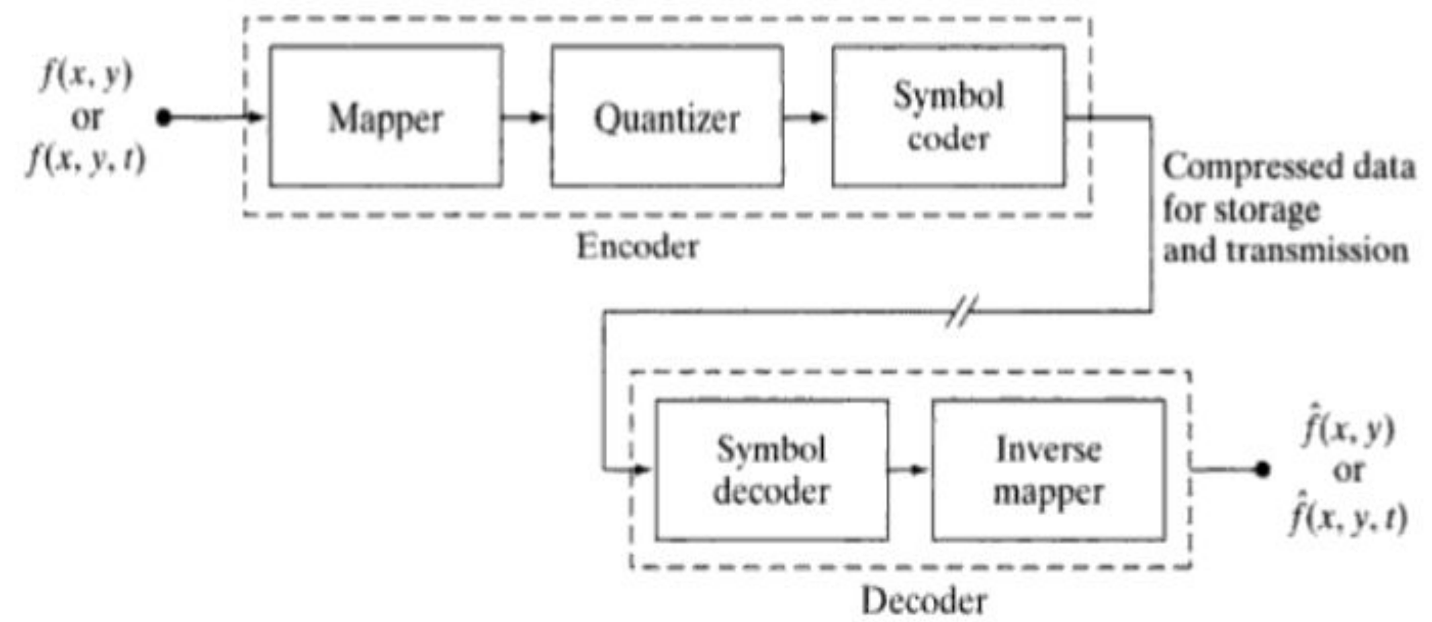
•**Quantizer:** reduces the accuracy of the mapper's output in accordance with some pre-established fidelity criteria.

- The goal is to keep irrelevant information out of the compressed representation.
- It is irreversible but it must be omitted when error free compression is desired.

Image Compression Model (cont'd)

- **Symbol encoder:** assigns the shortest code to the most frequently occurring output values – thus minimizing coding redundancy.

Image Compression Models (cont'd)



- Inverse operations are performed.

Huffman Coding (coding redundancy)

- A variable-length coding technique.
- Optimal code (i.e., minimizes the number of code symbols per source symbol).

Huffman Coding (cont'd)

- *Forward Pass*
  1. Sort probabilities per symbol
  2. Combine the lowest two probabilities
  3. Repeat *Step2* 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 (cont'd)

- ***Backward Pass***

  Assign code symbols going backwards

Original source			Source reduction							
Sym.	Prob.	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 (cont'd)

- L_{avg} using Huffman coding:

$$L_{avg} = E(l(a_k)) = \sum_{k=1}^{6} l(a_k)P(a_k) =$$

3x0.1 + 1x0.4 + 5x0.06 + 4x0.1 + 5x0.04 + 2x0.3=2.2 bits/symbol

- L_{avg} assuming binary codes:

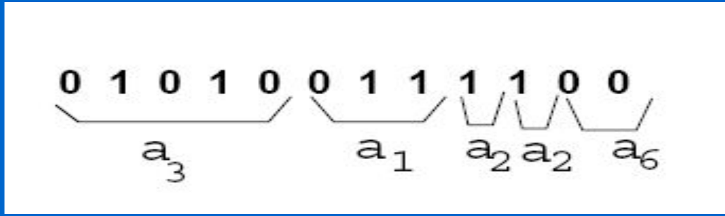
6 symbols, we need a 3-bit code

$(a_1: 000,\ a_2: 001,\ a_3: 010,\ a_4: 011,\ a_5: 100,\ a_6: 101)$

$$L_{avg} = \sum_{k=1}^{6} l(a_k)P(a_k) = \sum_{k=1}^{6} 3P(a_k) = 3\sum_{k=1}^{6} P(a_k) = 3 \text{ bits/symbol}$$

Huffman Coding/Decoding

- After the code has been created, *coding/decoding* can be implemented using a look-up table.
- Note that decoding is done unambiguously.



Original source		
Sym.	Prob.	Code
a_2	0.4	1
a_6	0.3	00
a_1	0.1	011
a_4	0.1	0100
a_3	0.06	01010
a_5	0.04	01011

Using the previously discussed Huffman decode the encoded string 0101000001010111110100

Original source		
Sym.	Prob.	Code
a_2	0.4	1
a_6	0.3	00
a_1	0.1	011
a_4	0.1	0100
a_3	0.06	01010
a_5	0.04	01011

a3 a6 a6 a2 a5 a2 a2 a2 a4.

Consider the simple 4×8, 8-bit image:

$$
\begin{array}{cccccccc}
21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\
21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\
21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\
21 & 21 & 21 & 95 & 169 & 243 & 243 & 243
\end{array}
$$

(b) Compress the image using Huffman coding.

(c) Compute the compression achieved and the effectiveness of the Huffman coding.

21	3/8	3/8	5/8
243	3/8	3/8	3/8
95	1/8	2/8	
169	1/8		

Source reductions

21	1	1	0
243	00	00	1
95	000	01	
169	001		

Code assignments

(c) Using Eq. (8.1-4), the average number of bits required to represent each pixel in the Huffman coded image (ignoring the storage of the code itself) is

$$L_{avg} = 1\left(\frac{3}{8}\right) + 2\left(\frac{3}{8}\right) + 3\left(\frac{1}{8}\right) + 3\left(\frac{1}{8}\right) = \frac{15}{8} = 1.875 \text{ bits/pixel.}$$

Thus, the compression achieved is

$$C = \frac{8}{1.875} = 4.27.$$

Because the theoretical compression resulting from the elimination of all coding redundancy is $\frac{8}{1.811} = 4.417$, the Huffman coded image achieves $\frac{4.27}{4.417} \times 100$ or 96.67% of the maximum compression possible through the removal of coding redundancy alone.

Arithmetic Coding (cont'd)

Encode message: $a_1\ a_2\ a_3\ a_3\ a_4$

Source Symbol	Probability
a_1	0.2
a_2	0.2
a_3	0.4
a_4	0.2

1) Assume message occupies [0, 1)

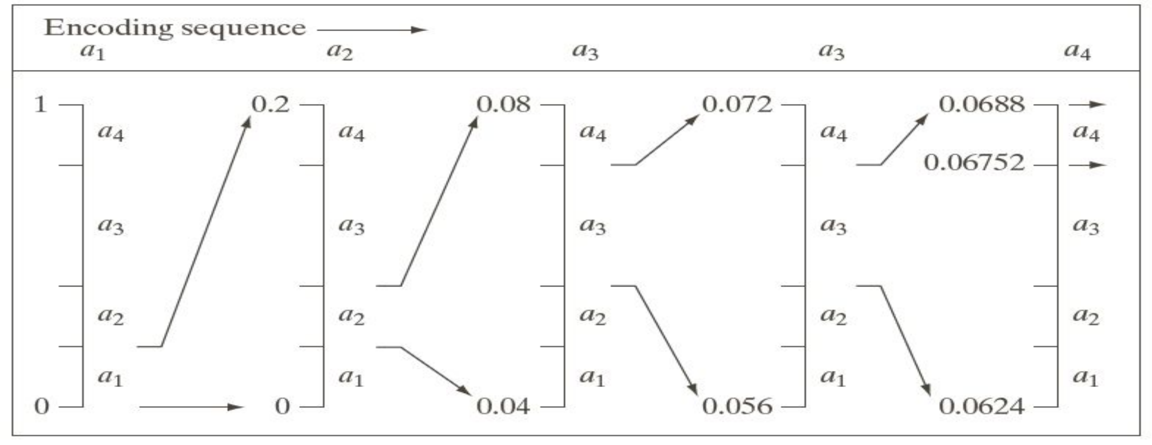
0 1

2) Subdivide [0, 1) based on the probability of α_i

Initial Subinterval
[0.0, 0.2)
[0.2, 0.4)
[0.4, 0.8)
[0.8, 1.0)

Example

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)



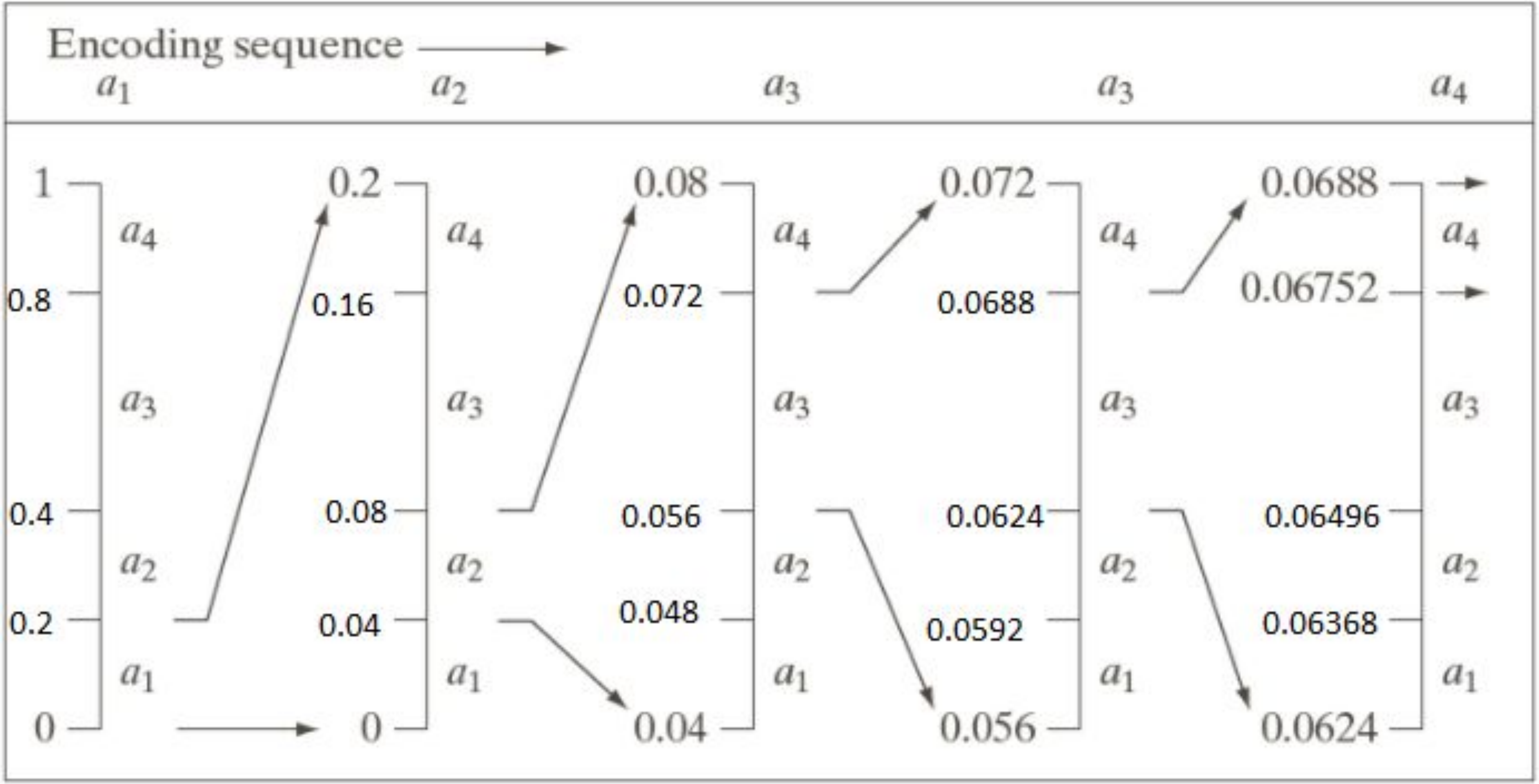
Encode

$a_1\ a_2\ a_3\ a_3\ a_4$

[0.06752, 0.0688)

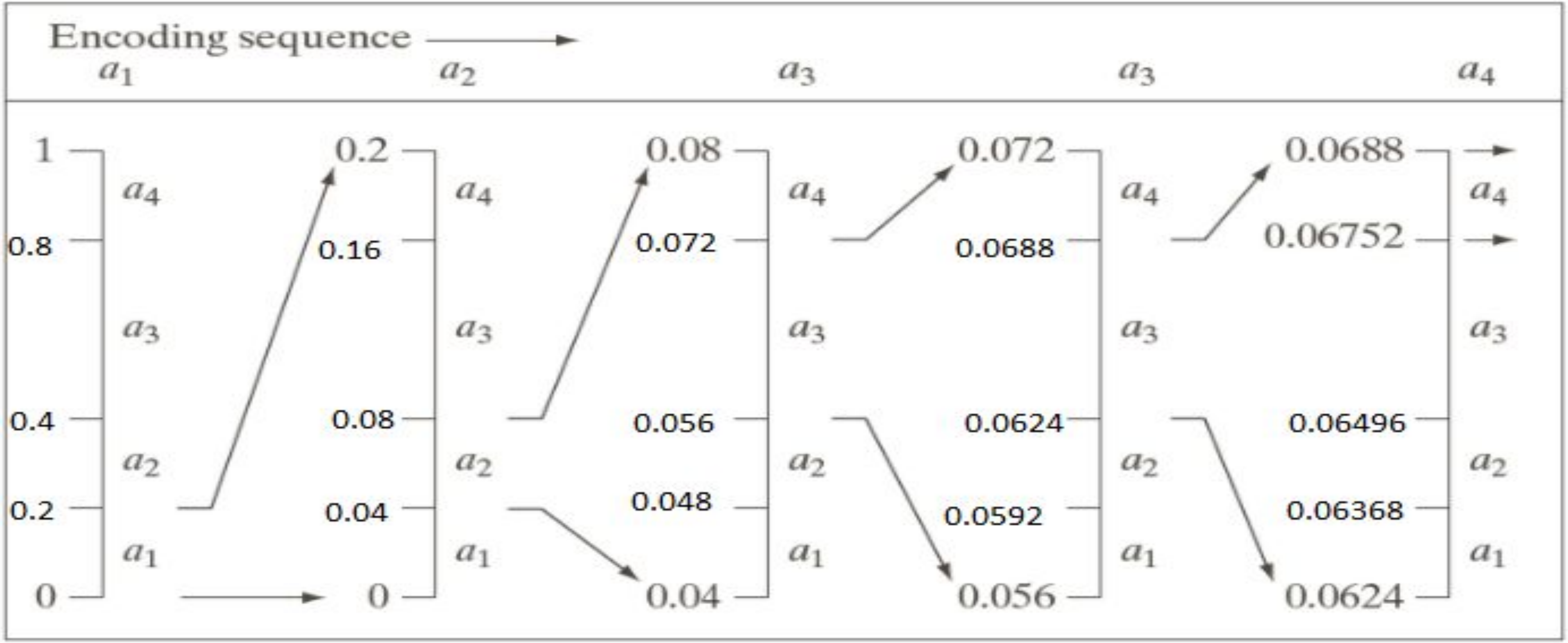
or,

0.068

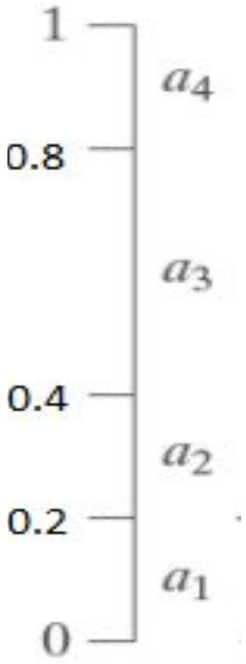


Encoding sequence
a_1
a_2
a_3
a_3
a_4
1
0.8
0.4
0.2
0
a_4
a_3
a_2
a_1
0.2
0.16
0.08
0.04
0
0.08
0.072
0.056
0.048
0.04
0.072
0.0688
0.0624
0.0592
0.056
0.0688
0.06752
0.06496
0.06368
0.0624

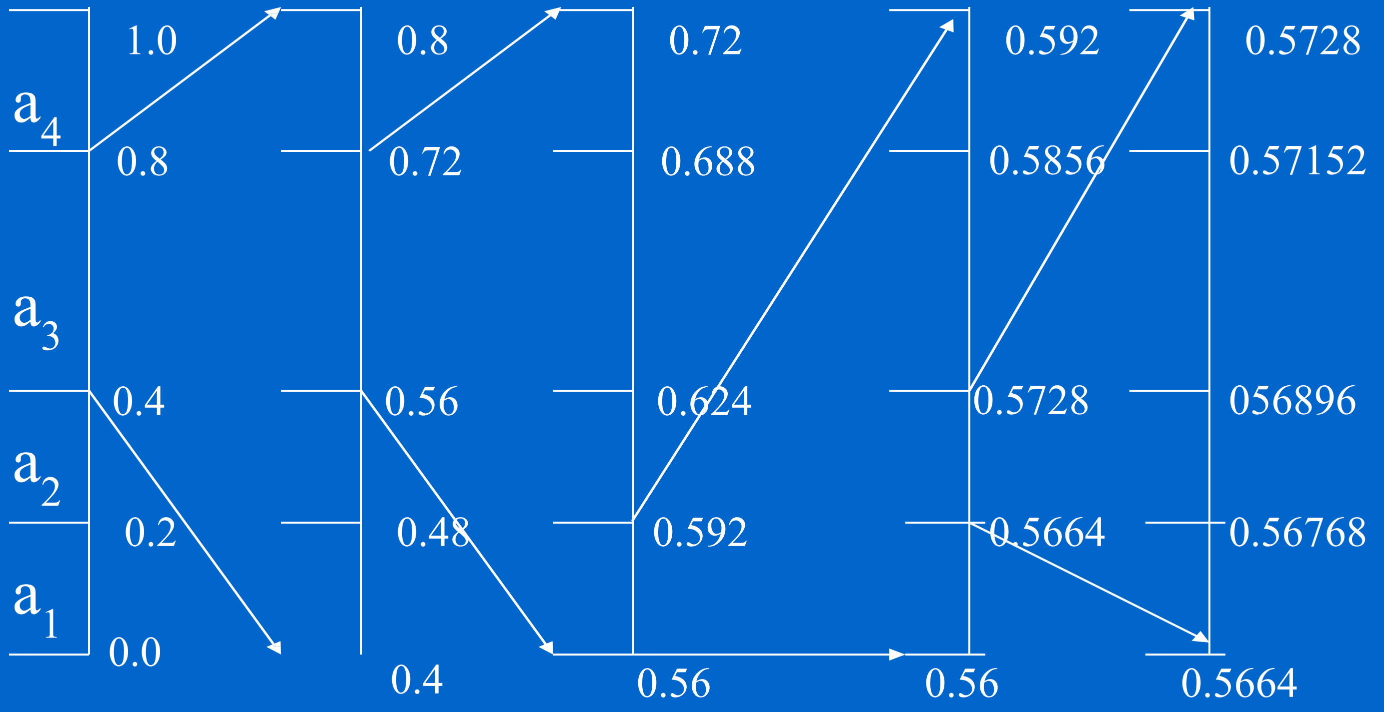
Arithmetic decoding of 0.068



Decode: 0.572



Arithmetic Decoding



Decode 0.572

a_3 a_3 a_1 a^2 a^4

LZW Coding (interpixel redundancy)

- Requires no priori knowledge of pixel probability distribution values.
- Lempel-Ziv-Welch code.
- Assigns fixed length code words to variable length sequences.
- Included in GIF and TIFF and PDF file formats

LZW Coding

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

Consider a 4x4, 8 bit image

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	-
511	-

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

24 Bits

9 Bits

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

Decoding LZW

- The dictionary which was used for encoding need not be sent with the image.

- Can be built on the "fly" by the decoder as it reads the received code words.

Symbol-Based Coding

In *symbol-* or *token-based* coding, an image is represented as a collection of frequently occurring sub-images, called *symbols*. Each such symbol is stored in a *symbol dictionary* and the image is coded as a set of triplets $\{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots\}$, where each (x_i, y_i) pair specifies the location of a symbol in the image and *token* t_i is the address of the symbol or sub-image in the dictionary.

ababcabac

a = 1

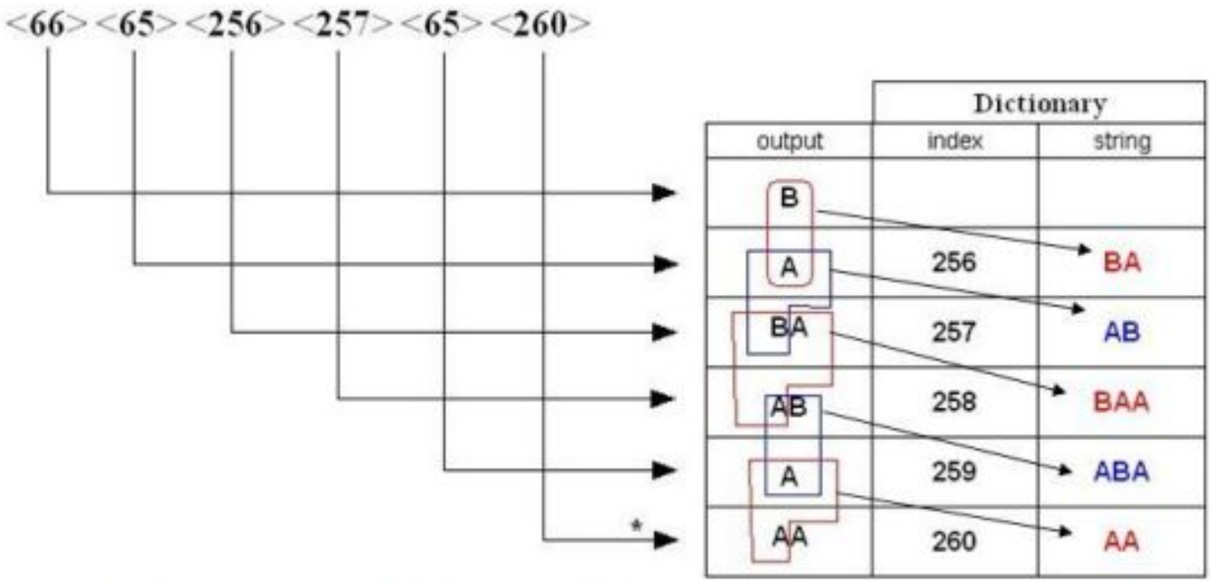
b = 2

c = 3

d = 4

letztes W.	aktuelles W.	Eintrag	Code Ausgabe
␣	a		
␣a	b	ab = 5	1
b	a	ba = 6	2
a	b		
ab	c	abc = 7	5
c	a	ca = 8	3
a	b		
ab	a	aba = 9	5
a	c	ac = 10	1
c	EOF		3

Use LZW to decompress the output sequence <66> <65> <256> <257> <65> <260>



1. **66** is in Dictionary; output **string(66)** i.e. B
2. **65** is in Dictionary; output **string(65)** i.e. **A**, insert BA
3. **256** is in Dictionary; output **string(256)** i.e. **BA**, insert **AB**
4. **257** is in Dictionary; output **string(257)** i.e. **AB**, insert **BAA**
5. **65** is in Dictionary; output **string(65)** i.e. A, insert **ABA**
6. **260** is **<u>not</u>** in Dictionary; output

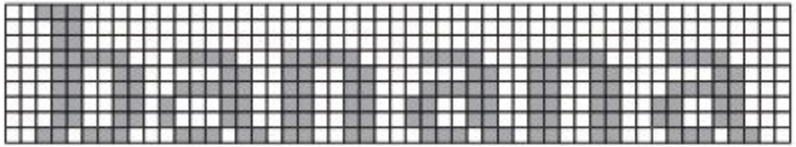
   previous output + previous output first character: AA**, insert AA**

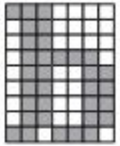
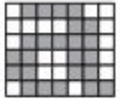
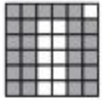
Dictionary

0	a
1	b

0 1 2 4 3

a b ab aba ba



Token	Symbol
0	
1	
2	

Triplet
(0, 2, 0) (3, 10, 1) (3, 18, 2) (3, 26, 1) (3, 34, 2) (3, 42, 1)

Compression ratio??

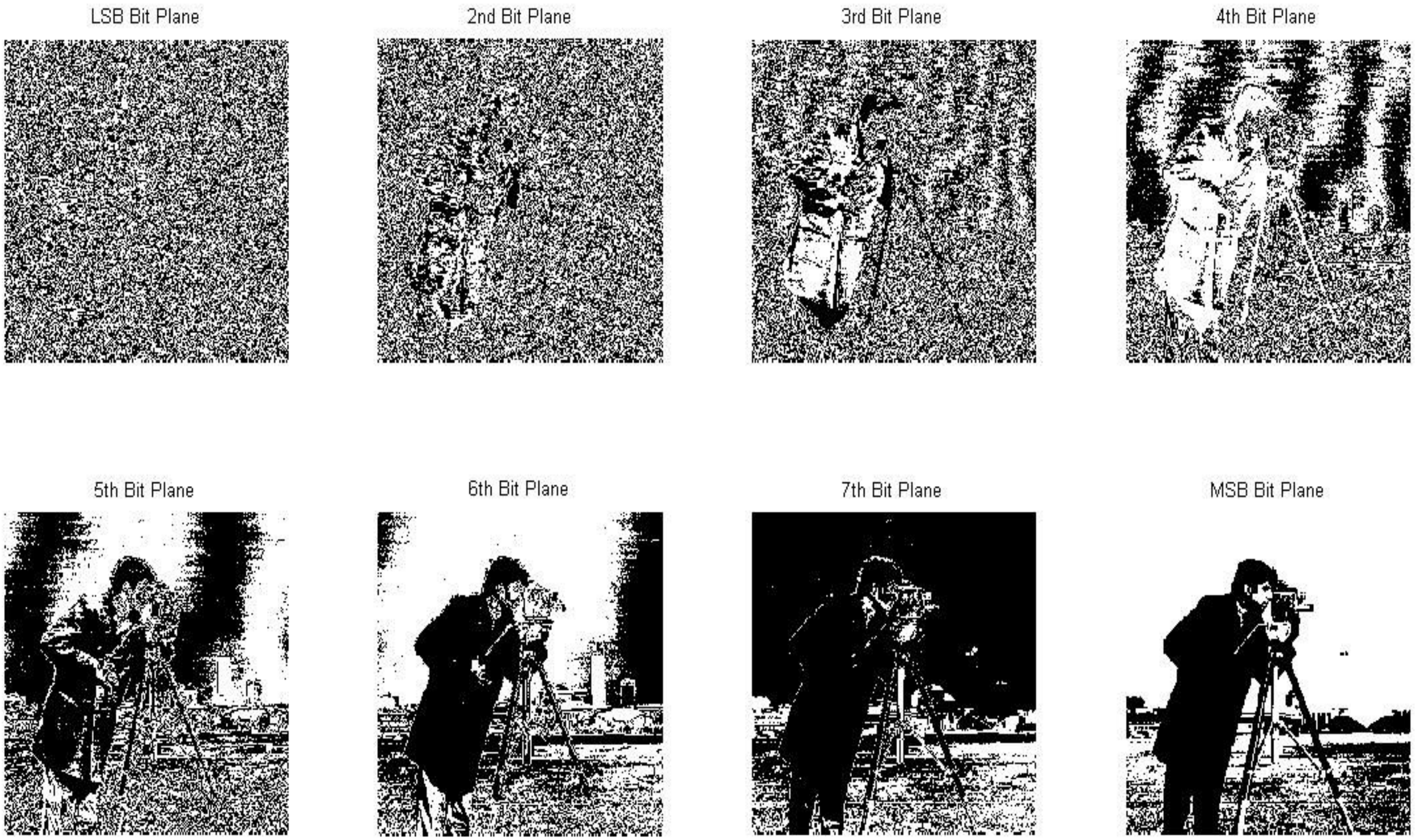
In this case, the starting image has 9x51x1 or 459 bits and, assuming that each triplet is composed of 3 bytes, the compressed representation has 6x3x8 +[(9x7)+(6x7)+(6x6)] or 285 bits; the resulting compression ratio= 1.61

Bit-Plane Coding

The run-length and symbol-based techniques of the previous sections can be applied to images with more than two intensities by processing their bit planes individually. The technique, called bit-plane coding, is based on the concept of decomposing a multilevel (monochrome or color) image into a series of binary images.

The intensities of an m-bit monochrome image can be represented in the form of the base-2 polynomial

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \ldots + a_1 2^1 + a_0 2^0$$


LSB Bit Plane
2nd Bit Plane
3rd Bit Plane
4th Bit Plane
5th Bit Plane
6th Bit Plane
7th Bit Plane
MSB Bit Plane

Bit plane slicing: solution??

127	128	127
128	127	128
127	128	127

0	1	0
1	0	1
0	1	0

1	0	1
0	1	0
1	0	1

1	0	1
0	1	0
1	0	1

1	0	1
0	1	0
1	0	1

1	0	1
0	1	0
1	0	1

1	0	1
0	1	0
1	0	1

1	0	1
0	1	0
1	0	1

1	0	1
0	1	0
1	0	1

Gray codes instead of binary!

128⇒ 10000000 ⇒ 11000000

127⇒ 01111111⇒ 01000000

$$g_i = a_i \oplus a_{i+1} \quad 0 \leq i \leq m - 2$$
$$g_{m-1} = a_{m-1}$$

Bit plane slicing: solution??

127	128	127
128	127	128
127	128	127

0	1	0
1	0	1
0	1	0

1	1	1
1	1	1
1	1	1

0	0	0
0	0	0
0	0	0

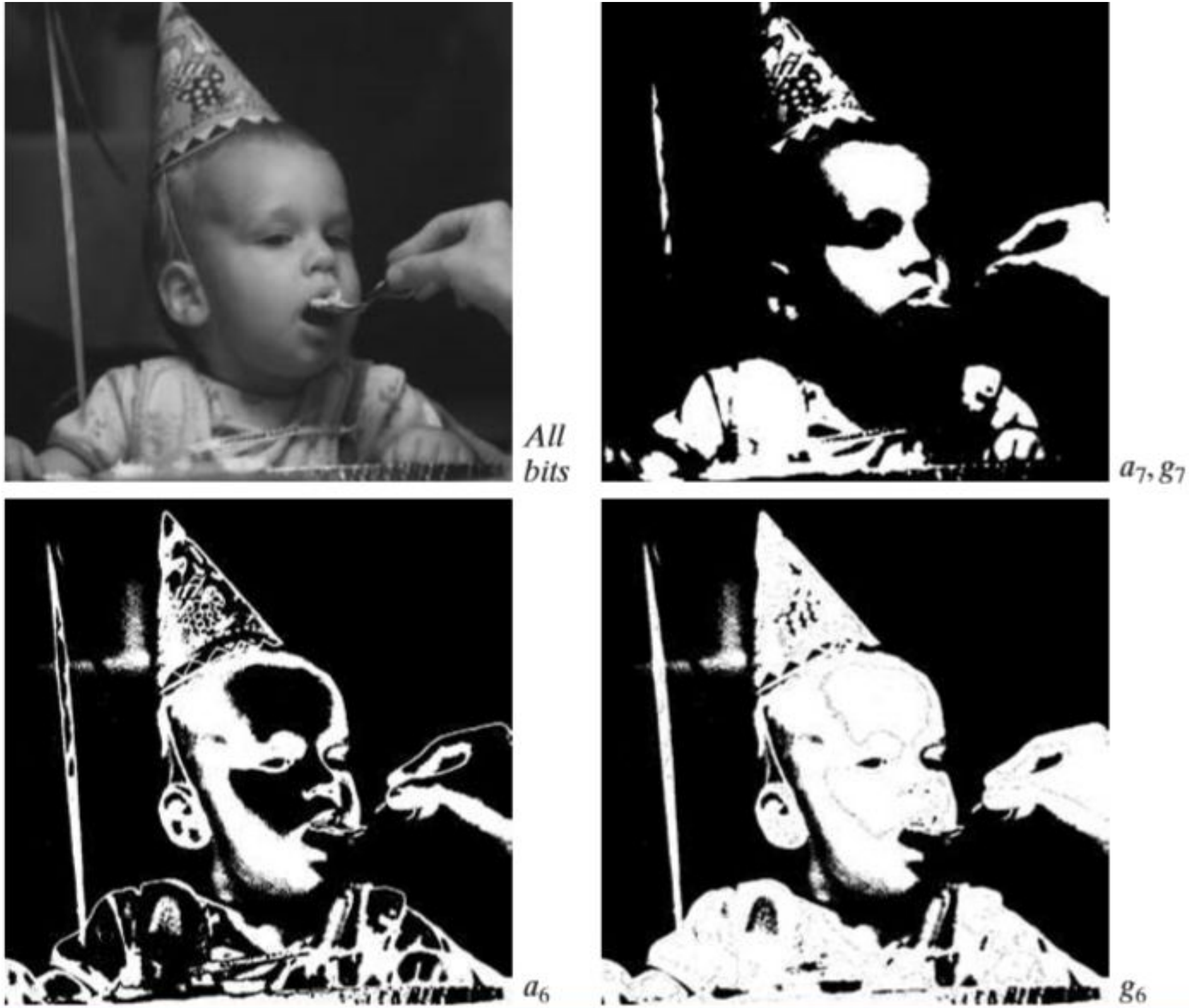
0	0	0
0	0	0
0	0	0

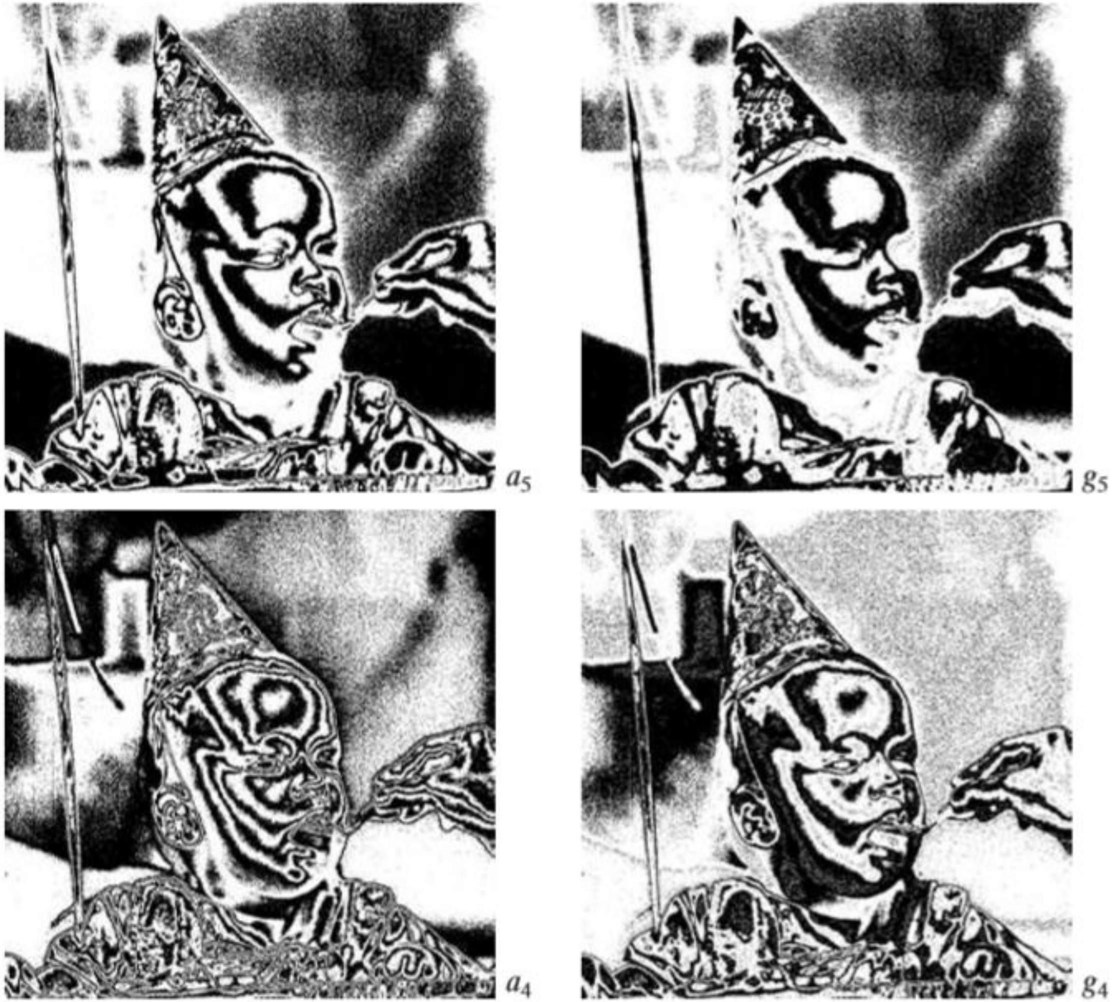
0	0	0
0	0	0
0	0	0

0	0	0
0	0	0
0	0	0

0	0	0
0	0	0
0	0	0

0	0	0
0	0	0
0	0	0


All bits
a_7, g_7
a_6
g_6


a_5
g_5
a_4
g_4

Predictive coding

The approach,commonly referred to as predictive coding,is based on eliminating the redundancies of closely spaced pixels—in space and/or time— by extracting and coding only the new information in each pixel.The new information of a pixel is defined as the difference between the actual and predicted value of the pixel

1. Lossless predictive coding

The system consists of an encoder and a decoder, each containing an identical *predictor*. As successive samples of discrete time input signal, $f(n)$, are introduced to the encoder, the predictor generates the anticipated value of each sample based on a specified number of past samples. The output of the predictor is then rounded to the nearest integer, denoted $\hat{f}(n)$, and used to form the difference or *prediction error*

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

The decoder reconstructs $e(n)$ from the received variable-length code words and performs the inverse operation

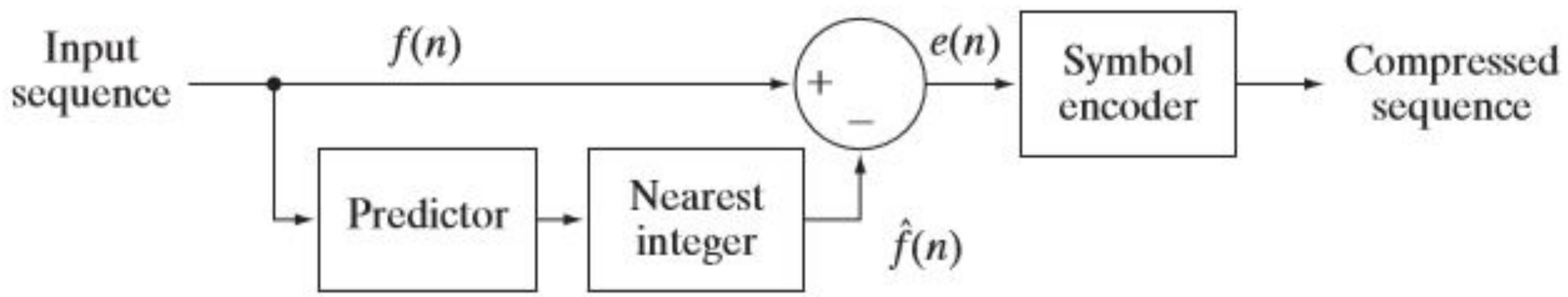
$$f(n) = e(n) + \hat{f}(n)$$

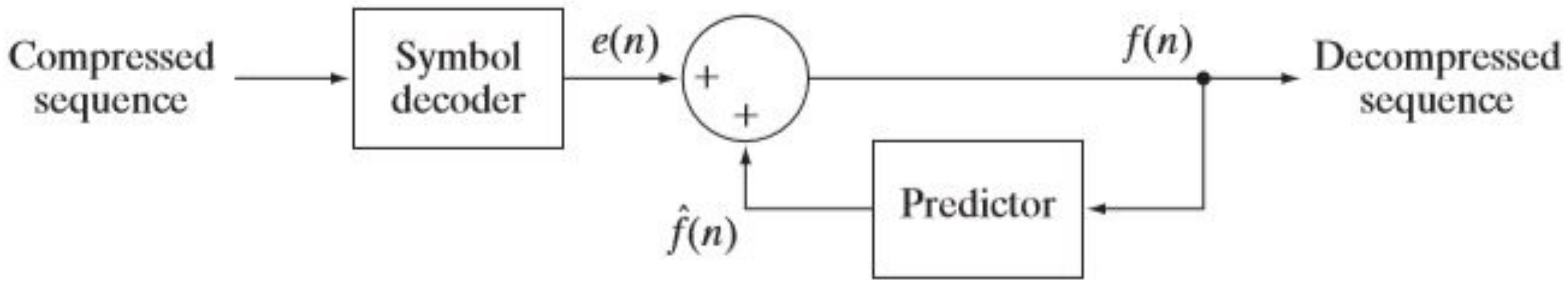
to decompress or recreate the original input sequence.

Various local, global, and adaptive methods (see the later subsection entitled Lossy predictive coding) can be used to generate $\hat{f}(n)$. In many cases, the prediction is formed as a linear combination of m previous samples. That is,

$$\hat{f}(n) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(n-i)\right]$$

where m is the *order* of the linear predictor, round is a function used to denote the rounding or nearest integer operation, and the α_i for $i = 1, 2, \ldots, m$ are prediction coefficients

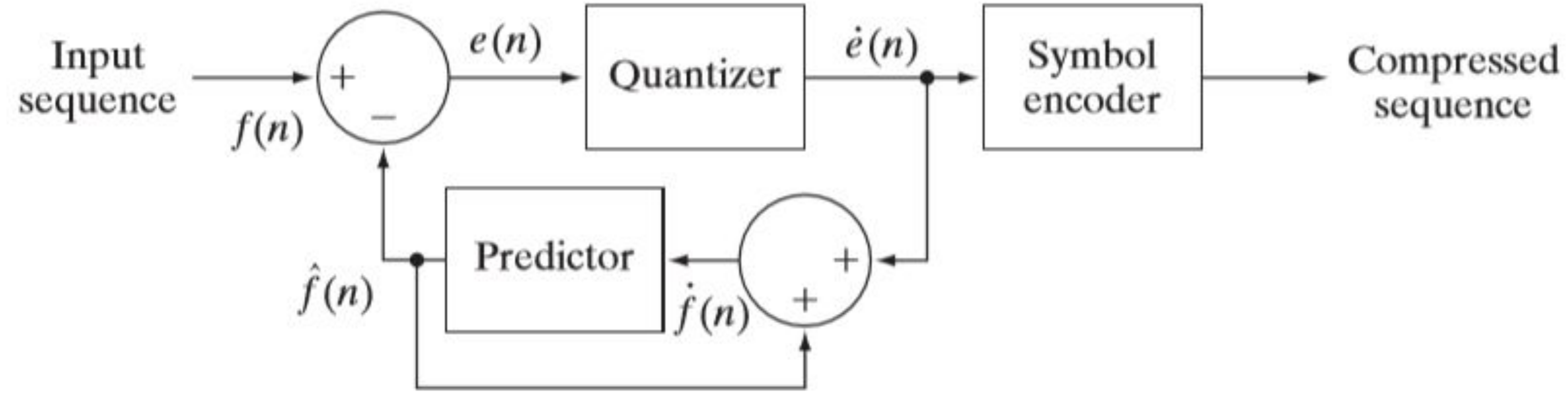

Input
sequence
$f(n)$
$e(n)$
Symbol
encoder
Compressed
sequence
Predictor
Nearest
integer
$\hat{f}(n)$


Compressed
sequence
Symbol
decoder
$e(n)$
$f(n)$
Decompressed
sequence
Predictor
$\hat{f}(n)$

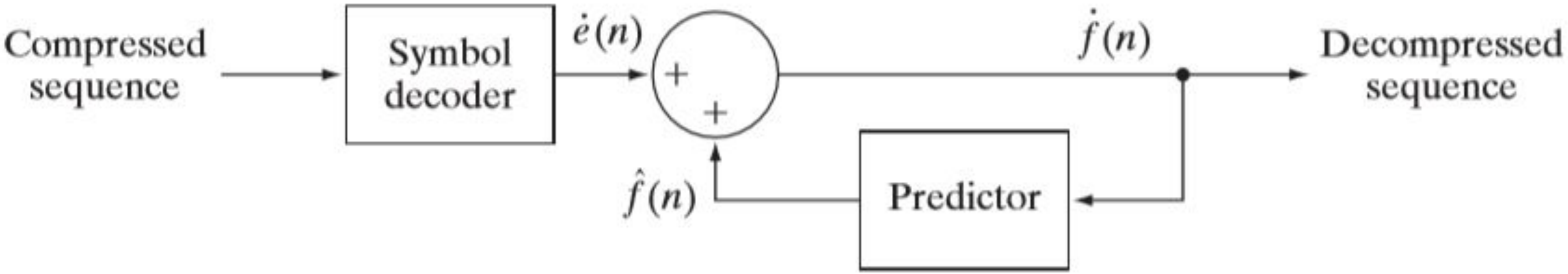
the m samples used to predict the value of each pixel come from the current scan line (called 1-D linear predictive coding), from the current and previous scan lines (called 2-D linear predictive coding), or from the current image and previous images in a sequence of images (called 3-D linear predictive coding). Thus, for 1-D linear predictive image coding, Eq. (8.2-32) can be written as

$$\hat{f}(x, y) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(x, y - i)\right]$$

Lossy predictive coding



Input sequence
$f(n)$
+
−
$e(n)$
Quantizer
$\dot{e}(n)$
Symbol encoder
Compressed sequence
Predictor
$\hat{f}(n)$
$\dot{f}(n)$
+
+



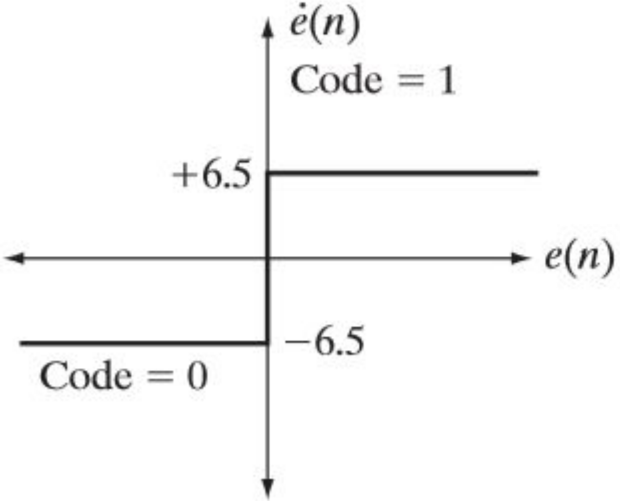
Compressed sequence
Symbol decoder
$\dot{e}(n)$
+
+
$\dot{f}(n)$
Decompressed sequence
$\hat{f}(n)$
Predictor

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

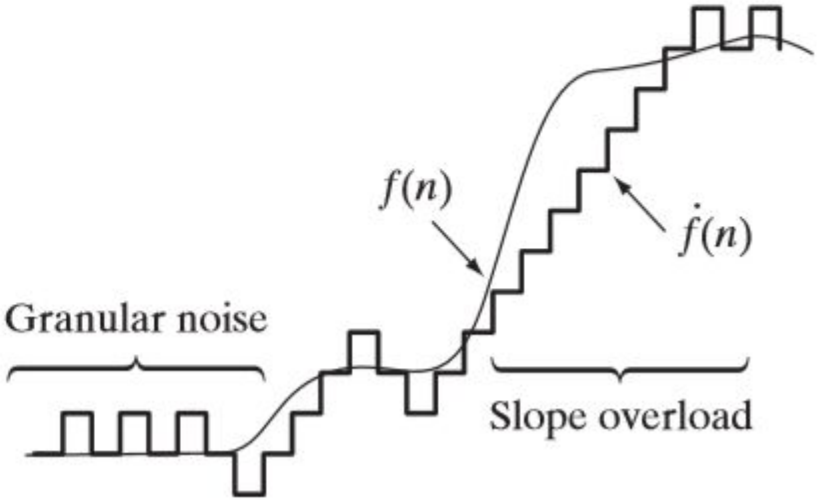
Delta modulation:

$$\hat{f}(n) = \alpha \dot{f}(n-1)$$

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

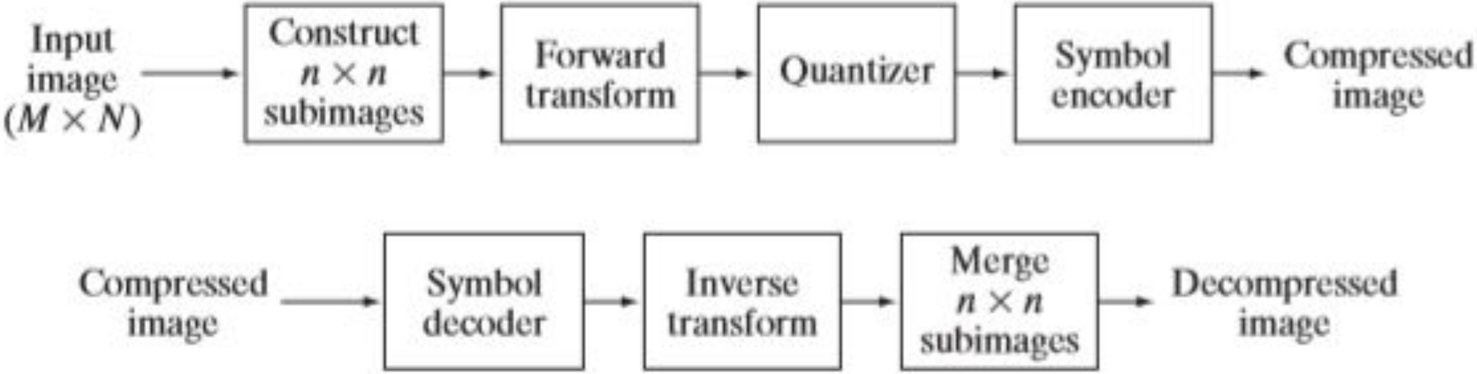


$\dot{e}(n)$
Code = 1
+6.5
$e(n)$
−6.5
Code = 0



$f(n)$
$\dot{f}(n)$
Granular noise
Slope overload

BLOCK TRANSFORM CODING



$g(x, y)$ of size $n \times n$ whose forward, discrete transform, $T(u, v)$, can be expressed in terms of the general relation

$$T(u, v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x, y) r(x, y, u, v) \tag{8.2-10}$$

for $u, v = 0, 1, 2, \ldots, n - 1$. Given $T(u, v)$, $g(x, y)$ similarly can be obtained using the generalized inverse discrete transform

$$g(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) s(x, y, u, v) \tag{8.2-11}$$

for $x, y = 0, 1, 2, \ldots, n - 1$. In these equations, $r(x, y, u, v)$ and $s(x, y, u, v)$ are called the *forward* and *inverse transformation kernels*, respectively.

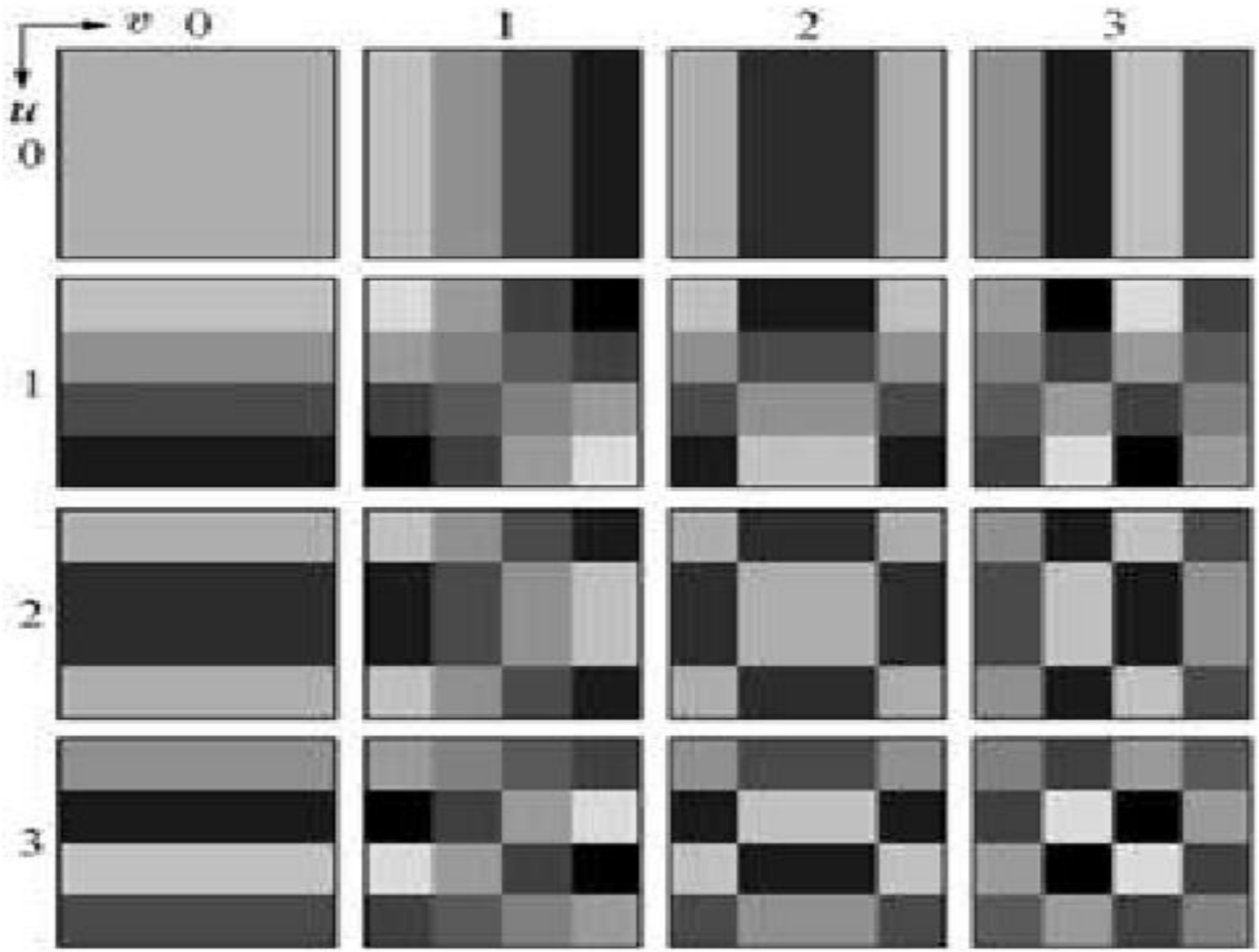
One of the transformations used most frequently for image compression is the *discrete cosine transform* (DCT). It is obtained by substituting the following (equal) kernels into Eqs. (8.2-10) and (8.2-11)

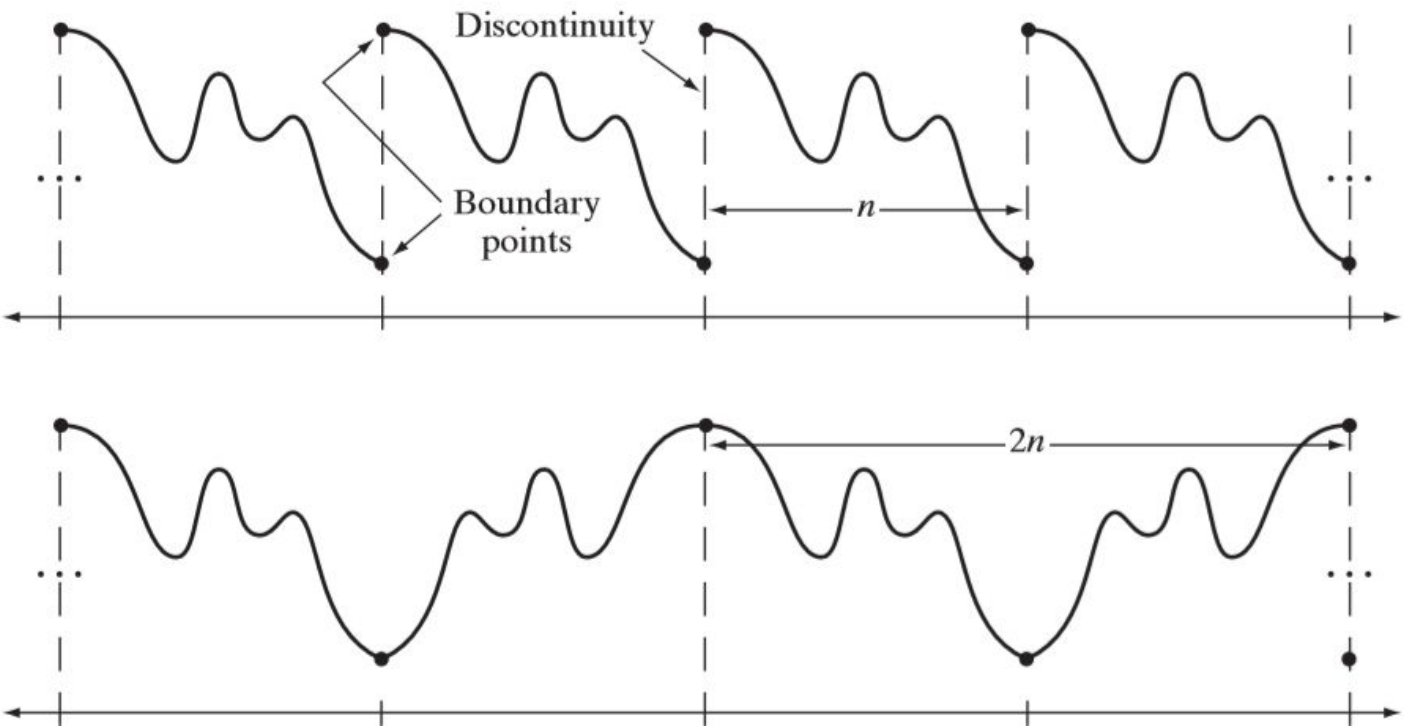
$$
\begin{aligned}
r(x, y, u, v) &= s(x, y, u, v) \\
&= \alpha(u)\alpha(v)\cos\left[\frac{(2x+1)u\pi}{2n}\right]\cos\left[\frac{(2y+1)v\pi}{2n}\right]
\end{aligned}
\tag{8.2-18}
$$

where

$$
\alpha(u) = \begin{cases} \sqrt{\dfrac{1}{n}} & \text{for } u = 0 \\ \sqrt{\dfrac{2}{n}} & \text{for } u = 1, 2, \ldots, n-1 \end{cases}
\tag{8.2-19}
$$

and similarly for $\alpha(v)$. Figure 8.23 shows $r(x, y, u, v)$ for the case $n = 4$.


v
u
0
1
2
3
0
1
2
3



a
b

FIGURE 8.25 The periodicity implicit in the 1-D (a) DFT and (b) DCT.

Jpeg

JPEG is an image compression standard which was accepted as an international standard in 1992.

Developed by the Joint Photographic Expert Group of the ISO/IEC for coding and compression of color/gray scale images.

Yields acceptable compression in the 10:1 range.

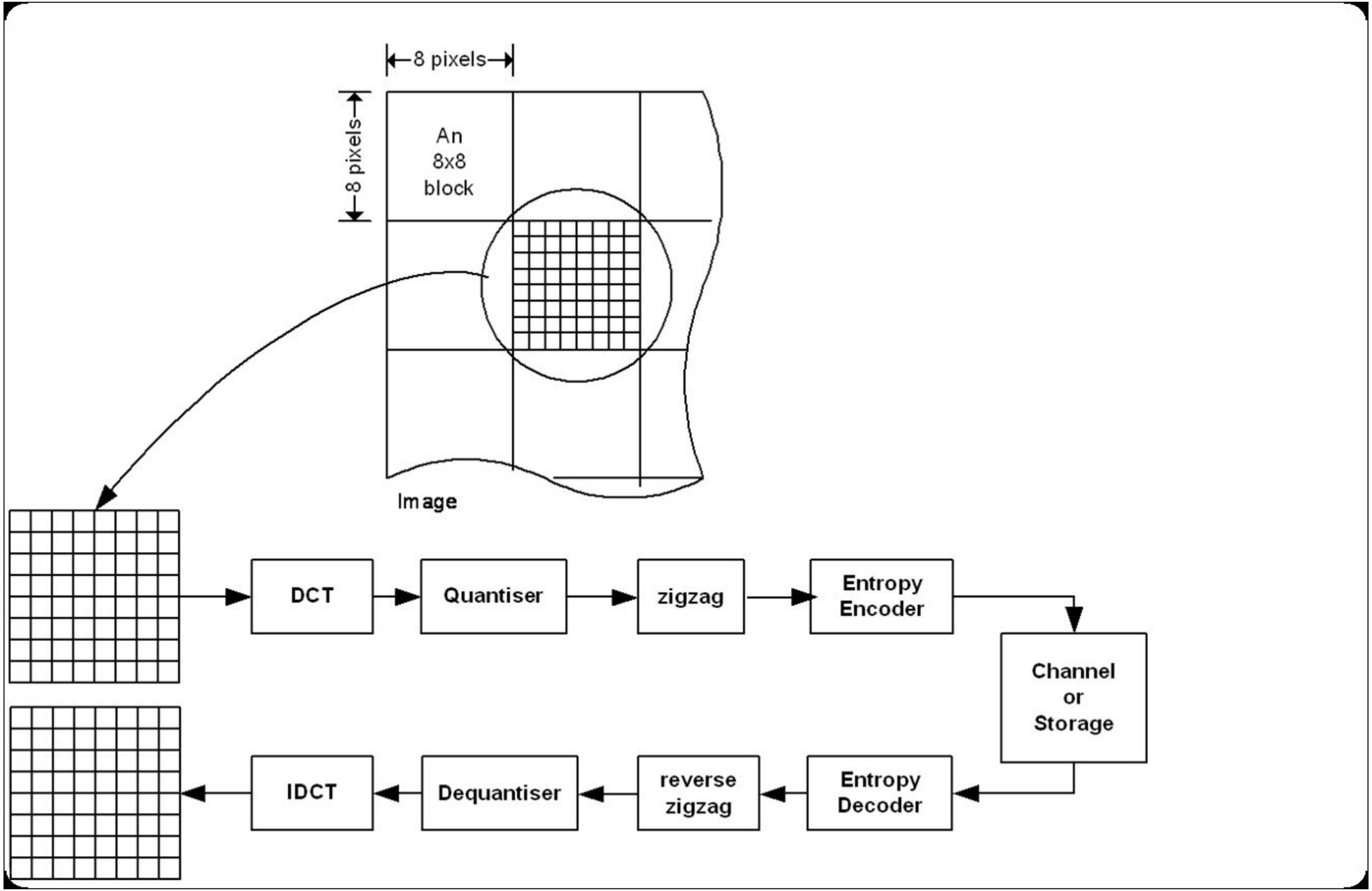
A scheme for video compression based on JPEG called Motion JPEG (MJPEG) exists

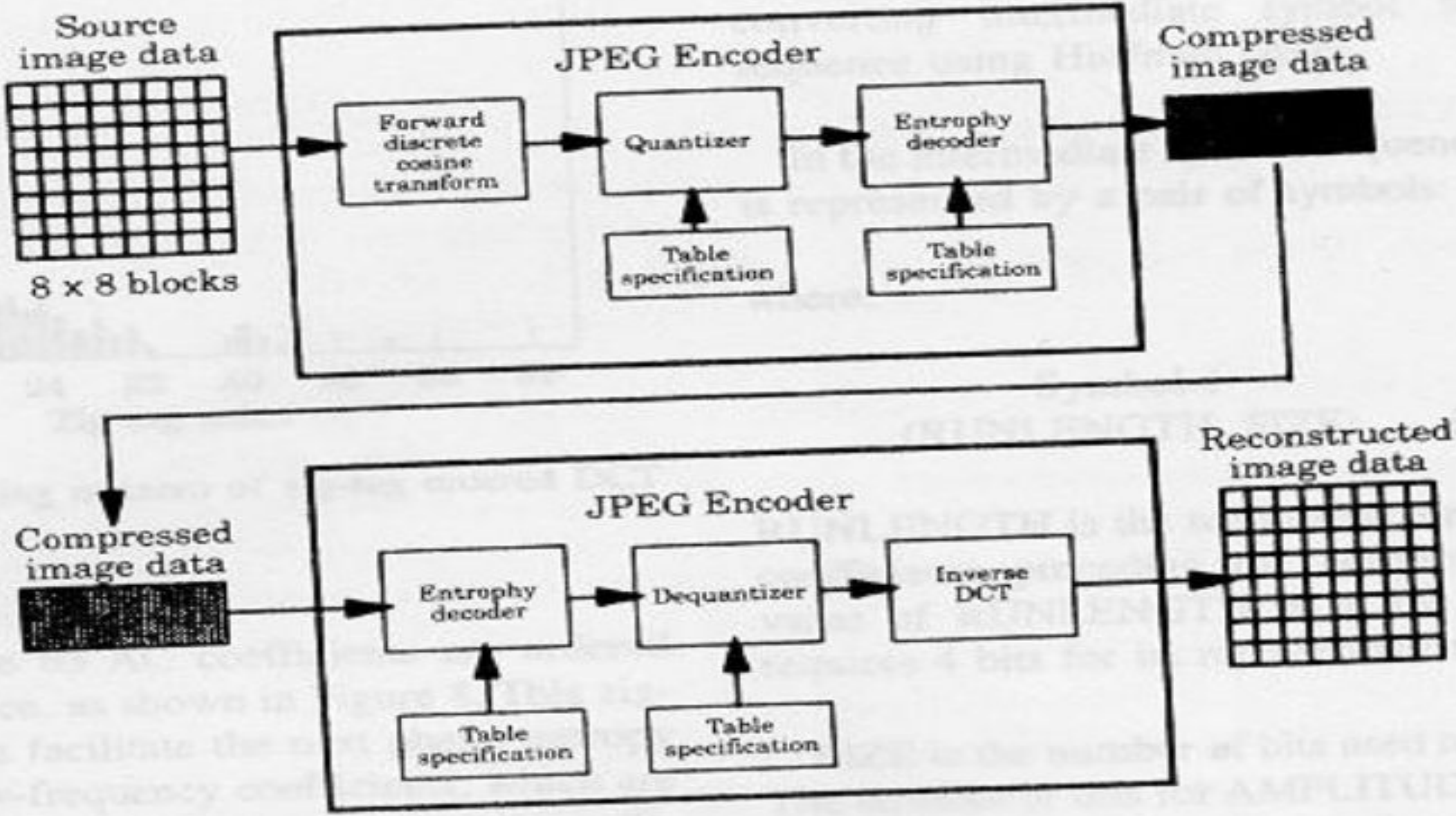
JPEG

Lossy Compression Technique based on use of Discrete Cosine Transform (DCT)

STEPS IN JPEG COMPRESSION:

1. Divide Each plane into 8x8 size blocks.
2. Compute DCT of each block
3. Treat separately DC components of each block.
4. Apply Quantization to discard values
5. Encode DC components and transmit data.


8 pixels
8 pixels
An
8x8
block
Image
DCT
Quantiser
zigzag
Entropy
Encoder
Channel
or
Storage
Entropy
Decoder
reverse
zigzag
Dequantiser
IDCT



Source image data
8 × 8 blocks
JPEG Encoder
Forward discrete cosine transform
Quantizer
Entrophy decoder
Table specification
Table specification
Compressed image data
Compressed image data
JPEG Encoder
Entrophy decoder
Dequantizer
Inverse DCT
Table specification
Table specification
Reconstructed image data

Steps in JPEG

1. Divide the image into 8x8 subimages;

For each subimage do:

2. Shift the gray-levels in the range [-128, 127] - DCT requires range be centered around 0

3. Apply DCT (i.e., 64 coefficients)

    1 DC coefficient: F(0,0)

    63 AC coefficients: F(u,v)

Image Compression Standards

JPEG Encoding - Example

Sub-Image

$$
\begin{matrix}
52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\
63 & 59 & 66 & 90 & 109 & 85 & 69 & 72 \\
62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\
63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\
67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\
79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\
85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\
87 & 79 & 69 & 68 & 65 & 76 & 78 & 94
\end{matrix}
$$

-128

$$
\begin{matrix}
-76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\
-65 & -69 & -62 & -38 & -19 & -43 & -59 & -56 \\
-66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\
-65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\
-61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\
-49 & -63 & -68 & -58 & -51 & -65 & -70 & -53 \\
-43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\
-41 & -49 & -59 & -60 & -63 & -52 & -50 & -34
\end{matrix}
$$

DCT

$$
\begin{matrix}
-415 & -29 & -62 & 25 & 55 & -20 & -1 & 3 \\
7 & -21 & -62 & 9 & 11 & -7 & -6 & 6 \\
-46 & 8 & 77 & -25 & -30 & 10 & 7 & -5 \\
-50 & 13 & 35 & -15 & -9 & 6 & 0 & 3 \\
11 & -8 & -13 & -2 & -1 & 1 & -4 & 1 \\
-10 & 1 & 3 & -3 & -1 & 0 & 2 & -1 \\
-4 & -1 & 2 & -1 & 2 & -3 & 1 & -2 \\
-1 & -1 & -1 & -2 & -1 & -1 & 0 & -1
\end{matrix}
$$

F(0,0) = (1/8) * addition of matrix = (1/8) *

F(0,0) = (1/8) * (-3317) = -414.62 =~ -415

Z represents quality

Z

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

T

−415	−29	−62	25	55	−20	−1	3
7	−21	−62	9	11	−7	−6	6
−46	8	77	−25	−30	10	7	−5
−50	13	35	−15	−9	6	0	3
11	−8	−13	−2	−1	1	−4	1
−10	1	3	−3	−1	0	2	−1
−4	−1	2	−1	2	−3	1	−2
−1	−1	−1	−2	−1	−1	0	−1

T/Z

−26	−3	−6	2	2	0	0	0
1	−2	−4	0	0	0	0	0
−3	1	5	−1	−1	0	0	0
−4	1	2	−1	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

$$
\begin{matrix}
-26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\
1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}
$$

Zigzag ordering → [-26 -3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB]

If DC coefficient of the transformed and quantized sub-image to its immediate left was -17, the resulting DPCM:

[-26-(-17)]=-9 Category: 4 Length: 7

101 0110

↑

(0111 -1)

[-26 -3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB]

1010110 0100 001 0100 0101 100001 0110 100011 001 100011 001 001 100101 11100110 110110 0110 11110100 000 1010

$$
\begin{matrix}
-26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\
1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}
\xrightarrow{\text{T*Z}}
\begin{matrix}
-70 & -64 & -61 & -64 & -69 & -66 & -58 & -50 \\
-72 & -73 & -61 & -39 & -30 & -40 & -54 & -59 \\
-68 & -78 & -58 & -9 & 13 & -12 & -48 & -64 \\
-59 & -77 & -57 & 0 & 22 & -13 & -51 & -60 \\
-54 & -75 & -64 & -23 & -13 & -44 & -63 & -56 \\
-52 & -71 & -72 & -54 & -54 & -71 & -71 & -54 \\
-45 & -58 & -70 & -68 & -67 & -67 & -61 & -50 \\
-35 & -47 & -61 & -66 & -60 & -48 & -44 & -44
\end{matrix}
$$

$$
\xrightarrow{\text{DCT}^{-1}}
\begin{matrix}
-416 & -33 & -60 & 32 & 48 & 0 & 0 & 0 \\
12 & -24 & -56 & 0 & 0 & 0 & 0 & 0 \\
-42 & 13 & 80 & -24 & -40 & 0 & 0 & 0 \\
-56 & 17 & 44 & -29 & 0 & 0 & 0 & 0 \\
18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}
\xrightarrow{+128}
\begin{matrix}
58 & 64 & 67 & 64 & 59 & 62 & 70 & 78n \\
56 & 55 & 67 & 89 & 98 & 88 & 74 & 69 \\
60 & 50 & 70 & 119 & 141 & 116 & 80 & 64 \\
69 & 51 & 71 & 128 & 149 & 115 & 77 & 68 \\
74 & 53 & 64 & 105 & 115 & 84 & 65 & 72 \\
76 & 57 & 56 & 74 & 75 & 57 & 57 & 74 \\
83 & 69 & 59 & 60 & 61 & 61 & 67 & 78 \\
93 & 81 & 67 & 62 & 69 & 80 & 84 & 84
\end{matrix}
$$

Error

−6	−9	−6	2	11	−1	−6	−5
7	4	−1	1	11	−3	−5	3
2	9	−2	−6	−3	−12	−14	9
−6	7	0	−4	−5	−9	−7	1
−7	8	4	−1	11	4	3	−2
3	8	4	−4	2	11	1	1
2	2	5	−1	−6	0	−2	5
−6	−2	2	6	−4	−4	−6	10

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
$-7, \dots, -4, 4, \dots, 7$	3	3
$-15, \dots, -8, 8, \dots, 15$	4	4
$-31, \dots, -16, 16, \dots, 31$	5	5
$-63, \dots, -32, 32, \dots, 63$	6	6
$-127, \dots, -64, 64, \dots, 127$	7	7
$-255, \dots, -128, 128, \dots, 255$	8	8
$-511, \dots, -256, 256, \dots, 511$	9	9
$-1023, \dots, -512, 512, \dots, 1023$	A	A
$-2047, \dots, -1024, 1024, \dots, 2047$	B	B
$-4095, \dots, -2048, 2048, \dots, 4095$	C	C
$-8191, \dots, -4096, 4096, \dots, 8191$	D	D
$-16383, \dots, -8192, 8192, \dots, 16383$	E	E
$-32767, \dots, -16384, 16384, \dots, 32767$	F	N/A

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

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).*

Table 8.19 (Con't)

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

■ The BMP file format uses a form of run-length encoding in which image data is represented in two different modes: encoded and absolute—and either mode can occur anywhere in the image. In *encoded* mode, a two byte RLE representation is used. The first byte specifies the number of consecutive pixels that have the color index contained in the second byte. The 8-bit color index selects the run's intensity (color or gray value) from a table of 256 possible intensities.

In *absolute* mode, the first byte is 0 and the second byte signals one of four possible conditions, as shown in Table 8.8. When the second byte is 0 or 1, the end of a line or the end of the image has been reached. If it is 2, the next two bytes contain unsigned horizontal and vertical offsets to a new spatial position (and pixel) in the image. If the second byte is between 3 and 255, it specifies the number of uncompressed pixels that follow—with each subsequent byte containing the color index of one pixel. The total number of bytes must be aligned on a 16-bit word boundary.

Second Byte Value	Condition
0	End of line
1	End of image
2	Move to a new position
3–255	Specify pixels individually

Decode the BMP encoded sequence $\{3, 4, 5, 6, 0, 3, 103, 125, 67, 0, 2, 47\}$.

Using the BMP specification given in Example 8.8 of Section 8.2.5, the first two bytes indicate that the uncompressed data begins with a run of 4s with length 3. In a similar manner, the second two bytes call for a run of 6s with length 5. The first four bytes of the BMP encoded sequence are encoded mode. Because the 5th byte is 0 and the 6th byte is 3, absolute mode is entered and the next three values are taken as uncompressed data. Because the total number of bytes in absolute mode must be aligned on a 16-bit word boundary, the 0 in the 10th byte of the encoded sequence is padding and should be ignored. The final two bytes specify an encoded mode run of 47s with length 2. Thus, the complete uncompressed sequence is {4, 4, 4, 6, 6, 6, 6, 6, 103, 125, 67, 47, 47}.

Consider a binary (black and white) image of size 8x8 pixels. Region splitting and merging algorithm is applied on the image. The quad tree generated is given by following sequence of symbols : g(g(wbbw)g(bg(bwwb)ww)g(bwww)g(wwbw)). In which the homogeneous square black pixels block are encoded by b, the homogeneoussquare white pixels block by w and the nonhomogeneous square pixel block by g is described. Calculate the number of white pixels and black pixels the image contains.Show your calculation.