Image compression through high value digital components analysis and non-uniform storage of pixel bits

Reza Askari Moghadam*
Faculty of New Sciences and Technologies
University of Tehran, Tehran, IR Iran, P.O.Box:14395-1561
r.askari@ut.ac.ir

Parviz Gharehbagheri
Tehran University, Tehran, IR Iran
p.gharehbagheri@yahoo.com

***Abstract*—In this article, we look into digital components analysis for image pixels by means of simple mathematical and statistical tools. This is to materialize by cutting non-uniform and low value pixels and also minimizing these values along with non-uniform storage of each pixel. The results would considerably prove the compression of images. The separated components, with low value and of different bit numbers, are used in approaches which are not based on data abundance due to high variance, heterogeneity and dispersion. What is proposed in this article for storage of separated high value is "Merge code Algorithm" (MCA), and low value components is "Radial Encoding".**

***Keywords*—**mathematical and statistical approaches; valuable digits analysis; merge code Algorithm; radial encoding.

I. Introduction

It has been years since different approaches whether in spatial dimension or frequency domain have been presented in image compression procedures and still have various applications today. What is new in this paper is innovative and blending ways to work in spatial dimension of pixels. This method works with the separation of valuable components and non-uniform storage of pixel bits and thus does the task of compression. The following points will clarify the thought behind this article: With the analysis of valuable components, the valuable digits, which store the main information, are separated and are stored again with a new introduced format called "IRN", which is known as a lossless approach. In digital data such as images, there are many pixels close to each other that their differences are small. Every two pixel data are saved and distinguished independently. After separating these small differences, the two pixel data would be uniform and it's enough to add differences to uniform data to reconstruct the original pixels data. Digital data can be minimized by cutting and separating their high and low values. The low value of each pixel can be compressed by novel "radial coding" which is described in this article. Some methods based on pixel data values analysis have been developed like Huffman method[1],[2] LZW[5] and PNG. LZ77 algorithm was used in PNG format [5]. Also, in some articles published in recent years move to front (MTF) compression and blending method of compression(advanced MTF) have been developed the acquire better capability in comparison to their former algorithms [3]. Using MLP neural networks and principal components analysis neural networks, principal components of data have been used in order to compress data much faster [4,6,7].

II. principles of working with pixels in spatial dimensions

Pixels data are variables which should be compressed in images. Each pixel, in spatial dimension consists of three main eight-bit data colors: red, green and blue. They are defined as:

$$(RGB)_{256} = R \times 256^2 + G \times 256^1 + B \times 256^0 \quad (1)$$

According to experimental and statistical observations, we can conclude that more than "50" percent of neighboring pixels have the same values in R, G and B order, and over "80" percent of the information is hidden in these significant digits. Numerical differences of neighboring pixels which have smaller differences compared with the values of pixels are used to compress data. This procedure described in next section.

III. separatioin operation

pixels data differences should be computed at first. The operation is shown in formula (2).

$$\Delta p_i = \left| p_{i+1} - p_i \right| \quad i = 1,2,3,\ldots,n \tag{2}$$

The signs related to each difference are stored in a separated bit (sign flags) which will be used during decoding phase. After getting the pixel image diffrences, we map this data in R, G, and B order as following:

$$R_k = \lfloor R \times K \rfloor, G_k = \lfloor G \times K \rfloor, B_k = \lfloor R \times K \rfloor \quad , K = 0.5 \tag{3}$$

Also the signs related to each map are stored in special bits (map flags) to be used while decoding. If we consider k= "0.5", map flag bits because of rounding, will be "0" and "1" for "0"and "0.5", respectively.

In this part, high value digits are analyzed:

$$R_{kh} = R_k / 10,\ G_{kh} = G_k / 10,\ B_{kh} = B_k / 10 \tag{4}$$

$$R_{kl} = R_k \bmod 10,\ G_{kl} = G_k \bmod 10,\ B_{kl} = B_k \bmod 10 \tag{5}$$

The base of every analyzed values is as follows:

$$b_{kl} = 10,\ b_{kh} = \lfloor (256/10) \times K \rfloor + 1 \tag{6}$$

Having all mentioned above, we could conclude that:

$$(RGB)_{256} \approx (R_{kh}\, G_{kh}\, B_{kh})_{b_{kh}} + (R_{kl}\, G_{kl}\, B_{kl})_{10} \tag{7}$$

The values of $(R_{kh}\, G_{kh}\, B_{kh})_{b_{kh}}$, are called "High value" and the values of $(R_{kl}\, G_{kl}\, B_{kl})_{10}$ are called "Low value".

In the rest of the discussion, we look through the storage methods in equation (7).

IV. Encoding algorithm of high and low value

The pixels below are a resized image of Chrysanthemum.bmp in "Windows 7", which have been implemented in this article.

Fig. 1. Chrysanthemum.bmp

We take all of these pixels to R,G,B format and analyze pixel values by the use of equations (3) to (7) by assuming the coefficient as k=0.5:

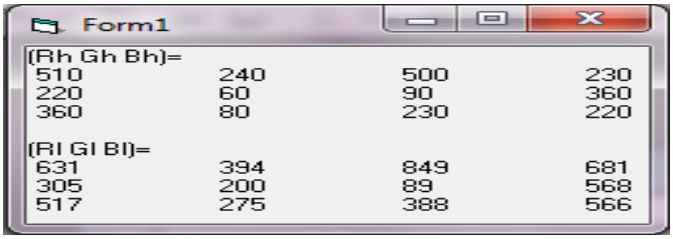

Fig. 2. Separating value *(high and low value)*

As you see in the figure above, the important and valuable part, $(R_{kh}\, G_{kh}\, B_{kh})_{b_{kh}}$, contains smaller and more significant amounts compared with the values of the second part meaning $(R_{kl}\, G_{kl}\, B_{kl})_{10}$. It can be prove that in many images more than 80% of image information is comprised in this part. Also, the abundance is more in the first part rather than the second, which increases the advantage of algorithms capability which are based on abundance for storing data. In the following portions of this article, we will attempt to store the more valuable parts by means of MCA algorithm . We store the data in a file with IRN extension. Less valuable parts will be stored after coding, which will be discussed later.

A. Merge Code Algorithm(MCA)

In this section "MCA" algorithm is discussed which is used for storing of high values of $(R_{kh}\, G_{kh}\, B_{kh})_{b_n}$. All valuable components are transferred on base "2", in other words, binary values:

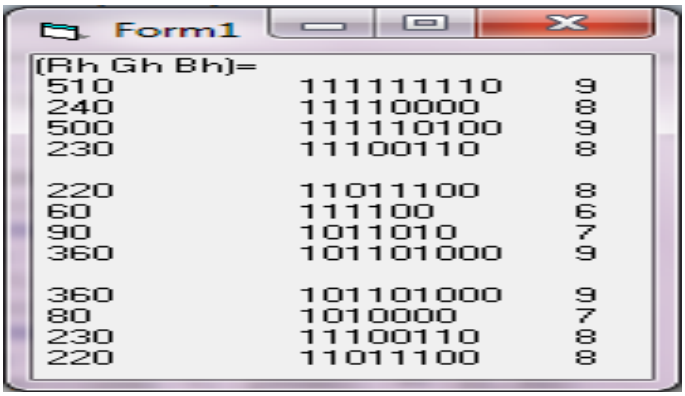

Fig. 3. Binary of high value

As it is shown in the figure, there are four types of bit data lengths (6, 7, 8, and 9). In order to uniform bit length in all data, we inject sufficient "0" bit into values with shorter length. So here, we change the "6" bit type into a "7" bit type by adding one "0" on the left. From among the remaining types, the "7" bit type with abundance of "3" has lower value than other types. Therefore, a "0" code is added to all "7" bit types so that all "7" bit data can be categorized as "8" bit types. Thus, by this manipulation, we have gained two types of data with "8" and "9" bits and abundance of "8" and "4". Even though "4" extra bits have been created in this approach, which seems an obvious disadvantage to the algorithm, this disadvantage is nothing compared to the total blending.

Now we give "0" code to "8" bit type and code "1" to the "9" bit type so that when coding and storing the actual number of bits, this number could be achievable by codes "0" and "1".

TABLE I. BIT TYPES

Binary Code	Len of binary code (Bit type)
111111110	9
11110000	8
111110100	9
11100110	8
11011100	8
111100	6
1011010	7
101101000	9
101101000	9
1010000	7
11100110	8
11011100	8

Now, table of frequency of "Bit type" is created:

TABLE II. FREQUENCY OF BIT TYPES

Bit type	Frequency (Sorted)
8	5
9	4
7	2
6	1

Because the bit type of "8" is nearest big bit type to "6" and "7", therefore we should convert them to bit type of "8" by adding "0" behind of binary code of them as below table:

TABLE III. BIT TYPES AFTER "MCA"

Binary Code	Length of binary code (Bit type)
111111110	9
11110000	8
111110100	9
11100110	8
11011100	8
00111100	6+2=8
01011010	7+1=8
101101000	9
101101000	9
01010000	7+1=8
11100110	8
11011100	8

TABLE IV. HEAD CODE FOR BIT TYPES

Type	Frequency (Sorted)	Head code
8	8	***0***
9	4	***1***

If this bit types merge to types of "8" and "9" but length , we have:

$$Ratio = \frac{112}{12 \times 24} = \frac{112}{288} \approx \%38$$

And if bit types merge to only type of "9" we have:

$$Ratio = \frac{108}{12 \times 24} = \frac{108}{288} \approx \%37 \quad (8)$$

Also we can use all of the bit types and coded them by variable length coding same "Huffman" or fixed length coding. This result demonstrates that in this example more than "80" percent of information in each pixel, which lies in the $(R_{kh}\, G_{kh}\, B_{kh})_{b_n}$, has occupied only "38" percent of a pixel size, and the "63" percent remaining which is derived from $(R_{kl}\, G_{kl}\, B_{kl})_{10}$ does not have a considerable value and can be stored in methods deleting some information while compression. The second point lies in this fact that in the valuable compressed data in IRN method, the abundance is more than the abundance in preliminary amounts of pixels. Therefore, these values can be compressed once again in methods such as LZW, but is not to be discussed in this article.

B. Radial Encoding

The second part of the equation (7), $(R_{kl}\, G_{kl}\, B_{kl})_{10}$ is the low value of pixels, which can be compressed by methods not dependent on frequency (abundance) due to its lack of high abundance and value. As it was stated before, in order to store low value numerical amounts of each pixel, as a result of low value and low abundance, we could use methods that do not depend on abundance and the loss of portions of information will not at all damage the whole file. The best methods are principal components analysis and neural networks, which have been treated in other essays. What we mainly focus here is a suggested way of encoding with radius of "1" and "2"; by radius here we mean the highest error that could possibly happen for each amount. Since low value amounts are ones and their value is between "0" and "9", in order for them to be saved, in encoding method with radius "1", we allocate "0" code for digits "0", "1" and "2", code "1" for digits "3", "4" and "5", code "2" for digits "6", "7" and "8" and code "3" for digit "9", then in decoding we replace number "2" with "0" code, number "4" with code "1", number "3" with code "2" and number "9" with code "3". Therefore, we use only "2" bits to store each and every single digit from "1" to "9" with ± 1 error. In encoding with radius 2, digits (0, 1, 2, 3, 4), we allocate "0" code and code "1" for (5, 6, 7, 8, 9) and in decoding we replace number "2" with "0" code and number "7" with code "1". In this way, we use only "1" bit to store each and every single digit with maximum error equal to ± 2 .

For instance, consider the amounts of ones and low values for R, G, and B are "1", "4" and "9" in order. In encoding with radius "1", rather than storing them, we store codes "0", "1" and "3" thus in this way, we will only encounter one unit of error while decoding. The biggest amount of integrated digit ones RGB is

$(999)_{10}$ which needs at most "9" bits to be stored, while the biggest digit ones code is $(333)_4 = (63)_{10}$ which needs at most "6" bits, the compression ratio in this method, known as lossy, is defined as follows:

$$compression\ Ratio = 1 - \frac{6}{9} \approx \%34 \tag{9}$$

Since the encoding method does not attain high degree of compression, it noticeably reduces the total compression ratio.

In any case, with regards to implemented methods in this article, the compression ratio would be as shown below:

$$Ratio = 1 - \frac{108 + (12 \times 6)}{12 \times 24} = 1 - \frac{108 + 72}{288} = 1 - \frac{180}{288} \approx \%40 \tag{10}$$

By the way, if we had used an encoding with radius "2", the compression ratio would have been more than "50" percent.

V. Defining the "IRN" format

To store values of $(R_{kh}\ G_{kh}\ B_{kh})_{b_{\blacksquare}}$, we define a new format as demonstrated below:

TABLE V. IRN STORAGE FORMAT

Sign bits	Bits indicating the number of main bits of each pixel	main bits of each pixel
(1)	(2)	(3)

1. *(1): This is the sign bits of each pixel (Difference & Divid).*
2. *(2): These bits are "MCA" codes, to specify the number of bits needed to store each pixel.*
3. *(3): The main bits to store each pixel, has been indicated according to experimental implementation on images. The average of bits needed to store the pixels is smaller than the bit diversity average.*

We should do and store six step of "IRN" algorithms same below steps:

1. *Calculate pixel differences of image according to equation (2).*
2. *Store the plus and minus signs in the previous clause in an array. (positive numbers=0 and Negative numbers=1)*
3. *Multiply the numerical values of pixels by (k=./5), k and gain the out coming values based on equation (3).*
4. *Analyze the new values according to equation (5) and equation (6), and gain the new bases according to equation (7).*
5. *Store* $(R_{kh}\ G_{kh}\ B_{kh})_{b_{\blacksquare}}$ *by means of the "MCA".*
6. *Store* $(R_{kl}\ G_{kl}\ B_{kl})_{10}$ *by using of "Radial Encoding"*

In "IRN" algorithm, for lossy compressing steps 1 and 2 should be changed.

VI. comparing compression degree

As it is shown here the Lena image (204 *204 pixel) has compressed with "IRN" Algorithm and does not have noticeable eye recognition difference with the original one, however, SNR will show an extent of the qualitative error in these four images:

Fig.5. original image (BMP).

Fig.6. decompressed image (Radius0)

Fig.7. PNG image.

Fig.8. TIFF image

Fig.9.decompressed image (Radius10)

Fig.10. decompressed image(Radius1)

The qualitative error of the two images would be:

$$SNR = 10\log_{10}\frac{\sum_{i=1}^{N_{row}}\sum_{j=1}^{N_{col}} I_{ij}^2}{\sum_{i=1}^{N_{row}}\sum_{j=1}^{N_{col}}(I_{ij} - mean(I_{ij}))^2} \tag{11}$$

$$Compreesion\ \ Ratio = 1 - \frac{size\ \ of\ \ compressed\ \ pic}{size\ \ of\ \ original\ \ pic} \tag{12}$$

Also, PSNR is most commonly used to measure the quality of reconstruction of lossy compression codecs (e.g., for image compression). The signal in this case is the original data, and the noise is the error introduced by compression. When comparing compression codecs, PSNR is an approximation to human perception of reconstruction quality. Although a higher PSNR generally indicates that the reconstruction is of higher quality, in some cases it may not. One has to be extremely careful with the range of validity of this metric; it is only conclusively valid when it is used to compare results from the same codec (or codec type) and same content. PSNR is most easily defined via the mean squared error (MSE). Given a noise-free m×n monochrome image I and its noisy approximation K, MSE is defined as:

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{13}$$

The PSNR (in dB) is defined as:

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned} \tag{14}$$

Here, MAX_I is the maximum possible pixel value of the image. When the pixels are represented using "8" bits per sample, this is "255". More generally, when samples are represented using linear PCM with B bits per sample, MAX_I is 2^B-1. [17].

Therefore:

TABLE VI. RESULTS OF PROPOSED ALGORITHM

Level of Radius	Compression Ratio	Size	SNR	PSNR
Fig.6(Lossless)	%30	85kb	No Noise	No Noise
Fig.7 (png)	7%	113kb	No Noise	No Noise
Fig.8 (tiff(Lzw))	15%	103kb	No Noise	No Noise
Fig.10(Radius1)	%48	62kb	87 db	159 db
Fig.9(Radius10)	%70	37kb	72 db	152 db

And this shows the desired quality and compression ratio of the decompressed image. Also Table (8) shows the comparison of proposed algorithm in lossless state and other formats.

Fig.11. original Lena image 512*512 pixel for comparing of lossless formats

TABLE VII. COMPARISON OF PROPOSED ALGORITHM AND OTHER 24 BIT LOSSLESS IMAGE COMPRESSION ALGORITHMS

Lossless image compression algorithms	Bit per pixel
Bmp	Original
Png(24 bit)	4.22
Tiff(32 bit)	5.17
Proposed(IRN)	4.5
Ref.[8]	4.27
Ref.[9]	4.4
Ref.[10]	4.84
Ref.[11]	4.58
Ref.[12]	4.54
Ref.[13]	4.87
Ref.[14]	4.4
Ref.[15]	4.62

VII. conclusion

The innovative method in this article could effectively analyze high value and low value figures of each pixel and store them on its exclusive format with the use of simple arithmetic and mathematical functions, which in comparison to other more sophisticated algorithms, could both compress and separate the low value data to send for other methods. Also, this new method improves the capability of a more and better compression along with other methods such as MTF and LZW due to its creation of more abundance in new compressed data compared to former and original data. This method has successful rate of compression in many cases and because of using simple mathematical functions it has good speed.

VIII. References

[1] Kh. Sayood, "Data Compression", University of Nebraska, Lincoln.
[2] Sh. Porwal, "Data Compression Methodologies for Lossless Data and Comparison between Algorithms", Department of Computer Science and Engineering Vedaant Gyan Valley,2013.
[3] D. Suarjaya, "A New Algorithm for Data Compression Optimization", Information Technology Department Udayana University Bali, Indonesia,2012.
[4] S. Kabiri, "principal component analysis technique", Electrical and Electronics Forum Iran, Tehran, 2009.
[5] G. Dudek, "Lossy dictionary-based image compression method", Department of Physical Chemistry and Technology of Polymers, Faculty of Chemistry, Silesian University of Technology, Ks. M. Strzody 9, 44-100 Gliwice, Poland, 2007.
[6] "Principals Component Analysis In Image Processing", Department of Computing and Control Engineering, Wiley Publishing, 2003.
[7] Y. Guowei, "Lossless Data Compression Methods Based on Neural Network"، Computer Center, Teachers' College of Qingdao University.
[8] M. Chen, P. Franti, and M. Xu, "Lossless Bit-plane Compression of Images with Context Tree Modeling", published in Green Circuits and Systems (ICGCS) International Conference on., pp. 605-610, 2010
[9] S. Dikbas, F. Zhai, "Lossless image compression using adjustable fractional line-buffer", Signal Processing: Image Communication., Vol. 25, pp. 345–351, 2010
[10] X. Xinfeng and H. Yong, "A Shortcut to Compressing Bayer-pattern Imagery Losslessly", published in Image and Signal Processing, 2009. CISP '09. 2nd International Congress on ., pp. 1-4, 2009
[11] H. Kikuchi, K. Funahashi, and Sh. Muramatsu , "SIMPLE BIT-PLANE CODING FOR LOSSLESS IMAGE COMPRESSION AND EXTENDED FUNCTIONALITIES", piblished in Picture Coding Symposium., pp. 1-4, 2009
[12] S.L. Concorde, "Simple and Efficient Lossless Coding for Continuous Tone Color Images", International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)., pp. 1-4, 2008
[13] Y. Cheng, J. Zhao, K. Xie and G. Zhang, "Fuzzy Neural Network Based Prediction Coding for Bayer Pattern Image", published in Cybernetics and Intelligent Systems IEEE Conference on., pp. 137-141, 2008
[14] A. Kingston, F. Autrusseau, "Lossless image compression via predictive coding of discrete Radon projections", Signal Processing: Image Communication., Vol 23, pp. 313– 324, 2008
[15] C.H. Chou and K.C. Liu, "Colour image compression based on the measure of just noticeable colour difference", published in IET Image Process., Vol. 2, No. 6, pp. 304–322, 2008
[16] S.L. Concorde, "Simple and Efficient Lossless Coding for Continuous Tone Color Images", International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)., pp. 1-4, 2008
[17] WebSite wikipedia, Internet: http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.
[18] J. Miano, Compressed image file formats: Jpeg, png, gif, xbm, bmp, 1999.