RRY025 -- Image Processing

# Exercises 7 – Lecture 9 – Compression I

## EX 1: Shannon’s coding theorem

Compute how much the three images below can be compressed by lossless compression (Shannon’s coding theorem):

cameraman.tif

pout.tif

Mikki1\_crop\_close.jpg

Which image can be compressed most by lossless compression and which most? Can you describe why the order is what it is? (Hint: Inspection of image histograms is helpful – it describes the *P*(*r*) that goes into the Shannon’s limit calculation…)

**ANSWER:** Example given as a .mlx file. Inspection of image histograms should indicate that images with “simpler” histograms have lower entropy (theoretical lower limit). This is perhaps intuitively clear: when the image is “simpler”, it has less information and it is possible to compress it more.

## EX 2. Huffman coding

Consider the following image:

4 4 3 6

5 8 2 1

3 4 7 6

3 2 4 1

Compute in Matlab how much the image can be compressed by lossless compression.

**ANSWER:** Using Shannon’s coding theorem, the entropy of the image is ~2.83 bits/px. Thus, the image can be compressed with a compression ratio of 8 / 2.83 = 2.83.

Compute (and draw) by hand the Huffman Tree and encoding dictionary.

**ANSWER:** One possible tree (numbers intentionally left blank):



Encode the image using the dictionary you derived. Then compute the average bits per pixel. Compare it to the theoretical limit. Is the result close to the theoretical limit?

**ANSWER:** The Huffman coding results in needing 46 bits (2.875 bits/px) to code the image. This is close to the entropy of the image (lower limit), which is 2.83.

Finally, perform the Huffman encoding using the Matlab’s *huffmandict* function. Read the help of *huffmandict* first. The function will very likely give you another dictionary than the one you derived, however, it also will have some “similarity” to what you derived. Is the *huffmandict* giving the same entropy as your by-hand calculation (it should)? Can you understand the difference/similarity between the by-hand and Matlab dictionaries?

## EX 1: Run-length coding, Huffman coding

Consider the binary-coded image:

0 0 0 1 1 0 1

0 0 0 0 1 1 1

1 1 1 1 0 0 1

1 0 0 1 1 1 1

1 1 1 1 0 0 0

0 0 1 1 1 1 1

1 1 0 0 1 1 1

Is it advantageous (in terms of file size) to use fixed length run length coding to encode the image instead of fixed length coding? For this, encode the entire image as a continuous bitstream (i.e., not row-wise).

Is it further advantageous to apply Huffman coding on the run length data? I.e., does the bitstream become shorter by doing that?

Hint for result check:

The relevant file sizes for this exercise are 49 bits, 37 bits, and 42 bits (in a random order). These numbers assume that the receiver knows which number (0 or 1) comes first, i.e., the first pixel value does not need to be transmitted.

**ANSWER:** Transferring the image as-is requires **49 bits** (49 one-bit numbers).

The run lengths are: 3 2 1 1 4 7 2 2 2 8 5 7 2 3 (14 run lengths).

If one uses the binary encoding: {1: 0; 2:1; 3:10; 4:11; 5: 100; 6: 101; 7: 110; 8: 111}, one gets by using 3 bits per run length. Thus, one needs 3 bits \* 14 run lengths = **42 bits**.

Huffman coding on the run lengths results in the total amount of **37 bits** to be transferred.

## EX 4. Pre-compression (=Lossy compression) in image and transform domains

Using *cameraman*, find out how much the lossy wavelet pre-compression improves the potential for compression in 1) the image domain, and 2) the transform domain.

Doing this in image domain means finding out how much smaller entropy an array (here: cameraman image) that has been pre-compressed (and reconstructed) has compared to the original image.

Doing this in the transform domain means comparing the amount of non-zero pixels in the pre-compressed transform image compared to the one that is non-pre-compressed.

What is “the right amount” of pre-compression? This you must find out by looking at the reconstructed image and deciding if it still is of acceptable quality.

One possible roadmap:

Perform pre-compression on the cameraman using a wavelet transform. Make a FWT using the Haar wavelet until level 5 and nullify globally x % of the smallest coefficients. You can do this using the Wavelet Analyzer’s ‘compression’ utility, but perhaps easiest is to use *wavedec2() and waverec2()*.

Let *I* be the cameraman image. Use:

[c, s] = wavedec2(I, 5, ‘Haar’)

to get the ‘c’(coefficients) and ‘s’ (sizes) structures for the decomposition.

Reshape the coefficient vectors to a viewable form: I\_FWT = reshape(c, 256, 256);

Then nullify x% of the coefficients with smallest absolute values, to yield I\_FWT\_mod.

Reconstruct the image using *waverec2()*, to yield *I\_rec*.

Then compare the entropy of *I* and *I\_rec*. Is there any improvement?

What do you think? Is it size-wise better to transfer: *I*, *I\_rec*, *I\_FWT*, or *I\_FWT\_mod*?

**ANSWER:** Entropy of the original image (*I*) is about 7 bits/px. Entropy of the image from which 90% of the smallest wavelet coefficients have been nullified (*I\_rec*) is about 6.85 bits/px. Thus, it does not help a lot to compress the image if it is transferred in the image domain (=waveform decoder).

The original image is an 8 bits/px image. The numerical values in the wavelet transform vary between ~[-1200, 6000]. Such numbers can be represented as 16-bit “half” precision floating point numbers in Matlab. To transfer the I\_FWT one thus needs 256 x 256 = 65 536 16-bit numbers. This is clearly undesirable.

In the pre-compressed wavelet image, I\_FWT\_mod, 90% of the coefficients are nullified; one thus needs on average only about 0.1 \* 256 x 256 ~= 6 534 16-bit numbers. This corresponds roughly to 6 534 \* 16 / 65 536 = 1.6 bits/pixel on average. It is thus likely most beneficial to transfer the I\_FWT\_mod.