Delft University of Technology

EE4740
Data Compression: Entropy and Sparsity Perspectives

Speech Signals Encoding by Huffman Code

Mini Project | Project 1

Author:
Yanqi Hong | 5884683
Y.Hong-8@student.tudelft.nl

April 12, 2024

1 Introduction

Huffman coding, introduced by David Huffman in 1952 [1], is a pioneering entropy encoding technique that proves particularly advantageous in the realm of audio compression. It optimizes compression by leveraging the probability distribution of quantized amplitude values, efficiently reducing data size while preserving intricate details. Through the assignment of variable-length codes to input characters based on their frequencies, wherein more common characters require fewer bits than their less common counterparts. Huffman coding stands as a widely adopted approach for lossless data compression, making it a versatile and robust solution in various applications.

This project investigates Huffman coding for speech compression. We'll build a Huffman table based on a speech signal, and then use that table to encode and decode other speech signals. The report will analyze how the size of the training speech signal set impacts the Huffman table size, computational complexity, and other relevant parameters.

2 Compression Process

2.1 Overview

To compress the speech signal based on Huffman code, we need six steps in this project which are normalization, quantization, Huffman code table calculation, encoding, decoding and result analysis. The provided speech data is divided into training and test sets. The normalization and quantization steps are performed identically on both sets to ensure consistent value ranges. The Huffman table is created using the training data's amplitude probabilities. This table is then used for encoding and decoding the test data. Finally, anti-normalization is applied to the decoded test data to recover the original signal's amplitude range. The specific details of how these steps interact can be found in Figure 1.

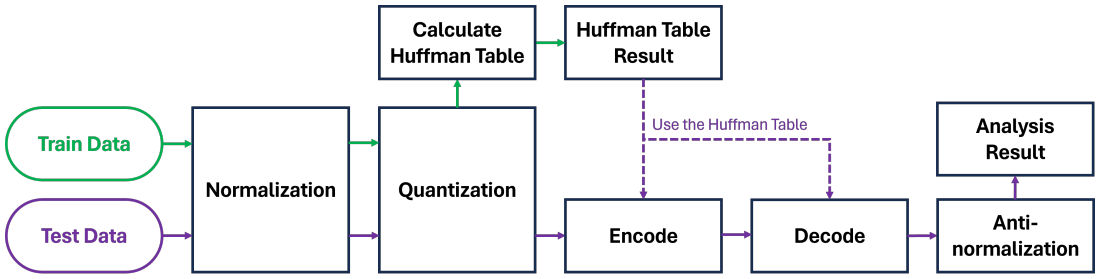


Figure 1: Flow chart of speech compression by Huffman code

2.2 Speech Signal Data

The provided speech comprises 16 audio segments, evenly distributed with 8 from male speakers and 8 from female speakers. Additionally, these 16 segments are derived from a pool of 4 males and 4 females, each contributing two distinct audio segments to create a diverse and comprehensive dataset. For each audio segments, the sample rate are same which are 8000. To replicate real-world scenarios where the data used to build Huffman code often differs from the audio to be compressed, the project involves partitioning the provided dataset into a training set and a test set. The training data and test data consist of distinct sets of audio segments, each assembled separately. Let the number of audio segments in the training set be denoted as M, and accordingly, the number of audio segments in the test set is $16 - M$. This division ensures an effective evaluation of the compression algorithm's performance on unseen data during testing.

2.3 Normalization and Quantization

Prior to constructing the Huffman table or encoding the signal, a crucial initial step involves pre-processing the audio data through normalization and quantization. This is essential because the Huffman table is constructed based on the probability distribution of amplitudes, and maintaining consistency in amplitude range between the training and testing data is vital. Normalization is employed to ensure that both the training and test datasets have the same amplitude range. Through normalization, the amplitude range is compressed into the interval $[-1, 1]$, and it is imperative to maintain uniformity in the normalization ratio for both the training data and the test data. This ensures that the amplitude characteristics are consistent, facilitating the subsequent steps of Huffman coding for effective compression.

Quantization is a critical component in audio compression based on Huffman code due to its role in representing continuous amplitude values with a discrete set of levels. In the context of audio compression, where storage or transmission bandwidth is limited, quantization helps reduce the amount of data needed to represent the original audio signal. The quantization formula, detailed in Equation 1, is expressed as follows: y denotes the value before quantization, and y_q denotes the quantized value. The parameter N signifies the quantization level [2]. Furthermore, to maintain uniform quantization levels, the step size for both the training set and the test set remains consistent.

$$y_q = round(\frac{y}{N}) \tag{1}$$

2.4 Huffman Code Table, Encoding and Decoding

Huffman coding is a method that generates prefix codes by analyzing the symbol frequencies within a dataset, specifically derived from the amplitude of the normalized and quantized train audio data. This intricate process begins by constructing a minimum heap of leaf nodes, progressively merging nodes with the lowest frequencies until a solitary root node persists. Subsequently, the edges in the Huffman tree are designated as either 0 or 1. The final binary code for each symbol is then determined by concatenating the bits traced along the path from the root to the respective leaf node.

The encoding and decoding processes involve the manipulation of amplitude data in normalized and quantized test audio. In the encoding phase, the algorithm identifies symbols in a Huffman table and replaces them with corresponding binary codes. If a symbol is not present in the Huffman table, the program skips the translation to binary code for that specific part.

On the other hand, during decoding, the objective is to retrieve the original amplitude from the given binary code. The program traverses the Huffman tree for each code, progressing through the tree until it reaches a leaf node. At this point, the decoding process concludes, and the symbol associated with the leaf code is recorded as the result. Iterating through this cycle, the final decoded amplitude is ultimately obtained.

3 Experiment

3.1 Experiment Introduction

To ascertain the optimal trade-off among Huffman table length, computational complexity, quantization level, training data size and bit loss, we employ a systematic grid search approach to unveil their interdependencies. The adjustable parameters in this investigation are the training data size and quantization level. Specifically, we vary the training data size across the range of $M = 1$ to $M = 15$, and the quantization level spans the set of $N = [128, 256, 512, 1024, 2048, 4096]$.

To mitigate the potential impact of individual audio segments being easier or more challenging to encode or decode, a random selection process is employed. Specifically, M segments are randomly chosen from the entire audio dataset, leaving the remaining $16-M$ audio segments designated as the test set. To enhance the precision of our experimental results, we conduct the experiment ten times, calculating the average value for each combination of the parameters M and N. This iterative approach ensures a more robust assessment of the chosen parameter values and their influence on the outcomes. Additionally, an observation reveals that post-decoding, the clarity of the decoded audio diminishes compared to the original test audio set. Consequently, this aspect will be explored further in the ensuing discussion.

3.2 Result Analysis

3.2.1 Huffman Table size

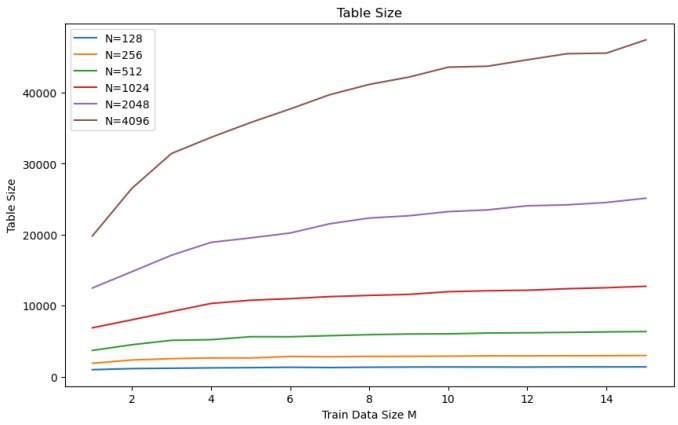


Figure 2: Huffman code table size of different quantization level and train data size

The size of the Huffman table signifies the how many binary codes listed in the table, and this size is linked to both the quantization level (N) and the size of the training data (M). As depicted in Figure 2, elevating quantization levels results in the emergence of larger table sizes. This trend is a consequence of the larger quantization level leading to a reduction in the step size or granularity of the quantized values. This reduction in step size enhances the precision of the quantization process, but it concurrently expands the range of possible values, necessitating a correspondingly larger table to accommodate and store these refined quantized values.

When calculating the Huffman code table, it is essential to consider the possibility that the training data may not cover the full quantization range. As illustrated in Figure 2, the table size increases with the augmentation of the training data size. A larger training dataset can cover a broader range of quantization values, contributing to the growth of the table length. Notably, when the quantization level N is higher, the phenomenon of larger training set sizes leading to increased Huffman table sizes becomes more pronounced. This is primarily attributed to the fact that higher quantization levels necessitate a larger volume of data to calculate the probabilities associated with each quantization value. Consequently, this phenomenon becomes more evident and pronounced as the quantization level increases.

3.2.2 Entropy and Average Code Length

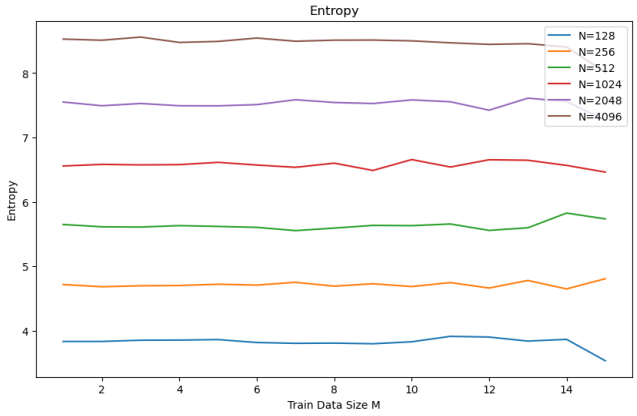


Figure 3: Entropy

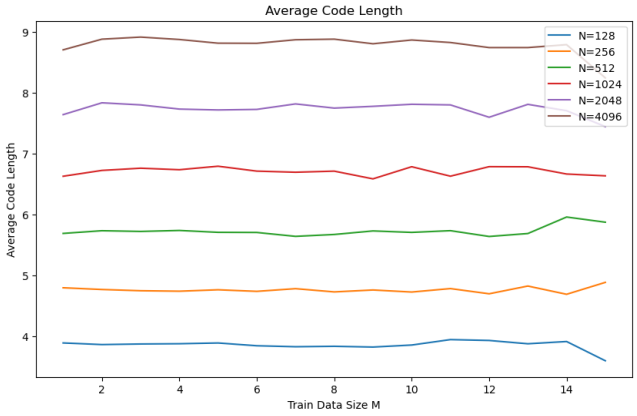


Figure 4: Average code length

Evaluating the results in a Huffman code table is facilitated by examining both entropy and average code length. The respective formulas for calculating entropy and average code length are provided in Equation 2 and Equation 3. In these equations, $P(x_i)$ denotes the probability of a symbol in the Huffman table, while $l(x_i)$ in Equation 3 represents the length of the binary code for the corresponding symbol. The plots in Figure 3 and Figure 4 illustrate that an increase in quantization level corresponds to higher entropy and longer average code lengths. Interestingly, these values appear unaffected by variations in the size of the training data. A higher quantization level, which means finer quantization with more intervals, leads to a broader spectrum of values that are less likely to occur. Consequently, this results in longer Huffman codes on average, as there are more unique symbols to encode. Simultaneously, higher quantization levels increase entropy. Entropy is a measure of unpredictability or information content. While finer quantization diversifies the possible outcomes, distributing the probabilities more uniformly across a larger set of outcomes. This uniform distribution enhances the uncertainty or randomness in the data, which is quantified as higher entropy. Therefore, while a higher quantization level can lead to longer Huffman codes, it also leads to higher entropy due to the increased number of potential outcomes and the more even spread of probabilities.

The figure depicted in Figure 5 illustrates the efficiency of the encoding process, represented by the values of $\frac{H(x)}{L(x)}$. A clear trend is evident, as all values are below one. This observation reinforces the claim that the average code length consistently equals or exceeds the entropy. Additionally, the plot highlights that lower quantization levels result in average code lengths closer to the entropy, indicating that lower quantization rates yield better efficiency in Huffman coding.

$$H(X) = -\sum_{i=1}^{n} P(x_i) \cdot \log_2(P(x_i)) \tag{2}$$

$$L(X) = \sum_{i=1}^{n} P(x_i) \cdot l(x_i) \tag{3}$$

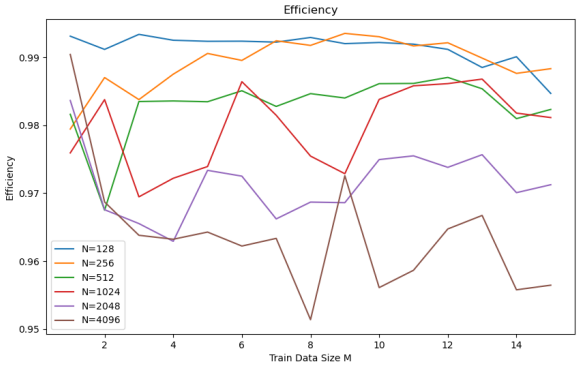


Figure 5: Efficiency of the Huffman code

3.2.3 Computational Complexity

In this experiment, we assess computational complexity by examining encoding and decoding times. As depicted in Figure 6 and Figure 7, these figures illustrate the time required for encoding and decoding under various scenarios. A noticeable observation is the consistent trend and pattern shared between encoding and decoding processes. Specifically, the computational time decreases as the size of the training dataset increases. This decrease in time can be attributed to the fact that a larger training dataset

results in a relatively smaller test set. Consequently, more test data must be encoded and decoded, requiring additional processing time. Additionally, an increase in the quantization level correlates with a rise in encoding or decoding time. This phenomenon occurs because a higher quantization level leads to a larger Huffman table (detailed in Chapter 3.2.1), necessitating the processing of more binary codes within the test dataset. Furthermore, the decoding time is significantly longer than the encoding time, a difference attributed to the encoding process involving the substitution of symbols with binary codes using a Huffman table. In contrast, the decoding process requires evaluating each binary code to determine if it signifies the conclusion of a symbol.

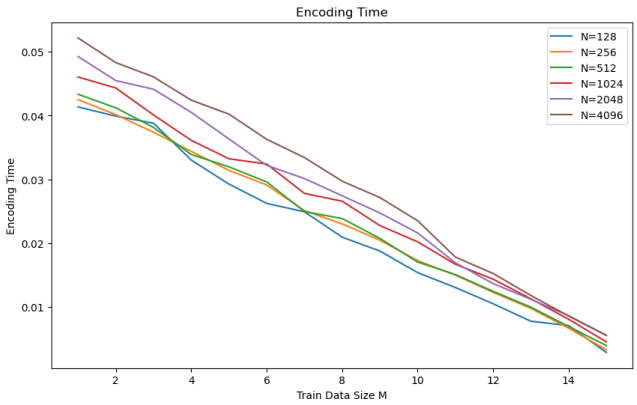


Figure 6: Encoding time

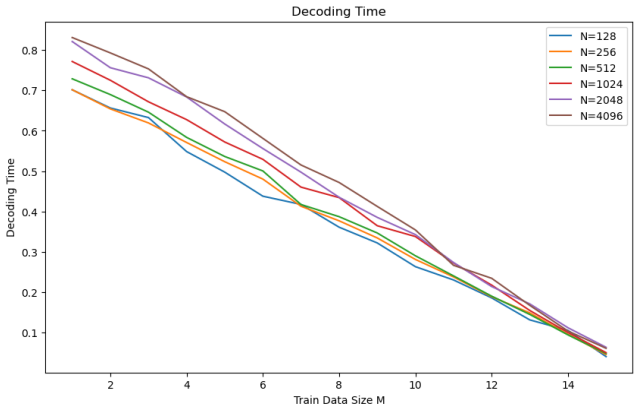


Figure 7: Decoding time

3.2.4 Space Saving Rate

The space saving rate serves as an intuitive metric for how many storage space is saved which shows the efficacy of a data compression algorithm . It is computed as $space\ saving\ rate = 1 - \frac{length(code)}{length(original\ data)}$, with the original data type defined as int16. As shown in Figure 8, an increase in quantization level correlates with a lower compression rate, whereas the size of the training data does not significantly affect the compression ratio. This phenomenon may be explained that higher quantization levels enhance data discretization, dividing the data range into more subintervals, each represented by a unique code. While the frequency of occurrence for each symbol decreases and the resulting Huffman code becomes shorter for every symbol, higher quantization levels necessitate the representation of more symbols. Consequently, the table size for higher quantization levels increases, leading to a lower space saving rate for higher quantization level.

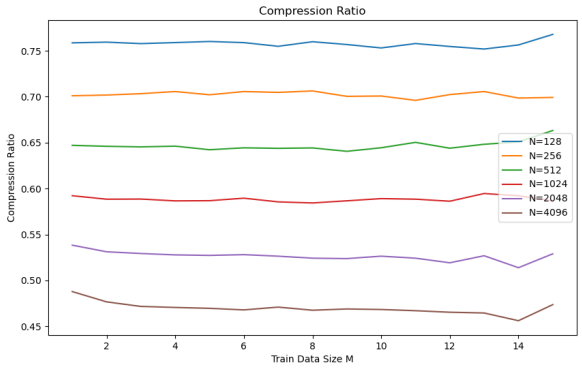


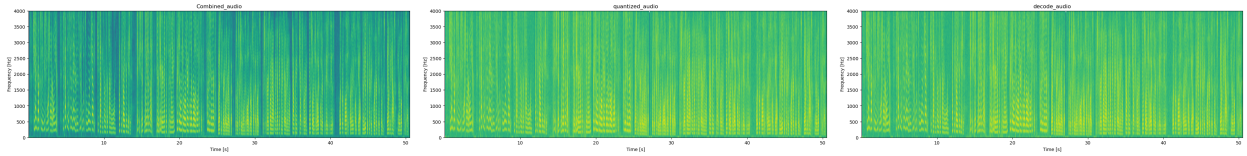
Figure 8: Space saving rate

3.2.5 Signal-to-Noise Ratio (SNR)

Based on the spectrogram depicted in Figure 9, it is evident that the processed audio lacks the clarity present in the original audio. A visual inspection reveals minimal disparity in the spectrogram between the quantized and decoded audio compared to the original audio. To quantify these observations, we employ Signal-to-Noise Ratio (SNR) as a quantitative metric to assess the audio clarity. The SNR is computed using the formula $SNR = 10 \log\left(\frac{\text{var}(Audio)}{\text{var}(Noise)}\right)$, with the noise variance defined as the variance of the first 1000 frames of the audio in this project. Upon calculation, we observe SNR values of 49.38, 41.59, and 41.59 for the original audio, quantized audio, and decoded audio, respectively. This discrepancy in SNR values indicates a decrease caused by the quantization process. It's noteworthy that Huffman coding being a form of lossless compression, does not exert an influence on the SNR.

In the depicted Figure 10, a discernible trend emerges, revealing that lower quantization levels result in a more substantial difference between the SNR of the original audio and that of the decoded audio. This phenomenon can be attributed to the precision limitations imposed by quantization, where precise amplitude values are confined to limited quantization intervals, leading to information loss. Remarkably, as the quantization level increases, the reduction in SNR difference is not as pronounced. For instance, a quantization level of 4096 demonstrates a doubling from the quantization level of 2048, yet the SNR differences

remain the same. This shows that when the quantization level increase beyond a certain point, it may fail to yield a proportional improvement in SNR difference.



(a) Original audio (b) Quantized audio (N=512) (c) Decoded audio

Figure 9: Spectrogram of audio

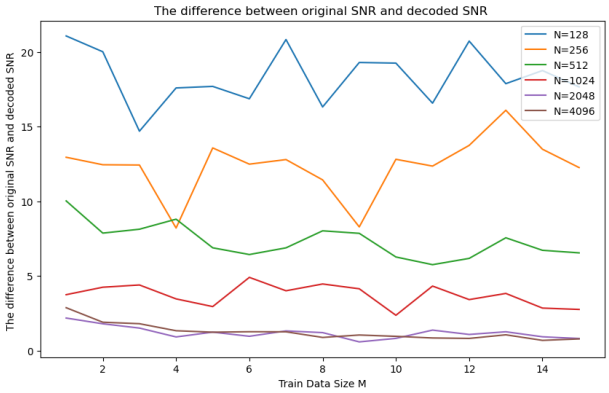


Figure 10: The difference between original SNR and decoded SNR

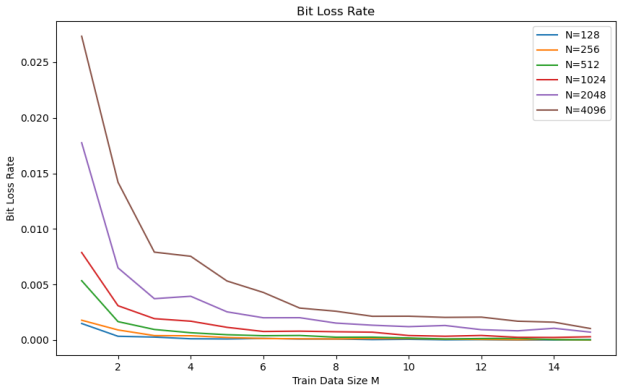


Figure 11: Bit loss rate

3.3 Bit Loss Rate

In Huffman coding-based audio compression, the loss of bits can significantly degrade the quality of the decompressed audio, occurring during both the encoding and decoding phases. During encoding, if the Huffman table fails to encompass all symbols in the test set, the encoding process may skip nonexistent symbols in the table, resulting in bit loss.

In Figure 11, the figure illustrates the rate at which bits are lost in the decoded audio compared to the original audio. Since the experiment is simulated in Python, there is no loss between the encoding and decoding data. From this figure, it is evident that higher quantization levels induce much higher bit loss rates compared to lower quantization levels, particularly evident in smaller training data sizes. This disparity arises because higher quantization levels require a larger number of symbols to represent the data, while smaller training sets may not provide sufficient probabilities for the Huffman table. However, examining the actual loss rate, it is found that less than 1% of bits are missing when using sample data larger than three segments. Since the missing bit values are not absent in the training set and the rate is low, they likely correspond to low-probability values. Although these bits are missing, they may not significantly disrupt human understanding of the speech.

On the other hand, if bits are lost between the encoding and decoding period due to communication error, the Huffman decoding process may fail to correctly reconstruct the original audio, leading to audible artifacts or even complete loss of sound. To tackle this problem, error detection and correction techniques can be employed. For example, parity bits can be added to the compressed data to detect errors. If an error is detected, the system can request retransmission of the data. For error correction, techniques like Reed-Solomon codes can be used [3]. These codes add redundancy to the data, allowing the system to correct errors without needing retransmission. Additionally, interleaving can be used to spread the error across the audio file, which can make the error less noticeable to the human ear. However, these techniques come at the cost of increased data size and computational complexity. It's a trade-off between robustness and efficiency.

4 Conclusion

In this project, we investigated the effectiveness of Huffman coding for speech compression. We successfully compress the amplitude value of speech signal by Huffman code. The experiment systematically explored the impact of training data size and quantization level on various variable crucial for speech compression. These variable includes the size of the Huffman table, the entropy and average code length, the computational complexity of encoding and decoding, the compression rate, SNR and bit loss rate.

The analysis unveils several trade-offs associated with Huffman coding for speech compression. Lowering the quantization level typically improves compression ratios by generating smaller, more efficient Huffman code tables, thereby reducing storage requirements, enhancing processing speed, and minimizing bit loss rates. However, this comes at the cost of decreased SNR. Furthermore, employing a larger training dataset inevitably reduces bit loss rates but also enlarges the Huffman code table. This underscores the delicate balance between achieving compression performance and preserving audio quality.

In summary, selecting the lowest values for the quantization level and the highest size of the training dataset is not an optimal strategy. Instead, a judicious choice of moderate values is essential to strike a balance between performance and efficiency.

References

[1] Huffman, D.A. (1952) A Method for the Construction of Minimum Redundancy Codes. Proceedings of the IRE, 40, 1098-1101. http://dx.doi.org/10.1109/JRPROC.1952.273898

[2] Schuller, G. (2020). Filter Banks and Audio Coding: Compressing Audio Signals Using Python. Springer Nature.

[3] Reed, I. S., & Solomon, G. (1960). Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics, 8(2), 300-304.