Journal homepage www.jzs.univsul.edu.iq
Journal of Zankoy Sulaimani
Part-A- (Pure and Applied Sciences)

New Image Compression/Decompression Technique Using Arithmetic Coding Algorithm

Nassir H. Salman[1]

1Faculty of science- Computer Science Dept.- Cihan University-100 street, Erbil, IRAQ
Email:nahu64@yahoo.com

Article info

Original: 21 July 2016
Revised: 12 October 2016
Accepted: 16 October 2016
Published online: 20 March 2017

Key Words:
arithmetic coding, image compression, lossless compression technique, decoding

Abstract

Arithmetic coding is a form of entropy encoding used in lossless data compression. In this article, new technique for image compression/decompression based on arithmetic coding decoding was used. The input image is divided into blocks, each block is 32 x 32 , 64 x 64, and 128 x 128 pixels. If the image is not multiple of the block size , the size of what remains of the image is calculated and columns and rows of zero values are added(padding) up to be all the blocks are 32 x 32,64 x 64..etc . The blocks are separated first as a stream for processing. The result is a stream of compressed tables after do arithmetic coding process and the store bytes are calculated during this process. Then compression ratio is calculated. Finally the stream of the compressed tables is recollected for decompressing (reconstruction image after decoding process) and this is done for each block. The algorithm is tested on different images and the results show a high space saving (88-90%), the process is fast, and it is easy to decode the binary bits to recover the original image exactly without any lose.

Introduction

Arithmetic coding is a data compression technique that encodes data (character, image,….) by creating a code which represents a fraction in the unit interval [0, 1]. The algorithm is recursive. And on each recursion, the algorithm successively partitions subintervals of the unit interval [0, 1]. This means in arithmetic coding, instead of using a sequence of bits to represent a symbol, a subinterval of the unit interval$[0, 1]$ is used to represent that symbol. In other words, the data is encoded into a number in the unit interval$[0, 1]$, and this technique can be implemented by separating the unit interval into several segments according to the number of distinct symbols. The length of each segment is proportional to the probability of each symbol, and then the output data is located in the corresponding segment according to the input symbol.[1]

Arithmetic coding (AC) is a very efficient technique for lossless data compression and it produces a rate which approaches the entropy of the encoded data (i.e a form of entropy encoding used in lossless data compression). AC is widely used in the well known image and video compression algorithms such as JBIG, JBIGS,JPEG2000 and H.264/AVC. AC is also a statistical coding technique which needs to work with a model that estimates the probability of each pixel at each iteration in the encoding and decoding processes. In this case it needs to find the probability distribution of symbols that maximize the compression efficiency [2].

Normally, a string of characters is represented using a fixed number of bits per character, as in the ASCII code. When a string is converted to arithmetic encoding, frequently used characters will be stored

with fewer bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total. In this case arithmetic coding differs from other forms of entropy encoding, such as Huffman coding, in that rather than separating the input into component symbols and replacing each with a code, arithmetic coding encodes the entire message into a single number, a fraction *n* where ($0.0 \leq n < 1.0$) [1].

In [2] an image is divided into non-overlapping blocks and then each block is encoded separately by using arithmetic coding. The proposed model provides a probability distribution for each block which is modeled by a mixture of non-parametric distributions by exploiting the high correlation between neighboring blocks. The Expectation-Maximization algorithm is used to find the maximum likelihood mixture parameters in order to maximize the arithmetic coding compression efficiency.

In [3] an efficient algorithm has been proposed for lossy image compression /decompression scheme using histogram based block optimization and arithmetic coding , the results shows that the algorithm gives better compression ratio as that of JPEG

In [4] a complete program was used for coding stream of symbols into stream of bits by using Arithmetic Coding Algorithm. The input to the coding function is array of "Data" such as [3 0 -1 129 9 -255 255 0 0 -3] and this program accept negative, positive and zero data, where in our work, image pixels values store as vector and used as input to coding and decoding functions**.**

The reset of the paper is organized as follows. In section 1 the basic concepts of arithmetic coding is introduced. The motivation and the coding theorem behind arithmetic coding is explained in section 2. In section 3 the arithmetic coding procedures is given. While the method of generating a binary sequence is presented in section 4. The proposed method and the algorithm steps is described in section 5. In section 6 the results of using arithmetic code to compressed image data is presented. Finally, the paper is end up with the conclusion.

Using Arithmetic Coding

The source symbols can be mapped to numbers. In the following discussion, we will use the mapping equation (1);[1][6]

$$X(a_i) = i, a_i \in \mathcal{A} \qquad (1)$$

Where, $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$ is the alphabet for a discrete source and X is a random variable. This mapping means that given a probability model for the source, we can obtain a probability density function (*pdf*) for the random variable ; equation (2):

$$Pr(X = i) = P(a_i) \qquad (2)$$

and the cumulative density function (*cdf*) can be expressed as

$$F_X(i) = \sum_{k=1}^{i} Pr(X = k). \quad (3)$$

Hence for each symbol a_i with a nonzero probability, there is a distinct value of $F_X(i)$. Since the value of $F_X(i)$ is distinct for distinct symbol a_i, it is able to use this fact in what follows to develop the arithmetic code. For more clear, the following example illustrates restricting the interval containing the tag for the input sequence (a_1, a_2, a_3).[6]

Consider a ternary alphabet $\mathcal{A} = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, and $P(a_3) = 0.2$. Using the equations (2) and (3) above, we have $F_x(1) = 0.7$, $F_x(2) = 0.8$, and $F_x(3) = 1.0$. Now let us consider an input sequence (a_1, a_2, a_3). Then this partitions the unit interval as shown in Figure 1.[1]

The partition in which the tag resides depends on *the first symbol of the sequence*. If the first symbol is a_1, then the tag lies in the interval [0.0, 0.7); if the first symbol is a_2, then the tag lies in the interval [0.7,0.8); if the first symbol is a_3, then the tag lies in the interval [0.8, 1.0). Once the interval containing the tag has been determined, the rest of the unit interval is *discarded*, and this *restricted interval is again divided in the same*

proportions as the original interval. Now the input sequence is (a_1, a_2, a_3). So, the first symbol is a_1, and the tag would be contained in the subinterval [0.0, 0.7). This subinterval is then subdivided in exactly the same proportions as the original interval, yielding the subintervals [0.00, 0.49), [0.49, 0.56),

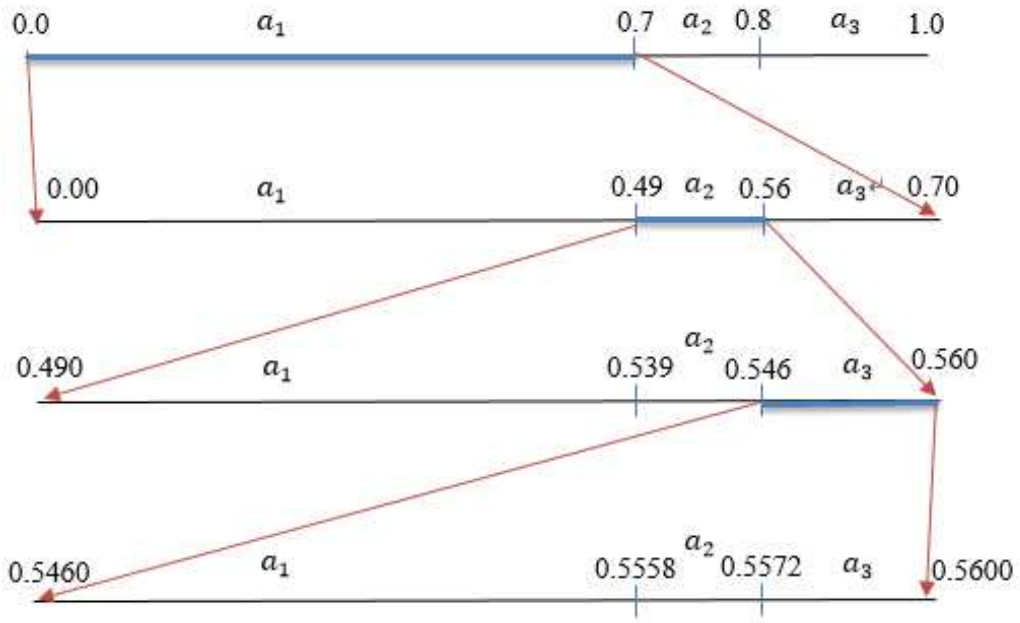


Figure -1: Restricting the interval containing the tag for the input sequence (a_1, a_2, a_3)

and [0.56, 0.70).The first partition [0.00, 0.49) corresponds to the symbol a_1, the second partition [0.49, 0.56) corresponds to the symbol a_2, and the third partition [0.56,0.70) corresponds to the symbol a_3. And because the second symbol in the sequence is a_2. Then the tag value is then restricted to lie in the interval [0.49, 0.56). Following is a simple calculating for these intervals:

0.7-0=0.7 starting with subinterval of a_1, then

0.7*probability=0.7 * .7=0.49 [0-0.49] for a_1, 0.7 *.8=0.56 [0.49 -0.56] represent a_2 interval, then continue for a_3 [0.56-0.7]. Now using interval of a_2 , the new intervals become:

0.56-0.49=0.07 *.7=0.049 +0.49=0.539 [0.49 -0.539] represent a_1,

0.07 *0.1=0.007 +0.539=0.546 [0.539-0.546] represent a_2

0.07 *0.2=0.014+0.546=0.560 , [0.546 -0.560] represent a_3

So, this interval **[0.49 -0.56] is partitioned** in the same proportion as the original interval in order to obtain the subinterval [0.49, 0.539) corresponding to the symbol a_1 , the subinterval [0.539,0.546) corresponding to the symbol a_2 and the subinterval [0.546,0.560) corresponding to the symbol a_3. Now the third symbol is a_3, then the tag will be restricted to the interval [0.546,0.560), which can be subdivided further by the same procedure described above.

Note that the appearance of each new symbol restricts the tag to a subinterval that is disjoint from any other subinterval that may have been generated using this process. For the sequence (a_1, a_2, a_3), since the third symbol is a_3, the tag is restricted to the subinterval [0.546, 0.560). If the third symbol is a_1 instead of a_3, the tag would have resided in the subinterval [0.49, 0.539), which is disjoint from the subinterval [0.546,0.560).

Arithmetic Coding Review

After reading the image, a matrix of pixel intensities will obtain. For a gray-scale image, the intensity values range from 0 to 255. All we have to do is make a list of all the intensity values i.e., 0 - 255 and find

out the no. of pixels that belong to each of these intensities (calculate statistical distribution of image pixels). And then dividing those numbers in each entry by the total number of pixels to obtain the frequency (probability) of each of these intensities in the input image. These intensities are your symbols with their corresponding probabilities. At this step you have the familiar parameters to use the Arithmetic Coding. So; any source coding technique is the final step in encoding and if you want to directly implement arithmetic coding method on image, you need to add a few steps as follows:

1. Read any image file format.
2. From the image matrix, find out the different intensity values that are used in the image and make out a list of them.
3. Find the frequency of occurrence (probability) of each of intensity values in the image.

Now the intensity values list make up the source symbols and frequency of occurrence will be their corresponding probabilities [5]. Then those parameters are supplied to arithmetic coding function.

Generating A Binary Sequence

A. Efficiency of the Arithmetic Code

Consider a source A that generates letters from an alphabet of size four and five with probabilities shown in tables (1 and 2) below. According to the equations (3 in section 2 ,and 4 below), A binary code for this source can be generated [6].

$$\overline{F_X}(x_i) = \sum_{k=1}^{i-1} P(X=k) + \frac{1}{2}P(X=i)$$

$$= F_x(i-1) + \frac{1}{2}P(X=i) \dots\dots\dots\dots\dots (4)$$

Equation(4) above means: for each x_i , $\overline{F_X}(x_i)$will have a unique value in the interval [0, 1] .This value can be used as a unique for x_i . Therefore, a binary code for $\overline{F_X}(x_i)$ can be obtained by taking the binary representation of this number (fraction number) and truncating it to $l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$ bits, where log (.) denotes the logarithm to base 2 as follows:

$$log_2(X) = log_{10}(X)/log_{10}(2),$$

For example for symbol no.1 in Table-1 below, the calculations are as follows:

$$\left\lceil \log \frac{1}{.25} \right\rceil + 1 = log_2 4 + 1 = log_{10}(4)/log_{10}(2) + 1 = 0.602059/0.30102 + 1 = 3$$

Table -1: Shows 4 symbols with their binary representation and binary code calculations

Symbol	$P(x)$	$F_X(x)$	i	$\overline{F}_X(x)$	$\overline{F}_X(x)$ in binary	$l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$	Codeword
1	0.25	0.25	1	0.125	0.001	3	001
2	0.5	0.75	2	0.5	0.10	2	10
3	0.125	0.875	3	0.8125	0.1101	4	1101
4	0.125	1.0	4	0.9375	0.1111	4	1111

The quantity $\overline{F}_X(x)$ is obtained using Equation (4), and the binary representation of $\overline{F}_X(x)$ is truncated to $l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$ bits to obtain the binary code. , $F_x(0) = 0$, see eq(4), and

$F_X(1) = (P(X=1) = 0.25$

$F_X(2) = P(X=1) + P(X=2) = 0.25 + 0.5 = 0.75$, but

$$\overline{F}_X(1) = F_X(1-1) + \frac{1}{2}P(X=1) = 0 + \frac{1}{2}\ x\ 0.25 = 0.125$$

$$\overline{F}_X(2) = F_X(1) + \tfrac{1}{2}P(X=2) = 0.25 + \tfrac{1}{2}\ x\ 0.5 = 0.5,$$

and so on, see Table -2.

Table -2: Shows 5 symbols with their binary representation and binary code calculation

Symbol	$P(x)$	$F_X(x)$	$\overline{F}_X(x)$	$\overline{F}_X(x)$ in binary	$l(x) = \left\lceil \log \frac{1}{P(x)} \right\rceil + 1$	Codeword
1	0.25	0.25	0.125	0.001	3	001
2	0.25	0.50	0.375	0.011	3	011
3	0.20	0.70	0.60	$0.1\overline{0011}$	4	1001
4	0.15	0.85	0.775	$0.110\overline{0011}$	4	1100
5	0.15	1.00	0.925	$0.111\overline{0110}$	4	1110

It can be proved that a code obtained in this fashion is a uniquely decodable code. For more about arithmetic coding details see [7-12].

Proposed Method and Algorithm Steps

Figure -2. Shows the steps of using Arithmetic Coding as follows:

1-Read any image file format (i.e; .jpg, gif,. png…etc)

2-Divide the image into blocks (each block size is 64 x 64), moving or sliding window which is designed to strip the image into blocks. The blocks are separated first as a stream for processing. Counter of number of blocks. Calculate the size of what remains of the image to complete it into block of size 64 x64 through zero padding. Call Arith_Code(f1) function to code image pixels values .The result is a stream of compressed tables , in this step:A complete program for coding stream of symbols into stream of bits by using Arithmetic Coding Algorithm [4] was used. The input to the function is array of "Data". Also in this step, the following were done:

- Compute the table of the symbols (pixels)
- Compute the probability of the symbols
- Apply Arithmetic coding using:code = arithenco(New_Data,Counts) to calculate the Store_Byte,Counts,Table of pixels.

3-Calculate the compressed image file size ,where the amount of data for the block stream of the original image without compression and the amount of data for the block stream after the compression were calculated.

4- Then calculate the space saving =(M*N*8-sum(BlockSize))/(M*N*8); origin image file size – compressed size divide by origin image. And the compression Ratio(CR%)=Uncompressed image size/compressed size

5-And finally the origin image was recovered from decompressed one, where the stream is recollected for decompressing process and this is done for each block: This step represents decoding process by Arithmetic Decoding using arithdeco(code, Counts, length(Data)) function;

6-Calculate the elapsed time(sec) for compressing /decompressing different images.

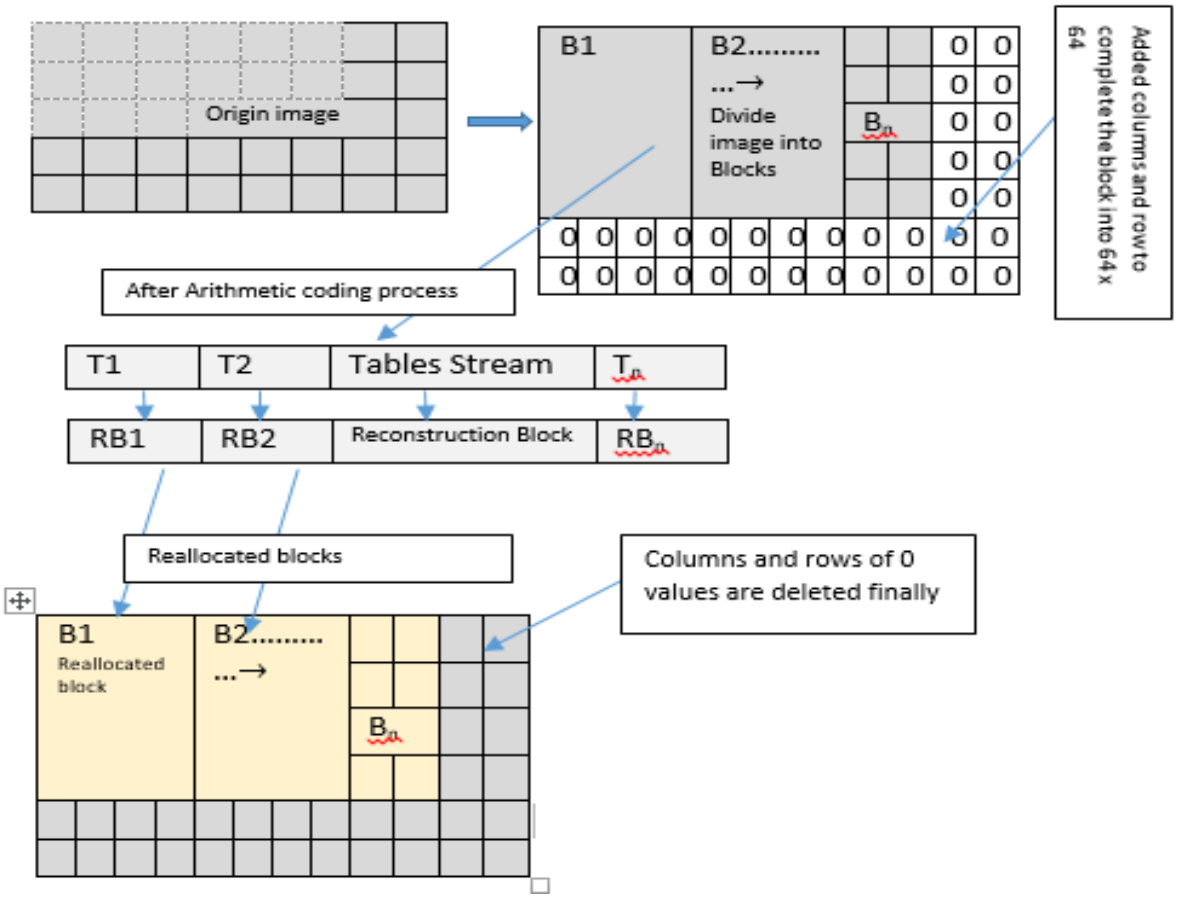


Figure- 2:Shows the proposed method and algorithm steps

Test and Results

First of all, many samples of gray scale medical images of different size and format were read and the following results are done:

1- Displaying the original data of the input image
2- The table of pixel values where the occurrence pixels display one time only,this means the values in the table included only the image data without repeating
3- Calculating the count of each occurrence of each pixel value
4- The probability of the table symbols vales are calculated. If the following data represents the original image data,

```
The original data is  = 25  35  34  47  75 112 134  75  72  49
26  46  58 115 105 122 102  76  95  36  26  43  68  84  55
99  67 104  98  19  27  50  69  44  70  83  68 156  58  17
27  58  51  61  87  63 117 177  25  20  29  56  59  75 100
55  90 203  47  21  35  47  50  74  89  80  79 190  64  24
42  74  57  58  89 122  96  89  50  23  48  79  99  77  97
138 158 104  37  35  49  80  84  91  68  72 182 179  47  45
%
```

Then the table of pixels values and the count of each occurrence as follows:

The table of pixel values is = 25 35 34 47 75 112 134 72 49
26 46 58 115 105 122 102 76 95 36 43 68 84 55 99
67 104 98 19 27 50 69 44 70 83 156 17 51 61 87
63 117 177 20 29 56 59 100 90 203 21 74 89 80 79
190 64 24 42 57 96 23 48 77 97 138 158 37 91 182
179 45

The Count of each occurence of each pixel value is = 2 3 1 4 3 1 1
2 2 2 1 4 1 1 2 1 1 1 1 1 3 2 2 2 1 2 1 1 2 3 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 2 2 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1

The values in the table included only the image data without repeating, where the original data represents the all input data (here are the pixels values).And the probability of the table symbols vales are calculated as follows:

The probability of table symbols value = 0.02 0.03 0.01
0.04 0.03 0.01 0.01 0.02 0.02
0.02 0.01 0.04 0.01 0.01 0.02
0.01 0.01 0.01 0.01 0.01 0.03
0.02 0.02 0.02 0.01 0.02 0.01
0.01 0.02 0.03 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.02
0.03 0.02 0.02 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01 0.01 0.01 0.01 0.01
0.01 0.01

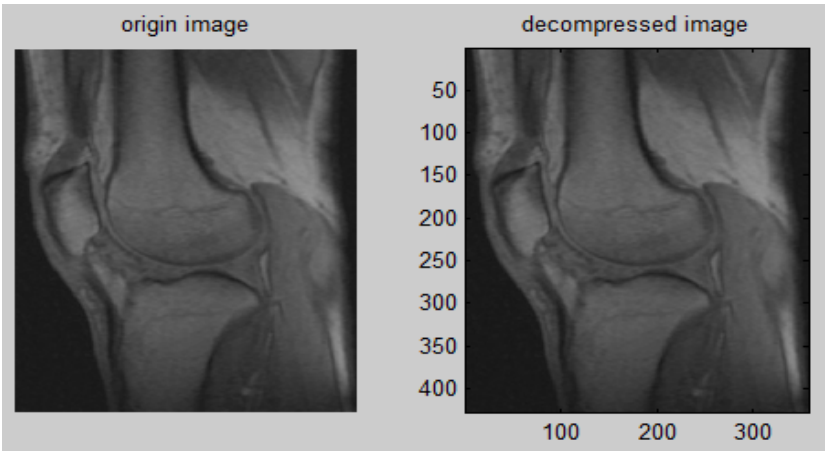
Simply the following calculations showed coding data using Arith_Code1(input image) [4]

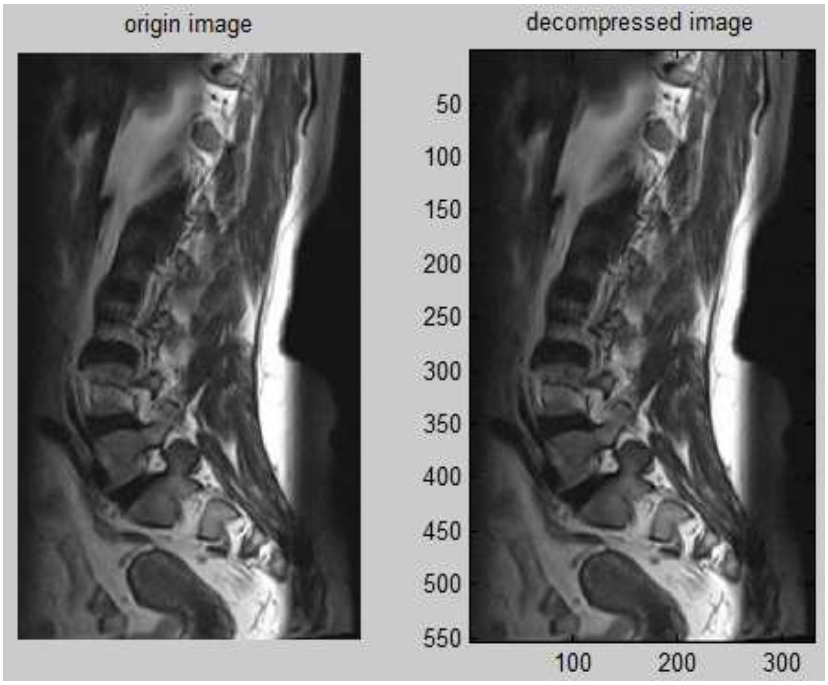
If the pixels values are as follows:

[10 8 8 10 9 9 9 12 0 0 10], this coding into

D_Bits = **<u>00010101</u>**10001110**<u>11111110</u>**0100000 = 31 bits converted into bytes
Store_Byte =21 142 254 64, where 21 represented in binary **<u>00010101</u>**
Counts = 3 2 3 1 2 , this means the value 10 is occurrence 3 times, and 8 is 2 times….etc
Table = 10 8 9 12 0, image pixels without repeating the occurence, see pixels values above

The following Figure 3 shows the origin image, and recovered image after compressing process decompressing processes using Arithmetic coding for some medical images got them from Rezgari Hospital, Erbil-Iraq





(a) (b)

Figure-3: Shows (a) the origin image,(b) recovered image using Arithmetic coding

Table -3: Shows compression ratio for three different images with 3 different block sizes

For block size 32 x 32

Image type and resolution	Image file size(bit)	Compression image file size(bit)	Space Saving *	CR%*	Elapsed time(sec)	Applications
331 x 553 .jpg	1464344	384578	73.737	3.80	75.7732	medical
357 x 429 .jpg	1225224	324607	73.506	3.77	53.3260	medical
500 x 362 .jpg	1448000	390733	73.016	3.70	70.7420	Normal image

* CR= Uncompressed size/Compressed size

* Space savings=(M*N*8-sum(BlockSize))/(M*N*8); [origin image file size – compressed size] divide by the origin image size

For block size 64 x 64

Image type and resolution	Image file size(bit)	Compression image file size(bit)	Space Saving*	CR* %	Elapsed time(sec)	Applications
331 x 553 .jpg	1464344	145604	90.057	10.05	77.8591	medical
357 x 429 .jpg	1225224	139349	88.627	08.79	57.0807	medical
500 x 362 .jpg	1448000	160892	88.889	09.00	79.6536	Normal image

For block size 128 x 128

Image type and resolution	Image file size(bit)	Compression image file size(bit)	Space Saving*	CR * %	Elapsed time(sec)	Applications
331 x 553 .jpg	1464344	536633	63.350	2.72	82.6872	medical
357 x 429 .jpg	1225224	441130	63.996	2.77	63.0074	medical
500 x 362 .jpg	1448000	553257	61.792	2.61	83.7527	Normal image

Conclusion

The space saving is large (about 90%) for medical images we used. The original image is recovered easy from the compression one without any lose (losseless compression tech). The processing time becomes short with using blocks compare with using one block (the whole image).The proposed method can be used for

image data or any array data to coding and encoding without any lose. Arithmetic coding is very useful with medical images compare with using .jpeg format (lossy) because it is losses, and the maximum CR is 10% which mean for example ,if the image file size is 100kbyte becomes 90 Kbytes.

Finally one advantage of arithmetic coding over other similar methods of data compression is the convenience of adaptation which is the changing of the frequency (or probability) tables while processing the data. The decoded data matches the original data as long as the frequency table in decoding is replaced in the same way and in the same step as in encoding. The synchronization is, usually, based on a combination of symbols occurring during the encoding and decoding process. The proposed method give good space saving and compression ratio, that is useful for sending compression images via e-mail (i.e. small size) for diagnostic, for storing a lot of image files in the hospital.

The best CR (10%)and saving space (90%) is for block size 64 x 64 , where for block size 128 x 128 is less ratios(see table 3. Also the processing time depend upon image size and the block size for each case individually , where the short time(fast) for block 32 x 32 comparing with other block sizes for the same images. The space saving is high with block size 64 x 64 , where the less space with block size 128 x 128 pixels.

At the end our idea is to do compression using gray image by dividing it into blocks not to use the whole image using AC algorithm. It is found that the AC algorithm is very good for small data (i.e;1D array numbers) for coding and decoding . But for the whole image (huge data) it is slow and the CR is small, that is why we divided our image into blocks (32 x32, 64 x 64, and 128 x 128) pixels to improve these factors. Finally most article used AC with small data not with huge data as we did in our article.

References

[1] Khalid Sayood. *"Introduction to Data Compression"* , 3rd edition Ch4-P/81. Elsevier Inc.(2006).

[2] Atef Masmoudi, William Puech, Afif Masmoudi. *"An improved lossless image compression based arithmetic coding using mixture of non-parametric distributions"*, Multimedia Tools and Applications, Vol. 74, Issue 23, pp. 10605-10619. (2015).

[3] Subarna Dutta etal. *"An efficient image compression algorithm based on histogram based block optimization and arithmetic coding"*. International journal of computer theory and engineering, Vol.4, No.6, (2012).

[4] Mohammed Siddeq. *"Arithmetic Coding and Decoding"*. Sept. 2011 Math works file exchange. See website: http://mamadmmx76.wix.com/mohammedsiddeq

[5] http://www.mathworks.com/matlabcentral/fileexchange/36377-arithmetic-encoding---decoding

[6] Yu-Yun Chang, *"Tutorial: Arithmetic Coding"*. National Taiwan University, Taipei, Taiwan, ROC

[7] Ian H. Willen, Radford M. Neal, and John G. Cleary. *"Arithmetic Coding For Data Compression"*. Communications of the ACM. Vol. 30, No. 6. (1987).

[8] David Salomon. *"Data Compression The Complete Reference"*. Third Edition-Ch2. P-106 , Springer-Verlag New York, Inc. (2004).

[9] David Salomon. *"A Concise Introduction to Data Compression"*, Ch4, p-123. Springer-Verlag London Limited (2008).

[10] David Salomon, Giovanni Motta, D. Bryant . *"Handbook of Data Compression"*. 5th edition .Ch5, p264. Springer-Verlag London Limited (2010).

[11] Ida Mengyi Pu . *"Fundamental Data Compression"* . Ch6.p-101. Elsevier Inc. (2006).

[12] Adam Drozdek . *"Elements of data compression"*, Brooks/Cole, Thomson learning, Inc.(2002).