ELE 725 Lab 2 Report – Lossless Coding

Name: Henry
FEAS
Toronto, Canada
ypzou@ryerson.ca

Abstract- In this lab, lossless compression though is explored though the usage of entropy-based, variable length coding. The coding technique used is Huffman encoding and though the lab it is shown that Huffman coding can be used to completely reconstruct the original coding sequence while compressing the sequence to a smaller size. A trade off of the Huffman encoding is the large the size of the sequence being compressed the larger the tree which takes a long time to traverse so for a real time application, the run time of encoding, decoding should be taken into account before implementation.

I. INTRODUCTION

The purpose of this lab is to investigate the properties of variable length coding and to construct a Huffman encoder, decoder. The key focuses are the construction of a Huffman encoder and the creation of the Huffman tree. The Huffman tree will be used to encode the input sequence based on entropy.

II. THEORY

Compression techniques is use for the purpose of reducing the amount of data used to convey the same amount of information. [1] There are two types of compression techniques: lossless and lossy; the lossless compression recovers the data exactly the same as the input while lossy recovers data approximately equal to input data with some data loss. [1] A method of measuring how good a compression is achieved is though the compression ratio which is given by:

$$\frac{size\ of\ original\ data}{size\ of\ compressed\ data} = Compression\ Ratio \quad (1)$$

One of the various lossless techniques is huff encoding in which the data in compressed based on entropy. Huffman code creates a prefix code in which the higher frequency symbols are less bits and is uniquely decodable.

The most important part to understand Huffman coding is the creation of the Huffman tree which can be obtained by first building a histogram of all the symbols used in the sequence. Though the histogram, a binary tree is generated based upon the lowest frequency being at the bottom. To generate the tree, all the symbols are created as a leaf of the tree. The leaves are weighted based upon the frequency of the symbol. The two leaves/nodes with the least frequency is attached together with the parent node equaling the sum of the two-child node's weight. This process is repeated until all node and leaves are in the tree.

III. METHODOLOGY

In part 1 of the lab, to create the entropy function used to calculate entropy, the built-in function unique and hist is used. The build in function unique sorts all the symbols based on acceding order and in the case of character, it is based on ascii. The build in function hist creates a histogram of frequency based upon the sequence and the values given which in this case is the unique symbols. By looping though the histogram of frequency, we can obtain the summation of all elements and assign the element with its frequency number. This allows us to divide all frequency by the total frequency which gives us probability and to send a sequence back to the user of the elements since they are reorder. The sequence "Huffman is the best compression algorithm" and the image lena.jpg was tested. To obtain image data the command imread is used. The commands rgb2ycbcr and rgb2gray transforms the rgb values into ycbcr and gray scale respectively. These data are put into the entropy function one channel at a time to entropy and compression.

The Huffman function was created by first creating the histogram by iterating though the sequence this time rather than unique as it makes the data much easier to manipulate than build in histogram function. Using this info, we create an array of size [2n-1][5] with N being the number of unique symbols there are.[2] The 1st column is for weight or frequency, 2nd column is for parent, 3rd column is for left child, 4th column is for right child, the 5th column is for the symbol value. The values of the array are preset to -1 and the first n rows are set to the symbol and there frequency respectively. In another array which hold the nodes/leaves not added to the tree yet, frequencies and row value are added to the array. The array is sorted using the sort rows which sorts the array in ascending order based upon value of the first row which is frequency. In a while loop of size n-1, the tree in created. In each iteration, the two nodes being added together is taken from the 1st row and 2nd row of the not yet added array. The new frequency of the parent node is calculated. The left, right child node is added to the parent node's row. The index of the parent node is added to the two children node. The new parent node is added the not yet added array with its weight and row value. Set the not yet added array equal to itself but down two rows and sort the array again. Increase the index and iterate though the entire process n-1 times which will build the huff man tree. Using the Huffman tree, we build a code book recursively. The base case of recursion is when the 2th column does not equal negative one meaning it is root node. By finding the matching symbol in 5th column of tree, we can find the starting leaves of the tree. We trace the sequence from the bottom up using the parent and matching the child node with the past row value to determine if a '0' or '1' is added to the sequence. As we are starting from the bottom up, the sequence is added in the following '0' + old sequence, this creates the prefix code as we did not start from the top. We do this for every symbol, to create a codebook and using the code book we encode the sequence based on what character is needed. This was applied to both sequence 'HUFFMAN IS THE BEST COMPRESSION ALGORITHM' and the image file.

For the decoding function, the code book from the encoding function was remade following the same steps. The sequence was iterated though one symbol at a time. The symbol is concatenate as symbol +past symbol and stored in past symbol. The past symbol is compared to the sequences in the code book to see if it matches any. If it matches, save the found variable as 1 and index of the found location. If found equals 1, reset found to 0 and reset old symbol to empty string and append the found symbol to the decoded message. This was applied to both sequence 'HUFFMAN IS THE BEST COMPRESSION ALGORITHM' and the image file. The return is of type cell array so using the cellfun which applies the entire cell with a specification function we can reobtain the sequence with its variable format; the string is of type char and image is of type uint8.

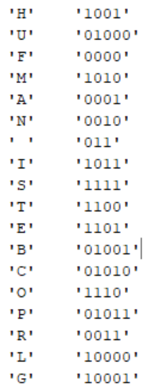
For encoding and decoding the image, the image data was spilt into 3 color channels. The encoding and decoding were applied to one row at a time. It is reconstructed though the cat building function which concatenates the three channels back together.

IV. RESULTS

The results of the entropy for 'HUFFMAN IS BEST COMPRESSION ALGOITHM' is 3.9883. The entropy value for grayscale image is 6.5265. The value of entropy for the image is 6.3139 for Y value, 4.6262 for Cb value, 4.4112 for the Cr value. These results indicate the number of bits needed to encode a media source in an ideal case. [1] The higher the number the more data is needed for the sequence.

The compression ratio for 'HUFFMAN IS BEST COMPRESSION ALGOITHM' is 2.0059. The compression ratio for the grayscale image is 1.2258 The value of compression ratio for the image is 1.2671 for Y value, 1.7293 for Cb value, 1.8136 for the Cr value

For the 'HUFFMAN IS BEST COMPRESSION ALGOITHM', the sequence obtained is 10010100000000000101000010010011101111111
0111100100111010110100111011111110001101
0101110101001011001111011111111110111110
0010011000110000100011110001110111100100
11010.

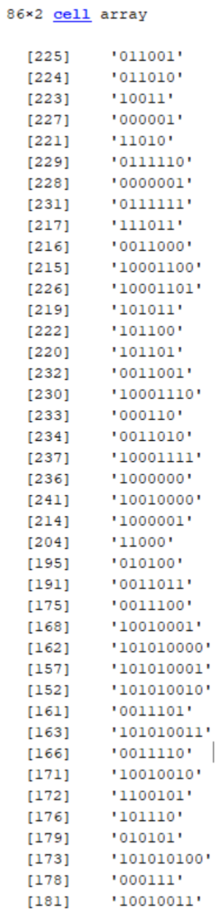


'H'	'1001'
'U'	'01000'
'F'	'0000'
'M'	'1010'
'A'	'0001'
'N'	'0010'
' '	'011'
'I'	'1011'
'S'	'1111'
'T'	'1100'
'E'	'1101'
'B'	'01001'
'C'	'01010'
'O'	'1110'
'P'	'01011'
'R'	'0011'
'L'	'10000'
'G'	'10001'

Figure 1: Code Book for Sequence

To find compression ratio, the character sequence size needs to be changed to bits. As each character is one byte or 8 bit which can be obtained by HTML.sizeof(). The character sequence is 36 so the size in bits is 288. The sequence is 209 so the compression ratio is 1.3846. The decode sequence is a cell with {HUFFMAN IS BEST COMPRESSION ALGOITHM}

The sample from the image is too long to include in this report but for a figure of how big the sequence is the sequence book taking only the first row and red column is:



```
  86×2 cell array

    [225]    '011001'
    [224]    '011010'
    [223]    '10011'
    [227]    '000001'
    [221]    '11010'
    [229]    '0111110'
    [228]    '0000001'
    [231]    '0111111'
    [217]    '111011'
    [216]    '0011000'
    [215]    '10001100'
    [226]    '10001101'
    [219]    '101011'
    [222]    '101100'
    [220]    '101101'
    [232]    '0011001'
    [230]    '10001110'
    [233]    '000110'
    [234]    '0011010'
    [237]    '10001111'
    [236]    '1000000'
    [241]    '10010000'
    [214]    '1000001'
    [204]    '11000'
    [195]    '010100'
    [191]    '0011011'
    [175]    '0011100'
    [168]    '10010001'
    [162]    '101010000'
    [157]    '101010001'
    [152]    '101010010'
    [161]    '0011101'
    [163]    '101010011'
    [166]    '0011110'
    [171]    '10010010'
    [172]    '1100101'
    [176]    '101110'
    [179]    '010101'
    [173]    '101010100'
    [178]    '000111'
    [181]    '10010011'
```

Figure 2: Code Book of first row of red channel

The encoding of the image is successful as the image is the same as the decoded image which is shown from figure 3 and 4.

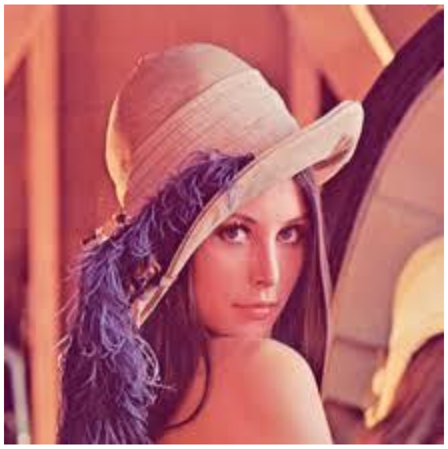


Figure 3: Image put into Huffman

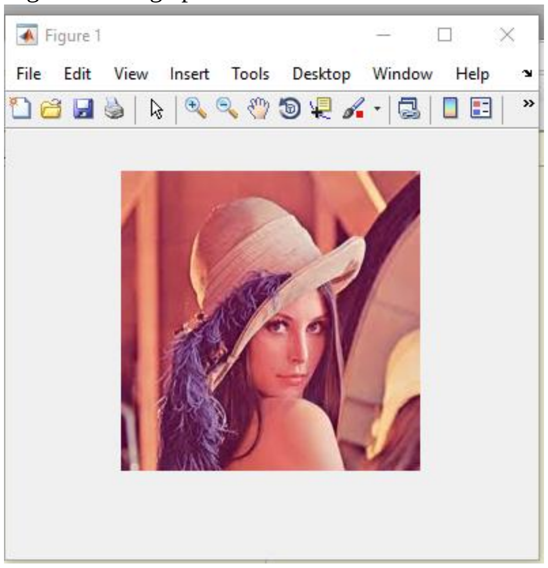


Figure 4: Reconstructed image after Huffman

V. DISCUSSION

We can observe though the compression ratio of HUFFMAN IS THE BEST COMPRESSION ALGOITHM that the value will never be able to reach the ideal value and is always lower than the ideal compression ratio. Also, the sheer size of lossless compression and the amount of time it takes to run is shown to be long for images. To compute the 220 pixels by 220 pixels by 3 color channels image. It was impossible for my computer to compute even as the size of the three would be the size of at least 100 000 which made encoding the image take forever and I did not have the patience to wait for it. The reason I spilt the channels and decided to encode one row at a time is because it speeds up the run time to tens of minutes. Even with only 220 elements, the size of the code book was 86 so with 220 by 220 the size would drastically increase. This of course increases the coded sequence size because it is no longer constructing a tree from the entire image but only a small section of the image. Also, it should be noted that system memory may become an issue with huge size recursive calls as they need to be stored in system memory until it is returned. To have an effective compression, run time must be weight against size of sequence.

VI. CONCLUSION

In conclusion, the lab successful demonstrated that lossless compression using Huffman coding will be able to recover all the input data while compressing the data sent. An important aspect to remember is the run time of this algorithm vs the size of the sequence being compressed. For real time application, it is not suggested to run lossless compression as the delay it causes would take too long.

REFERENCE

[1] Naimul Khan. (2020, Winter). Week4 – Compression Basics and Lossless Compression. Ryerson University.

[2] Faculty of Engineering and Architectural Science. (2020, Winter). ELE725 Lab 2 Manual: Lossless Coding. Ryerson University.