6 – IMAGE COMPRESSION (B)

ERROR-FREE COMPRESSION

In many applications, error-free compression is the only acceptable means of data compression. Examples are

- archival of medical or business documents
- processing satellite imagery
- digital radiography

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

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

Compression ratios of 2 to 10 are generally achievable.

Variable-length Coding

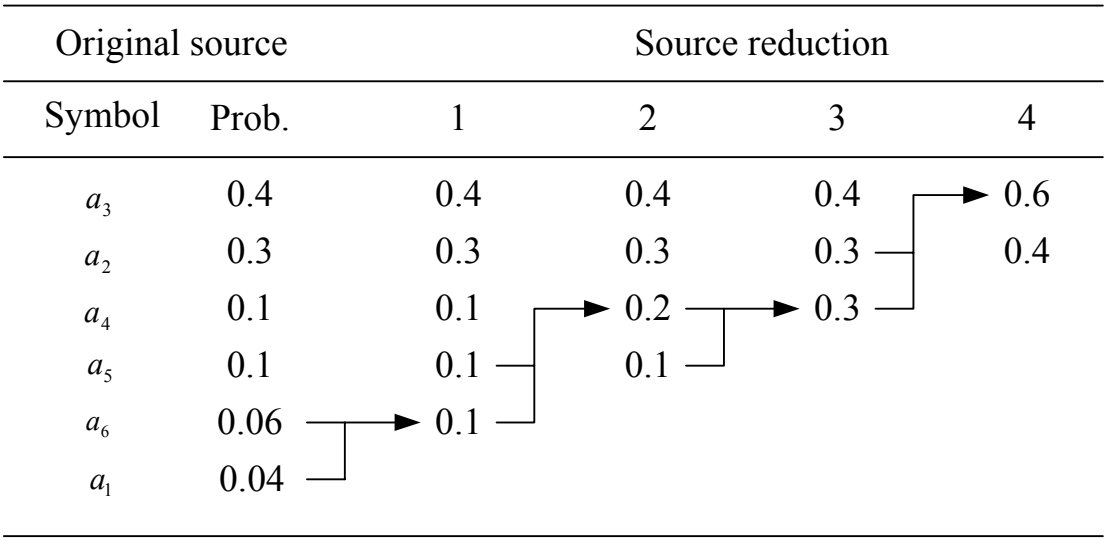
Coding redundancy normally is present in any natural binary encoding of the gray levels in an image. It can be eliminated by coding the gray levels so that L_{avg}, the average number of bits required to represent each pixel, is minimized. To do so requires construction of a variable-length code that 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.

Original source		Source reduction			
Symbol	Prob.	1	2	3	4
a_3	0.4 1	0.4 1	0.4 1	0.4 1	0.6 0
a_2	0.3 00	0.3 00	0.3 00	0.3 00	0.4 1
a_4	0.1 011	0.1 011	0.2 010	0.3 01	
a_5	0.1 0100	0.1 0100	0.1 011		
a_6	0.06 01010	0.1 0101			
a_1	0.04 01011				

The average length of this code is

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

The entropy of the source is

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

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.

- The code is a *block code*, because each source symbol is mapped into a fixed sequence of code symbols.
- It is *instantaneous*, because each code word in a string of code symbols can be decoded without referencing succeeding symbols.
- It is *uniquely decodable*, because any string of code symbols can be decoded in only one way. Any string of Huffman-encoded symbols can be decoded by examining the individual symbols of the string in a left to right manner. For the binary code in the example, a left-to-right scan of the encoded string 010100111100 reveals that the first valid code is 01010 (a_6). The next valid code is 011 (a_4). The completely decoded message is $a_6 a_4 a_3 a_3 a_2$.

For the general case of J source symbols, $J-2$ source reductions must be performed and $J-2$ code assignments made. Thus construction of the optimal Huffman code for an image with 256 gray levels requires 254 source reductions and 254 code assignments. In view of the computational complexity, other near-optimal schemes have been devised; these sacrifice coding efficiency for simplicity in code construction.

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.

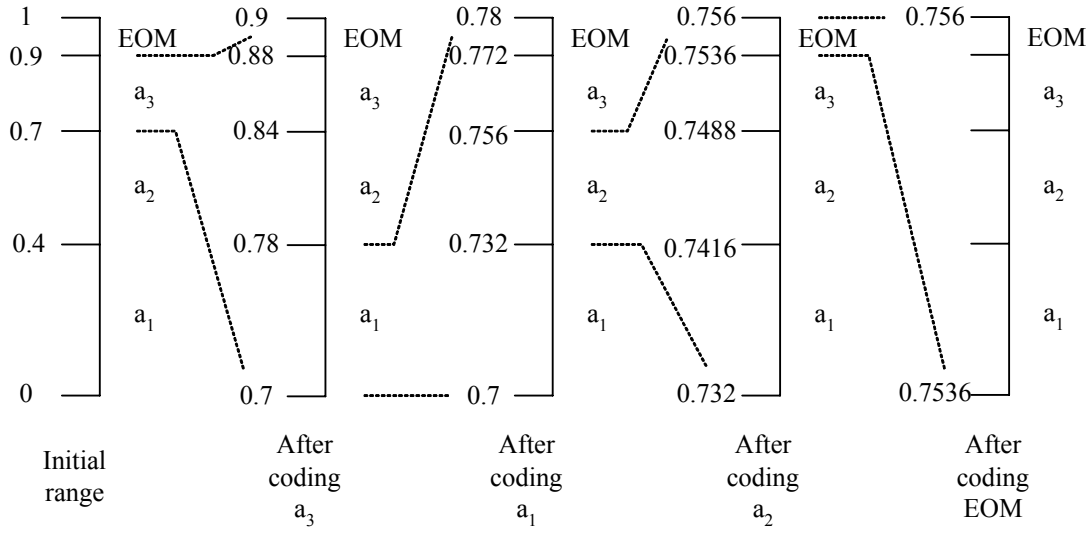
As the message becomes longer, the interval needed to represent it becomes smaller, and the number of bits needed to specify the interval grows. Successive symbols of the message reduce the size of the interval in accordance with the symbol probabilities generated by the model. The more likely symbols 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)

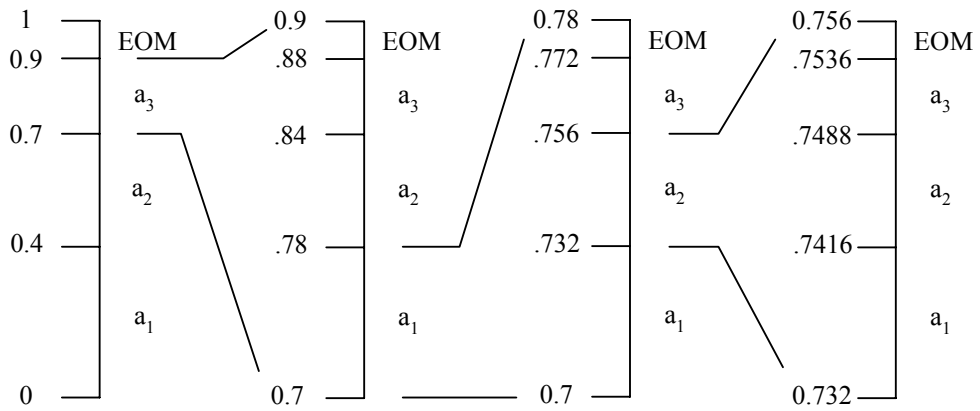
Suppose we wish to transmit the message $a_3a_1a_2$(EOM).

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 10-ary units/symbol.

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.

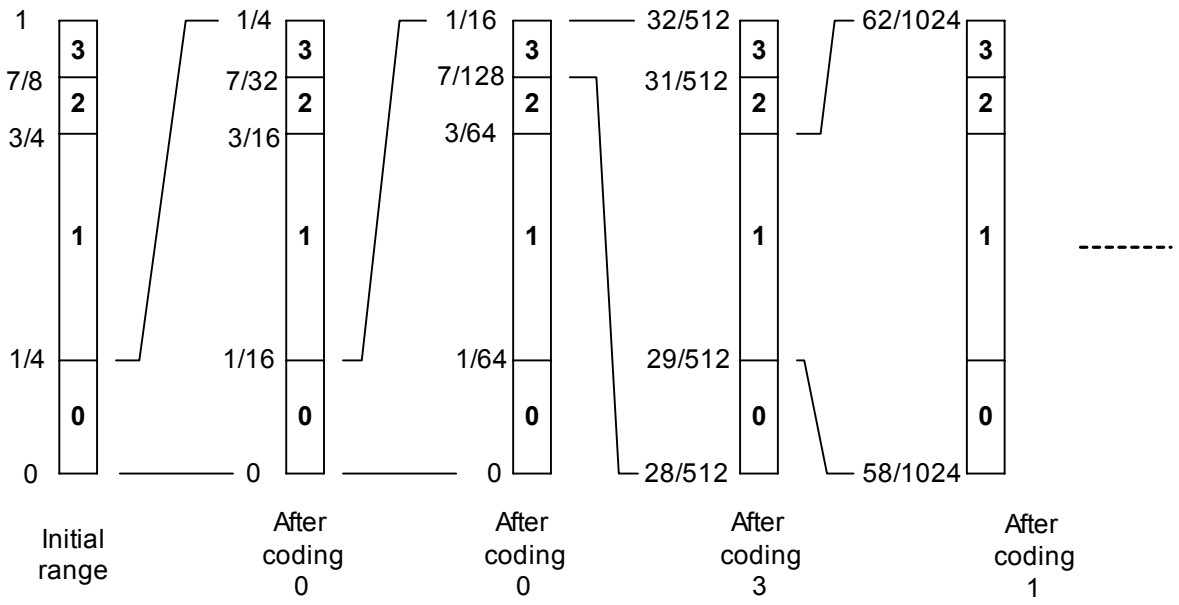


When applied to images, we need to know the histogram probability values. If the image size is known (or transmitted in advance), then the EOM symbol is not used. In practice, arithmetic coding is not used by itself but used as part of an image compression scheme.

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...



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:

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

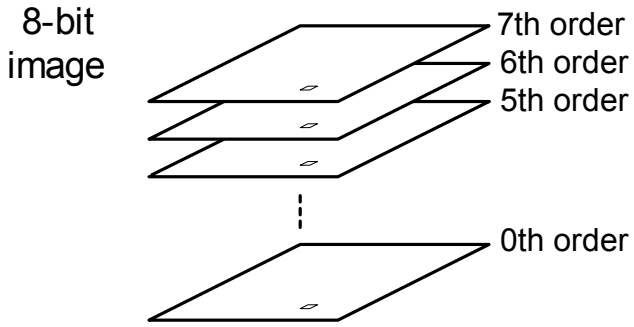
For example,

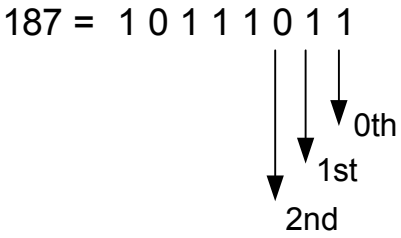
$$\begin{aligned} 255 &= 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ 187 &= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \end{aligned}$$

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

The zeroth-order bit plane is generated by collecting the a_0 bits of each pixel, while the $(m-1)$th-order bit plane contains the a_{m-1} bits or coefficients. In general, each bit plane is numbered from 0 to $m-1$ 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.





Original

Plane 7

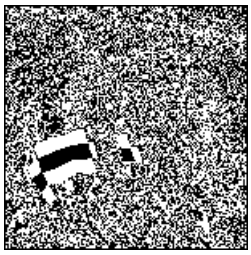
Plane 6

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.

Example

$$\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{array}$$

Using the first convention, the run length code is

$$\underbrace{1,8,}_{R1}\ \underbrace{0,8,}_{R2}\ \underbrace{1,2,1,2,3,}_{R3}\ \underbrace{0,1,1,1,5,}_{R4}\ \underbrace{1,1,1,5,1}_{R5}\ \underbrace{1,6,2}_{R6}$$

(Rn is row n.)
Using the second convention, the run length code is

$$\underbrace{8,}_{R1}\ \underbrace{0,8,}_{R2}\ \underbrace{2,1,2,3,}_{R3}\ \underbrace{0,1,1,1,5,}_{R4}\ \underbrace{1,1,5,1}_{R5}\ \underbrace{6,2}_{R6}$$

Additional compression can usually be realized by variable-length coding the run lengths themselves. In fact, the black and white run lengths may be coded separately using variable-length codes that are specifically tailored to their own statistics.

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.

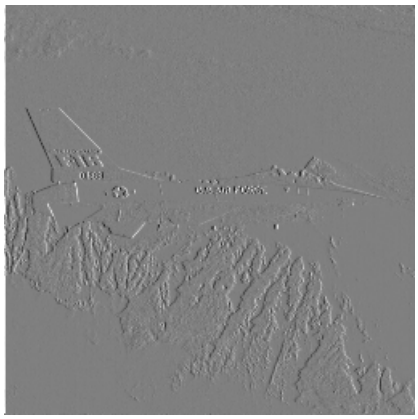
We can code the image below by using four bits to code differences in the range $R = [-7, +7]$. These four bits give us 16 (2^4) codewords, but there are only 15 values in the range. The remaining codeword can be used to flag pixels for which the gray-level difference exceeds the range. The values of these pixels are output using the full eight bits.

For this image, it turns out that difference values in the range R comprise 84% of the total number, N. The number of bits required to encode the difference image can be computed as follows.

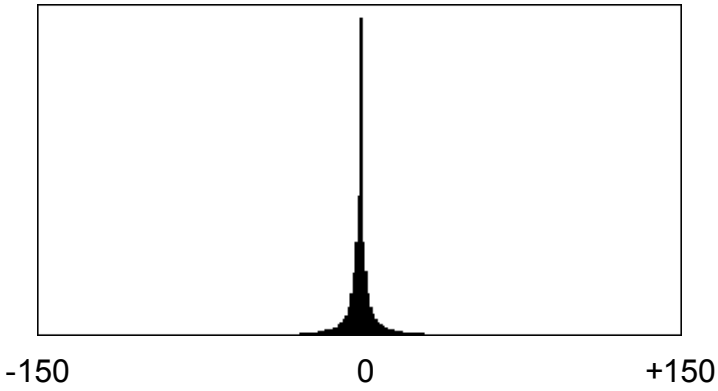
Pixels in range R : $0.84N \times 4 = 3.36N$ bits
Pixels outside range R: $0.16N \times 12 = 1.92N$ bits

$$\text{Compression ratio } = \frac{8N}{3.36N + 1.92N} = 1.52$$

Original image



Difference image - image showing gray-level difference of adjacent pixels. (Gray-levels offset by +128.)



Histogram of gray-level differences.

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.

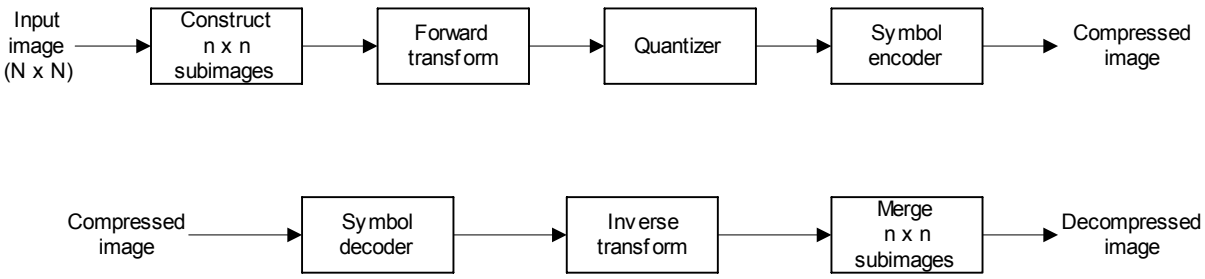
Transform Coding

In *transform coding*, a reversible, linear transform (such a the Fourier transform) is used to map the image into a set of transform coefficient, which are then quantized and coded.

In a typical transform coding system, the encoder performs four relatively straightforward operations: 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 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.

The quantization stage then selectively eliminates or 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.



Transform coding based on the Karhunen-Loeve (KLT), discrete Fourier (DFT), discrete cosine (DCT), Walsh-Hadamard (WHT), and various other transforms have been constructed and studied extensively. The choice of a particular transform in a given application depends on the amount of reconstruction error that can be tolerated and the computational resources available. Compression is achieved during the quantization of the transformed coefficients (and not during the transformation step).

- The nonsinusoidal transforms (such as the WHT or Haar transform) are the simplest to implement.
- The KLT minimizes the mean-square error for any input image and any number of retained coefficients. However, because the KLT is data dependent, its implementation is a nontrivial computational task. Hence, the KLT is seldom used in practice.
- In terms of information-packing ability, the DCT is generally superior superior to the DFT and WHT. Most practical transform coding systems are based on the DCT, which provides a good compromise between information-packing ability and computational complexity.

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.