An approach for Image Compression Using Adaptive Huffman Coding

[1]Jagadeesh B, [2]Ankitha Rao

[1]Vidyavardhaka college of Engineering, Dept of E&C, Mysore, India

[2] NMAMIT, Dept of E & C, Nitte, Mangalore, India

Abstract— Color image processing is an area that has been gaining importance because of the significant increase in the use of digital images over the internet. Compression deals with techniques for reducing the storage required to save an image, or the bandwidth required to transmit it. In this paper, the encoding is done using Adaptive Huffman Coding, which is based on binary tree and in decoding we traverse the tree in a manner identical to that in the encoding procedure. This coding technique can be either lossy or lossless. The lossless compression method can be extended to all type of files as long as the file size is less than the buffer size

Keywords-Color image processing, Adaptive Huffman Coding, binary tree

1. INTRODUCTION

Uncompressed multimedia (graphics, audio and video) data requires considerable storage capacity and transmission bandwidth. Despite rapid progress in mass-storage density, processor speeds, and digital communication system performance, demand for data storage capacity and data-transmission bandwidth continues to outstrip the capabilities of available technologies. The recent growth of data intensive multimedia-based web applications have not only sustained the need for more efficient ways to encode signals and images but also have made compression of such signals central to storage and communication technology. In lossless compression schemes, the reconstructed image, after compression, is numerically identical to the original image. However lossless compression can only achieve a modest amount of compression. An image reconstructed following lossy compression contains degradation relative to the original. Often this is because the compression scheme completely discards redundant information. However, lossy schemes are capable of achieving much higher compression. Under normal viewing conditions, no visible loss is perceived (visually lossless).

II. METHODOLOGY

The image is (QCIF-176x144) is divided into 8x8 blocks. Discrete Cosine Transform is applied to the 8x8 blocks. The obtained DCT values are then quantized using Q-50 scalar quantization. The quantized output is encoded using Adaptive Huffman Coding. The reverse procedure is followed during decompression.

A. Adaptive Huffman Coding

Huffman coding requires knowledge of the probabilities of the source sequence. If this knowledge is not available, Huffman coding becomes a two-pass procedure: the statistics are collected in the first pass, and the source is encoded in the second pass. In order to convert this algorithm into a one-pass procedure, adaptive algorithms were independently developed by Faller and Gallagher, to construct the Huffman code based on the statistics of the symbols already encountered. These were later developed by Knuth and Vitter. Theoretically, if we wanted to encode the (k+1) th symbol using the statistics of the first k symbols, we could recomputed the code using the Huffman coding procedure each time a symbol is transmitted, however, this would not be a very practical approach due to the large amount of computation involved- hence , the adaptive Huffman coding procedures In order to

describe how the adaptive Huffman code works, we add two parameters to the binary tree: the weight of each leaf, which is written as a number inside the node, and a node number. The weight of each external node is simply the number of times the symbol corresponding to the leaf has been encountered. The weight of each internal node is the sum of the weights of its offspring. The node number y_j is a unique number assigned to each internal and external node. If we have an alphabet of size *n*, then the *2n-1* internal and external nodes can be numbered as $y_1,\ldots, y_{2n-1}$ such that if x_j is the weight of node y_j, we have $x_1 \leq x_2 \leq \ldots\ldots \leq x_{2n-1}$. Furthermore, the nodes y_{2j-1} and y_{2j} are offspring of the same parent node, or siblings, for $1 \leq j \leq n$, and the node number for the parent node is greater than y_{2j-1} and y_{2j}. These last two characteristics are called sibling property, and any tree possesses this property is a Huffman tree. In the adaptive Huffman coding procedure, neither transmitter nor receiver knows anything about the statistics of the source sequence at the start of transmission. The tree at both the transmitter and receiver consists of a single node that corresponds to all symbols not yet transmitted (NYT) and has a weight of zero. As transmission progresses, nodes corresponding to symbols transmitted will be added to the tree, and the tree is reconfigured using an update procedure. Before the beginning of transmission, a fixed code for each symbol is agreed upon between transmitter and receiver. A simple code is as follows: If the source has an alphabet (a_1, a_2, …., a_m) of size m, then pick e and r such that $m=2^e+r$ and $0 \leq r \leq 2e$. The letter a_k is encoded as the (e+1)-bit binary representation of *k-1*, if $1 \leq k \leq 2r$; else, a_k is encoded as the e-bit binary representation of *k-r-1*. For example, suppose *m=26*, then *e=4* and *r=10*. The symbol a_1 is encoded as 00000, the symbol a_2 is encoded as 00001, and the symbol a_{22} is encoded as 1011. When a symbol is encountered for the first time, the code for the NYT node is transmitted, followed by the fixed code for the symbol. A node for the symbol is then created, and the symbol is taken out of the NYT list. Both transmitter and receiver start with the same tree structure. The updating procedure used by both transmitter and receiver is identical. Therefore, the encoding and decoding processes remain synchronized.

B. Updating procedure

The update procedure requires that the nodes be in a fixed order. This ordering is preserved by numbering the nodes. The largest node number is given to the root of the tree, and the smallest number is assigned to the NYT node. The numbers from the NYT node to the root of the tree are assigned in increasing order from left to right, and lower level to upper level. The set of nodes with the same weight makes up a block. Figure 2 is a flowchart of the updating procedure. The function of the update procedure is to preserve the sibling property. In order that the update procedures at the transmitter and receiver both operate with the same information, the tree at the transmitter is updated after each symbol is encoded, and the tree at the receiver is updated after each symbol is decoded. The procedure operates as follows: After a symbol has been encoded or decoded, the external node corresponding to the symbol is examined to see if it has the largest node number in its block. If the external node does not have the largest node number, it is exchanged with the node that has the largest node number in the block, as long as the node with the higher number is not the parent of the node being updated. The weight of the external node is then incremented. If we did not exchange the nodes before the weight of the node is incremented, it is very likely that the ordering required by the sibling property would be destroyed. Once we have incremented the weight of the node, we have adapted the Huffman tree at that level. We then turn our attention to the next level by examining the parent node of the node whose weight was incremented to see if it has the largest number in its block. If it does not, it is exchanged with the node with the largest number in the block. Again, an exception to this is when the node with the higher node number is the parent of the node under consideration. Once an exchange has taken place (or it has been determined that there is no need for an exchange), the weight of the parent node is incremented. We then proceed to a new parent node and the process is repeated. This process continues until the root of the tree is reached. If the symbol to be encoded or decoded has occurred for the first time, a new external node is assigned to the symbol and a new NYT node is appended to the tree. Both the new external node and the new NYT node are off springs of the old NYT node. We increment the weight of the new external node by one. As the old NYT node is the parent of the new external node, we increment its weight by one and then go on to update all the other nodes until we reach the root of the tree. Assume we are encoding the message [a a r d v a r k], where our alphabet consists of the 26 lowercase letters of the English alphabet. The updating process is shown in Figure 3.2. We begin with only the NYT node. The total number of nodes in this tree will be 2*26-1=51, so we start numbering backwards from 51 with the number of the root node being 51. The first letter to be transmitted is ***a***. As *a* does not yet exist in the tree, we send a binary code 00000 for ***a*** and then add ***a*** to the tree.

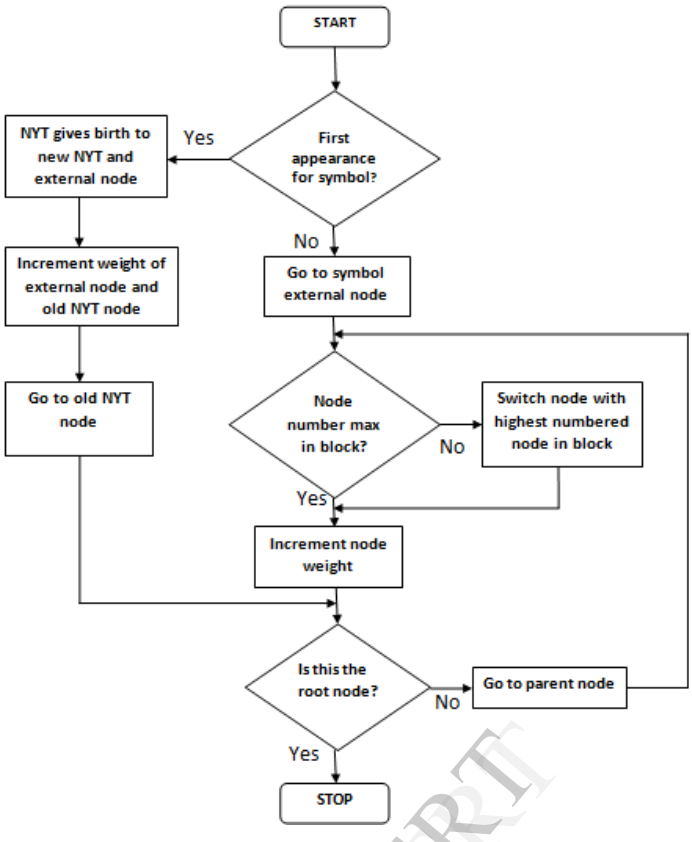


Figure 2: Update procedure for the adaptive Huffman coding algorithm

The NYT node gives birth to a new NYT node and a terminal node corresponding to ***a***. The weight of the terminal node will be higher than the NYT node, so we assign the number 49 to the NYT node and 50 to the terminal node corresponding to the letter ***a***. The second letter to be transmitted is also ***a***. This time the transmitted code is 1. The node corresponding to ***a*** has the highest number (if we do not consider its parent), so we do not need to swap nodes. The next letter to be transmitted is ***r***. This letter does not have a corresponding node on the tree, so we send the codeword for the NYT node, which is 0 followed by the index of ***r***, which is 10001. The NYT node gives birth to a new NYT node and an external node corresponding to ***r***. Again, no update is required. The next letter to be transmitted is ***d***, which is also being sent for the first time. We again send the code for the NYT node, which is now 00 followed by the index for ***d,*** which is 00011. The NYT node again gives birth to two new nodes. However, an update is still not required. This change with the transmission of the next letter, ***v***, which has also not yet been encountered. Nodes 43 and 44 are added to the tree, with 44 as the terminal node corresponding to ***v***. We examine the grandparent node of ***v*** (node 47) to see if it has the largest number in its block. As it does not, we swap it with node 48, which has the largest number in its block. We then increment node 48 and move to its parent, which is node 49. In the block containing node 49, the largest number belongs to node 50. Therefore, we swap nodes 49 and 50 and then increment node 50. We then move to the parent node of node 50, which is node 51. As this is the root node, all we do is increment node 51.

D. Encoding procedure

The flowchart for the encoding procedure is shown in figure 4. Initially, the tree at both the encoder and decoder consists of a single node, the NYT node. Therefore, the codeword for the very first symbol that appears is a previously agreed-upon fixed code.

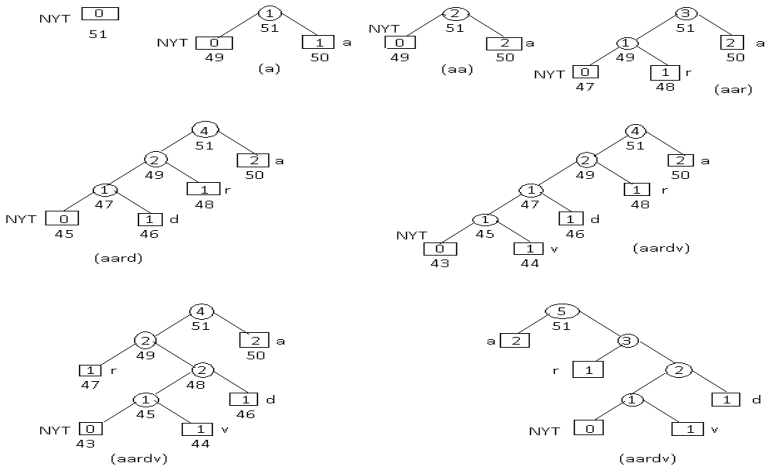


Figure 3: Adaptive Huffman Binary tree construction

After the very first symbol, whenever we have to encode a symbol that is being encountered for the first time, we send code for the NYT node, followed by the previously agreed-upon fixed code for the symbol. The code for the NYT node is obtained by traversing the Huffman tree from the root to the NYT node. This alerts the receiver to the fact that the symbol whose code follows does not as yet have a node in the Huffman tree. If a symbol to be encoded has a corresponding node in the tree, then the code for the symbol is generated by traversing the tree from the root to the external node corresponding to the symbol. To see how the coding operation functions, we use the same example that was used to demonstrate the update procedure. In the example, we used an alphabet consisting of 26 letters. In order to obtain our prearranged code, we have to find m and e such that $2^e+r=26$, where $0 \le r \le 2^e$. It is easy to see that the values of ***e***=4 and ***r***=10 satisfy this requirement. The first symbol encoded is the letter ***a***. As ***a*** is the first letter of the alphabet, ***k***=1. As 1 is less than 20, ***a*** is encoded as the 5-bit binary representation of *k-1*, or 0, which is 00000. The Huffman tree is then updated as shown in the figure. The NYT node gives birth to an external node corresponding to the element ***a*** and a new NYT node. As ***a*** has occurred once, the external node corresponding to ***a*** has a weight of one. The weight of the NYT node is zero. The internal node also has a weight of one, as its weight is the sum of the weights of its offspring. The next symbol is again ***a***. As we have an external node corresponding to symbol ***a***, we simply traverse the tree from the root node to the external node corresponding to *a* in order to find the codeword. This traversal consists of a single right branch. Therefore, the Huffman code for the symbol ***a*** is 1. After the code for ***a*** has been transmitted, the weight of the external node corresponding to ***a*** is incremented, as is the weight of its parents. The third symbol to be transmitted is ***r***. As this is the first appearance of this symbol, we send the code for the NYT node followed by the previously arranged binary representation for ***r***. If we traverse the tree from the root to the NYT node, we get a code of 0 for the NYT node. The letter ***r*** is the 18th letter of the alphabet; therefore, the binary representation of ***r*** is 10001. The code for the symbol ***r*** becomes 010001. The tree is again updated as shown in the figure, and the coding process continues with symbol ***d***. Using the same procedure for ***d***, the code for the NYT node, which is now 00, is sent, followed by the index for ***d***, resulting in the codeword 0000011. The next symbol ***v*** is the 22nd symbol in the alphabet. As this is greater than 20, we send the code for the NYT node followed by the 4-bit binary representation of ***22-10-1=11***. The code for the NYT node at this stage is 000, and the 4-bit binary representation of 11 is 1011; therefore, ***v*** is encoded as 0001011. The next symbol is ***a***, for which the code is 0, and the encoding proceeds.

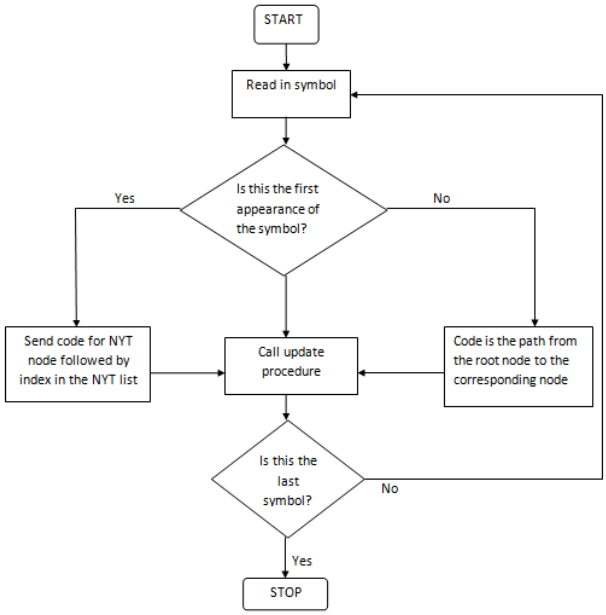


Figure 4: Encoding procedure of Adaptive Huffman Coding

E. Decoding procedure

The flowchart for the decoding procedure is shown in Figure 5. As we read the received binary string, we traverse the tree in a manner identical to that used in the encoding procedure. Once a leaf is encountered, the symbol corresponding to that leaf is decoded. If the leaf is the NYT node, then we check the next *e* bits to see if the resulting number is less than ***r.*** If it is less than ***r***, we read in another bit to complete the code for the symbol. The index for the symbol is obtained by adding one to the decimal number corresponding to the ***e-*** or ***e+1***-bit binary string. Once the symbol has been decoded, the tree is updated and the next received bit is used to start another traversal down the tree. To see how this procedure works, let us decode the binary string generated in the previous example. The binary string generated by the encoding procedure is

000001010001000001100010110

Initially, the decoder tree consists only of the NYT node. Therefore, the first symbol to be decoded must be obtained from the NYT list. We need in the first 4 bits, 0000, as the value of ***e*** is four. The 4 bits 0000 correspond to the decimal value of 0. As this is less than the value of ***r***, which is 10, we read in one more bit for the entire code of 00000. Adding one to the decimal value corresponding to this binary string, we get the index of the received symbol as 1. This is the index for ***a***; therefore, the first letter is decoded as ***a***. The tree is updated as shown in fig. The next bit in this string is 1. This traces a path from the root node to the external node corresponding to ***a***. We decode the symbol ***a*** and update the tree. In this case, the update consists only of incrementing the weight of the external node corresponding to ***a***. The next bit is a 0, which traces a path from the root to the NYT node. The next 4 bits, 1000, correspond to the decimal number 8, which is less than 10, so we read in more bit to get the 5-bit word 10001. The decimal equivalent of this 5-bit word plus one is 18, which is the index for ***r.*** We decode the symbol ***r*** and then update the tree. The next 2 bits, 00, again trace a path to the NYT node. We read the next 4 bits, 0001. Since this corresponds to the decimal number 1, which is less than 10, we read another bit to get the 5-bit word 00011. To get the index of the received symbol in the NYT list, we add one to the decimal value of this 5-bit word. The value of the index is 4, which corresponds to the symbol ***d***. continuing in this fashion, we decode the sequence *aardva*. This Adaptive Huffman coding method can also be used to compress any uncompressed file such as text files, sound clips etc. It can also be used in recursive mode.

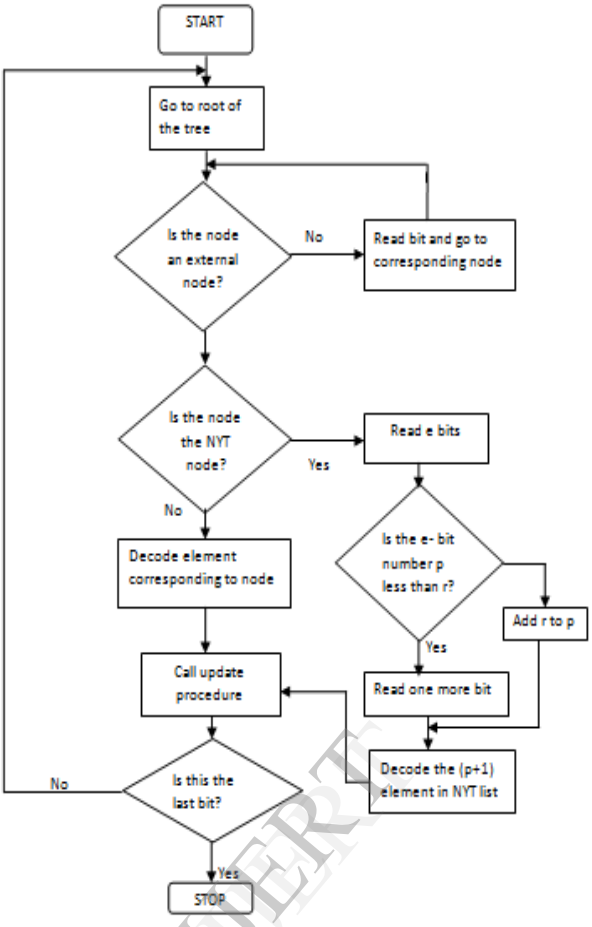


Figure 5: Decoding procedure of Adaptive Huffman Coding

III. IMPLEMENTATION:

IMPLEMENTATIONS ON TRANSMITTER SIDE

Image preparation: The given image is divided into 8x8 blocks. These 8x8 blocks are saved in a separate file.
Discrete Cosine Transform: For each 8x8 block, DCT is applied. This is implemented using the matrix method.
Quantization: The obtained DCT values are quantized using Q-50 scalar quantization standard.
Encoding: The encoding is done using Adaptive Huffman Coding, which is based on binary tree. At the start of encoding procedure, a binary tree structure is initialized. As we read the quantized input file, the tree structure is updated after reading each character. For a character already existing in the tree structure, it is coded as the binary traversal root to the node containing that character.

IMPLEMENTATIONS ON RECEIVER SIDE

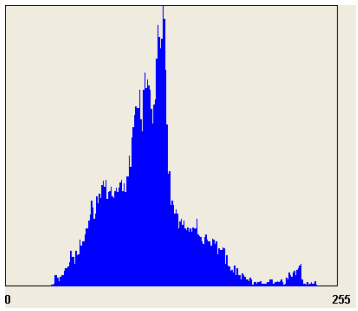
Decoding: As we read the received binary string, we traverse the tree in a manner identical to that in the encoding procedure. Once the character has been decoded, the tree is updated and the next received bit is used to start another traversal down the tree from the root node.
Dequantization: The dequantization is done using Q-50 standard table.
Inverse DCT: Inverse DCT is done using 8x8 matrix multiplication to get 8x8 blocks.
Reconstruction of image: The original image is reconstructed using the obtained 8x8 blocks.

IV. RESULTS

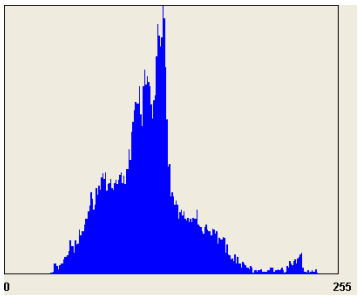
- **LOSSLESS PREDICTIVE MODE**

IMAGES ***HISTOGRAMS***

Original image



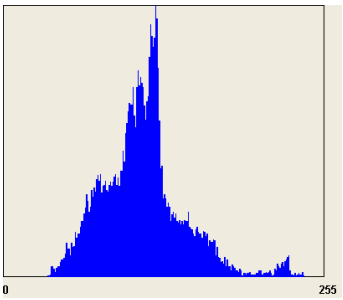
Reconstructed image



- **LOSSY PREDICTIVE MODES**

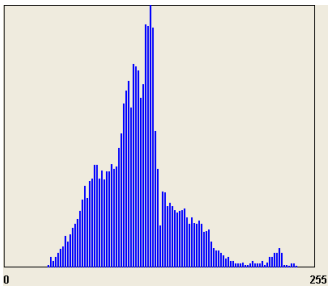
IMAGES **HISTOGRAMS**

Original image

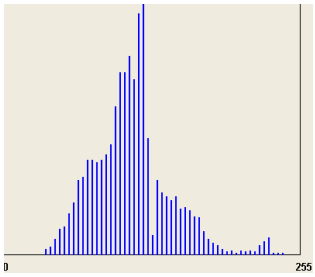


RECONSTRUCTED IMAGES

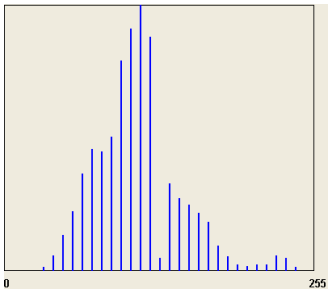
MODE 1:



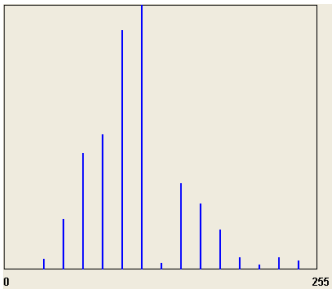
MODE 2:



MODE 3:



MODE 4:



V. CONCLUSION

This paper is based on implementing Adaptive Huffman coding for image compression. This method can be used both for lossy and lossless compression. It provides better compression ratios when compared to other lossless coding methods like LZW coding method, JPEG lossless compression. The performance of this method increases by using better predictive methods. The input file size is limited by the constraints of buffer size. The compression ratio, for a given quality level, obtained by implementing of lossy predictive modes is more compared to that obtained by other lossy methods. The image quality can be increased by the use of better transforms and suitable filters at the receiver side, for lossy methods of compression using transforms. The compression ratio can be increased by using higher level languages which have instructions for direct bit manipulation The lossless compression method can be extended to all type of files as long as the file size is less than the buffer size. It can be programmed in such a way so as to obtain the desired compression ratio by using recursive coding method.

VII. REFERENCES

[1] Vartika Singh **"**A Brief Introduction on Image Compression Techniques and Standards" *International Journal of Technology and Research Advances Volume of 2013 issue II.*

[2] Mamta Sharmap "Compression Using Huffman Coding*" IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.5, May 2010.*

[3] Sindhu M, Rajkamal R"Images and Its Compression Techniques A Review "*International Journal of Recent Trends in Engineering, Vol 2, No. 4, November 2009.*

[4] C.Saravanan,R. Ponalagusamy "Lossless Grey-scale Image Compression using Source Symbols Reduction and Huffman Coding".

[5] Mayur Nandihalli,Vishwanath Baligar "lossless gray scale images using dynamic array for predction and applied to higher bitplan" *International Journal of Scientific and Research Publications, Volume 3, Issue 1, January 2013.*

[6] Bhooshan, S., Sharma, S.: An efficient and selective image compression scheme using huffman and adaptive interpolation. In: Image and Vision Computing, New Zealand (2009)

[7] Gabriela Dudek , Przemyslaw Borys , Zbigniew J. Grzywna :- Lossy dictionary based image compression method. Image and Vision Computing 25(2007) 883-889.

[8] Somchart, C., Masahiro, I., Somchai, J.: A new unfield lossless/lossy image compression based on a new integer dct. IEICE Trans. Inf. Syst. E88-D, 1598–1606 (2005)