Lab 2: Lossless Coding

Rami Saad
Section 4
Toronto, Canada
rami.saad@ryerson.ca

***Abstract* - In this lab the effects of utilizing entropy-based, variable length coding techniques in relation to compressing data was observed. More specifically the effects of using Huffman encoding to reduce data size was measured. Data that had a lower entropy had a better results in terms of Huffman encoding as the higher probability symbols had a comparatively better outcome of receiving small prefix codes**

Huffman; Entropy; Encode; Decode; Compression; Tree

I. Introduction

The purpose of this lab was to understand the principles of Lossless coding. More specifically, the effects of how Huffman coding compares to both the Theoretical best entropy and regular uncompressed data. A string test sequence and an image files probability vectors and entropy were calculated and then used to create a Huffman tree, followed by an encoder and decoder, the results of which were noted.

The lab was primarily conducted using Matlab,. Matlab has a suite of built in function such as sort(), unique() and rgb2ycbcr() which make sorting and converting color spaces simple. Matlab also have unique data types such as cells which were utilized in this lab.

II. Theory

It is know that known that with VLC, variable-length coding, certain parts of call numbers appear more frequently than others, so it would be more practical to assign fewer bits as their code words [2], the more frequently appearing symbols are coded with fewer bits per symbol. Less bits are usually needed to represent the whole collection of data.

In this Lab the application of lossless compression techniques are observed. More specifically the Huffman coding in which compression is performed by an encoder and decompression is performed by a decoder. Huffman encoding code words have a unique property such that any code cannot be the prefix of another code, thus they are called prefix codes.

If the total number of bits required to represent the data before compression is B_0 and the total number of bits required to represent the data after compression is B_1, then we define the compression ratio as compression ratio

$$\text{compression ratio} = \frac{B_0}{B_1} \tag{1}$$

The entropy α of a data source with alphabet S = s1,s2,...,sn is defined as [2]:

$$\alpha = H(S) = \sum_{i=1}^{n} p_i \cdot \log_2 \frac{1}{p_i} \tag{2}$$

$$= -\sum_{i=1}^{n} p_i \cdot \log_2 p_i \tag{3}$$

The term $log_2 \frac{1}{p_i}$ indicates the amount of information [2] contained in si, which corresponds to the number of bits needed to encode. The less probability a symbol has the more self information is required, this is considered as the best case theoretical scenario.

The definition of entropy is aimed at identifying often occurring symbols in the data as good candidates for short prefix code words.

III. Methodology

Calculating Entropy of a Source

Subsection (1)

1. A function named myEntropy was created which takes in one input argument name input and returns 3 output arguments symbol, prob_vec, and entropy.
2. The function first checks if the input is of the data type "char", if it is the function converts the characters to their ASCII number equivalent and stores it back in the input vector.
3. The inbuilt Matlab function unique() is used to find all unique numbers in the input vector and returns these unique numbers in the symbol output vector
4. This symbol output vector variable is used in later sections.
5. Next, two for loops are utilized to find the frequency of occurrence each number in the input vector and stores that result in a temporary vector.
6. This vector then undergoes a scalar division with the total size of the input vector to find the weighted percentage probability vector
7. This is stored in the prob_vec output variable
8. Finally using equation (3) the entropy was calculated

Subsection (2)

1. The myEntropy function was used on the test sequence "HUFFMAN IS THE BEST COMPRESSION ALGORITHM"
2. The Entropy of a gray scaled image was calculated with the myEntropy function
3. A RGB image was convert to YCbCr color space using the builtin Matlab function rgb2ycbcr()
4. The entropy for each channel was then calculated using the myEntropy function

Subsection (4)

1. The compression ratio of the test sequence, grayscale image and the YCbCr color space image was calculated using equation (1)

Huffman Encoder

Subsection (1 and 2)

1. The createTree Matlab function was created to construct a Huffman tree and converts that tree to a list of code words
2. The createTree function has two input arguments symbol and prob_vec which are the unique symbol data and probability vector both of which were calculated in part 1
3. The createTree also has two output arguments code and tree which are the code words and Huffman tree respectively
4. The huffman tree was created using the 2d array outlined in the lab manual
5. First the size of the symbol vector is calculated and used with (2n-1) x 4, where n is the number of unique symbols, this array is initiated with -1 values
6. The prob_vec vector is used to fill in the weight column of the Huffman tree
7. Next using the Huffman tree algorithm in which the smallest two weighted values are chosen to be the two first leafs of the tree, then a new node is created and function repeats recursively, the tree is built
8. To get the code words the Huffman tree is then traversed for each leaf in the weight column until it reaches the root node
9. If the child is a left child then then a 1 value is recorded and if the child is a right child then a 0 is recorded
10. Right before this result is stored the values are flipped such that they appear in the correct order
11. Both the tree and code words are then returned from the function

Subsection (3)

1. The huffEncode function takes 4 input arguments sym, code, data, and type which are the symbol vector, code words, data to be encoded and the type of data respectively
2. The function has one output argument which is the encoded result
3. First the function checks if the type is a "char", if it is a char it converts the data to their ASCII equivalent
4. Using for loops the data input is checked against the symbol vector and then using the code words LUT the data is encoded
5. This result is stored in the encode output argument

Subsection (4)

1. The huffDecode function takes 6 input arguments, sym, code, encoded, h, w, type which refer to the symbol vector, code words, encoded data, height of data vector, width of data vector, and type of data vector respectively.
2. It has one output argument which is the decoded data
3. The function first initializes the output vector with zeros using the h and w input variables
4. The encoded data stream is check against the code words using the symbol vector if there are no matches the next encoded data element is included in the next check
5. Once the data is decoded if it is of type char it is convert to characters using the char() function and if it is of type image then it is converted to unsigned integers using the uint8() function

IV. Results

Calculating Entropy of a Source

```
%% Part I: Calculating Entropy of a Source
%%  Steps 1 & 2

% test sequence
test_seq = 'HUFFMAN IS THE BEST COMPRESSION ALGORITHM';
```

Figure 1: Test Sequence input Data

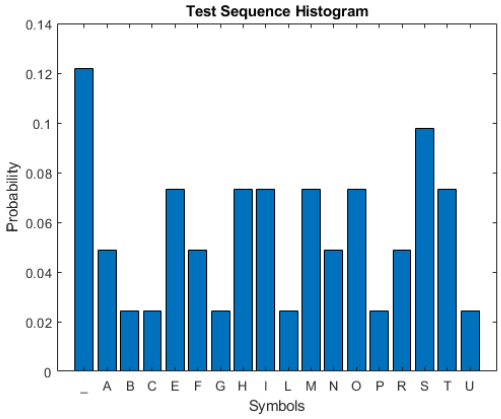

Figure 2: Test Sequence Histogram

Figure 3: Gray scale image input

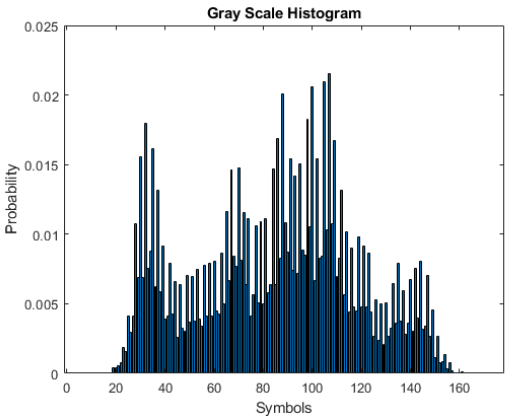

Figure 4: Gray scale image Histogram

Figure 5: Peppers input Image

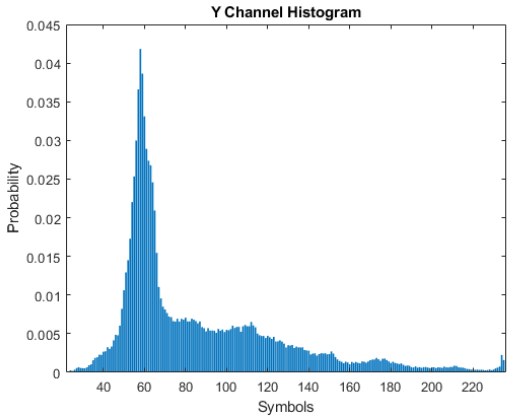

Figure 6: Y Channel Histogram

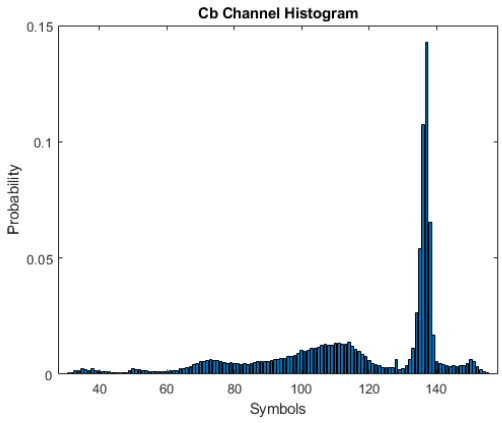

Figure 7: Cb Channel Histogram

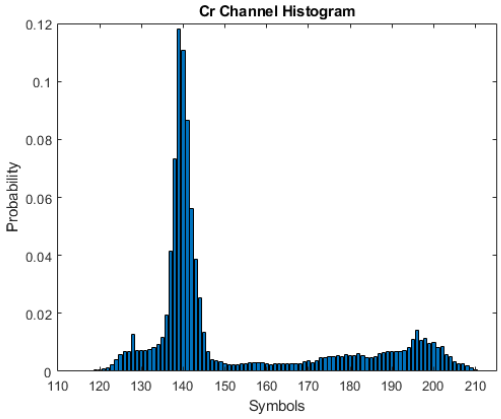

Figure 8: Cr Channel Histogram

Table 1: Entropy of Input Data

	Test Seq.	Gray scale	Y	Cb	Cr
Entropy	3.9883	6.8181	6.7729	5.7535	5.3164

Table 2: Compression Ratio: Input Data vs Encoded Data

	Test Seq.	Gray scale	Color
Compression Ratio Theory	2.0059	1.1733	1.3451
Compression Ratio Real	1.9878	1.1688	None

0.1220	30	-1	-1
0.0488	22	-1	-1
0.0244	19	-1	-1
0.0244	19	-1	-1
0.0732	25	-1	-1
0.0488	22	-1	-1
0.0244	20	-1	-1
0.0732	26	-1	-1
0.0732	26	-1	-1
0.0244	20	-1	-1
0.0732	27	-1	-1
0.0488	23	-1	-1
0.0732	27	-1	-1
0.0244	21	-1	-1
0.0488	23	-1	-1
0.0976	28	-1	-1
0.0732	28	-1	-1
0.0244	21	-1	-1
0.0488	24	3	4
0.0488	24	7	10
0.0488	25	14	18
0.0976	29	2	6
0.0976	29	12	15
0.0976	30	19	20
0.1220	31	21	5
0.1463	31	8	9
0.1463	32	11	13
0.1707	32	17	16
0.1951	33	22	23
0.2195	33	24	1
0.2683	34	25	26
0.3171	34	27	28
0.4146	35	29	30
0.5854	35	31	32
1	-1	33	34

Figure 9: Sample Test Sequence Tree

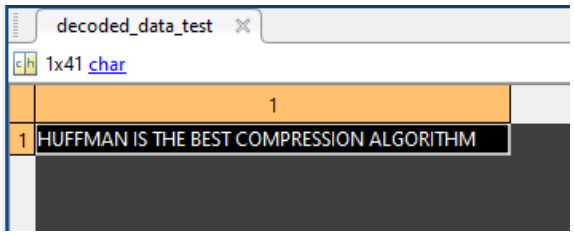

Figure 10: Decoded Test Sequence

Figure 11: Decoded Image

V. Discussion

In part one, the entropy were calculated and the result populates Table 1. First, the test sequence entropy is less than the 8 bits per character normally required. This outcome, can be attributed to the fact that there are very view symbols in the message and each symbol has a relatively high probability. This is more apparent when compared to the results of the gray scale and YCbCr channels since they both have a higher symbol counts they entropy is also higher. With the Cb and Cr channels because they have certain clusters of high probability, those high probabilities can be given shorter prefix codes when compared to the Y channel. This is the cause for both of those channels having a smaller entropy.

The Huffman tree algorithm works as follows. First the two smallest weighted symbols are created as the left and right child a parent node. Those child nodes are then removed from the list of potential children of the Huffman tree and the parent node is added to that list with the weight of its combined children. This general algorithm is repeated until there is only one node left with a weight of 1. This node is the parent node and the tree would be considered complete. a sample completed tree can be seen in figure 9.

With Table 2 it can be observed that the real compression ratio was very close to the theoretical compression ratio calculated using the entropy. The theoretical entropy was always better because the average code word length is slightly more for the Huffman encoding compared to just the theoretical calculation but this will be closer to the theoretical as the number of symbols in the data encode increases.

VI. Conclusion

In this lab, the benefits of lossless compression were observed. More specifically with Huffman coding, the greater the probability of symbols in data the less bits are needed for encoding. Furthermore as the data size increase the more closer fit to the theoretical entropy and compression ratio can be achieved.

References

[1] "Lab 2: Lossless Compression," D2L, 2020.

[2] P. Ze-Nian Li and Mark S. Drew Fundamentals of Multimedia. Boston, MA: Course Technology Cengage Learning, 2014.