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…)

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.

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

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?

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 3: Run-length coding, Huffman coding

Consider the binary 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.

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*?