Lossless Image Compression based on Data Folding

Suresh Yerva, Smita Nair, Krishnan Kutty

CREST, KPIT Cummins Info systems Ltd.
Pune, India
{Suresh.Yerva, Smita.Nair, Krishnan.Kutty}
@kpitcummins.com

***Abstract*— The paper presents an approach for lossless image compression in spatial domain for continuous-tone images using a novel concept of image folding. The proposed method uses the property of adjacent neighbor redundancy for prediction. In this method, column folding followed by row folding is applied iteratively on the image till the image size reduces to a smaller pre-defined value. For column folding, elements of even columns are subtracted from elements of odd columns. Thereafter, row folding is applied on odd columns in a similar fashion. In row folding, even rows are subtracted from odd rows and the resultant odd rows are used for next iteration. The difference data, thus obtained, is stored in a tile format; which along with the reduced image is encoded and transmitted. The proposed method is compared with the existing standard lossless image compression algorithms and the results show comparative performance. Data folding technique is a simple approach for compression that provides good compression efficiency and has lower computational complexity as compared to the standard SPIHT technique for lossless compression.**

***Keywords*- lossless image compression, data folding, JPEG-LS, SPIHT**

I. INTRODUCTION

Compression is the core module for storing and sending data; hence it has a gamut of applications [1]. Lossy compression is majorly used for network related applications whereas lossless compression is used for storage related applications. Possible applications of lossless compression include: file zipping; compression of audio, images/videos, executable programs, text, source code etc.

Any lossless coding system basically consists of three steps. They are: transformation, data-to-symbol mapping and lossless symbol coding [2]. Transformation converts the image data into a form that can be compressed more efficiently by further stages. But sometimes, it might be desirable to operate directly on the original data without incurring the additional cost of applying transformation; in this case transformation is set to the identity mapping. Data-to-symbol mapping converts the input data into symbols that can be efficiently coded by final stage. Lossless symbol coding generates a binary bit stream by assigning binary code words to the input symbol. The first two stages can be regarded as preprocessing stages for mapping the data into a form that can be coded more efficiently by the last stage. Most popular standards for lossless image compression are JPEG-LS (lossless mode) and lossless SPIHT.

Compression efficiency and computational complexity are two important parameters for designing data compression algorithm as communication bandwidth and computing power vary from environment to environment. Hence, the need for data compression with good compression efficiency and low computational complexity is more apparent than ever. We propose a fast method to compress the image data; which also provides good compression efficiency. For images, where there is not much high-frequency content in the image, the results obtained by the proposed method are pretty close to that of JPEG-LS and SPHIT.

The remainder of this paper is organized as follows: Section II describes the background concepts of existing standards. Section III describes the proposed method. Section IV details experimental results. Section V explains some issues of data folding followed by conclusion and references.

II. BACKGROUND

A. SPIHT

Lossless SPIHT makes use of S+P Transform which is similar to the Haar multi-resolution image representation. S-transform followed by pre-defined predictor function is applied on the image. After that, any of standard coding techniques such as Huffman Coding (HC), Arithmetic Coding (AC) etc. is applied to get the compressed the image. S-transform is defined as follows:

A sequence of integer's c[n], n=0… N-1 with N even, can be represented by the two sequences. The two sequences are calculated as follows.

$$l[n] = \left\lfloor \frac{(c[2n] + c[2n+1])}{2} \right\rfloor, n = 0, \dots, \frac{N}{2} - 1 \quad (1)$$

$$h[n] = c[2n] - c[2n+1], n = 0, \dots, \frac{N}{2} - 1 \quad (2)$$

Where $\lfloor . \rfloor$ corresponds to downward truncation. The sequences l[n] and h[n] form the S transform of c[n], as shown in Fig. 1.

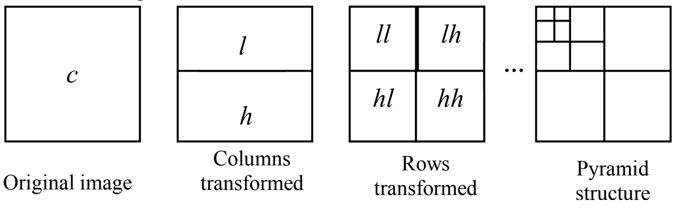


Fig. 1: Construction of an image multi-resolution pyramid from one-dimensional transformation

The inverse transformation is given by:

$$c[2n] = l[n] + \left\lfloor \frac{(h[n]+1)}{2} \right\rfloor \qquad (3)$$

$$c[2n+1] = c[2n] - h[n] \qquad (4)$$

More information about SPIHT can be found in [3].

B. JPEG-LS

LOCO-I (LOw COmplexity LOssless COmpression for Images) algorithm has been standardized as JPEG-LS. JPEG-LS is the low-complexity lossless and near-lossless compression standard. It is based on a simple fixed context model, which approaches the capability of the more complex universal techniques for capturing high-order dependencies. The model is tuned for efficient performance in conjunction with an extended family of Golomb-type codes, which are adaptively chosen. LOCO-I attains compression ratios similar or superior to those obtained with state-of-the-art schemes based on arithmetic coding, at a much lower complexity level.

More information about JPEG-LS can be found in [4, 5].

III. PROPOSED METHOD

Data folding is an innovative way for lossless image compression. The compression idea is based on spatial resolution for lossless image compression called corrugation/ data folding [6, 7, 8, 9, 10, 11]. Corrugation/ Data folding exploit pixel redundancy in 2-D images in either consecutive rows or columns. The idea is to subtract even pixels from odd pixels and store the difference data in a buffer. Odd pixels are stored in another buffer for further iterations. In data folding, column folding followed by row folding is applied. In column folding, pixels used for subtraction are column adjacent whereas in row folding, the pixels are row adjacent. The pixel redundancies are rearranged in a tile format and source encoding technique is applied at the end before transmitting the data. The idea is to reduce the image size iteratively in terms of dimensions - rows or columns by 2.

At the decoder, Huffman decoding is applied followed by data unfolding which is similar to data folding.

A. Encoder

1) Data folding: Data folding is an iterative procedure, column folding followed by row folding, that is repeated at every image level. Original image (i.e. input image) must be square.

Fig. 3(a) depicts the flowchart of data folding algorithm. Fig. 3(b) illustrates the algorithm for the first level of data folding. In this paper, labels N, k, n represent maximum no. of iterations possible for the original image, no. of iterations performed at the encoder and level of folding/unfolding respectively. Maximum number of iterations is equal to binary logarithm of width or height of the original image. For all matrices that are used in this paper, index of the first element is (1, 1).

Define a buffer ‘F’ (also called image matrix) whose size is equal to that of original image. Original image is considered as input image for the first iteration. Initially, the buffer ‘F’ is empty.

In column folding, odd columns of the input image are subtracted from its right adjacent even columns and stored in first half columns of the empty portion of the buffer ‘F’. Odd columns are stored in a different buffer S’ which is taken as input image to row folding. Following equations depicts column folding technique.

$$F(X+a, Y+b) = S(a, 2b-1) - S(a, 2b)$$

$$S'(a,b) = S(a, 2b-1)$$

$$a \in [1, W] \; and \; b \in \left[1, \frac{W}{2}\right] \qquad (5)$$

Where, S = input image
W = width of input image
S’ = modified input image
X = starting x-coordinate of empty portion of ‘F’
Y = starting y-coordinate of empty portion of ‘F’

Row folding is similar to column folding. In row folding, odd rows are subtracted from its adjacent even rows and stored in first half rows of empty portion of ‘F’. Odd rows are stored in a different buffer S’’ which is taken as input to next iteration. Observe that, input image to column folding is always square $(2^{N-n+1} * 2^{N-n+1})$ whereas it is rectangular$(2^{N-n+1} * 2^{N-n})$ to row folding.

Difference data thus obtained through all iterations is stored in the tile format, as shown in Fig. 2.

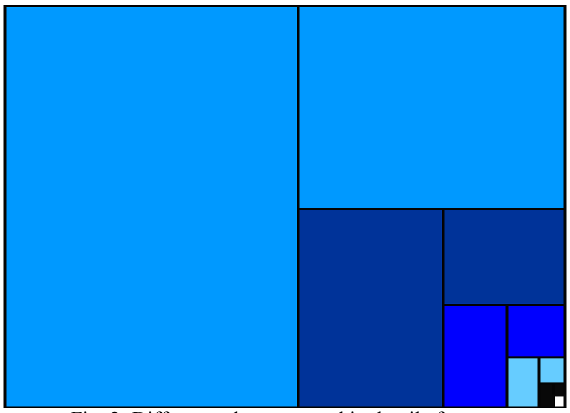
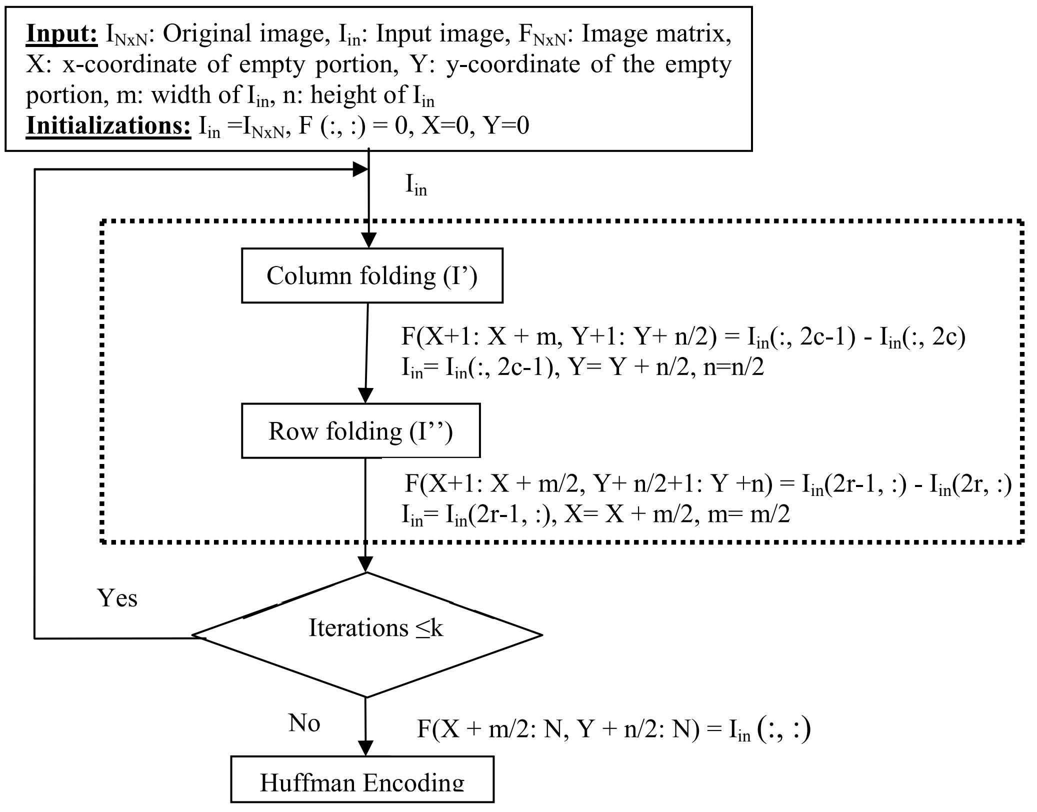


Fig. 2: Difference data arranged in the tile format

Input image obtained after all iterations is copied into remaining portion of matrix ‘F’. The magnitude of difference data ranges from -255 to +255. Add 255 to each element of image matrix in order to map the data to a positive range [0, 510].


Input: I_{NxN}: Original image, I_{in}: Input image, F_{NxN}: Image matrix, X: x-coordinate of empty portion, Y: y-coordinate of the empty portion, m: width of I_{in}, n: height of I_{in}
Initializations: $I_{in} = I_{NxN}$, F (:, :) = 0, X=0, Y=0
I_{in}
Column folding (I')
F(X+1: X + m, Y+1: Y+ n/2) = I_{in}(:, 2c-1) - I_{in}(:, 2c)
I_{in}= I_{in}(:, 2c-1), Y= Y + n/2, n=n/2
Row folding (I'')
F(X+1: X + m/2, Y+ n/2+1: Y +n) = I_{in}(2r-1, :) - I_{in}(2r, :)
I_{in}= I_{in}(2r-1, :), X= X + m/2, m= m/2
Yes
Iterations ≤k
No
F(X + m/2: N, Y + n/2: N) = I_{in} (:, :)
Huffman Encoding

(a)

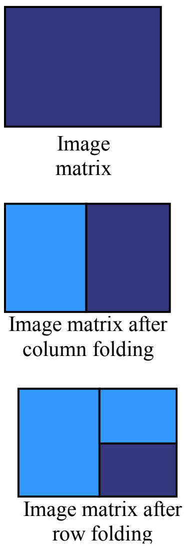


Image matrix
Image matrix after column folding
Image matrix after row folding

(b)

Fig. 3: (a) Flowchart of data folding, (b) Illustration of data folding for the first iteration

2) *Coding:* This difference matrix can be entropy encoded by using various algorithms like Huffman Coding (HC), Arithmetic encoding (AC). We have encoded it using Huffman algorithm. Data obtained by each level will be encoded separately. Calculate the histogram and probability distribution to help in Huffman table calculation. The encoding step also called as source encoding could be carried out in different ways. The final result obtained after encoding the data of each level would be compressed data for the input image. The ratio of output compressed file to input image is the quantitative measure for the algorithm. The dictionary used for entropy encoding, which is a mapping between pixel value and corresponding bit value; is also transferred along with the encoded bit file. The compression is measured as number of bits used per pixel and is calculated as input file size divided by sum of size of the output bit file (i.e. the matrix obtained after source encoding), sum of sizes of the dictionaries (i.e. Huffman tables) and y bits (to send the iterations to the decoder).

$$Compression\ ratio = \frac{I}{I' + D + y} \tag{6}$$

Where, I = Raw image size (image size at start)
I'= Size of output bit file
D= Sum of sizes of all dictionaries
y= no. of bits required to encode no. of iterations

The no. of iterations required (k) for decoding and dictionary will also be sent along with encoded data to the decoder.

B. Decoder

The decoder will perform Huffman decoding followed by data unfolding. Huffman decoding is done using Huffman table. Data unfolding is an iterative procedure similar to data folding; row unfolding followed by column unfolding that is repeated at each image level. Here, no. of iterations is equal to that of data folding. Matrix elements which were folded in higher iterations during encoding will be unfolded first. Data folding starts with the first element of the image whereas data unfolding starts with bottom right elements of the image.

The procedure is illustrated for a 16x16 image in Fig. 4. Assume that only 3 levels of folding is applied at the encoder side.

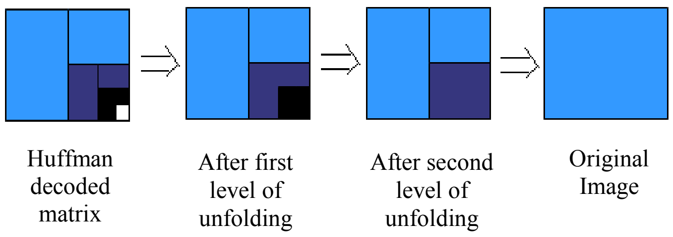

Huffman decoded matrix
After first level of unfolding
After second level of unfolding
Original Image

Fig. 4: Illustration of data unfolding algorithm

For row unfolding, initialize two buffers; say 'K' and 'L' of size $(2^{(N-k)+n} * 2^{(N-k)+n-1})$. Consider the example which is illustrated in Fig. 4. Then, N=4, k=3. For the first iteration, value of n is 1. Initially, size of both buffers 'K' and 'L' are 4x2.

Copy the elements from the bottom right sub-matrix of size $(2^{(N-k)+n} * 2^{(N-k)+n-1})$ into the buffer 'K' which is taken as input image for row unfolding. In row/column unfolding, two consecutive rows/columns are unfolded at a time. In row unfolding, copy (row wise) the elements of second half rows of 'K' into 'L' to calculate odd rows of 'L'. The above mentioned rows of 'K' are copied into 'L' in the same order they appear in 'K'. At the same time, subtract elements of these rows from elements of the rows which are $2^{(N-k)+n-1}$distance above (row wise) from them in order to calculate even rows of 'L'. Following equations depicts the row unfolding technique.

$$\begin{gathered} L(2a-1,b) = K\left(a + 2^{(N-k)+n-1}, b\right) \\ L(2a,b) = K\left(a + 2^{(N-k)+n-1}, b\right) - K(a,b) \\ a \in [1, W/2] \; and \; b \in [1, W] \end{gathered} \tag{7}$$

Where, W = height of buffer 'K'

Observe that, buffer 'K' is exactly equal to the input of row folding during (k-n)th level of data folding. For above mentioned example, pixels which are 2 distant apart are used to find out even rows. Elements of 'L' matrix after row unfolding are shown in terms of elements of matrix 'K' below.

$$L = \begin{bmatrix} \mathrm{K}(3,1) & \mathrm{K}(3,2) \\ (\mathrm{K}(3,1) - \mathrm{K}(1,1)) & (\mathrm{K}(3,2) - \mathrm{K}(1,2)) \\ \mathrm{K}(4,1) & K(4,2) \\ (\mathrm{K}(4,1) - \mathrm{K}(2,1)) & (\mathrm{K}(4,2) - \mathrm{K}(2,2)) \end{bmatrix}$$

After row unfolding, replace the elements of sub-matrix $(2^{(N-k)+n} * 2^{(N-k)+n-1})$ from the bottom right with buffer 'L'. Similarly, for column unfolding, initialize two buffers; say 'P' and 'Q'. Copy the elements of sub-matrix of size $(2^{(N-k)+n} * 2^{(N-k)+n})$ from the bottom right into the buffer 'P' which is taken as input image for column unfolding. In column unfolding, copy the elements of second half columns of 'P' to calculate odd columns of 'Q'. At the same time, subtract elements of these columns from the elements of columns which are $2^{(N-k)+n-1}$ distant apart (column wise) from it. This procedure is repeated till the whole data is unfolded.

C. Computational Complexity

As we mentioned in earlier sections, data folding algorithm requires less computations than SPIHT.

Consider a square image of size 2^N*2^N (=M) as input image. Then, computational complexity of data folding algorithm is calculated as follows:

For the first level of folding,

Cost associated with column folding= 2^N*2^{N-1}

Cost associated with row folding= $2^{(N-1)}*2^{(N-1)}$

Hence, Cost associated with first level of data folding= $3*2^{(N-1)}*2^{(N-1)}$

Similarly, cost associated with second level of data folding= $3*2^{(N-2)}*2^{(N-2)}$

.
.
.

Hence, computational complexity of data folding algorithm= $3*2^{(N-1)}*2^{(N-1)} + 3*2^{(N-2)}*2^{(N-2)} +...+3= (2^N*2^N)-1 =$ O(M)

Complexity of S-transform is O(M) and that of prediction is also O(M). In data folding, column/row folding requires only one subtraction and one assignment operation per 2 pixels. In SPIHT, two additions (i.e. one addition and one subtraction) and one left shift operation is needed for S-transform whereas four additions, two multiplications and one left shift operation per pixel is required for prediction.

Hence, we can conclude that computational complexity of data folding is lower than SPIHT. Computational complexity of data unfolding is equal to that of data folding. So, the decoding process is also much faster than that of SPIHT.

IV. Results

We have performed maximum levels of folding on all images. Huffman encoding is used after the data folding stage is complete. Any other coding technique can be used instead of Huffman coding. The values for bits per pixel (bpp) can be further improved by applying arithmetic coding instead of Huffman coding.

We have applied this approach on a variety of images shown in Fig. 5. As seen in Table (1), the bits per pixel (bpp) of the proposed method are close to that of standard algorithms, when the image is smooth. However, as mentioned earlier, the algorithm requires very few computations as compared to JPEG-LS and SPIHT techniques. The algorithm is suitable for those images whose pixel values vary slowly in the 2D-space, i.e., the imager doesn't have too many high-frequency contents – like edges, corners etc in it.

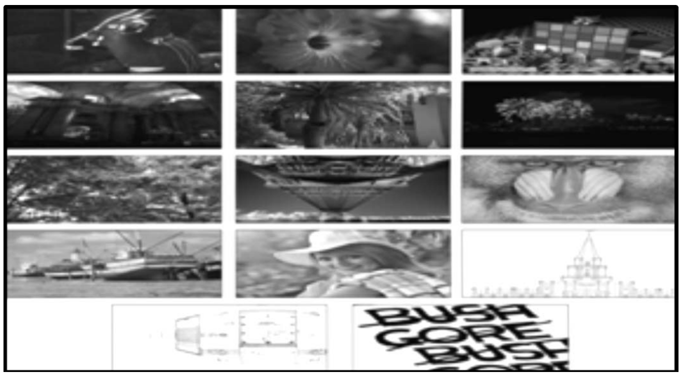


Fig. 5: Images respectively (in raster fashion): Deer, Flower, Artificial, Cathedral, Big tree, Fireworks, Leaves, Bridge, Baboon, Boat, Elaine, Temple, Pod rear draw, Cmp5-text

Table 1: Results of Data Folding Algorithm, JPEG-LS and SPIHT

Image	Size	Proposed method (bpp)	SPIHT (bpp)	JPEG-LS (bpp)
Deer	2048x2048	4.47	4.2	4.22
Flower	1024x1024	2.95	2.02	1.86
Artificial	2048x2048	2.04	1.5	0.95
Cathedral	1024x1024	5	4.28	4.26
Big tree	4096x4096	4.69	3.83	3.87
Fireworks	2048x2048	2.4	1.58	1.35
Leaves	2048x2048	5.78	4.47	4.48
Bridge	512x512	5.88	5	4.91

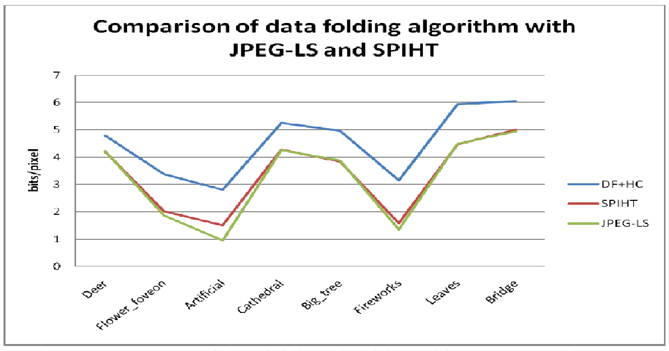


Fig. 6: Comparison of data folding algorithm with JPEG-LS and SPIHT

As seen in Fig. 6, results are relatively closer when the images are smooth enough. We have tested our algorithm over different classes of images, namely photographic, line drawings and text. Typically, intensity changes between neighbor pixels are very small (except at boundaries) in photographic images whereas in line/text images, difference between neighbor pixels would be either too high or low. For line/ text images, results of the algorithm are either better or at least close to that of standard algorithms.

Table 2*:* Comparison of Data Folding Algorithm with existing Algorithms for different class of images

Image category	Image Name	Size	Algorithm (bpp)		
			Data folding	SPI HT	JPEG -LS
Photographic	Baboon	512*512	7.22	5.76	5.94
	Boat	512*512	5.84	4.23	4.26
	Elaine	512*512	5.86	4.2	4.31
Line/ Text	Temple	1024*1024	2.66	1.73	1.39
	Pod rear draw	512*512	3.48	2.67	2.28
	Cmp5-text	256*256	6.31	4.62	3.56

V. ISSUES IN DATA FOLDING

We know that, in a typical image, adjacent (except at boundaries) pixel values are very close to each other. So, if we consider adjacent pixels for subtraction, we can guarantee that prediction error will be very small.
In data folding, the pixels that are considered for subtraction are $2^{(l-1)}$ (l=level) distant. Even if we stop folding at level 'k' (<N), we can't say whether the magnitude of difference data values is small. Hence, we cannot guarantee that data folding gives comparatively good results for all images.
Consider only first row of a 16x16 image. Let's apply column folding on it. Its effect on the first row is:

In level 1, PD=1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

In level 2, PD=2

1	3	5	7	9	11	13	15

In level 3, PD=4

1	5	9	13

In level 4, PD=8

1	9

Here, PD= Pixel Distance

It means, for NxN image,

In level 1, PD = 2^0

In level 2, PD = 2^1

.
.
.

In level m (<=L), PD= 2^{m-1}

The pixel distance is not a big problem if image intensities vary slowly along rows/columns (i.e. very smooth images).

VI. CONCLUSION

Data folding is an algorithm for lossless image compression. It is a simple and faster method for still image lossless compression. Computational complexity of this algorithm is lower than most popular standards like JPEG-LS and SPIHT. It works comparatively better for smooth images. It gives relatively good results for text images than photographic images. But it faces the problem of higher magnitudes while calculating difference data. This is due to more pixel distance in higher levels. Consequently, compression efficiency gets saturated after certain levels of data folding.

REFERENCES

[1] Gonzalez and Woods, *Digital Image Processing*, 3rd ed., vol. 8, pp. 409-518, 2007.

[2] Alan Conrad Bovik, *The Essential guide to Image Processing*, vol. 16, pp. 385-419, 2009.

[3] Amir Said, and William A. Pearlman, "An Image Multiresolution representation for Lossless and Lossy Compression," in SPIE Symposium on Visual Communications and Image Processing, Cambridge, MA, Nov. 1993.

[4] Marcelo Weinberger, Gadiel Seroussi and Guilermo Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS," Computer Systems Laboratory, HPL Tech. Rep. 98-193, Nov. 1998.

[5] G.K. Wallace, "The JPEG still picture compression standard," Communications of the ACM, vol. 34, pp. 30-44, Apr. 1991.

[6] Bin Zhu, Swanson, and Tewfik, "Image coding by folding," in International conference on Image Processing, Santa Barbara, CA, USA, 1997.

[7] Chin-Chen Chang, Chi-Shiang Chan and Ju Yuan Hsiao, "Lossless Image Compression Based on Two-Way Smaller Difference," in International Conference on Advanced Information Networking and Applications, Xi'an, China, 2003.

[8] Sabrina D. Boler, Karen L. Baker, Robert E. Gruhl, Robert D. Young and Thomas W. Getzinger, "Lossless Manipulation of Media Objects," U.S. Patent 7 239 328 B2, Jul. 3, 2007.

[9] Louis Lippincott, "Data-Modifying Run Length Encoder to avoid Data Expansion," US. Patent 7 327 289 B1, Feb. 5, 2008.

[10] Patrice Y. Simard, III; Erin L. Renshaw; James Russel Rinker and Henrique Malvar, "Segmented Layered Image System," US 7 376 266 B2, May 20, 2008.

[11] Patrice Y. Simard, Henrique S. Malvar and Erin L. Renshaw, "Clustering," US 7 376 275 B2, May 20, 2008.