A fast and improved image compression technique using Huffman coding

Shrikant M. Patil 202SP024
Department of Electronics and Communication Engineering
National Institute of Technology Surathkal, Karnataka
Shrikant.patil2197@gmail.com
January 2021

***Abstract*: Compression of image is an important task for reducing the storage space required for an image and also the bit rate required for its transmission. There are many techniques available for this purpose which are broadly classified as lossy and lossless methods of compression. The purpose of this project is to analyze Huffman coding method for compression of images by calculating the parameters like compression ratio, bits per pixel, peak signal to noise ratio for the various input images with different resolutions. Project also deals with understanding the basic 'under the hood' process going on while Huffman coding algorithm is applied to an image. Conventional or traditional Huffman coding method scans the entire image whole at once. In this project, new method is introduced which splits the input image into sufficient number of blocks and applies Huffman coding algorithm to each block separately. These small compressed blocks of image are decoded separately and merged back to one image at the end. This compression method is proved to be advantageous as it provides better compression ratio, increased data security and low bit rate for image transmission compared to the traditional Huffman coding.**

1. Introduction:

Image compression using Huffman coding is the simplest method and easier to implement [1]. As it is lossless compression method the compression ratio achieved by this method is not encouraging. In spite of low compression ratio, Huffman coding is most popular for text compression and few specific applications in image compression. For example, compression of medical images can't tolerate any loss in image data at the cost of low compression ratio. Lossless methods can't achieve compression below the entropy which is dependent on source image pixel probabilities but it is completely lossless i.e., reconstructed image from compressed image is exactly same as original image. In spite of low compression ratio lossless compression is popular for compression of medical images and all kind of images where any loss can't be tolerated. Lossy methods on other hand use other redundancies in original image in addition to statistical redundancies as in case of lossless methods like Huffman coding.

This motivates to invent new methods in lossless compression to improve the compression ratio. This project demonstrates one such of method which aims at improving the compression ratio in lossless image compression using Huffman coding method.

Almost all lossy compression techniques like transform coding, wavelet coding generally deals with smaller blocks of image separately to improve the processing speed and compression results of algorithm. This project researches the application of this simple idea in case Huffman coding to verify if it results in any better compression.

Any compression technique can be described by the following block diagram.

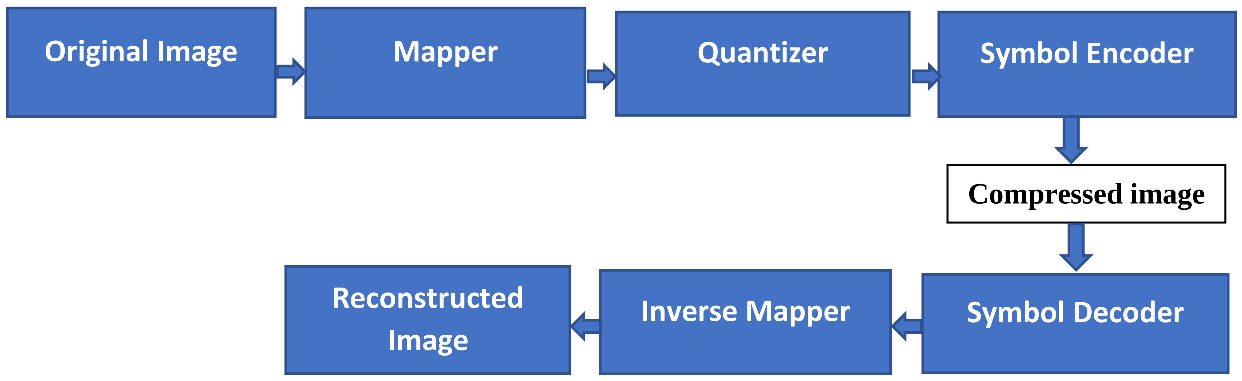


Fig 1. Systematic Block Diagram of Image Compression

As seen from Fig 1. any changes in block diagram can't improve lossless coding hence improvement at input side is only possible. So, research in modification at the input side is the only way to improve the lossless Huffman coding. As input image dismantling is the common process in almost every compression technique.

The proposed method is as shown in flowchart below (Fig 2.)

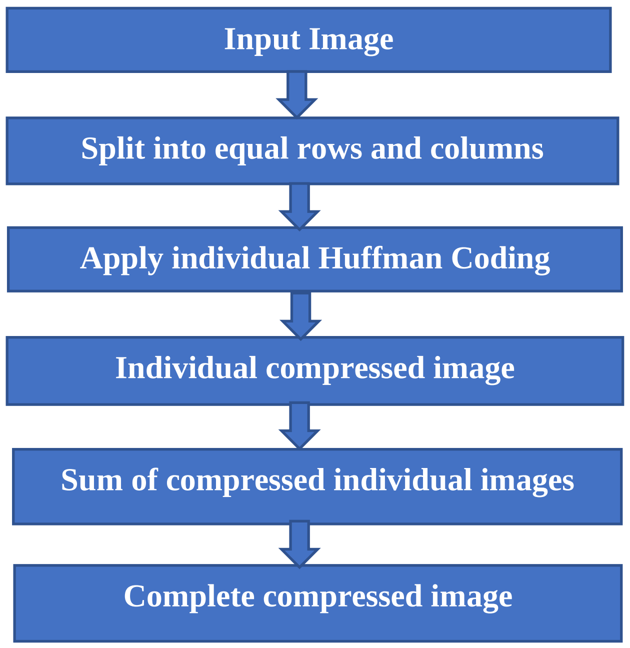

Input Image
Split into equal rows and columns
Apply individual Huffman Coding
Individual compressed image
Sum of compressed individual images
Complete compressed image

Fig 2. Flowchart of proposed method

2. Background:

A. Huffman coding algorithm:

Huffman coding comes under lossless compression technique with more prominent features in various applications such as medical survey and analysis, technical drawing [1] etc. It is step by step process and involves variable length codes for input characters which uses basically the statistical redundancy in input data. Original image can be restored with the help of digital image restoration [1]. The code length of character depends on how frequently it occurs in the input. Code word length is shorter for the more frequently used characters.

Huffman coding is best suited for text compression. Pixel values in images can be considered as different characters in the image. Usually, pixel values are of size 8 bit representing 256 different intensity levels. While for compression purpose this intensity levels can be quantized into 25 different pixel values without significant loss of information. This pixel values are better to round up to nearest possible 10th position value (e.g., 153 can be quantized to 150 while 157 to the nearest 160) such that no significant change in image appearance.

Following are the steps involved in Huffman coding [2]:

a) Build the list of all characters/symbols (in this case, the pixel values) in descending order of their probability.
b) Construct a tree with a symbol at every leaf from the bottom up.
c) Two symbols with smallest probability are selected, added to the top of the partial tree. Deleted from the list and added with an auxiliary symbol representing both of them.
d) When the list is reduced to just one symbol, the tree is complete.
e) The tree is traversed to determine the codes of symbols.

Huffman code should pass prefix code test (i.e., one code should not be the prefix of other code) so as to be uniquely decodable.

B. Important compression parameters:

a) Compression Ratio:

The ratio of size of uncompressed original image to the size of compressed image is called as compression ratio [1].

$$\text{Compression ratio} = \frac{size\ of\ original\ uncompressed\ image}{size\ of\ compressed\ image}$$

More the compression ratio better is the compression achieved.

b) Mean Square Error:

It is used to analyze the quality of image after compression. It is given as follows [1]:

$$\text{MSE} = \frac{1}{MN}\sum_{I=0}^{M-1}\sum_{J=0}^{N-1}[f(i,j) - f'(i,j)]\text{^}2$$

Where, M x N is image resolution

f (i, j) = original image pixel value

f’ (i, j) = compressed image pixel value

c) Peak signal to noise ratio:

Psnr is given by ratio of peak or maximum signal power to noise encountered in the signal [1].

$$\text{PSNR} = 10\log_{10}\frac{PEAK}{MSE\text{^}2}\ \text{dB}$$

d) Bits Per Pixel:

It is information (bits) stored per pixel of image. This is ratio of number of bits in the compressed image to total number of pixels in original image [1].

$$\text{BPP} = \frac{no.of\ bits\ in\ the\ compressed\ image}{total\ no.of\ pixel\ in\ the\ image}$$

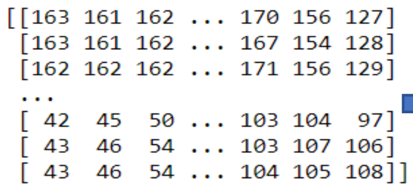
3. Implementation Details:

If Huffman coding is directly applied on the given image, as any image can have many numbers of distinct pixels the code for least probable pixel becomes of large length. This results in compressed size being of more size than the original image. Hence sufficient but not lossy quantization is used.

Following are the basic implementation steps used in the project. Same implementation is utilized for proposed method where input image is split into smaller parts and compressed separately and then reconstructed & merged back into complete image.

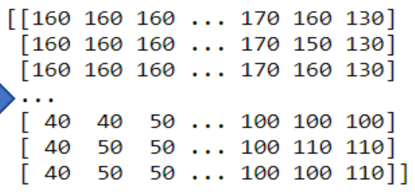
Step 1)

Input image pixel values are as shown in Fig 3 and Quantized image is shown in Fig 4. e.g., 153 can be quantized to 150 while 157 to the nearest 160.



```
[[163 161 162 ... 170 156 127]
 [163 161 162 ... 167 154 128]
 [162 162 162 ... 171 156 129]
 ...
 [ 42  45  50 ... 103 104  97]
 [ 43  46  54 ... 103 107 106]
 [ 43  46  54 ... 104 105 108]]
```

Fig 3.

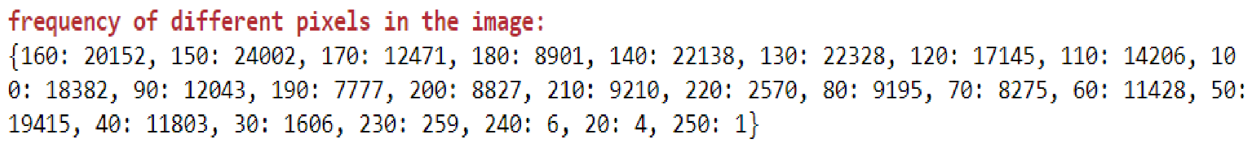


```
[[160 160 160 ... 170 160 130]
 [160 160 160 ... 170 150 130]
 [160 160 160 ... 170 160 130]
 ...
 [ 40  40  50 ... 100 100 100]
 [ 40  50  50 ... 100 110 110]
 [ 40  50  50 ... 100 100 110]]
```

Fig 4.

Step 2)

Frequency of occurrence of different pixel values in the input image is calculated as shown in Fig 5.

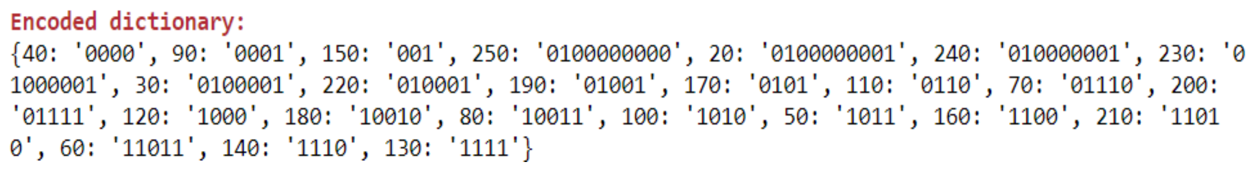


```
frequency of different pixels in the image:
{160: 20152, 150: 24002, 170: 12471, 180: 8901, 140: 22138, 130: 22328, 120: 17145, 110: 14206, 10
0: 18382, 90: 12043, 190: 7777, 200: 8827, 210: 9210, 220: 2570, 80: 9195, 70: 8275, 60: 11428, 50:
19415, 40: 11803, 30: 1606, 230: 259, 240: 6, 20: 4, 250: 1}
```

Fig 5. Frequency dictionary of image

Step 3)

With image as list of pixel values priority list is formed such that smallest frequency value is always popped first. Available Heapq python module is used to form the Huffman tree. Then Huffman tree is traversed from root node to character node to obtain the code of that particular character. All pixel values are then encoded as shown in Fig 6.



```
Encoded dictionary:
{40: '0000', 90: '0001', 150: '001', 250: '0100000000', 20: '0100000001', 240: '010000001', 230: '0
1000001', 30: '0100001', 220: '010001', 190: '01001', 170: '0101', 110: '0110', 70: '01110', 200:
'01111', 120: '1000', 180: '10010', 80: '10011', 100: '1010', 50: '1011', 160: '1100', 210: '1101
0', 60: '11011', 140: '1110', 130: '1111'}
```

Fig 6. Encoded dictionary of image

Note: It is seen from the Fig 5 and Fig 6 that most probable pixel value is '150' which is encoded with least length code as '001'.

This encoded dictionary is then applied to all pixels in the image and encoded complete image is then saved as .bin file which is final compressed image.

Step 4)

Compressed .bin file and encoded dictionary is the passed to the decoder function which in turn returns the reconstructed image file.

Original image and reconstructed or decoded images are shown in Fig 7.

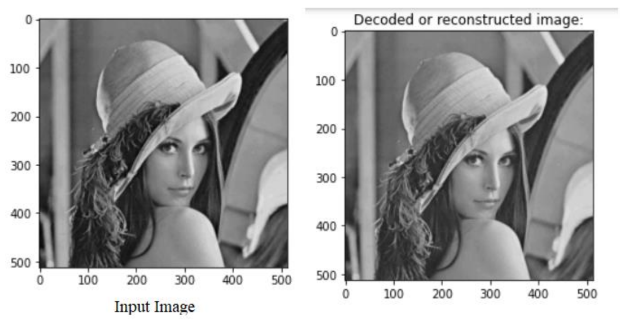


Fig 7.

4. Results:

Following are the results for different combinations of different parameters such as no. of blocks (in which image is split into), resolution, compression ratio, PSNR, bits per pixel etc.

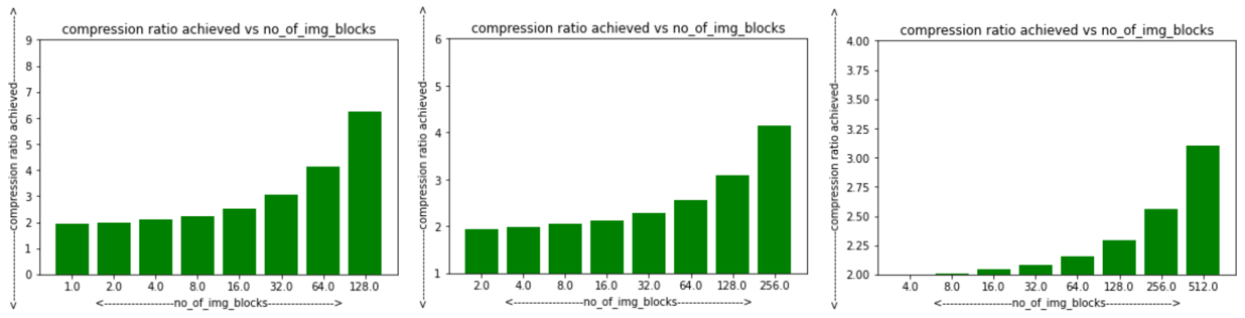


Fig 8. Graphs obtained for compression ratio achieved vs resolutions 256x256, 512x512, 1024x1024

[from left to right] for different no. of blocks chosen.

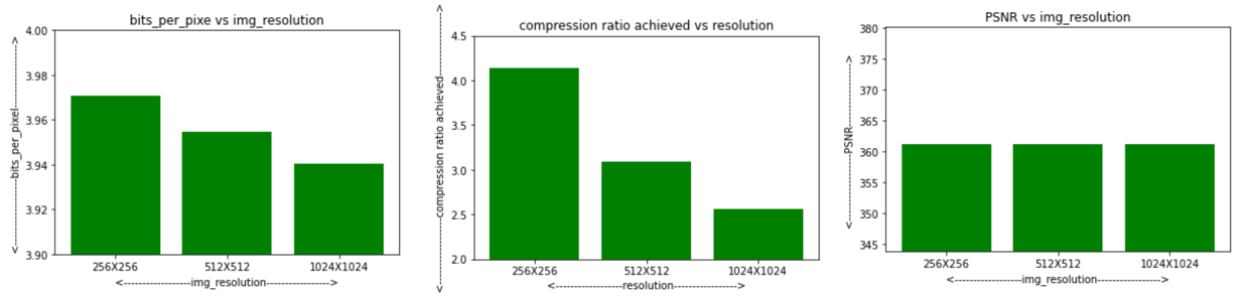


Fig 9. Graphs obtained for (bits per pixel vs resolutions 256x256, 512x512, 1024x1024),

(compression ratio vs resolution), (PSNR vs resolution) [from left to right]

5. Conclusion:

a) As seen from Fig 8. Exponential increase in compression ratio as image is split into more smaller size blocks and compression ratio achieved by this proposed method is better for low resolution images compared to higher resolution.

b) As seen from Fig 9. By this proposed method bits per pixel and compression ratio achieved are better for images with lesser resolution while no changes are observed in the PSNR for different resolution choice.

At the end it can be concluded that results achieved by this method are more promising for compression ratio.

6. References:

[1] *A Fast and Improved Image Compression Technique Using Huffman Coding* Rachit Patel, Virendra Kumar, Vaibhav Tyagi, Vishal Asthana, Department of ECE ABES-IT, Ghaziabad, India. presented at the IEEE Wisp NET 2016 conference.

[2] *Image processing and compression* course, Aparna P., Department of ECE NITK 2020-2021