Differential Lossless Compression

The principle is to compare each pixel *p* to a *reference pixel*, which is one of its previously-encoded immediate neighbors, and encode *p* in two parts: a prefix, which is the number of most-significant bits of *p* that are identical to those of the reference pixel, and a suffix, which is the remaining least-significant bits of *p*. For example, if the reference pixel is 10110010 and *p* is 10110100, then the prefix is 5, because the five most-significant bits of *p* are identical to those of the reference pixel, and the suffix is 00. Notice that the remaining three least-significant bits are 100 but the suffix does not have to include the 1.

The prefix part is Huffman coded and since we expect most suffixes to be small, it makes sense to write the suffix on the output stream un-coded.

So for the above example the code of *p* will be the five bits *010|00*.

A simple black and white digitized image consists of a rectangular or square array of pixels (say 256 by 256). Each of these pixels is assigned a number that indicates how light or dark that particular pixel is in the image; usually there are 256 gray levels, with 0 corresponding to black (no brightness on your screen) and 255 to white (maximum brightness). This means that one pixel uses 8 bits; for the whole 256x256 image this gives 256 * 256 * 256 = 16777216 bits, which is 16777216/256 = 65536 bytes or 65536/1024 = 64 KB. Color images of course use a bit more memory and movies, with 18-25 images per second, use much more memory.

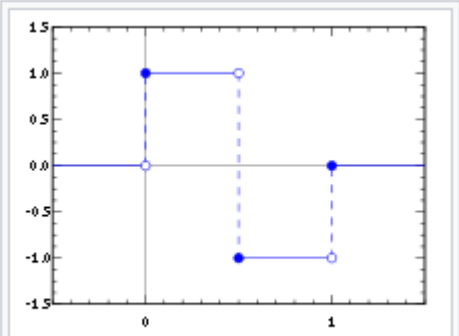
That is why it is very important to be able to compress images. In many cases, you can obtain images that are so close to the original image as to be virtually indistinguishable, but that are in fact stored on only a fraction of the space originally

needed. For some applications, you may even be happy with a noticeable distortion if the image still looks pretty good, provided the memory savings are really huge.

There exist several types of algorithms that compress images. We shall illustrate one of them in this lab, called "subband filtering"; it is related to a mathematical concept called the "wavelet transform".

1D Haar Wavelet Transform: Recursive Definition

The Haar wavelet is also the simplest possible wavelet. The technical disadvantage of the Haar wavelet is that it is not continuous, and therefore not differentiable. This property can, however, be an advantage for the analysis of signals with sudden transitions, such as monitoring of tool failure in machines.



The Haar wavelet

Two iterations of the 2D Haar wavelet decomposition on the Lenna image. The original image is high-pass filtered, yielding the three detail coefficients subimages (top right: horizontal, bottom left: vertical, and bottom right: diagonal). It is then low-pass filtered and downscaled, yielding an approximation coefficients subimage (top left); the filtering process is repeated once again on this approximation image

Horizontal Differencing and Averaging

The basic principle is very simple. Imagine taking out just one horizontal line from the black and white image. The gray levels for the pixels in this line form a sequence of numbers between 0 and 255. In this sequence many values are very close to their neighbors - sudden jumps only occur if there was a sudden transition there in brightness (say from a light object to a dark wall) in the image. So a piece of this sequence could look like:

45 45 46 46 47 48 53 101 104 105 106 106 107 106 106 106.

If you are given any 2 numbers **a** and **b** then you completely characterize them by giving their average s = (a+b)/2 and their difference d = a-b. (Because a = s+d/2, b = s-d/2). The averages and differences for successive pairs in our sequence are:

s:	45	46	47.5	77	104.5	106	106.5	106
d:	0	0	-1	-48	-1	0	1	0

The sequence of differences has many more really small entries than large entries, and such sequences are easy to compress. Moreover, we can now easily make a small change to the **d**-sequence that would make it even more compressible. For instance, if we replace in the **d**-sequence every entry that is a 0 or 1 or -1 by 0:

d': 0 0 0 -48 0 0 0 0

then this **d'**-sequence is highly compressible.

If we now recomputed the original sequence (rounding off if necessary, so that we get integers) by a' = s+d'/2 , b' = s-d'/2, then we get:

45 45 46 46 47 47 53 101 104 104 106 106 106 106 106 106

which is very close to the original.

Vertical Differencing and Averaging

Since images are 2-dimensional, we have to do this averaging and differencing in two dimensions as well. We can first do it within every row, transforming our 256x256 image into two arrays (one of averages, one of differences) of 256x128 entries each; for each of these we can then do the same vertically, so that in the end we have four arrays of 128x128. Here is a simple example:

45	47	101	101
46	46	103	103
47	47	103	101
48	48	55	55

After averaging and differencing within every row:

averages		differences	
46	101	-2	0
46	103	0	0
47	102	0	2
48	55	0	0

After averaging and differencing in two directions:

	horizontal averages		horizontal differences	
vertical averages	46	102	-1	0
	47.5	78.5	0	1
vertical differences	0	-2	-2	0
	-1	47	0	2

In these four little arrays, the top left one corresponds to averaging in both directions; this array typically has sizeable entries for all pixels. The other three arrays typically have most of their entries very small, and can thus be highly compressed.