Master MsCV 1/MAIA Image Processing Project

Basic Compression Methods

Authors:
Brianna Burton
Doiriel Vanegas

Supervisor:
Dr. Christophe Stoltz

December 15, 2017

1 Introduction

Data compression is key to modern communications: it is prevalent to every device we could think of [1]. To most people, it is vitally important to send and store as little information as possible to send it quicker and save space, without losing too much or any information. Data compression allow an input stream to be converted into an output stream with a smaller size. There are many specialized algorithms for certain applications. For example, arithmetic coding allows a sequence of data to be encoded in one code. The method begins with an interval, and shortens the interval based on the probability of each symbol in the sequence. High probability symbols shorten the interval less, and correspond to less bits in the output stream. Arithmetic coding has slight advantages to Huffman coding, as Huffman coding assigns integer codes to individual symbols. Huffman coding serves as the basis for several popular programs run on various platforms. It is a lossless data compression method which takes a set of symbols and their probabilities and returns a binary code (set of codewords) with minimum expected codeword length. Lastly, JPEG encoding is especially important for those who want to share and save images. It is a lossy coding technique, but the images do not appear to be compressed. The JPEG encoder is able to split the image into many uniformly sized blocks, transformed to the frequency domain, quantized, and encoded. In this report, we explore these three coding types, as algorithms in Matlab.

2 Arithmetic Coding

2.1 Arithmetic Encoding

Unlike other coding methods, arithmetic coding does not assign integer codes to individual symbols, but a long code for the entire stream [1]. The arithmetic coding function accepts a list of symbols, their respective probabilities, and a sequence to code. First, the probabilities (and respective symbols) are sorted from highest to lowest. Then, a high and low range are determined for each symbol, producing an interval. The sequence is then coded using the following three formulas:

$$range = oldhigh - oldlow \tag{1}$$

$$newhigh = oldlow + range * highrange(X) \tag{2}$$

$$newlow = oldlow + range * lowrange(X) \tag{3}$$

where oldlow is initialized to 0, oldhigh to 1, and X is the symbol in the sequence to code. The newlow is saved in a binary stream to output. Next, the old high and low are replaced by the new high and low and the loop continues until the end of the sequence to code is reached. All values in this binary stream are values between [0,1). Only the last entry in the binary stream is required to decode the sequence, so the 0. is discarded and only the decimal part of the code is encoded in binary format to complete the encoding.

2.2 Arithmetic Decoding

Decoding begins by inputting the a list of symbols, their respective probabilities, and the last entry in the binary stream. First the binary formatted number is decoded into a decimal. Then, the probabilities (and respective symbols) are sorted from highest to lowest, and a high and low range are determined for each symbol, producing an interval like in the encoding process. Then the decoded sequence is calculated as follows: the decimal is checked to determine which interval it falls into and the corresponding symbol is the first entry in the decoded sequence. Then, the decimal is converted to a new decimal by:

$$newdecimal = \frac{olddecimal - lowrange(X)}{range(X)} \tag{4}$$

This new decimal is evaluated to determine which interval it belongs to, the corresponding symbol is assigned to the decoded sequence, and the calculations are performed again, until newdecimal equals one, indicating the entire stream has been decoded. For decimals on the border between two intervals, the lowrange is included in each interval.

3 Huffman Coding

The algorithm starts by building a list of all the alphabet symbols in descending order of their probabilities. It then constructs a tree, with a symbol at every leaf, from the bottom up. This is done in steps, where at each step the two symbols with smallest probabilities are selected, added to the top of the partial tree, deleted from the list, and replaced with an auxiliary symbol representing the two original symbols. When the list is reduced to just one auxiliary symbol (representing the entire alphabet), the tree is complete. The tree is then traversed to determine the codes of the symbols [1].

For implementing Huffman Coding in Matlab, two functions were written:

- `huffman_encoding.m` takes a probability vector and its corresponding list of symbols (alphabet), builds the Huffman tree and generates the Huffman codebook containing the binary codes for each symbol (codewords).
- `huffman_image.m` takes a gray image and returns the codebook contaning codewords for each gray value present on the image. The function first calculates the probability vector for the image by computing its histogram and in a second step calls `huffman_encoding.m` to generate the final codebook.

4 Simplified JPEG Encoding

JPEG is the most common imaging format in the world. Matlab has many built-in functions in the image processing toolbox to convert images to JPEG, but in this section, we describe a method that takes the raw data and encodes the JPEG, using our own Huffman coding function.

4.1 8x8 Block Extractor

An image is accepted into the function as an array. The image is first divided into non-overlapping blocks of 8 x 8 pixels. To ensure that the data can be divided, a condition resizes the image into a width and height of integer multiples of 8. Next, the gray levels of the image are shifted from unsigned to signed by subtracting 2^{m-1} from every pixel in the image. Shifting reduces the dynamic range for the discrete cosine transform. To facilitate this, the number of gray levels is found, and then m is determined by

$$m = log_2(numbergreylevels) \tag{5}$$

Next, the 2D discrete cosine transform (DCT) is computed on each 8x8 block as

$$T = HFH^T \tag{6}$$

where F is an 8 x 8 block of the image, and H is a transformation matrix found through the function dctmtx(8), and H^T is its transpose. If a matrix has m x n blocks of size 8 x 8, then the DCT will have the same dimensions.

Each 8 x 8 block is then normalized and quantized using a predefined transformation normalization array Z, as:

$$quantization = round(T/Z) \tag{7}$$

and the quantization has the same dimensions as the original image.

4.2 Zig-Zag Encoding

Zig-zag (entropy) encoding rearranges the pixels in each block so that the spatial frequency is increased, and often, long runs of zeros are found at the end, enabling the stream to be truncated. Each 8 x 8 block is divided into a one dimensional array of length 64, and the pixels are reordered using a zig-zag reordering pattern. The quantization is now a stream of 64 x total number of blocks.

4.3 Conversion of quantized vectors into the JPEG defined bitstream

Each block in the quantization stream is then truncated after the last non-zero element to reduce the size of the data. An end of block symbol defined as the maximum value of the initial image plus one is defined to ensure that there are no repetitions of this symbol within the JPEG code. Each encoded and truncated block is appended by the end of block symbol and the next block added. This produces a one dimensional array of the shortened quantized streams, which is then encoded using the Huffman code as described earlier.

References

[1] Salomon, D. *Data Compression: The Complete Reference* (Springer, London, 2007), fourth edn.