10 – IMAGE COMPRESSION (B)

ERROR-FREE COMPRESSION

In many applications, error-free compression is required, e.g., in archival and transmission of medical images. Furthermore, many lossy coding schemes make use of lossless coding components to minimize the redundancy of the signal being processed.

Error-free compression techniques generally are composed of two relatively independent operations:

1. coding the representation to eliminate coding redundancies (symbol coding)
2. devising an alternative representation of the image in which its interpixel redundancies are reduced (mapping), and

Achievable compression ratios are in the range of 2 to 10.

Variable-length Coding

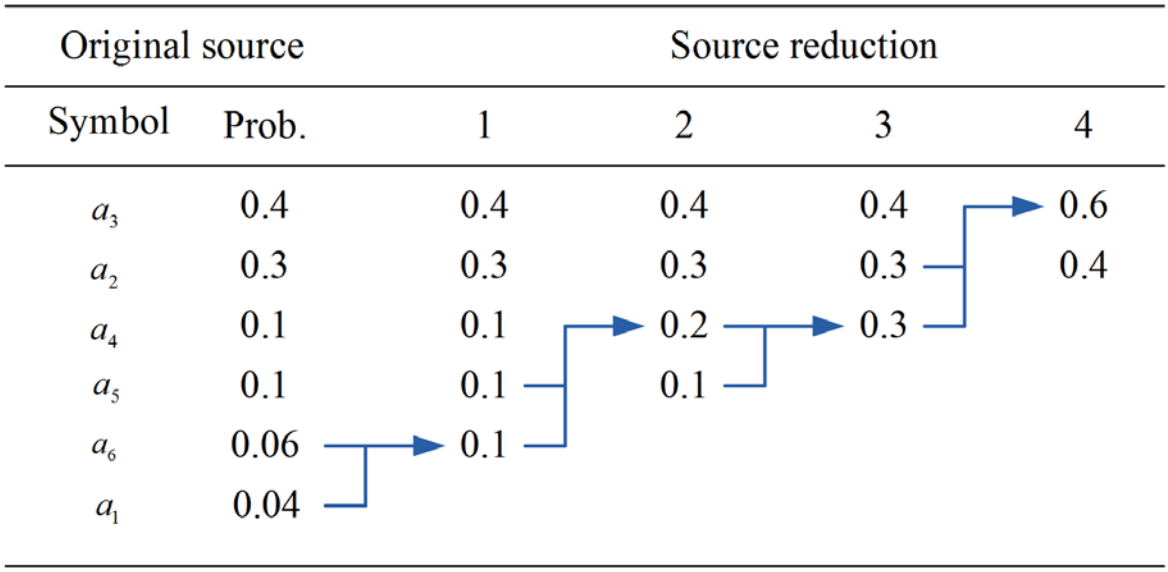
Coding redundancy normally is present in any natural binary encoding of the gray levels in an image. It can be eliminated by coding the gray levels so that $\bar{L}$, the average number of bits required to represent each pixel, is minimized. To do so requires construction of a variable-length code that assigns the shortest possible code words to the most probable gray levels. We look at Huffman coding and arithmetic coding.

Huffman coding

When coding the symbols of an information source individually, *Huffman coding* yields the smallest possible number of code symbols per source symbol.

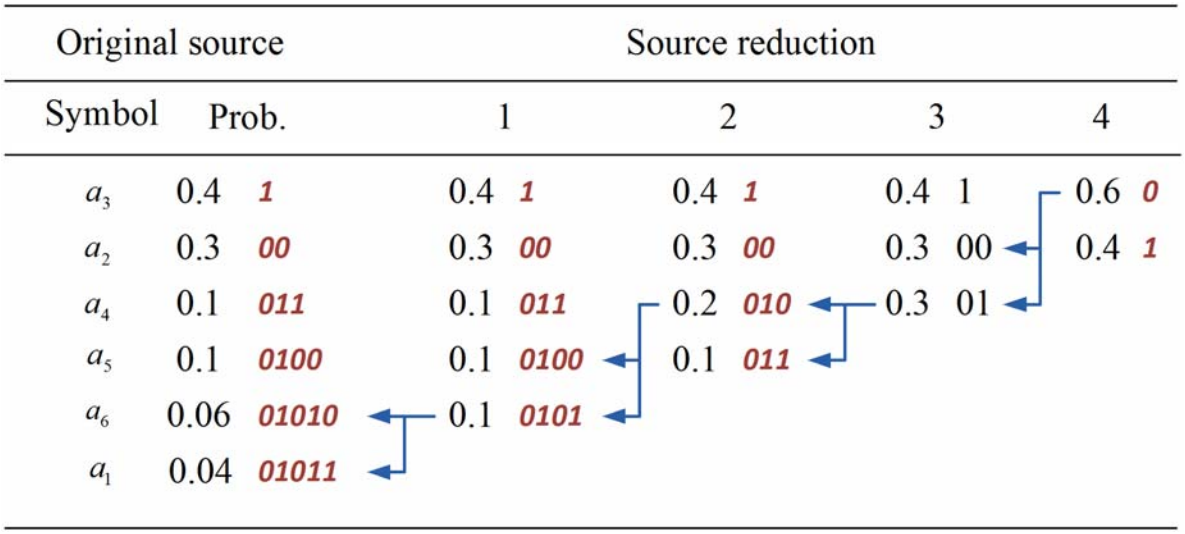
The first step is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction.

1. The set of source symbols are ordered from top to bottom in decreasing probabiliy values.
2. The bottom two probabilites, 0.06 and 0.04, are combined to form a "compound symbol" with probabilty 0.1.
3. This compound symbol and its associated probability are placed in the first source reduction column so that the probabilities of the reduced source are also ordered.
4. The process is then repeated until a reduced source with two symbols is reached.



The second step is to code each reduced source, starting with the smallest source and working back to the original source.

1. The minimal length binary code for a two-symbol source is the symbols 0 and 1. These symbols are assigned to the two symbols on the right (reversing the order of the 0 and 1 would work just as well).
2. As the reduced source symbol with probability 0.6 was generated by combining the two symbols in the reduced source to its left, the 0 used to code it is now assigned to both these symbols, and a 0 and 1 are arbitrarily appended to each to distinguish them from each other.
3. This operation is then repeated for each reduced source until the original source is reached.



The average length of this code is

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

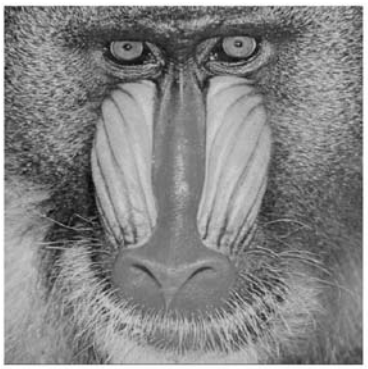
The entropy of the source is

$$H = -\sum_{j=1}^{6} P(a_j) \log_2 P(a_j) = 2.14 \text{ bits}$$

and the resulting code efficiency is

$$\eta = \frac{2.14}{2.2} = 0.973$$

Huffman's procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols be coded one at a time. After the code has been created, coding and/or decoding is accomplished in a simple look-up table manner. For natural images, Huffman coding will generally result in $\eta > 99\%$. It is used in JPEG and many other image and video compression standards.



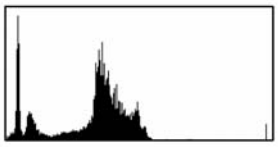
$H = 7.358$ bits

$\overline{L}$ (Huffman) $= 7.381$ bits

$\eta = 99.7\%$

CR $= 1.08$

Gray ...	Frequency	Huffman Code	Code Length
81	0.004665	00101101	8
82	0.004532	00110010	8
83	0.005005	00011100	8
84	0.005116	00001111	8
85	0.004925	00100010	8
86	0.004818	00100011	8
87	0.005314	00000110	8
88	0.005043	00011001	8
89	0.005199	00001110	8
90	0.005051	00011000	8
91	0.005302	00000111	8
92	0.005367	00000011	8
93	0.005367	00000010	8
94	0.005238	00001000	8
95	0.005501	1111110	7
96	0.005493	1111111	7
97	0.005482	00000001	8
98	0.005661	1111011	7



$H = 6.667$ bits

$\overline{L}$ (Huffman) $= 6.694$ bits

$\eta = 99.6\%$

CR $= 1.20$

Gray Level	Frequency	Huffman Code	Code Length
0	0.000029	101010000110110	15
1	0.000516	01101001000	11
2	0.001356	0000110100	10
3	0.000997	0110100110	10
4	0.001334	0000110110	10
5	0.002288	010000001	9
6	0.001879	011111011	9
7	0.002482	000111011	9
8	0.002052	011000010	9
9	0.003415	10101010	8
10	0.008637	0101000	7
11	0.026743	10110	5
12	0.036456	00111	5
13	0.028364	10011	5
14	0.008737	0100110	7
15	0.003171	11001110	8
16	0.002582	000100001	9
17	0.001564	110101110	9

Arithmetic coding

In arithmetic coding, a message is represented by a real floating point number between 0 and 1. We do not have a one-to-one correspondence between source symbols and code words.

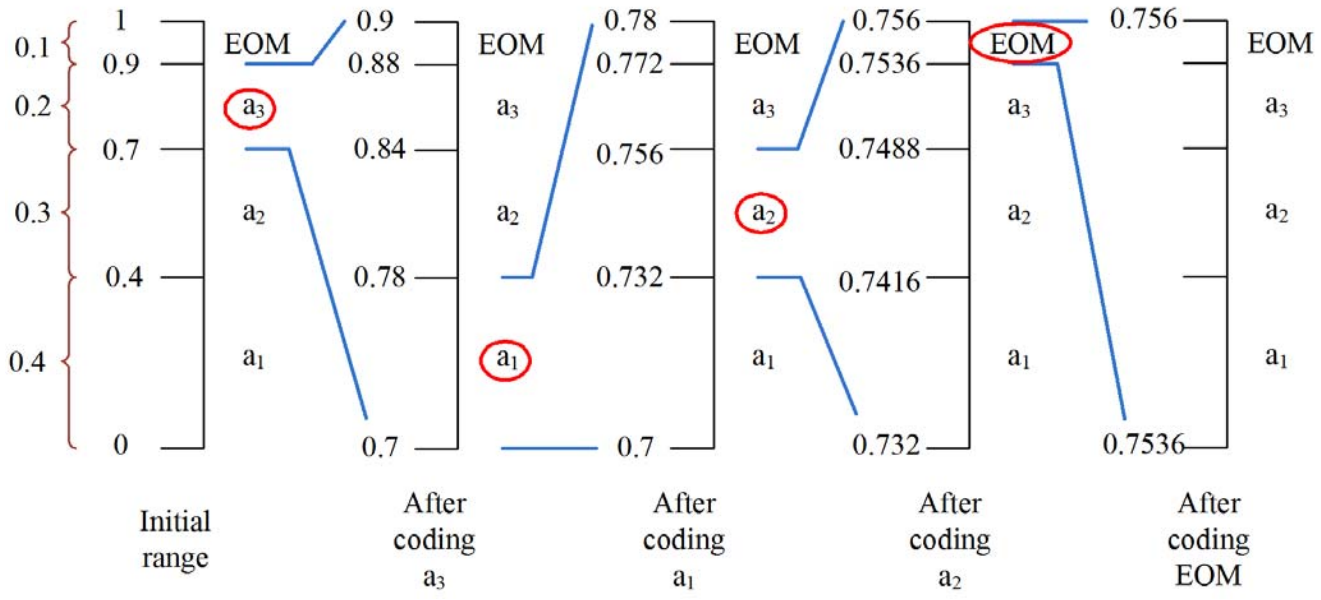
As the message becomes longer, the interval needed to represent it becomes smaller, and the number of bits needed to specify the interval grows. Successive symbols of the message reduce the size of the interval in accordance with the symbol probabilities generated by the model. The more likely symbols reduce the range by a smaller amount than the unlikely symbols do, and hence introduce fewer bits to the message.

Suppose the model consists of three symbols: a_1, a_2 and a_3. An additional symbol EOM is needed to signify the end of a message. The probabilities of the symbols are

Source Symbol	Probability	Initial Subinterval
a_1	0.4	[0.0, 0.4)
a_2	0.3	[0.4, 0.7)
a_3	0.2	[0.7, 0.9)
EOM	0.1	[0.9, 1.0)

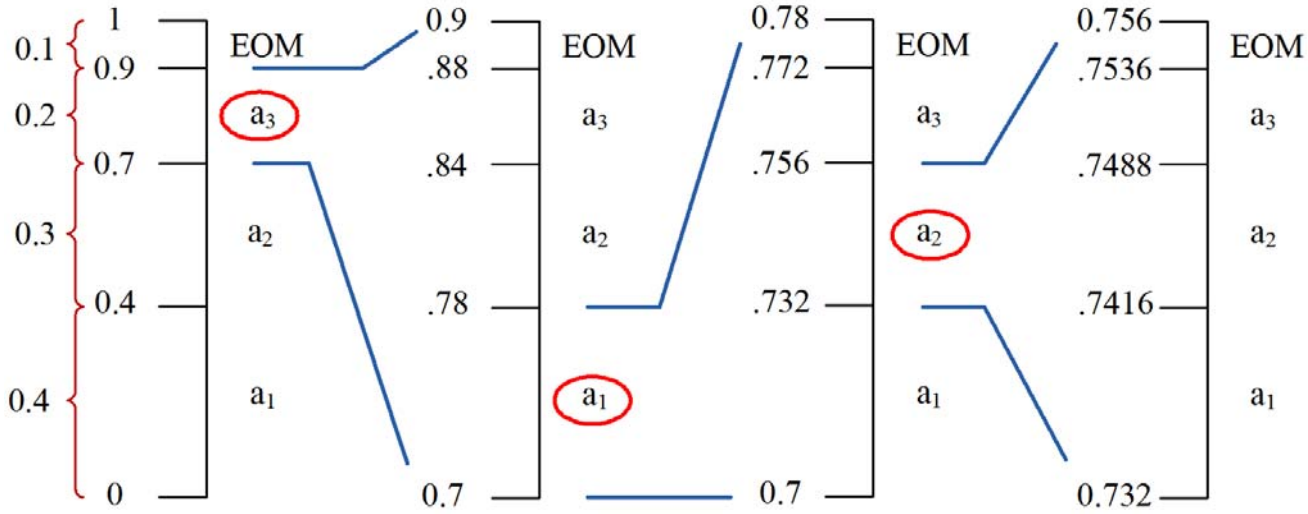
Suppose we wish to transmit the message $a_3a_1a_2$(EOM).

1. Initially, both encoder and decoder know that the range is $[0, 1)$.
2. After sending the first symbol a_3, the encoder narrows it to $[0.7, 0.9)$, the range that the model allocates to this symbol.
3. The second symbol a_1 will narrow this range to the first two-fifths of it, since a_1 has been allocated $[0, 0.4)$. This produces $[0.7, 0.78)$ since the previous range was 0.2 units long and two-fifths of that is 0.08.
4. The next symbol a_2 is allocated $[0.4, 0.7)$, which when applied to $[0.7, 0.78)$ gives the smaller range $[0.732, 0.756)$.
5. After coding the last symbol, EOM, the range is reduced to $[0.7536, 0.7560)$. Any number within this subinterval, say 0.754, can be used to represent the message.



Three decimal digits are used to represent the four-symbol message. This translates into 0.75 decimal digits per source symbol. In comparison, the entropy of the source is 0.56 decimal digits (or Hartleys).

To decode the message, all that the decoder needs to know is any number within the final range. On seeing the number (0.754), it can immediately deduce that the first symbol is a_3, since the range lies entirely within the intial subinterval allocated to a_3 by the model. It can now simulate the operation of the encoder to decode the message until the EOM symbol is decoded, which signifies the end of message.



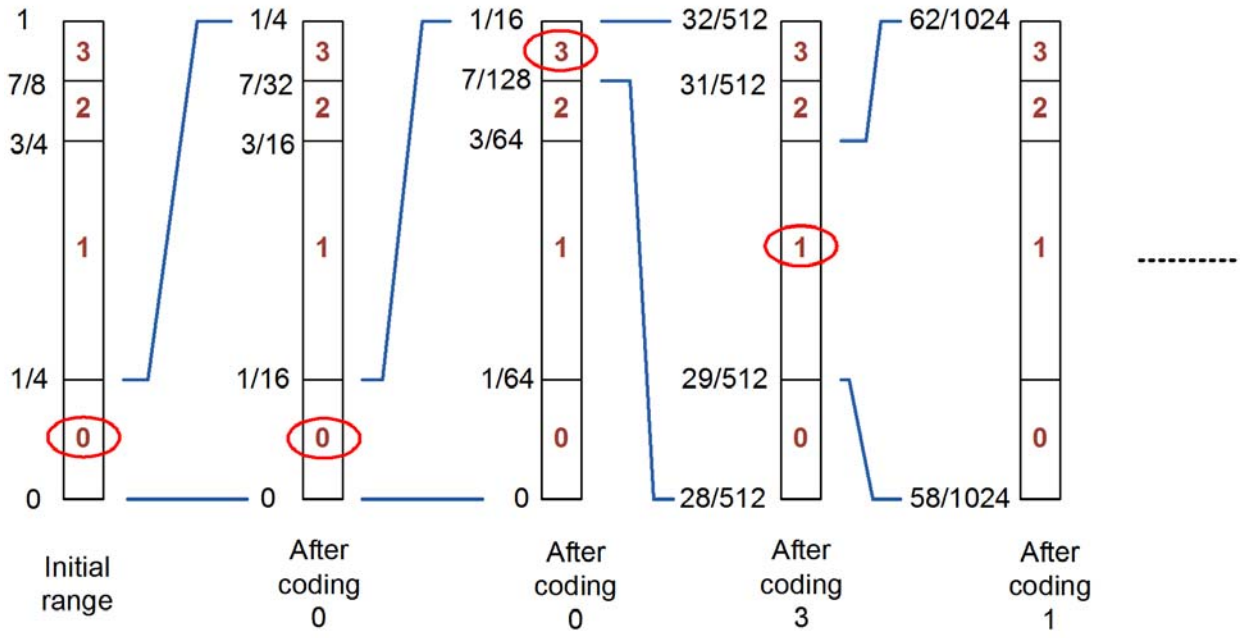
When applied to images, we need to know the histogram probability values. If the image size is known (or transmitted in advance), then the EOM symbol is not used. Arithmetic coding forms part of many image/video compression schemes, e.g., JPEG-2000.

Example

Consider a 4-level image with this histogram:

Gray level :	0	1	2	3
Probability:	1/4	1/2	1/8	1/8

Suppose we wish to transmit the pixel sequence 0031...



Huffman coding and arithmetic coding both require *a priori* knowledge of the probability of occurrence of the symbols to be encoded. There are techniques that do not have this requiement, e.g., the Lempel-Ziv-Welch (LZW) coding technique used in popular imaging file formats such as GIF and TIFF.

Bit-Plane Coding

Bit-plane coding is a technique for reducing an image's interpixel redundancies. It is based on the concept of decomposing a multilevel (monochrome or color) image into a series of binary images and compressing each binary image via one of several binary compression methods.

Bit-plane decomposition

The gray levels of an m-bit gray-scale image can be represented in the form of the base 2 polynomial:

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

In the case of an 8-bit image, we have

$$(a_7 \times 2^7) + (a_7 \times 2^7) + \ldots + (a_1 \times 2^1) + (a_0 \times 2^0)$$

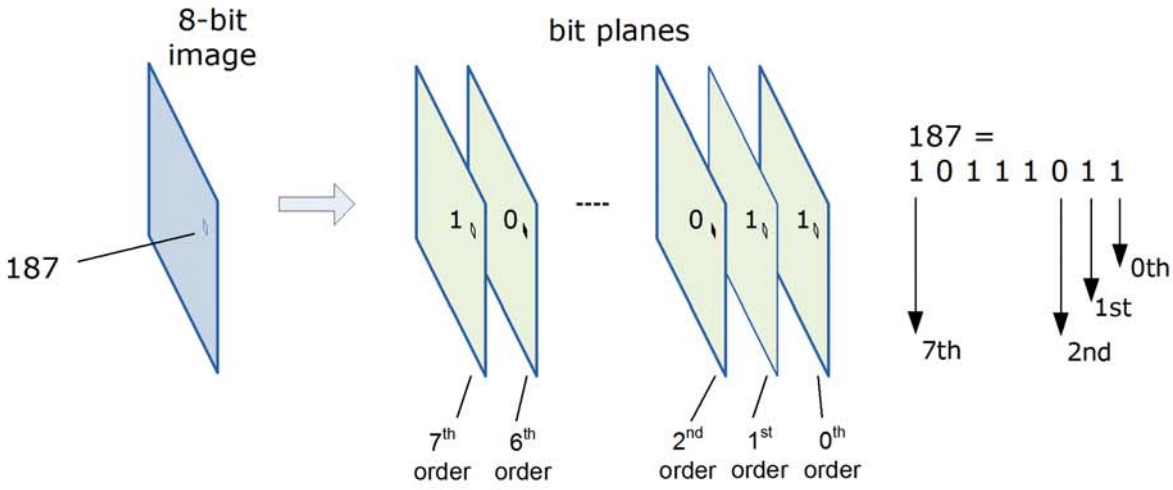
For example,

$$\begin{aligned} 255 &= 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ 187 &= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \end{aligned}$$

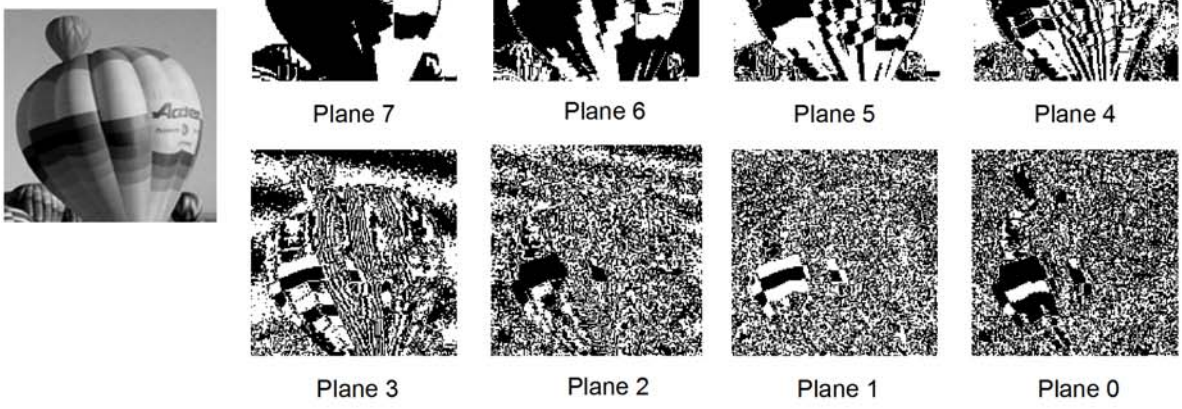
A simple method of decomposing the image into a collection of binary images is to separate the 8 coefficients of the polynomial into 8 1-bit *bit planes*.

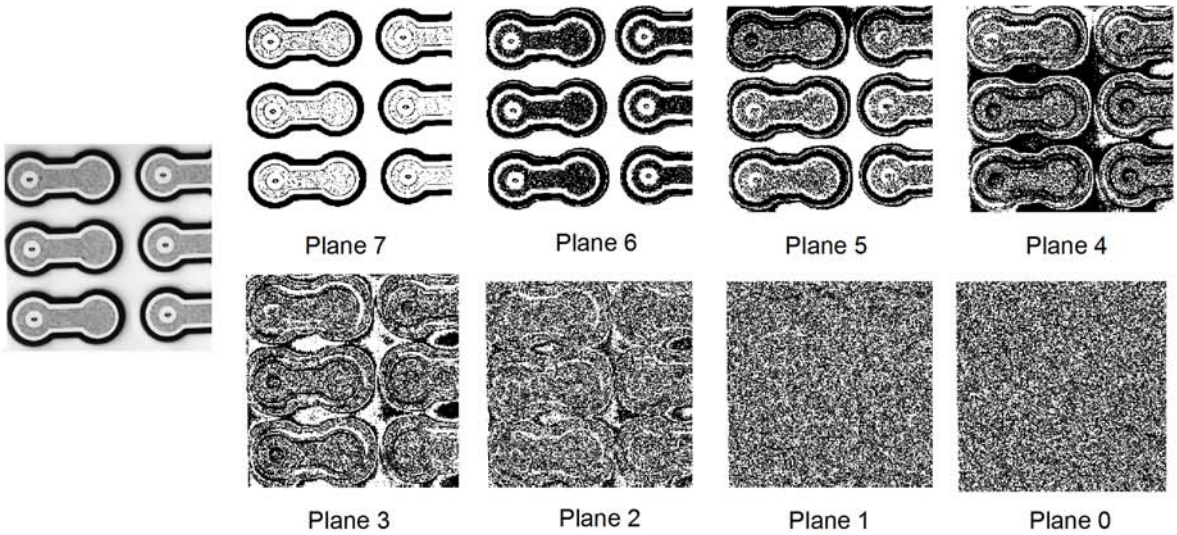
The zeroth-order bit plane is generated by collecting the a_0 bits of each pixel, while the 7th-order bit plane contains the a_7 bits or coefficients. In general, each bit plane is numbered from 0 to 7 and is constructed by setting its pixels equal to the values of the appropriate bits of polynomial coefficients from each pixel in the original image.

Each bit plane is a binary image which can be coded separately.


8-bit
image
187
bit planes
1
0

0
1
1
7th
order
6th
order
2nd
order
1st
order
0th
order
187 =
1 0 1 1 1 0 1 1
7th
2nd
1st
0th


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


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

One-dimensional run-length coding

In *run-length coding*, each row of an image or bit plane is represented by a sequence of lengths that describe successive runs of black and white pixels. We code each contiguous group of 0's or 1's encountered in a left to right scan of a row by its length. Two possible conventions for determining the value of the run are:

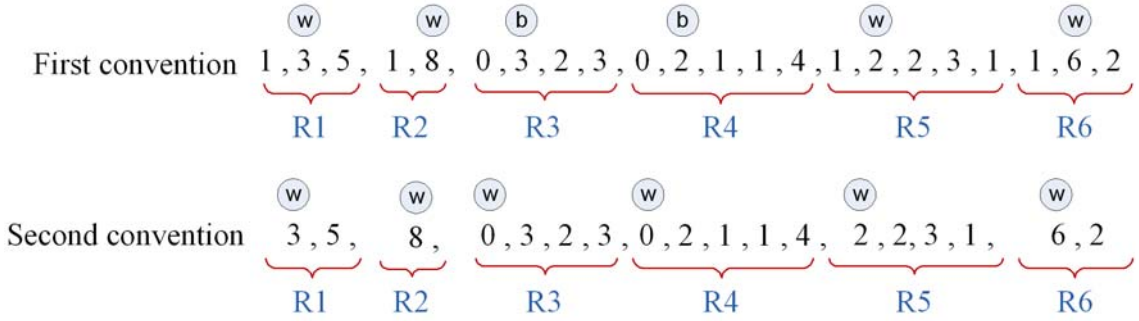
1. Specify the value of the first run of each row.
2. Assume that each row begins with a white run, whose run length may in fact be zero.

Run-length coding is used, e.g., in JPEG and BMP files.

Example

									Run lengths
R1	1	1	1	0	0	0	0	0	3,5
R2	1	1	1	1	1	1	1	1	8
R3	0	0	0	1	1	0	0	0	3,2,3
R4	0	0	1	0	1	1	1	1	2,1,1,4
R5	1	1	0	0	1	1	1	0	2,2,3,1
R6	1	1	1	1	1	1	0	0	6,2

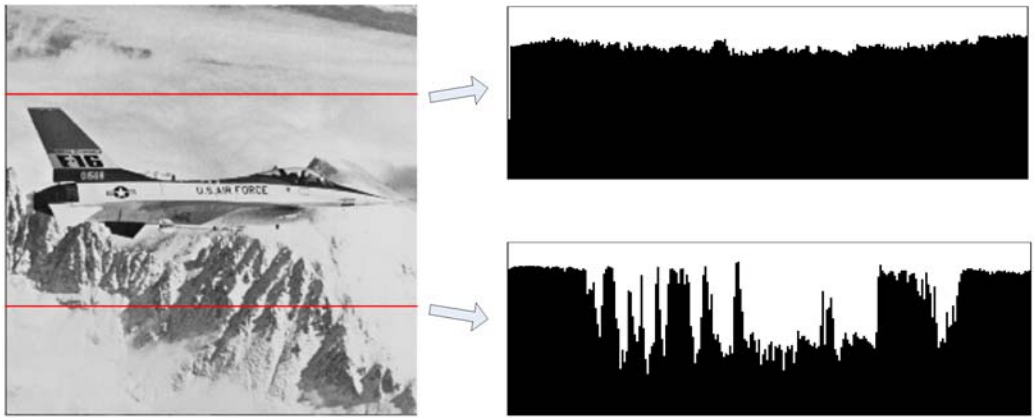
Run length code



Additional compression may be obtained by applying variable-length coding on the run lengths themselves.

Delta Compression

In delta compression (or differential coding), we code an image in terms of the difference in gray level between each pixel and the previous pixel in the row.



The first pixel is represented by its absolute value, but subsequent values are represented as differences, or "deltas". Most of these differences are very small because in typical images, gray level usually changes gradually rather than abruptly. These small differences can be coded using fewer bits. Thus delta compression exploits interpixel redundancy to create coding redundancy.

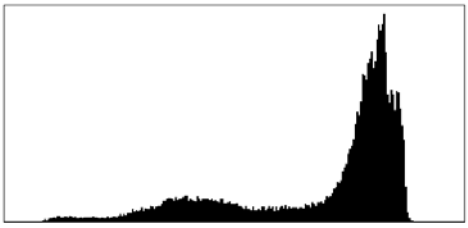
50	57	59	51	48	49	52	52
66	40	48	40	39	46	60	55
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Original image

50	+7	+2	-8	-3	+1	+3	0
+14	-26	+8	-8	-1	+7	+14	-5
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

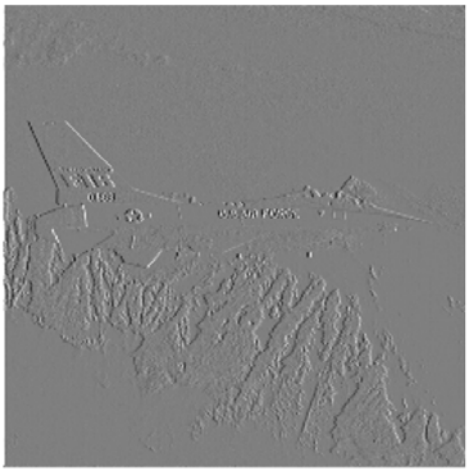
"Difference" image

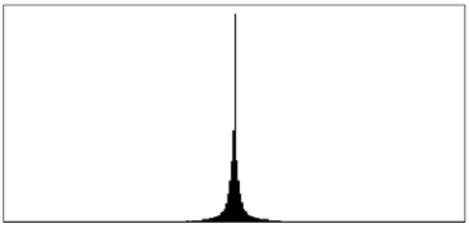
Another popular error-free compression technique is *lossless predictive coding*.



$H = 6.713; \overline{L}_{Huff} = 6.744$

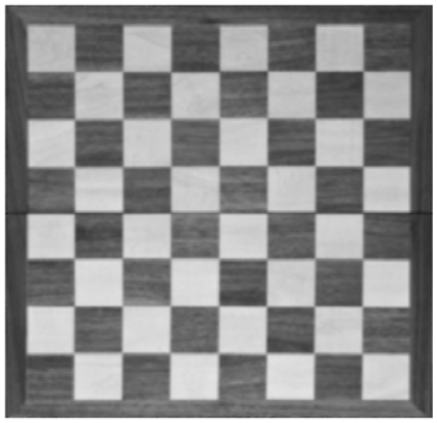
CR = 1.2

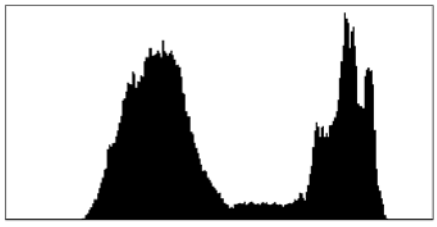




$H = 4.326; \overline{L}_{Huff} = 4.360$

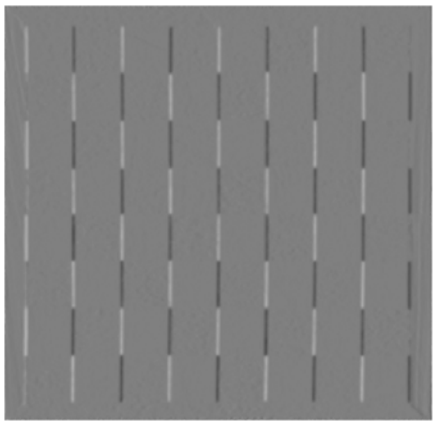
CR = 1.8

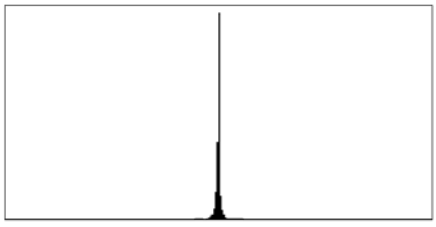




$H = 7.185; \overline{L}_{Huff} = 7.223$

CR = 1.1





$H = 2.951, \overline{L}_{Huff} = 2.988$

CR = 2.7

LOSSY COMPRESSION

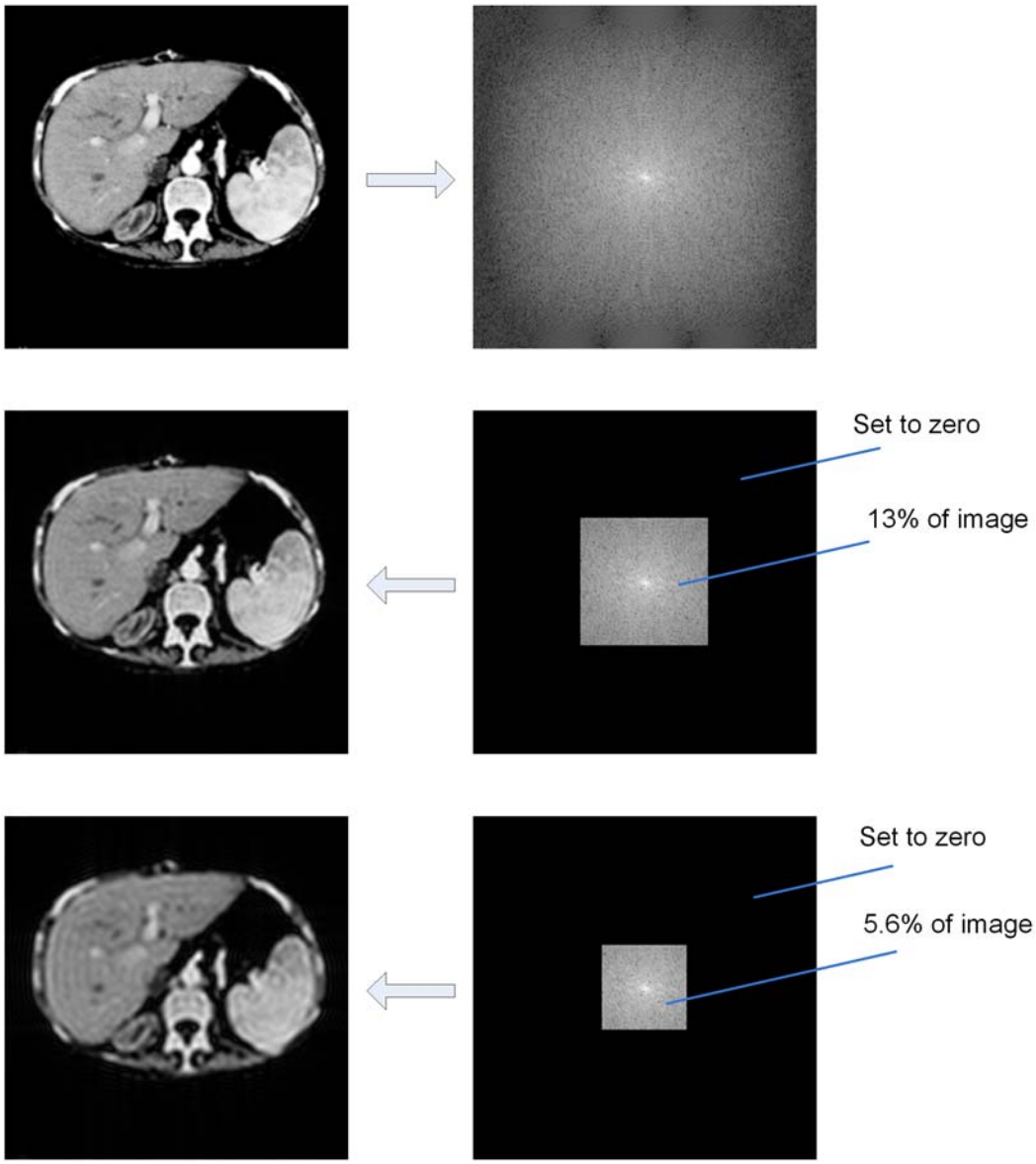
Lossy compression is based on the concept of compromising the accuracy of the reconstructed image in exchange for increased compression. Many lossy encoding techniques are capable of reproducing recognizable monochrome images from data that have been compressed by $> 30 : 1$ and images that are virtually indistinguishable from originals at 10:1 to 20:1. Error-free encoding of monochrome images, however, seldom results in more than a 3:1 reduction in data. The principal difference between these two approaches is the presence or absence of the quantizer block.

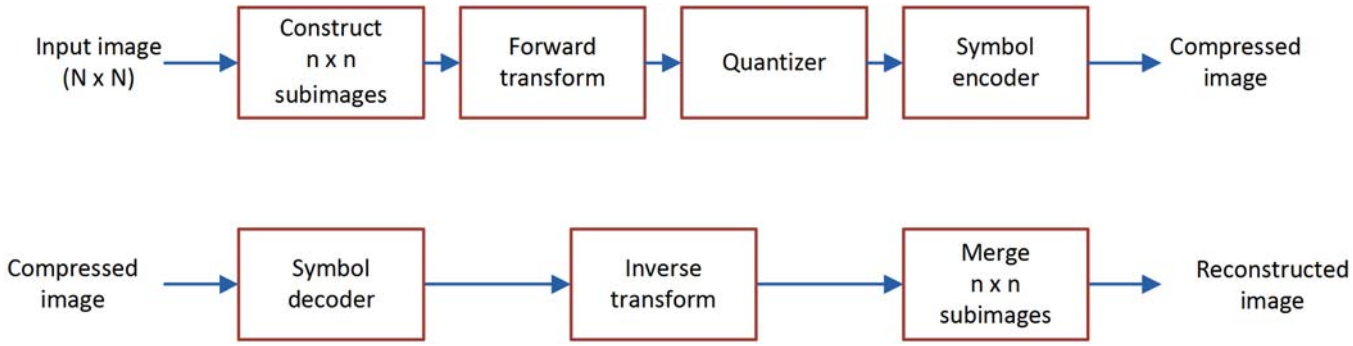
Common methods of lossy compression include

- lossy predictive coding
- optimal quantization
- transform coding

Transform Coding

In *transform coding*, a linear transform such as the Karhunen-Loeve transform (KLT), discrete Fourier transform (DFT), discrete cosine transform (DCT) or the wavelet transform, is used to map the image into a set of transform coefficients, which are then quantized and coded. The goal of the transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into the smallest number of transform coefficients.





A typical transform coding system comprises several stages: subimage decomposition, transformation, quantization and coding.

An $N \times N$ input image is subdivided into subimages of size $n \times n$, which are then transformed to generate $(N/n)^2$ subimage transform arrays.

The quantization stage then selectively eliminates and more coarsely quantizes the coefficients that carry the least information. The encoding process terminates by coding (normally using a variable-length code) the quantized coefficients.

Compression is achieved during the quantization of the transformed coefficients (and not during the transformation step).

Subimage size selection affects transform coding error and computational complexity. In most applications, images are subdivided so that the correlation (redundancy) between adjacent subimages is reduced to some acceptable level, and so that n, the subimage dimension, is an integer power of 2. In general, both the level of compression and computational complexity increase as the subimage size increase as n increases. The most popular subimage sizes are 8×8 and 16×16.

JPEG Compression

Original

1:1

7:1

12:1

18:1

23:1

JPEG
(41:1)

JPEG 2000
(41:1)