# Project 2 Report

## Huffman Coding & LZW Coding:

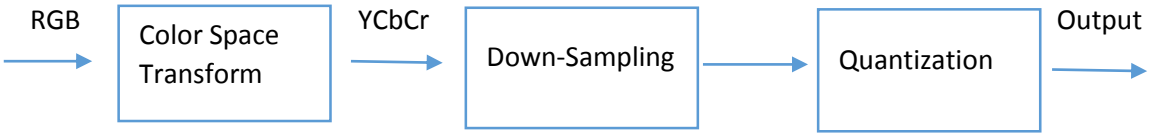
I use a joint Huffman coding (treat every 16 bits as a symbol), and have dictionary initialized with 256 base code for single symbol for my LZW coding. For both encoding methods, I use binary strings (e.g. “010110”) to store the codes for different symbols, then concatenate all strings to obtain a final binary sequence to represent a data file. The binary sequence will be split for every 8 bits as a byte to construct a byte array for file output.

The advantage is being easier for generating new code and code assignment to symbols. But the main drawback is the inefficiency in time since the memory will need more time to deal with the long string then do conversion instead of managing the byte directly.

Moreover, both encoders do not have very good performance as I expected. The compression ratio for Huffman coding ranges from 1.15 to 1.25, and the LZW has slightly higher one. Compared to FLAC which can typically reach a compression ratio of 60% to 70%, my algorithm might be too naïve for being used in practice. One possible reason is that FLAC also adapts linear prediction to convert the samples, then uses entropy coding to store the differences between predictors and actual samples. Mine just simply encodes the original sample bytes based on the Huffman tree built on the samples. Since many temporally adjacent samples are the same or similar, we can save man the y bits to encode the high frequency symbols thus reducing the code length in the bit sequence.

## Lossy Image Compression IM3:

The main technique used in my IM3 format encoding is YCbCr color space down-sampling and Midrise Quantization, which simply follow this processing flow:




And the decompressing process right follows the inverse direction. The compressor first converts the pixel byte values from RGB space to YCbCr space, in which we can do down-sampling onto the chroma components (Cb and Cr) with scale 4:2:0. That is, we retrieve every 2 * 2 block from the two dimensional pixel array, calculating the average of the chroma components values Cb and Cr (1 out of 4), and keeping the luminance component Y. Then it quantizes the YCbCr values in range (0, 255) to 16 different reconstruction levels of values in range (0, 15), by dividing a constant value and rounding to integers, which can be represented by only half of a byte (4 bits) each instead of a byte (8 bits) originally.

The quantization that reduces half of bits to encode the color information of a pixel apparently contributes 50% to the total compression ratio. The down-sampling reserves the Y component for every pixel but only takes 1 sample of Cb and Cr components out of 4 pixels, thus gaining another 50% data size reduction. In total, the compressed image will be around 1/4 of the size of the uncompressed image. Thus the compression ratio is about 4 for all.

For an image of resolution 256*256, The mean square error (MSE) would be (sample file from Project 1) $\frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j)-K(i,j)=377$ , and the reconstruction quality measured by PSNR is $10*\log_{10}\left(\frac{MAX^2}{MSE}\right)=22.37\ (dB)$