CSCI 576 Assignment 2

Instructor : Parag Havaldar
Assigned on 02/23/2015
Solutions due 03/13/2015 by 12pm noon

Late Policy: None, unless prior arrangement has been made

Question 1: Generic Compression Problem

The following sequence of real numbers has been obtained sampling a signal:
5.8, 6.2, 6.2, 7.2, 7.3, 7.3, 6.5, 6.8, 6.8, 6.8, 5.5, 5.0, 5.2, 5.2, 5.8, 6.2, 6.2, 6.2, 5.9, 6.3, 5.2, 4.2, 2.8, 2.8, 2.3, 2.9, 1.8, 2.5, 2.5, 3.3, 4.1, 4.9
This signal is then quantized using the interval [0,8] and dividing it into 32 uniformly distributed levels.

- What does the quantized sequence look like? For ease of computation, assume that you placed the level 0 at 0.25, the level 1 at 0.5, level 2 at 0.75, level 3 at 1.0 and so on. This should simplify your calculations. Round off any fractional value to the nearest integer levels
- How many bits do you need to transmit it?
- If you need to encode the quantized output using DPCM. Compute the successive differences between the values – what is the maximum and minimum value for the difference? Assuming that this is your range, how many bits are required to encode the sequence now?
- What is the compression ratio you have achieved?
- Instead of transmitting the differences, you use Huffman coded values for the differences. How many bits do you need now to encode the sequence?
- What is the compression ratio you have achieved now?

Question 2: Arithmetic Compression

Consider two symbols, A and B, with the probability of occurrence of 0.8 and 0.2, respectively. The coding efficiency can be improved by combining N symbols at a time (called “symbol blocking”). Say N = 3, so you are grouping symbols of 3 and giving them a unique code. (Assume that each symbol occurrence is independent of previous symbol occurrences).

- How many types of different outcomes are there and what are their probabilities?
- Show the arrangement of symbols on the unit interval [0, 1] and determine the arithmetic code for the three-symbol sequence.
- What is the average code word length? Is it optimum?
- How many bits are required to code the message “ABABBAABBAAABBB"
- How could you do better than the above code length?

Question 3: Programming on Vector Quantization

This assignment will increase your understanding of image compression. We have studied JPEG compression in class, which works by transforming the image to the frequency domain and quantizing the frequency coefficients in that domain. Here you will implement a common but contrasting method using "vector quantization". Quantization or more formally scalar quantization, as you know, is a way to

represent (or code) one sample of a continuous signal with a discrete value. Vector quantization on the contrary codes a group or block of samples (or a vector of samples) using a single discrete value or index.

Why does this work, or why should this work? Most natural images are not a random collection of pixels but have very smooth varying areas – where pixels are not changing rapidly. Consequently, we could pre-decide a codebook of vectors, each vector represented by a block of two pixels (or four pixels etc) and then replace all similar looking blocks in the image with one of the code vectors. The number of vectors, or the length of the code book used, will depend on how much error you are willing to tolerate in your compression. More vectors will result in larger coding indexes (and hence less compression) but results are perceptually better and vice versa. Thus vector quantization may be described as a lossy compression technique where groups or blocks of samples are given one index that represents a code word. In general this can work in k dimensions, but we will limit your implementation to two dimensions and perform vector quantization on an image.

When forming vector quantization you need to create a code book, – the size or type of vector you will use and the number of vectors. Let's assume that your vectors are *two adjacent pixels* side by side. For your assignment you will take as input a parameter *N*, which is the number of vectors in your codebook. You may assume this is a *N* is a power of 2 and thus after quantization each vector will need an index with log*N* bits. Your code will be called as follows and will result in a side by side display of the original image and your result after vector compression and decompression.

MyCompression.exe myImage.rgb N

Here are the steps that you need to implement to compress an image.

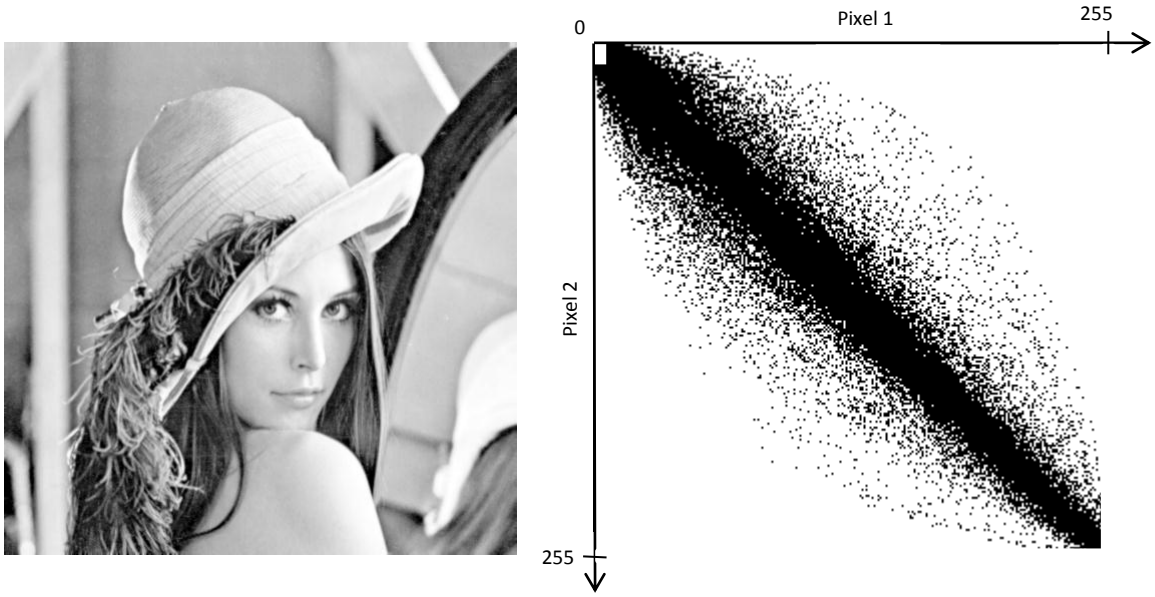
1. Understanding your two pixel vector space to see what vectors your image contains
2. Initialization of codewords - select N initial codewords
3. Clustering vectors around each code word
4. Refine and Update your code words depending on outcome of 3.

Repeat steps 3 and 4 until code words don't change or the change is very minimal.

5. Quantize input vectors to produce output image

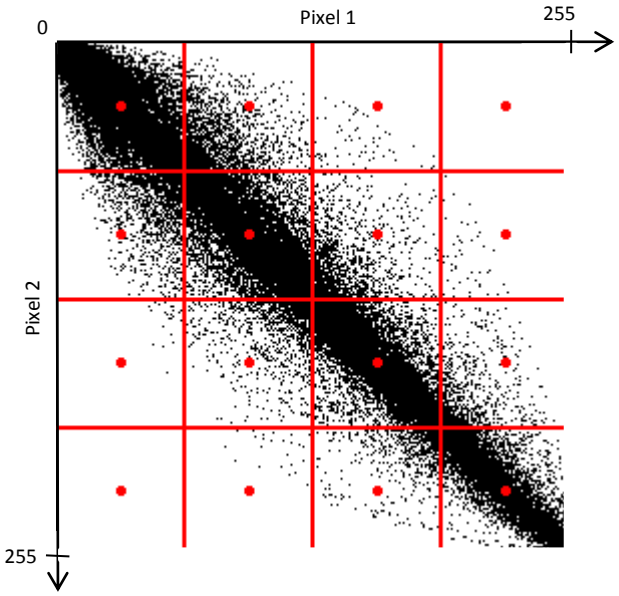
Step 1 - Understanding your vector space:

Let's look at the traditional Lena image below on the left. When creating a code book we need to decide two things – the size of the vector and the number of vectors. We have already established that we want to create a codebook of 2 pixel vectors. The space of all possible 2 pixel vectors is 256x256, and obviously not all of them are used. The right image shows a plot of which 2 pixels vectors are present in your image. Here a black dot shows a vector [pixel1, pixel2] being used in the image. This naturally is a sparse set because all combinations of [pixel1, pixel2] are not used.



Step 2 - Initialization of codewords

Next we need to choose the *N* vectors of two pixels each that will best approximate this data set, noting that a possible best vector may not necessarily be present in the image but is part of the space. We could initialize codewords using a heuristic, which may help cut down the number of iterations of the next two steps or we may choose our initial codewords in a random manner. The figure below shows a uniform initialization for N=16 where the space of vectors is broken into 16 uniform cells and the center of each cell is chosen as the initial codeword.

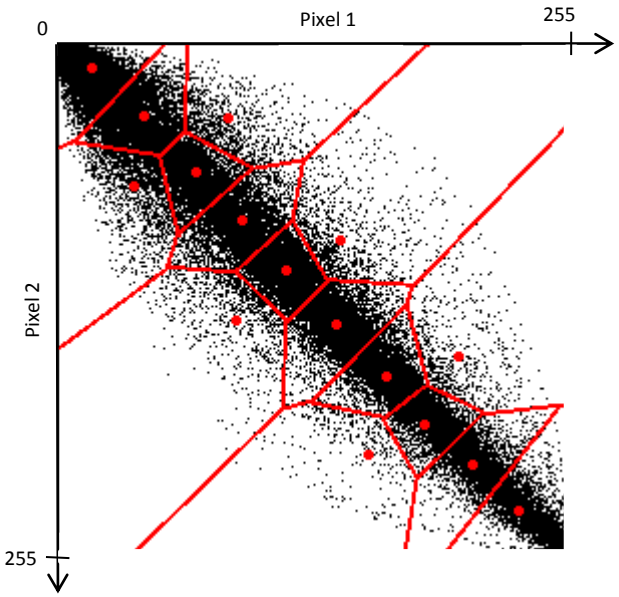


Every pair from the input image pixels would be mapped to one of these red dots during the quantization. In other words - all vectors inside a cell would get quantized to the same codebook vector. Thus compression is achieved by mapping 2 pixels in the original image to an index which needs log(N) bits or 4 bits in this example. In other words we are compressing down to 2 bits per pixel.

While the compression ratio is good, we see why this quantization is very inefficient: Two of the cells are completely empty and four other cells are very sparsely populated. The codebook vectors in the six cells adjacent to the x = y diagonal are shifted away from the density maxima in their cells, which means that the average quantization error in these cells will be unnecessarily high. Thus, six of the 16 possible pairs of pixel values are wasted, six more are not used efficiently and only four seem probably well used. This results in large overall quantization error for all codewords, also known as the mean quantization error. The next steps aim to reduce this overall mean quantization error.

Step 3 and 4:

The figure below show a much better partitioning and assignment of codewords, which is how vector quantization should perform. The cells are smaller (that is, the quantization introduces smaller errors) where it matters the most—in the areas of the vector space where the input vectors are dense. No codebook vectors are wasted on unpopulated regions, and inside each cell the codebook vector is optimally spaced with regard to the local input vector density.



This is done iteratively performing steps 3 and 4 together. Normally, no matter what your starting state is, uniformly appointed codewords or randomly selected codewords, this step should converge to the same result.

In step 3, you want to clusterize all the vectors, ie assign each vector to a codeword using the Euclidean distance measure. This is done by taking each input vector and finding the Euclidean distance between it

and each codeword. The input vector belongs to the cluster of the codeword that yields the minimum distance.

In step 4 you compute a new set of codewords. This is done by obtaining the average of each cluster. Add the component of each vector and divide by the number of vectors in the cluster. The equation below gives you the updated position of each codeword y_i as the average of all vectors x_{ij} assigned to cluster i, where m is the number of vectors in cluster i.

$$y_i = \frac{1}{m} \sum_{j=1}^{m} x_{ij}$$

Steps 3 and 4 should be repeated, updating the locations of codewords and their clusters iteratively until the codewords don't change or the change in the codewords is small.

Step 5 – Quantize input vectors to produce output image:

Now that you have your code vectors, you need to maps all input vectors to one of the codewords and produce your output image. Be sure to show them side by side.