Al-Mustaqbal University College

Department of Computer Techniques Engineering

Multimedia Computing

Lecturer : Ali Al-Safi

2022/2021

18 & 19. Image compression techniques

Image compression is a type of data compression applied to digital images, to reduce their cost for **storage or transmission**. Data compression refers to the process of reducing the amount of data required to represent a given quantity of information.

- The reduced file is called the compressed file and is used to reconstruct the image
- Resulting in the decompressed image is the original image, is called the uncompressed image file.

$$Compression\ Ratio = \frac{Uncompressed\ File\ Size}{Compressed\ File\ Size} = \frac{size_U}{size_C}$$

Example: the original image is 256×256 pixel, single band (gray scale), 8-bit per pixel. This file is 65,536 bytes (64K). After compression the image file is 6,554 byte. The compression ratio is:

SizeU/SizeC = 65536/6554= 9.999=10 this can be written as

10:1

This is called a "10 to 1" compression or a "10 times compression", or it can be stated as "compressing the image to 1/10 original size.

Image compression types

There are two primary types of image compression methods and they are:

1. Lossless Compression

- This compression is called lossless because no data are lost (with no loss in image quality)

- And the original image can be recreated exactly from the compressed data.
- Lossless compression is generally used for text or spreadsheet files, where losing words or financial data could pose a problem
- The Graphics Interchange File (GIF) is an image format used lossless compression

2. *Lossy Compression.*

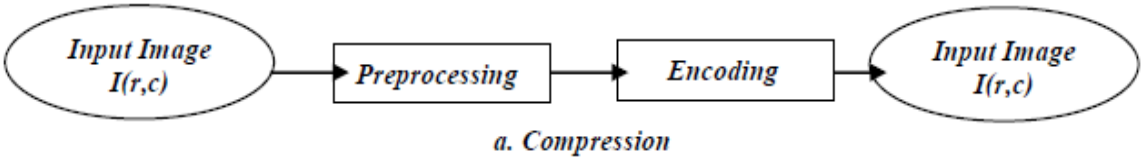
- Lossy Compression is the class of data encoding methods that allows a loss in some of data images.
- The uncompressed image cannot be same the original image file.
- Lossy methods can provide high degrees of compression and result in smaller compressed files, but some number of the original pixels, sound waves or video frames are removed forever.

Compression System Model

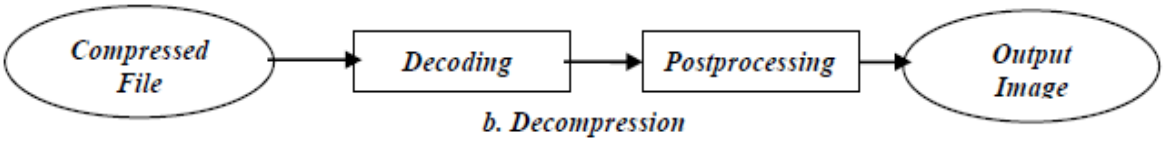
The compression system model consists of two parts: the compressor (Encoding) and the decompressor (Decoding).

Compressor: consists of preprocessing stage and encoding stage.

Decompressor: consists of decoding stage followed by a post processing stage, as following figure:



a. Compression



b. Decompression

Figure (1): Compression System Model.

Before encoding, *preprocessing* is performed to prepare the image for the encoding process. After the compressed file has been decoded, *post processing* can be performed to eliminate some of the undesirable artifacts brought about by the compression process. Often, many practical compression algorithms are a combination of a number of different individual compression techniques.

Entropy

An important concept here is the idea of measuring the average information in an image, referred to as entropy. The entropy of N×N image can be calculated by this equation.

$$Entropy = -\sum_{i=0}^{L-1} P_i \log_2(P_i) \qquad \text{(in bits per pixel)}$$

Where

P_i = The probability of the i_{th} gray level n_k/N^2

n_k= the total number of pixels with gray value k.

L= the total number of gray levels (e.g. 256 for 8-bits)

Example

Let L=8, meaning that there are 3 bits/ pixel in the original image. Let that number of pixel at each gray level value are equal (they have the same probability) that is:

$$P_0=P_1=P_2=\ldots\ldots=P_7=1/8$$

Now, we can calculate the entropy as follows:

$$Entropy = -\sum_{i=0}^{7} P_i \log(P_i) \quad = -\sum_{i=0}^{7} \frac{1}{8} \log_2(\frac{1}{8}) = 3$$

This tell us that the theoretical minimum for lossless coding for this image is 8 bit/pixel

Note:*Log₂(x) can be found by taking Log₁₀(x) and multiplying by 3.33 bacuase 1/Log₁₀(2) =3.32192809488736234*

Lossless Compression Algorithms

1. Run-length Encoding (RLE)

- This encoding method is frequently applied to graphics-type images (or pixels in a scan line) - simple compression algorithm in its own right.
- It is also a component used in JPEG compression pipeline.
- Example: Original sequence: 111122233333311112222

  can be encoded as: (4,1),(3,2),(6,3),(4,1),(4,2)

How Much Compression?The savings are dependent on the data: In the worst case (random noise) encoding is heavier than original file.

2. Simple Repetition Suppression

- Replace series with a token and a count number of occurrences.
- Usually need to have a special flag to denote when the repeated token appears.
- Simplicity is its downfall: poor compression ratios.
- Compression savings depend on the content of the data.

Example:

894000000000000000000000000000000

We can replace with: **894f32** /where f is the flag for zero.

3. Pattern Substitution

- This is a simple form of statistical encoding.
- Here we substitute a frequently repeating pattern(s) with a code.
- *The code is shorter than the pattern giving us compression.*
- The simplest scheme could employ predefined codes:

Example: Basic Pattern Substitution

Replace all occurrences of pattern of characters `and' with the predefined code '&', So: ***and you and I*** becomes: **& you & I**

4-Shanon-Fano Method

A Shannon–Fano tree is built according to a specification designed to define an effective code table. The actual algorithm is simple:

1. For a given list of symbols, develop a corresponding list of probabilities or frequency counts so that each symbol's relative frequency of occurrence is known.
2. Sort the lists of symbols according to frequency, with the most frequently occurring symbols at the left and the least common at the right.
3. Divide the list into two parts, with the total frequency counts of the left part being as close to the total of the right as possible.
4. The left part of the list is assigned the binary digit 0, and the right part is assigned the digit 1. This means that the codes for the symbols in the first part will all start with 0, and the codes in the second part will

all start with 1.

5. Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding code leaf on the tree.

Example: The source of information A generates the symbols {A0, A1, A2, A3 and A4} with the corresponding probabilities {0.4, 0.3, 0.15, 0.1 and 0.05}. Encoding the source symbols using binary encoder and Shannon-Fano encoder gives:

Source Symbol	P_i	Binary Code	Shannon-Fano
A0	0.4	000	0
A1	0.3	001	10
A2	0.15	010	110
A3	0.1	011	1110
A4	0.05	100	1111
L_{avg}	**H = 2.0087**	**3**	**2.05**

The Entropy of the source is:

$$H = -\sum_{i=0}^{4} Pi \log_2 Pi = 2.0087 \text{ bit/symbol}$$

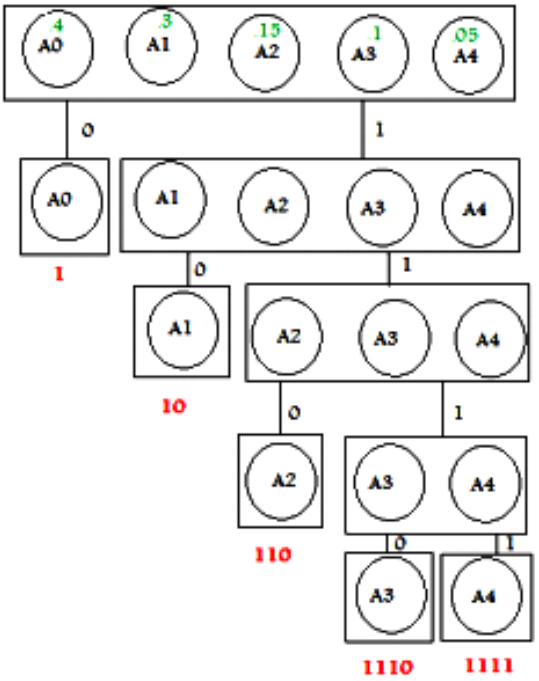
Since we have 5 symbols ($5<8=2^3$), we need 3 bits at least to represent each symbol in binary (fixed-length code). Hence the average length of the binary code is

$$Lavg = \sum_{i=0}^{4} Pi\, li = 3\,(0.4 + 0.3 + 0.15 + 0.1 + 0.05) = 3 \text{ bit/symbol}$$

Thus the efficiency of the binary code is

$$\eta = \frac{H}{Lavg} = \frac{2.0087}{3} = 67\%$$

Shannon-Fano code is a top-down approach. Constructing the code tree, we get



The average length of the Shannon-Fano code is

$$\text{Lavg} = \sum_{i=0}^{4} \text{Pi li} = 0.4 * 1 + 0.3 * 2 + 0.15 * 3 + 0.1 * 4 + 0.05 * 4 = 2.05 \text{ bit/symbol}$$

Thus the efficiency of the Shannon-Fano code is

$$\eta = \frac{H}{\text{Lavg}} = \frac{2.0087}{2.05} = 98\%$$

This example demonstrates that the efficiency of the Shannon-Fano encoder is much higher than that of the binary encoder.

Huffman Code

The Huffman coding algorithm comprises two steps, reduction and splitting. These steps can be summarized as follows:

1) Reduction

a) List the symbols in descending order of probability.

b) Reduce the *r* least probable symbols to one symbol with a

probability equal to their combined probability.

c) Reorder in descending order of probability at each stage.

d) Repeat the reduction step until only two symbols remain.

2) Splitting

a) Assign *r* ,... 1, 0 to the *r* final symbols and work backwards.

b) Expand or lengthen the code to cope with each successive split.

Example: Design Huffman codes for A = {$a1$, $a2$, … … . $a5$},having the probabilities

{0.2, 0.4, 0.2, 0.1, 0.1}.

Symbol	Step 1	Step 2	Step 3	Step 4	Codeword
a_2	0.4 →	0.4 →	0.4 →	0.6 0	1
a_1	0.2 →	0.2 →	0.4 0	0.4 1	01
a_3	0.2 →	0.2 0	0.2 1		000
a_4	0.1 0	0.2 1			0010
a_5	0.1 1				0011

Letter	Probability	Codeword
a_2	0.4	1
a_1	0.2	01
a_3	0.2	000
a_4	0.1	0010
a_5	0.1	0011

The average code word length:

$$Lavg = \sum_{i=0}^{4} Pi\ li$$

$$L_{avg} = \sum_{i=0}^{4} P_i l_i$$

L = 0.4 × 1 + 0.2 × 2 + 0.2 × 3 + 0.1 × 4 + 0.1 × 4 = **2.2 *bits/symbol***

The source entropy:

$$\boldsymbol{H(S) = \sum p_i \log_2 \frac{1}{p_i}} \text{or} \qquad \boldsymbol{H(S) = -\sum p_i \log_2 p_i}$$

(Y) = −[0.4ln0.4 + 2 × 0.2ln0.2 + 2 × 0.1ln0.1]/ln2 = **2.12193 bits/symbol**

The code efficiency:

$$\eta = \frac{\mathrm{H}}{\mathrm{Lavg}}$$

η =(2.12193 / 2.2)× 100 = 96.45%