**Image Compression**

CPE 462

Intro to Image Processing and Coding

Final Report

Troy D’Angelo

Jacob Heutlinger

Edward Runyon

I pledge my Honor that I have abided by the Stevens Honor System.

Table of Contents

[Project Accomplishments 1](#_Toc419316954)

[Program Explanation 1](#_Toc419316955)

[Image Encoder 1](#_Toc419316956)

[columns.m / rows.m 2](#_Toc419316957)

[quant.m 2](#_Toc419316958)

[bitStream.m 3](#_Toc419316959)

[Image Decoder 4](#_Toc419316960)

[columnsInv.m / rowsInv.m 4](#_Toc419316961)

[quantInv.m 5](#_Toc419316962)

[psnr.m 5](#_Toc419316963)

[Main Code 6](#_Toc419316964)

[Group Member Contributions 7](#_Toc419316965)

# Project Accomplishments

Our group accomplished all the necessary task detailed in the project outline given to us for image compression and decompression. We were able to implement a sub-band (or wavelet) image encoder and decoder using MATLAB. This involved performing sub-band decomposition, scalar quantization, and entropy encoding for an input image. The inverses of these functions were also performed in order to reconstruct the original image. The program also allows the user to input their own parameter for the quantization step size during the scalar quantization portion of the code. The original and reconstructed images are also compared in the end to determine the Peak-Signal-To-Noise ratio, which represents an accurate evaluation of the performance of the encoder and decoder.

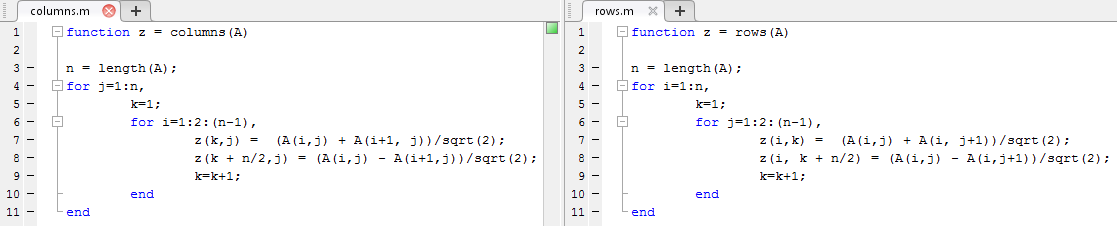
# Program Explanation

## Image Encoder

There are a couple of steps that need to be taken in order to use our encoding scheme. The sums and differences between the columns and rows need to be calculated to generate the edge information and a transformed matrix. We then quantize the transformed matrix and write the now quantized and transformed matrix into a binary file. What follows is the main function as well as a description of the individual functions:

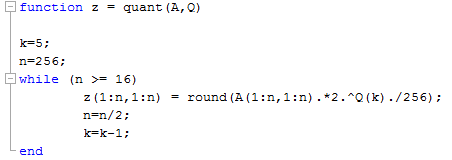
## 

### columns.m / rows.m



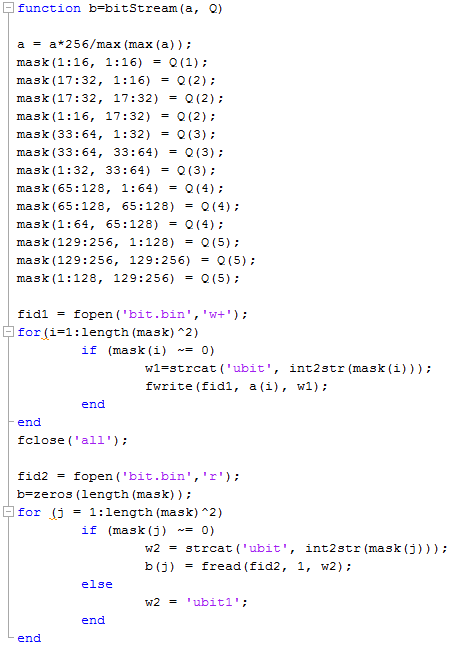
Both rows.m and columns.m take in the image in question (‘A’) and run a loop over either the rows or columns of the image, calculating the sums and differences of the consecutive elements. This is the Haar Discrete Wavelet Transformation procedure. It is important to note that we had to divide each sum or difference by the square root of 2 in order to keep the energy of the transformed image the same. Each use of the rows/columns combination essentially means a level of wavelet transformation. Both functions output a wavelet-transformed image.

### quant.m



The next step in our program asks for the quantization step size, as seen in the quant.m function. This program takes in the wavelet-transformed image (‘A’) after columns.m and rows.m are performed, as well as the previously described quantization mask (‘Q’). That mask (along with its values) is applied to the picture, giving different bit values (as dictated by the ‘Q’ vector) to different regions of the image. It is important to note that this function assumes an overall image width and height of 256 pixels, although this value could be changed to a user input to accommodate other image sizes. After assigning these values to different areas of the image, a quantized and wavelet-transformed image is now outputted.

### bitStream.m

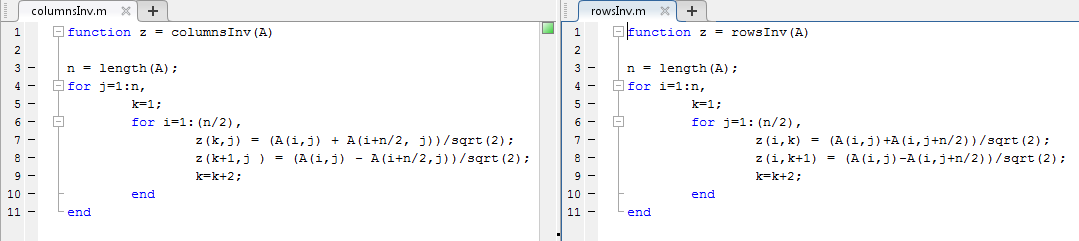


The generated data is now written to a file named bit.bin. Above is our entropy encoding function. Similar to the previous functions, this takes in an image (‘A’) which has been transformed and quantized, and the quantization mask vector (‘Q’). The (‘Q’) vector components are assigned to areas of the graph and are used to determine the amount of bits that are used to code a certain area of the graph. With this information, a bit.bin file is created, which is the compressed image file.

## Image Decoder

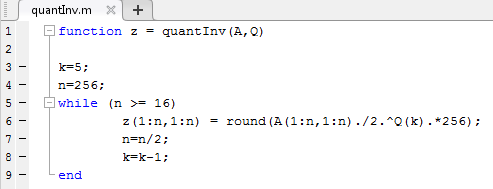
In order to decode the image we have now created, we essentially created a process that is identical to the encoder but just works inversely. First, the generated bit file is imported and read. The matrix of this image is then de-quantized and reproduced to generate the original image. The output of this image is compared to the original input image and the PSNR (Peak-Signal-To-Noise ratio) is calculated. Because the user is allowed to input their own quantization step of 1 to 8 bits, the PSNR tends to vary depending on what value is input.

### columnsInv.m / rowsInv.m



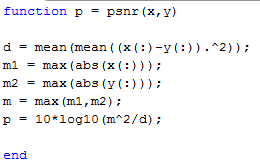
The inverse sums and differences calculations are made by adding and subtracting the appropriate pairs of numbers in the matrix and dividing by two. These values are renormalized and placed in their appropriate positions in the target matrix.

### quantInv.m



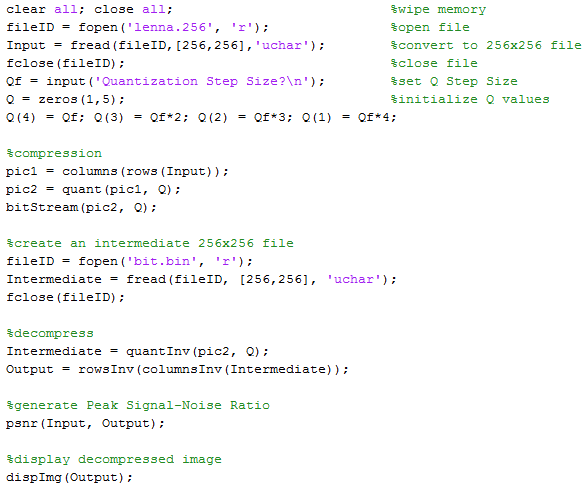
This function quantizes the matrix back to its original 8-bits.

### psnr.m



This function receives the original and reconstructed images and computes the Peak-Signal-To-Noise ratio, a measure of closeness between the two images. This allows us to compare the quality after compression and decompression of the original image.

## Main Code



The main function is a script that allows the compressor and decompressor to be run by simply typing “main” in the command line and prompting the user to input the quantization step size. It starts with a clear all / close all to clear any variables and close any open windows that may have resulted from a previous iteration of this script or another project. With a clean start, lenna.256 is read into a 256x256 double for data manipulation. The user is then prompted to choose the step size for a five-element quantization mask. This allocates a particular number of bits for storage in the transformed matrix, allowing the program to effectively compress the original image. Next, the “rows” and “columns” functions collect the sums and differences of column and row elements in “Input”, then outputs a wavelet-transformed image. The “quant” function applies a mask onto the transformed image, which produces an image for pic2. Finally, the “bitStream” function converts the image into a .bin file. To begin decompression, the .bin file is read in and stored it to an intermediate variable. The inverse functions “quantInv”, “rowsInv”, and “coumnsInv” are applied to complete decompression. Finally, using psnr.m, a peak signal-to-noise ratio between the original and final images is calculated to determine how close the decompressed image is to the original file.

# Group Member Contributions

For this project the whole group was very involved with all parts. When working on it all members were together and collaborating equally while coming up with ideas, planning, programming and testing/debugging. Additionally, all members were involved with writing this report.