International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

A Two Stage Low Complexity Lossless Coding Algorithm for Medical Images

Karthick Kumaran Ayyalluseshagiri Viswanathan

Member IEEE

Running Title: *Tailored For Ultra Low Power Devices*

Abstract: ***Nowadays a large number of various medical images are generated from hospitals and medical centers with sophisticated image acquisition devices. This paper deals with the techniques to decrease the communication bandwidth and to save the transmitting power in wireless medical devices. Digital image consumes huge memory and thus digital image data compression is necessary in order to solve this problem. In medical applications such as disease diagnosis, the loss of information is unacceptable; hence medical images should be compressed lossless. Wireless medical devices injected into the body are battery operated devices and hence the lifetime of the battery should last for a long time. Complexity of the compression algorithm is directly related to the power consumption of the processor and hence there is a need for simple compression algorithms. We proposed 2 simple algorithms which can be combined together or used separately based on the necessity to address this requirement.***

Keywords: Medical Image compression, Lossless compression, Wireless medical images, medical images, low power devices

1. Introduction

Medical imaging is a powerful and useful tool for radiologists and consultants, allowing them to improve and facilitate their diagnosis. Worldwide, X-ray images represents 60% of the total amount of radiological images, the remaining consists of more newly developed image modalities such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Ultrasound (US), Positron Emission Tomography (PET), Nuclear Medicine (NM) and Digital Subtraction Angiography(DSA).

Image communication systems for medical images have bandwidth and image size constraints that result in time consuming transmission of uncompressed raw image data. Thus, image compression is a key factor to improve transmission speed and storage. It exploits common characteristics of most images that are the neighboring picture elements (pixels) are highly correlated. It means a typical still image contains a large amount of spatial redundancy in plain areas where adjacent pixels have almost the same values.

Compression is the process of storing or packing data in a format that requires less space than the initial or original data. Compression techniques can be classified into lossy and lossless.

Lossy compression permits some signal degradation and provide higher compression ratios in comparison with lossless techniques. This is used in applications dealing with speech and video signals where some loss of information can be tolerated.

Lossless compression does not permit any loss of information and allows the original signal to be recovered exactly. This is used in a wide range of medical applications and under special circumstances such as disease diagnostic. In such applications, loss of information cannot be tolerated. Thus, rather than lossy compression with relatively high compression ratio, lossless compression methods are favored.

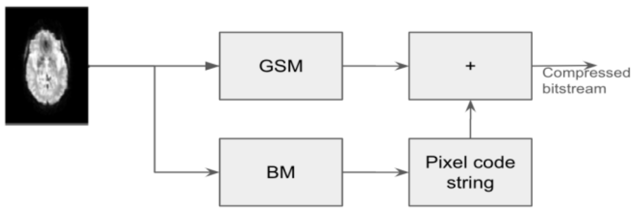
2. Existing Techniques/Algorithms

- Discrete Cosine Transform (DCT) based JPEG
- Discrete Wavelet Transform (DWT) based JPEG2000
- Lossless JPEG (JPEG-LS)
- Lossless JPEG 2000

First of these two algorithms are lossy compression algorithms and the rest are lossless compression algorithms. The weakness of these methods comes from its computational load and complexity (encoding and decoding complexity). Moreover, they have been tested only for non-medical images.

3. Proposed Algorithm

We present a new coding algorithm for medical images. This algorithm is absolutely lossless and based on pixel redundancy reduction technique.



This algorithm comprises of 2 stages. First stage is based on pixel redundancy reduction technique using only two matrices for coding and decoding processes. Second stage removes the redundancy present in the first stage.

1) Stage-1 Compression Algorithm

Stage-1 is based on only two matrices, binary matrix and grayscale matrix. Stage-1 compression is performed based on the following steps.

a) Read the original image matrix, OR
b) Construct the Binary Matrix, BM

- First element in the BM is set to 1. Rest all are set as follows:
- $[BM]_{i,j} = 0, [OR]_{i,j} = [OR]_{i,j-1}$
- $1, [OR]_{i,j} \neq [OR]_{i,j-1}$

c) Construct the Grayscale Matrix, GSM
- First element in the GSM is set to the first value in the original matrix OR. The remaining elements of GSM are calculated as follows:
- $[GSM]_k = \text{NULL}, [OR]_{i,j} = [OR]_{i,j-1}$
- $[OR]_{i,j}, [OR]_{i,j} \neq [OR]_{i,j-1}$

d) Compressed bit-stream is the combination of both the matrices.

e) Binary matrix consists only of 0's and 1's. So 1-bit is enough to store each element of Binary matrix. A byte is formed by combining consecutive 8 bits in the Binary matrix. So Binary matrix will always takes memory footprint of (width * height) bits = (width * height / 8) bytes.

f) Grayscale matrix elements can have any value between 0 and 255. So, each element in grayscale matrix requires 1-byte.

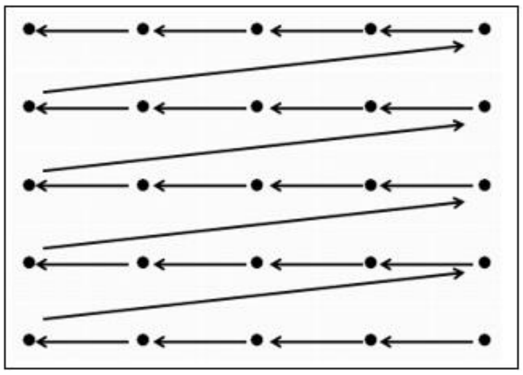


Figure 1: Original Pixels Comparison

Let us take an example to demonstrate this.

Original Matrix [OR]:

10	10	10	10
20	20	20	20
30	30	30	30
30	30	30	30

The pixel value varies from 0 to 255 and it takes one byte for each pixel. So above matrix takes 16 bytes.

Binary Matrix [BM]:

1	0	0	0
1	0	0	0
1	0	0	0
0	0	0	0

Each element of the Binary matrix takes only 1-bit and so the above matrix takes 16 bits which is 2 bytes.

From the above matrix, combine consecutive 8-bits to form byte values. With the above available 16 bits, byte values are formed as follows:

[10001000, 10000000] => [136, 128]

Grayscale Matrix [GSM]:

[10 20 30]

As the elements of Grayscale matrix takes 1-byte, the above matrix takes 3 bytes.

Compressed Bitstream:
It is a combination of GSM & BM. So it takes 2 + 3 = 5 bytes of memory footprint as follows:

[136, 128, 10, 20, 30]

2) Stage-2 Compression Algorithm

Pixel Code String Algorithm	
0000 0001 To 1111 1111	One pixel in color 1 To One pixel in color 255
0000 0000 0LLL LLLL	L pixels (1 – 127) in color 0
0000 0000 1LLL LLLL CCCC CCCC	L pixels (3 – 127) in color C (L > 2)

- Above table is constructed based on prefix codes
- No codeword will be the prefix for any other code.
- Designed to compress Binary matrix of Stage-1
- Binary matrix will have more number of zeroes (0's) for medical images. So the probability of zero is more for Binary Matrix
- Hence a separate codeword is designed in the above table for the symbol 0 (color 0)
- And at last the table is tailored specially for Binary Matrix constructed with medical images.
- Count consecutive same byte values, C, in the Binary matrix and assign it to L.
- Let us take an example to demonstrate this. If the output of the Binary matrix is

[0, 0, 0, 0, 0, 0, 0, 0, 1, 255, 1, 1, 1, 1, 1, 1, 1, 1, 255, 255, 255, 255, 255, 255, 255, 255, 255]

Input has 27 byte values.
Let us form the L and C for each of the consecutive byte value.

[0, 0, 0, 0, 0, 0, 0, 0]	=> L = 8, C = 0
[1]	=> L = 1, C = 1
[255]	=> L = 1, C = 255
[1, 1, 1, 1, 1, 1, 1, 1]	=> L = 8, C = 1
[255,255,255,255,255,255,255,255,255]	=> L = 9, C = 255

Let us form the pixel code string with the above L and C values (Refer the pixel code string table to form this).

L	C	Pixel Code String	Pixel Code String (Decimal)
8	0	0000 0000 0000 1000	0 8
1	1	0000 0001	1
1	255	1111 1111	255
8	1	0000 0000 1000 1000 0000 0001	0 136 1
9	255	0000 0000 1000 1001 1111 1111	0 137 255

Output of pixel code string has 80 bits which is equal to 10 bytes where as the input has 27 bytes.

[0, 8, 1, 255, 0, 136, 1, 0, 137, 255]

3) Stage-2 Decompression Algorithm

- Read the compressed Bitstream
- Convert the decimal values to 8-bit Binary values
- Read 8-bits of data from the compressed Bitstream
- If they are not equal to '0000 0000' then copy that to output (decompressed).
- If it equals '0000 0000' then read the next 8 bits.
- Split the 1st bit and form 'L' with the next 7 bits.
- If the 1st bit equals '0' then store 'L' times 0 (decimal) to the output (decompressed).
- If the 1st bit equals '1' then read the next 8 bits and form 'C'. Store 'L' times the value 'C' to the output (decompressed).
- Goto step 3 till we reach the end of the compressed bitstream.

Pixel Code String Algorithm	
0000 0001 to 1111 1111	One pixel in color 1 to One pixel in color 255
0000 0000 0LLL LLLL	L pixels (1 – 127) in color 0
0000 0000 1LLL LLLL CCCC CCCC	L pixels (3 – 127) in color C (L > 2)

Example:
[0, 8, 1, 255, 0, 136, 1, 0, 137, 255]

Pixel Code String (Decimal)	Pixel Code String (Binary)	L	C
0	0000 0000	8	0
8	0000 1000		
1	0000 0001	1	1
255	1111 1111	1	255
0	0000 0000	8	1
136	1000 1000		
1	0000 0001		
0	0000 0000	9	255
137	1000 1001		
255	1111 1111		

L = 8, C = 0 => [0, 0, 0, 0, 0, 0, 0, 0]
L = 1, C = 1 => [1]
L = 1, C = 255 => [255]
L = 8, C = 1 => [1, 1, 1, 1, 1, 1, 1, 1]
L = 9, C = 255 => [255,255,255,255,255,255,255,255,255]
So the decompressed output is as follows:
[0, 0, 0, 0, 0, 0, 0, 0, 1, 255, 1, 1, 1, 1, 1, 1, 1, 1, 255, 255, 255, 255, 255, 255, 255, 255, 255]

A. Stage-1 Decompression Algorithm
1) Get the 2 matrices, GSM and BM.
2) Expand the Binary Matrix – Convert to Binary values
3) Read one element from GSM and assign to VAL.
4) Read one bit of BM
5) Copy the VAL to the output (decompressed)
6) Read the next bit of BM
7) If the bit equals '0' copy the VAL to the output
8) If the bit equals '1' read the next element from GSM
9) Repeat from Step 6 till the end of the BM.

Example:

Let the compressed bitstream be [136, 128, 10, 20, 30] with BM = [136, 128] and GSM = [10, 20, 30]

Expand the Binary Matrix:

BM Decimal Value	BM (8-bit Binary representation)
136	1000 1000
128	1000 0000

BM bit	Output (Decompressed)	Description
1	10	**
0	10	##
0	10	##
0	10	##
1	20	$$
0	20	##
0	20	##
0	20	##
1	30	$$
0	30	##
0	30	##
0	30	##
0	30	##
0	30	##
0	30	##
0	30	##

** - Read the first element from GSM and store to output
## - Copy the previous value from GSM to output
$$ - Read the next element from GSM and store to output

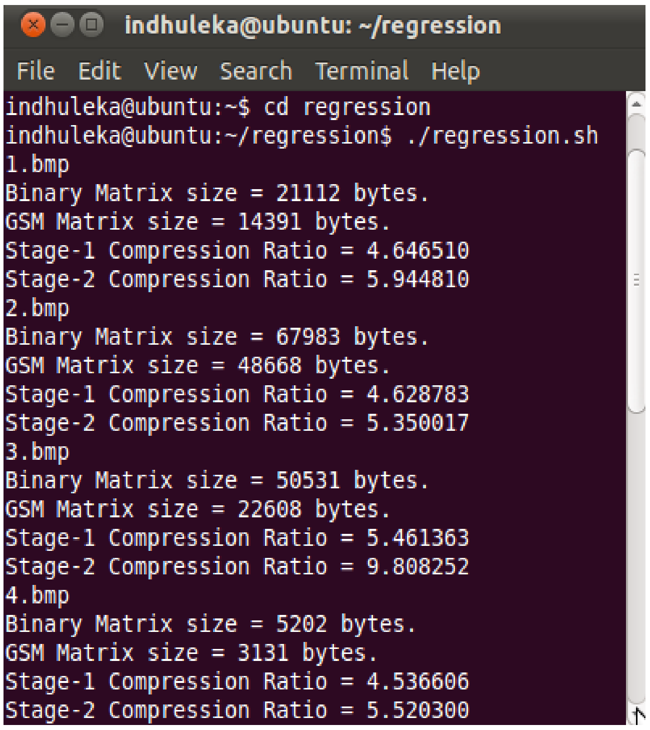
4. Implementation

The above mentioned algorithms were implemented in C under Ubuntu Linux 12.04 because we want to publish our work to the open source community. Also it is very easy to write a shell script under Linux to run the same algorithm multiple times. We developed a shell script, *regression.sh*, to automate the algorithm. We have chosen C language because the code written in C can be ported to any hardware/processor. The medical devices will have an embedded microcontroller inside it. So the code can be easily ported to that microcontroller using its associated development tools. The above algorithms were tested with 230 images of different resolutions and different types.

a) *Regression setup*

A shell script was written named *regression.sh* to run the same algorithm with different images. The script reads each image from the image database and calls the C written algorithm by passing the image through command line arguments. Our image database consists of 230 images with different resolutions and different categories namely CT, MRI, lung, X-Ray etc.,

A snapshot of our regression setup is shown below.



```
indhuleka@ubuntu:~$ cd regression
indhuleka@ubuntu:~/regression$ ./regression.sh
1.bmp
Binary Matrix size = 21112 bytes.
GSM Matrix size = 14391 bytes.
Stage-1 Compression Ratio = 4.646510
Stage-2 Compression Ratio = 5.944810
2.bmp
Binary Matrix size = 67983 bytes.
GSM Matrix size = 48668 bytes.
Stage-1 Compression Ratio = 4.628783
Stage-2 Compression Ratio = 5.350017
3.bmp
Binary Matrix size = 50531 bytes.
GSM Matrix size = 22608 bytes.
Stage-1 Compression Ratio = 5.461363
Stage-2 Compression Ratio = 9.808252
4.bmp
Binary Matrix size = 5202 bytes.
GSM Matrix size = 3131 bytes.
Stage-1 Compression Ratio = 4.536606
Stage-2 Compression Ratio = 5.520300
```

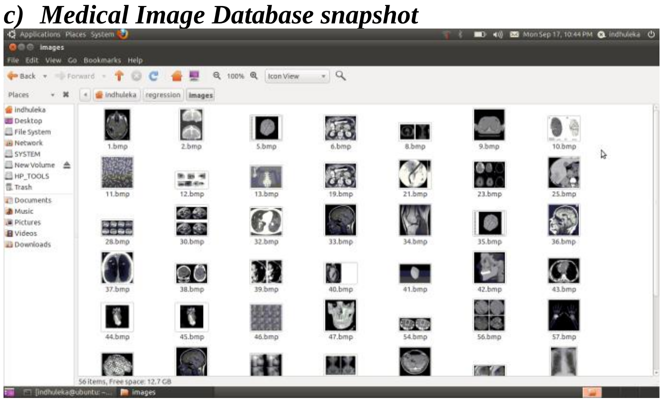
indhuleka@ubuntu: ~/regression
File Edit View Search Terminal Help

b) *Regression Results*

We collected the results of our algorithm in a spread sheet format to compute the compression ratio of each stage. A snapshot of that is shown in the below table.

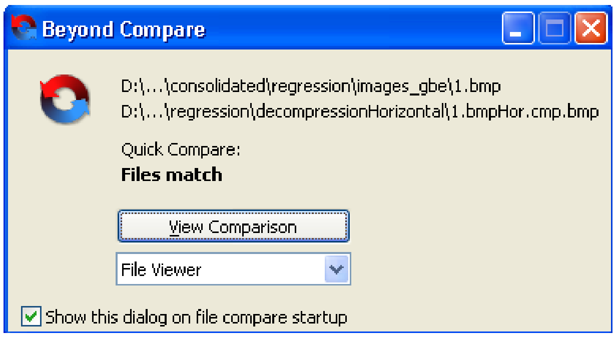
Image	Input Size	Stage-1 compress size	Stage-2 compress size	Stage-1 comp ratio	Stage-2 comp ratio
1.bmp	169974	36581	28592	4.65	5.94
2.bmp	544942	117729	101858	4.63	5.35
3.bmp	405326	74217	41325	5.46	9.81
4.bmp	42694	9411	7734	4.53	5.52
5.bmp	189238	43489	37202	4.35	5.09
6.bmp	207958	43549	36213	4.78	5.74
7.bmp	186358	40144	32032	4.64	5.81
8.bmp	255094	59486	44909	4.29	5.68

c) *Medical Image Database snapshot*



d) *Lossless compression*

An algorithm is said to be lossless if the decompressed image exactly bit-matches with the input image of the compression algorithm. Hence we wrote the equivalent decompression algorithm in C. We bit-matched all the 230 images in the database with the reconstructed images generated from the decompression algorithms. We used the Beyond compare software freely available in the Internet to compare the input image and the reconstructed image.



Beyond Compare

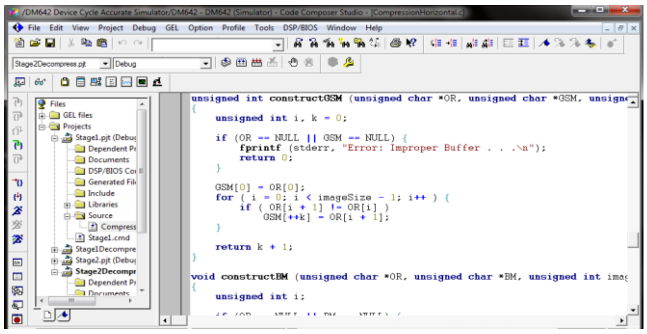
D:\...\consolidated\regression\images_gbe\1.bmp
D:\...\regression\decompressionHorizontal\1.bmpHor.cmp.bmp

Quick Compare:
Files match

e) *Porting on Embedded Processor*

These types of low complexity algorithms are required for the implementation in low power embedded processors. So we ported the C written algorithms (both Stage-1 and Stage-2) on a low power DSP processor DM642 for measuring the code size and for validating the algorithms. We also verified that the algorithms are lossless by decompressing the compressed bit-stream with the help of the equivalent decompression algorithms which were also written in C.

A snapshot of the Code Composer Studio IDE is shown below.



```
unsigned int constructGSM (unsigned char *OR, unsigned char *GSM, unsigne
{
    unsigned int i, k = 0;

    if (OR == NULL || GSM == NULL) {
        fprintf (stderr, "Error: Improper Buffer . . .\n");
        return 0;
    }

    GSM[0] = OR[0];
    for ( i = 0; i < imageSize - 1; i++ ) {
        if ( OR[i + 1] != OR[i] )
            GSM[++k] = OR[i + 1];
    }

    return k + 1;
}

void constructBM (unsigned char *OR, unsigned char *BM, unsigned int imag
{
    unsigned int i;
```

5. Performance Comparison

a) *Compression Ratio*

The results are obtained from testing of the 230 images available in the image database. The performance of the proposed algorithm is compared with other research results. The result comparisons are shown below.

Method	Technique	Performance (Avg)
Lurawave sw	Wavelets-LS	2.7
JPEG Encoder	JPEG-LS	2.9
Proposed Algorithm	Stage-1	4.3
Proposed Algorithm	Stage-2	5.9

From the above table, it is clear that the stage-2 proposed compression algorithm gives the best compression ratio.

b) *Memory code size on DM642 DSP Processor*

The memory required to fit the code in memory is computed using Code Composer Studio IDE for DM642 DSP processor and is shown below

Algorithm	Code Memory size in bytes
Stage-1 compression	928
Stage-2 compression	1312
Stage-1 Decompression	672
Stage-2 Decompression	1056
JPEG-LS	175424

So, the proposed algorithm consumes less memory when compared to the standard JPEG-LS compression algorithm and can be implemented even in a microcontroller with limited memory attached to it.

6. Conclusion

In this paper an efficient, simple lossless image compression technique is proposed with a remarkable compression ratio and greatly reduced computation load while keeping low complexity compared with other methods. Algorithm complexity is directly related to the power consumption of the processor. As the medical devices injected into the body are battery powered devices and the power consumption should be kept as low as possible to increase the lifetime of the battery. Mainly these kinds of algorithms are targeted in applications involving medical devices powered with ultra low power processors like MSP430.

Moreover, it is not necessary to compress the images using both the stages. We can turn off stage-2 compression if the battery is low which in turn reduces the complexity of the algorithm to slightly increase the lifetime.

References

[1] S.E.Ghrare, M.A. Mohd. Ali, K. Jumari and M. Ismail, An Efficient Low Complexity Lossless Coding Algorithm for Medical Images, 2009 American Journal of Applied Sciences