BRACT’s Vishwakarma Institute of Information Technology

Image Compression Using Huffman Coding

Siddhant Shinde, Rhishikesh Sonawane, Siddhesh Songire, Shreyas Tornekar, Sakshi Wani

# Abstract

Images are very important documents nowadays; to work with them in some applications they need to be compressed, more or less depending on the purpose of the application. The need for an efficient technique for compression of Images ever increasing because the raw images need large amounts of disk space seems to be a big disadvantage during transmission & storage. There are various algorithms that performs this compression in different ways; some are lossless and keep the same information as the original image, some others loss information when compressing the image. Image can be represented with minimum number of bits by using image compression. When images are transferred over the network it requires space for storage and time to transmit image.

## Introduction

A good Image will be analysed in terms of better picture quality parameters along with memory space requirement which is very important for an image analysis. Image compression is one very important application of Digital Image Processing. The aim of image compression is to remove unwanted information from image so that it can be able to transmit or store data in an efficient form. Compression basically means removing unwanted information from image which only lead to the enhancement of memory space requirement without affecting quality of image.

The field of image compression continues to grow at a rapid pace. As we look to the future, the need to store and transmit images will only continue to increase faster than the available capability to process all the data. Image compression involves reducing the size of image data files, while retaining necessary information. Retaining necessary information depends upon the application. Image segmentation methods, which are primarily a data reduction process, can be used for compression. The reduced file created by the compression process is called the compressed file and is used to reconstruct the image, resulting in the decompressed image.

There are different techniques and they all have their own advantages and disadvantages. Huffman coding is a lossless data compression technique. Huffman coding is based on the frequency of occurrence of a data item i.e. pixel in images. The technique is to use a lower number of bits to encode the data in to binary codes that occurs more frequently. It is used in JPEG files.

## Literature Survey

## FUNDAMENTALS FOR COMPRESSION

Compression can be divided into two categories, as Lossy and Lossless compression. Lossy compression means that some data is lost when it is decompressed. Lossy compression bases on the assumption that the current data files save more information than human beings can "perceive”. Thus the irrelevant data can be removed. Lossless compression means that when the data is decompressed, the result is a bit-for-bit perfect match with the original one. The name lossless means "no data is lost", the data is only saved more efficiently in its compressed state, but nothing of it is removed.

## Huffman Coding

Huffman coding is classical data compression techniques invented by David Huffman. It is optimal prefix code generated from set of probabilities and has been used in various compression applications. These codes are of variable code length using integral number of bits. This idea causes a reduction in the average code length and thus overall size of compressed data is smaller than the original. Huffman code procedure is based on the two observations [3].

a) More frequently occurred symbols will have shorter code words than symbol that occur less frequently.

b) The two symbols that occur least frequently will have the same length.

The Huffman code is designed by merging the lowest probable symbols and this process is repeated until only two probabilities of two compound symbols are left and thus a code tree is generated and Huffman codes are obtained from labeling of the code tree.

Let us suppose, we need to store a string of length 1000 that comprises characters a, e, n, and z. To storing it as 1-byte characters will require 1000 byte (or 8000 bits) of space. If the symbols in the string are encoded as (00=a, 01=e, 10=n, 11=z), then the 1000 symbols can be stored in 2000 bits saving 6000 bits of memory.

The number of occurrence of a symbol in a string is called its frequency. When there is considerable difference in the frequencies of different symbols in a string, variable length codes can be assigned to the symbols based on their relative frequencies. The most common characters can be represented using shorter codes than are used for less common source symbols. More is the variation in the relative frequencies of symbols, it is more advantageous to use variable length codes for reducing the size of coded string.

Since the codes are of variable length, it is necessary that no code is a prefix of another so that the codes can be properly decode. Such codes are called prefix code (sometimes called "prefix-free codes", that is, the code representing some particular symbol is never a prefix of the code representing any other symbol). Huffman coding is so much widely used for creating prefix codes that the term "Huffman code" is sometimes used as a synonym for "prefix code" even when such a code is not produced by Huffman's algorithm.

Huffman was able to design the most efficient compression method of this type: no other mapping of individual source symbols to unique strings of bits(i.e. codes) will require lesser space for storing a piece of text when the actual symbol frequencies agree with those used to create the code.

## Image Compression

Huffman coding is one of the basic compression methods, that have proven useful in image and video compression standards. When applying Huffman encoding technique on an Image, the source symbols can be either pixel intensities of the Image, or the output of an intensity mapping function.

The first step of Huffman coding technique is to reduce the input image to a ordered histogram, where the probability of occurrence of a certain pixel intensity value is as

prob\_pixel = numpix/totalnum

where numpix is the number of occurrence of a pixel with a certain intensity value and totalnum is the total number of pixels in the input Image.

Now, there are 2 essential steps to build a Huffman Tree :

1. **Build a Huffman Tree :**
   1. Combine the two lowest probability leaf nodes into a new node.
   2. Replace the two leaf nodes by the new node and sort the nodes according to the new probability values.
   3. Continue the steps (a) and (b) until we get a single node with probability value 1.0. We will call this node as **root**
2. **Backtrack from the root, assigning ‘0’ or ‘1’ to each intermediate node, till we reach the leaf nodes**

In this example, we will assign ‘0’ to the left child node and ‘1’ to the right one.

Now, let’s look into the implementation :

**Step 1 :**  
**Read the Image into a 2D array**(**image**)

If the Image is in .bmp format, then the Image can be read into the 2D array

**Step 2**  
**Define a struct which will contain the pixel intensity values**(**pix**), their corresponding **probabilities**(**freq**), the **pointer** to the left(**\*left**) and right(**\*right**) **child nodes** and also the string array for the Huffman code word(**code**).

These structs is defined inside main(), so as to use the maximum length of code(maxcodelen) to declare the code array field of the struct pixfreq

**Step 3**  
**Define another Struct which will contain the pixel intensity values**(**pix**), their corresponding **probabilities**(**freq**) and an additional field, which will be used for storing the **position** of new generated **nodes**(**arrloc**).

**Step 4**  
**Declaring an array of structs**. Each element of the array corresponds to a node in the Huffman Tree.

**Step 5**  
**Initialize the two arrays *pix\_freq* and *huffcodes* with information of the leaf nodes.**

**Step 6**  
**Sorting the *huffcodes* array according to the *probability of occurrence* of the *pixel intensity values***

**Step 7**  
**Building the Huffman Tree**

We start by combining the two nodes with lowest probabilities of occurrence and then replacing the two nodes by the new node. This process continues until we have a root node. The first parent node formed will be stored at index nodes in the array pix\_freq and the subsequent parent nodes obtained will be stored at higher values of index.

**Step 8**  
**Backtrack from the root to the leaf nodes to assign code words**

**Final Step**  
**Encode the Image**

We also have to calculate the average number of bits required to represent each pixel.

## Conclusion

Image compression plays a vital role in saving memory storage space and saving time while transmitting images over the network. The compression technique increase storage capacity and transmission speed. Using compression coding techniques the share size in image secret sharing is reduced. By using Huffman coding the image is compressed by 40%. Huffman algorithm is comparatively easier because of its simpler mathematical calculation in order to find the various parameters.

The average number of bits required for every pixel in our given input image is 6.353724.

## Future Work

We all are in the revolution of computer technology but computers are yet not able to be as fast as the human brain in image processing. If we try to capture an image in a high-quality DSLR camera then it can be good but not as good as our vision. This new era of technology leading to more and more data usage too. So image compressing and processing are important in future too.

## Reference

1. <https://en.wikipedia.org/wiki/Huffman_coding>
2. <https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>
3. <https://www.programiz.com/dsa/huffman-coding>
4. <https://youtu.be/co4_ahEDCho>
5. <https://www.javatpoint.com/huffman-coding>
6. Sartaj Sahani: Data structures, Algorithms and Applications in C++
7. Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest, Clifford Stein : Introduction to Algorithms