

Comparison between Image Compression Algorithms

Abdullatif zankawi. adel Alateyah

ABSTRACT

Data compression is an essential part of computerized applications and processes. Images make up a significant portion of transmitted and stored data. There are various ways of image compression that have been studied and improved upon. This paper focuses on three main image compression algorithms: Run Length Encoding, Huffman Encoding, and JPEG compression. While there are a large number of available image compression algorithms, each of the aforementioned methods is selected due to their widespread usage, compression characteristics, and representation of different aspects in image compression. These three compression techniques are discussed and compared with each other, highlighting advantages and optimal conditions.

Keywords: *Image Compression, Run Length Encoding, Huffman Encoding, JPEG Compression*

Date of Submission: 13-12-2017

Date of acceptance: 22-12-2017

I. INTRODUCTION

In today's highly digital world, the exchange of information has evolved into various innovative methods and practices. As such, the accurate and efficient transfer of data has become a primary concern. The data can be transmitted through other forms, such as text or sound files, but one of the most common file type is the image. Digital images are used in various fields, including storage of imaged medical records and archiving copies of evidence for forensic and federal purposes. In web pages, majority of the downloaded bytes are constituted by images (Grigorik) which take up the clients' bandwidths, slowing down the browser's speed in rendering the site's content.

In order to transmit images in a more cost-effective manner, several streamlining techniques have been applied to these files. One such optimization is to reduce the size of an image through compression. Image compression is defined as the application of an encoding method onto an image such that it takes up less storage space than the original file. Using compression, the redundancy within the image is reduced, allowing for more efficient data storage and transmission (Wei).

Image optimization is a fusion of art and science, the search for a technique to compress a file given a specific image. Since a universal compressor does not exist, compression techniques must adapt to their input. There are various aspects within a file that have to be taken into consideration, including format, quality, dimensions, and image content (Grigorik).

II. OBJECTIVE

The objective of this paper is to discuss three primary image compression algorithms and to determine which methods are best suited to certain situations. In order to fully understand these algorithms, the underlying concepts of image compression must also be discussed. These include the redundancies found in image data, as well as the types of compression techniques available for use.

III. BACKGROUND

a. Principles of Image Compression

The process of image compression can be defined as the representation of an image in a more compact form. Compression is done with the goal of having a resulting image that consumes less storage space than the original file. This reduction is achieved through removal of information which are deemed unimportant or redundant, allowing the file to be represented with a smaller amount of bytes.

Redundancy is a key point in image compression. Redundancy may occur from the source or receiver of image information. Source images are known to have repeating chunks of data; groups of pixels or strings of information may be duplicated throughout the image file. As such, these redundancies can be removed in order to reduce the amount of information on the image file. The redundancy can also stem from the receiver of the information. If the receiver is not able to determine the difference between two data points, then these points should be considered as having the same value, which can thus be classified as a redundancy.

There are three basic ways in which compression can be attained through redundancy

removal. These methods either remove duplicate pixel information, reduce data representation strings, or integrate visually indistinguishable information.

Pixel Redundancy

Pixel redundancy, also known as spatial redundancy, relies on the statistical dependence of adjacent pixels with each other. In most images, it can be observed that the constituting pixels possess an evident correlation with their neighboring pixels. If data can be inferred from the previous pixel, then some part of the current pixel must contain redundant information, and is a candidate for data disposal. Reduction can be done by transforming the original data contained within the image file. Instead of holding its original information, an image pixel can instead be represented by the difference between itself and the previous pixel. From this compressed form, the image can be reconstructed using a reference pixel. Images which can be completely reconstructed through this method are noted as having a reversible mapping (Wahba and Maghari).

Coding Redundancy

The information contained within an image is represented through code - a collection of symbols each having a predefined value. Code words are used to define a specific portion of the image, often with a fixed length or number of bits. Often times, the pixel information takes up less space than the allotted number of bits on the storage. In order to reduce the file size, code words are abbreviated to represent more information. A common technique is to utilize lookup tables to map a set of code words to the original data. The process is streamlined by pulling statistics from the original file, such as the frequency of each block of information. Code words of different lengths is used, usually assigning shorter code words for information that recurs more often.

Visual Redundancy

Redundancy can also be noted from the receiver's standpoint. Majority of image files are viewed by humans, so in most cases the Human Visual System (HVS) acts as the signal receiver of the information (Dhawan). Human vision has been proven, through various scientific experiments, to respond with varying sensitivity to different aspects of visual information. The human eye is able to dictate which parts of an image are deemed unimportant to the scene's overall content, which can thus be discarded. Pixels which may have different encoded values or digital representations may be considered as virtually equal by the human visual system. In these cases, the information can be

classified as redundant, as there is fundamentally no difference from the receiver's point of view. Image compression takes advantage of this and attempts to reduce visually redundant information.

b. Types of Image Compression

Depending on the type of algorithm used, the image compression may be either lossless or lossy. As suggested by its name, lossless compression produces the image's original value - without any loss of data - once decompressed. This type of compression is often used for images that contain text data, wherein the exact representation of the original form is necessary. Lossy algorithms, on the other hand, decompress to an approximation of the image's original value. It is mostly applied on photo, audio, and video files, as loss of resolution or quality is less noticeable for these examples.

Lossless Image Compression

The first type of image compression is labeled as "lossless". The principal objective of lossless image compression is to reduce the number of bits that represent the image file in storage, without losing any data. When using a lossless compression technique, it is expected that the image's decompression process reconstructs the exact image before its compression. An illustration of lossless compression can be seen in the Morse Code, wherein each letter corresponds to a series of dots and dashes. A message encoded using the Morse Code will produce the exact same message once decoded.

Lossless image compression is often required in medical applications in order to prevent liabilities and disputes over incorrect data. A slight inaccuracy in the image rendered could cause crucial errors in diagnosis, thus the requirement for compression that fully preserves complete information. Images which are expected to undergo significant editing or multiple compressions also often use lossless compression. The number of accumulated errors through lossy changes or processes may exceed the acceptable range for compression.

In some cases, it is difficult to identify a pattern or portion of an image that can be discarded. Non-natural images, which often use indexed colors, are structured in such a way that even a minor error in value could translate into a substantial change in the color representation (Taubman and Marcellin). Examples of these are computer-generated text and graphic images such as lineart and comics, which are more convenient to represent using lossless compression techniques.

Lossy Image Compression

A lossy image compression identifies data which can be discarded with no significant impact on the final output. Essentially, pixel information which can be considered as “unimportant” (Mahoney) or irrelevant are removed when using this compression method. Lossy compression algorithms can be likened to creating an abridged version of a novel. The process identifies key points of the original data, generating a more compact book but with the necessary information given to the reader. Lossy compression is often useful in removing imperceptible details in audio and video clips. This technique is applied by older color televisions, wherein the color signal is transmitted with less resolution than its monochrome counterpart. This is due to the fact that the human eye is more sensitive to brightness than color differences, and thus would need less information to process the former than the latter. In lossy image compression, the original message cannot be completely reconstructed thus labelling it as an “irreversible compression” (Kodituwakku). However, studies within the lossy compression topic have been continuously aiming at “visually lossless” compression, which produces images at the compression quality of lossy compression but with the least amount of distortion.

The system for lossy image compression is composed three parts: the source encoder, the quantizer, and the entropy encoder. The source encoder transforms the original information of the image, often using a linear conversion such as the Discrete Fourier Transform (DFT). Then the quantizer applies further compression by reducing the precision of the values generated by the

transform. Finally, an entropy encoder, such as the Huffman encoder, creates a more compact version of these quantized values (Dhawan).

IV. IMAGE COMPRESSION ALGORITHMS

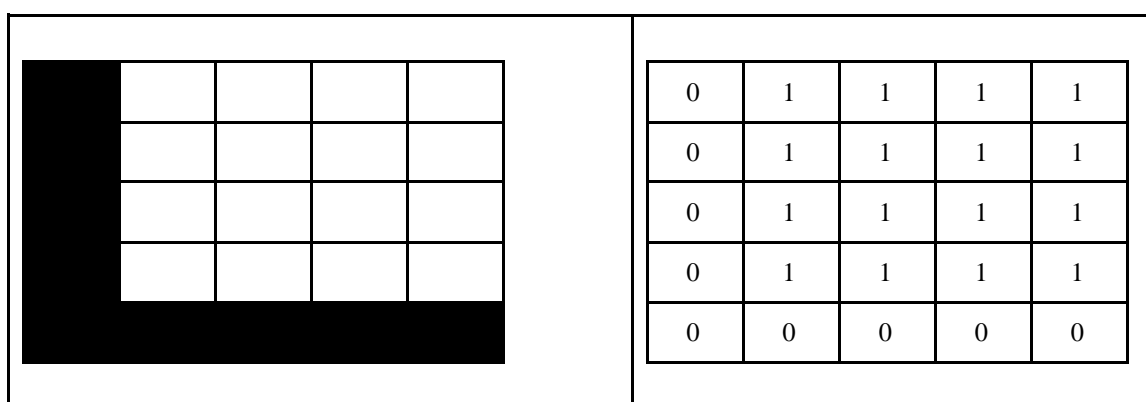
a. Run Length Encoding

One of the most basic methods of lossless compression is through Run Length Encoding. Given a sequence of data, RLE identifies any repeated bytes and represents the entire chunk with the repeated byte and its number of repetitions. Run Length Encoding algorithms are best suited for images with distinct portions having the same sample values. These images are often represented through “black” and “white” runs, wherein the state of the pixel is either 0 or 1.

Examples of these images include graphics, binary images, and images created through computer programs (Taubman and Marcellin). Essentially, image data containing minimal correlation between pixels can be compressed appropriately by the Run Length Encoding algorithm. Run Length Encoding is known to be inefficient at times, but is used by other algorithms as well.

Illustration

The algorithm used by Run Length Encoding checks for any repeating symbols and classifies these portions as “runs”. Therefore for any string of symbols, RLE produces a series comprised of runs and literal text, also referred to as “non-runs”. To illustrate, below is a sample representation of an image with a black X shape on a white background.



The image can be represented as 011110111101111011110000. Since Run Length Encoding identifies adjacent occurrences with equal value and replaces them with a single symbol and count, the resulting code of this image would be something like 0[1,4]0[1,4]0[1,4]0[1,4][0,5]. Note that the bits enclosed in brackets are those

representing the runs, while bits without brackets are the non-runs.

Implementation

Since the algorithm for Run Length Encoding is straightforward, it is one of the easiest compression methods to implement. Non-runs are

encoded as they are, while runs are converted into three-byte groups. The first byte is a predefined special character denoting that the group is a run, often the least used or non-existent symbol in the language set. The second byte is the character to be

repeated, while the third byte represents the number of times the character is to be repeated. For the example mentioned earlier, the actual encoding would be as follows:

0	*	1	4	0	*	1	4	0	*	1	4	0	*	1	4	*	0	5
run of 4 "1"s				run of 4 "1"s				run of 4 "1"s				run of 4 "1"s				run of 5 "0"s		

Assuming each symbol is allotted a byte, the size is reduced from the original 25 bytes (011110111101111011110000) to 19 bytes (0*140*140*140*14*05).

Due to its simplicity, Run Length Encoding can also prove to be inefficient in some cases. Take for example the following image and its representation in zeroes and ones.

	<table border="1"> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> </table>	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0																				
1	1	0	0	1	1																				
0	0	1	1	0	0																				
1	1	0	0	1	1																				

The original size is 24 bytes (00110011001100110011) while the output of the Run Length Encoding algorithm is 36 bytes (*02*12*02*12*02*12*02*12*02*12*02*12). It can be seen that coding characters with only two repetitions would actually increase the file size.

Encoding, the Huffman algorithm does pre-processing of the image data in order to assign the weights accurately. A binary tree is then constructed using this mapping, the process of which is expounded in the following section. As only the leaves of the binary tree are assigned values, the Huffman algorithm also removes the need for codeword boundaries, a common problem with variable-length codes (Wahba and Maghari).

b. HuffmanEncoding

Named after its developer, David Huffman, the Huffman Encoding algorithm is one of the most widely used component in compression techniques such as GZIP and JPEG (Blelloch). The basic concept of the Huffman Encoding algorithm is to represent more common data with shorter keywords in order to reduce the size of a file. It relies on a finite set of symbols mapped with their weights or probabilities within the given message. Unlike the previous compression technique, Run Length

Illustration

Consider an image composed of a finite set of colors, represented by the letters A to E in the image below. The Huffman Encoding algorithm expects a value associated to each symbol. In this case, the number of times the symbol appears in the message is used as the weight.

A	A	B	B	C	<table border="1"> <thead> <tr><th>Symbol</th><th>Weight</th></tr> </thead> <tbody> <tr><td>A</td><td>4</td></tr> <tr><td>B</td><td>2</td></tr> </tbody> </table>	Symbol	Weight	A	4	B	2
Symbol	Weight										
A	4										
B	2										
E	D	E	D	D							
D	C	A	C	D							

E	E	E	A	E	C	5
C	E	E	C	E	D	5
					E	9

Implementation

The construction of the binary tree follows a simple set of instructions. First, a tree with only a root node is created for each symbol. From this forest, the two trees with the least weight are combined into a larger tree. Often times, the tree with the lower weight is placed on the left branch, but this ultimately does not matter. The resulting tree will be then assigned a weight equal to the sum

of the weights of its component trees and returned into the forest. The process is repeated, once again combining the two least-weight trees, until all nodes are part of the entire tree. Once the binary tree is completed, all left branches are assigned 0 while all right branches are assigned one. From the previous example, the binary tree is created through the following steps:

Symbol	Weight	Symbol	Weight	Symbol	Weight	Symbol	Weight
E	9	E	9	C & D	10	C & D	10
C	5	B & A	6	E	9	(B & A) & E	15
D	5	C	5	B & A	6		
A	4	D	5				
B	2						

Final Tree:

Symbol	Weight
(C & D) & ((B & A) & E)	10

Visually, the binary tree would be constructed like the following, with the branch and node values assigned.

Symbol	Huffman Code
A	101
B	100
C	00
D	01
E	11

Finally, the Huffman Code is used to transmit the original image. Assuming each original symbol is allotted 8 bits, the size is reduced from the original 200 bits (25 symbols) to 56 bits (1011011001000011011101010100101000111111101110011110011), although for image compression, the data must still be grouped into bytes (Mahoney). Even with the binary tree included in the compressed file, a significant amount of space is still saved by the Huffman Encoding algorithm.

c. JPEG Compression

For images in color or grayscale, the JPEG compression algorithm is commonly used. It is a lossy type of compression designed to work with photographs and artwork in a natural or real-world environment. The compression scheme of JPEG relies on the limitations of the human visual system. Depending on how pixels are spaced out, the human eye’s sensitivity to brightness changes accordingly. Color variation is also perceived less when the details are grouped closely together (Mahoney).

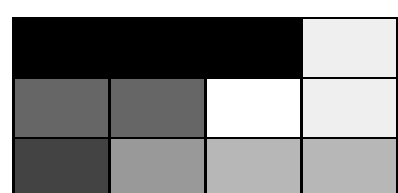
The JPEG compression algorithm starts by converting the RGB color input into the YCbCr space. The chroma components may be scaled down due to the previously mentioned human visual limitations. The converted image is then partitioned into 8x8 blocks without overlapping. For each block, a discrete cosine transform is applied,

converting the color levels into coefficients in the frequency domain (Dhawan). Normalization of these coefficients is done afterwards. In this step, information is discarded, thus making the JPEG compression a lossy algorithm. The resulting quantized coefficients are traversed in a zigzag pattern, and further compressed using a lossless algorithm. In order to decode the compressed file, the compression steps are simply reversed.

In JPEG compression algorithms, image quality is traded off for size. Since most photographic images have a large number of visually-redundant data, the loss in quality is not easily detected by the human vision, and can be labelled as visually lossless. However, in higher compression ratios, the boundaries of the 8x8 blocks become more apparent and the algorithm produces less satisfactory output images (Blelloch).

Illustration

Compression is done in 8x8 blocks, which are represented by the intensity of each of the 64 pixels within the block. For convenience, the following representation uses a 4x4 block with sample values of intensity. Preprocessing can also include subtracting 127 from each element of the array, transforming the domain from the original intensity values [0, 255] to one that is centered around zero [-127, 128].

	<table border="1"> <tbody> <tr> <td>3</td> <td>19</td> <td>10</td> <td>190</td> </tr> <tr> <td>49</td> <td>62</td> <td>231</td> <td>166</td> </tr> <tr> <td>26</td> <td>114</td> <td>99</td> <td>85</td> </tr> </tbody> </table>	3	19	10	190	49	62	231	166	26	114	99	85	<table border="1"> <tbody> <tr> <td>-124</td> <td>-108</td> <td>-117</td> <td>63</td> </tr> <tr> <td>-78</td> <td>-65</td> <td>104</td> <td>39</td> </tr> <tr> <td>-101</td> <td>-13</td> <td>-28</td> <td>-42</td> </tr> </tbody> </table>	-124	-108	-117	63	-78	-65	104	39	-101	-13	-28	-42
3	19	10	190																							
49	62	231	166																							
26	114	99	85																							
-124	-108	-117	63																							
-78	-65	104	39																							
-101	-13	-28	-42																							

REFERENCES

- [1]. Belloch, Guy E. "Introduction to data compression." *Computer Science Department, Carnegie Mellon University* (2001).
- [2]. Dhawan, Sachin. "A review of image compression and comparison of its algorithms." *International Journal of Electronics & Communication Technology, IJECT* 2.1 (2011): 22-26.
- [3]. Gonzales, Rafael C., and Richard Eugene Woods. *Digital Image Processing*. Pearson Education, 2009.
- [4]. Grigorik, Ilya. "Image Optimization." *Web Fundamentals*, Google Developers, 26 Sept. 2017.
- [5]. Kodituwakku, S. R., and U. S. Amarasinghe. "Comparison of lossless data compression algorithms for text data." *Indian journal of computer science and engineering* 1.4 (2010): 416-425.
- [6]. Ludman, Lonnie C. *Fundamentals of Digital Signal Processing*. Wiley, 1986.
- [7]. Mahoney, Matt. *Data Compression Explained*. mattmahoney.net/dc/dce.html.
- [8]. Taubman, David, and Michael Marcellin. *JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards and Practice*. Vol. 642, Springer Science & Business Media, 2012.
- [9]. Wahba, Walaa Z., and Ashraf YA Maghari. "Lossless Image Compression Techniques Comparative Study." *International Research Journal of Engineering and Technology (IRJET)*, e-ISSN (2016): 2395-0056.
- [10]. Wei, Wei-Yi. "An introduction to image compression." *National Taiwan University, Taipei, Taiwan, ROC* (2008).

Abdullatif zankawi. adel Alateyah "Comparison between Image Compression Algorithms ." *International Journal of Engineering Research and Applications (IJERA)* , vol. 7, no. 12, 2017, pp. 54-61.