600-656 Experimental Methods
Computation lab 1

Jonathan Newnham

April 19, 2010

1 Introduction

There are many ways of representing images. It is usually possible to remove a lot of the high-frequency information from images without changing them noticeably. This is how JPEG image compression works. In this assignment we investigate the Fourier and Discrete Cosine Transforms that allow such manipulation.

2 Preliminary Study

2.1 Bit-by-bit Correlation

A binary signal and a reference waveform were provided. A short matlab routine (`cm1.m`) [1] plots the signal offset that gives the largest correlation with the reference waveform (Figure 1).

We see quite clearly from the figure that the best correlation is achieved by offsetting the signal by 235. This corresponds to a signal of

```
110100000100100101110000110110111001000101101100101100111011010111010100110111100111101011
000101000011110100111100010001110100010011001001110011010101101001010111111011101111000001
10011000010000101010100011000111111100101001000000010111101100000011100.
```

This exercise demonstrates the usefulness of the correlation method – it is much easier to see that the signals are correlated from the correlation graph than from looking at the two signals side-by-side or superimposed. If both signals were periodic this would result in regular spikes in the graph with a period related to the beat frequency between the signals.

2.2 Fourier Transform

In this section we investigate the Discrete Fourier Transform of x, the correlation between a signal and an orthogonal basis of sine and cosine waves:

$$a_m \equiv \frac{2}{N}\sum_{n=0}^{N-1} x_n \cos\left(\frac{2\pi mn}{N}\right)$$

$$b_m \equiv \frac{2}{N}\sum_{n=0}^{N-1} x_n \sin\left(\frac{2\pi mn}{N}\right)$$

for $m \in 0..(M-1)$ and the inverse transform

$$\hat{x}_n \equiv \frac{a_0}{2} + \sum_{m=0}^{M-1} a_m \cos\left(\frac{2\pi mn}{N}\right) \sin\left(\frac{2\pi mn}{N}\right)$$

for $n \in 1..N$.

The DFT converts a signal x to its frequency-space representation a_m and b_m. If $M = N/2$ then no information is lost. We can then convert back ($\hat{x}$).

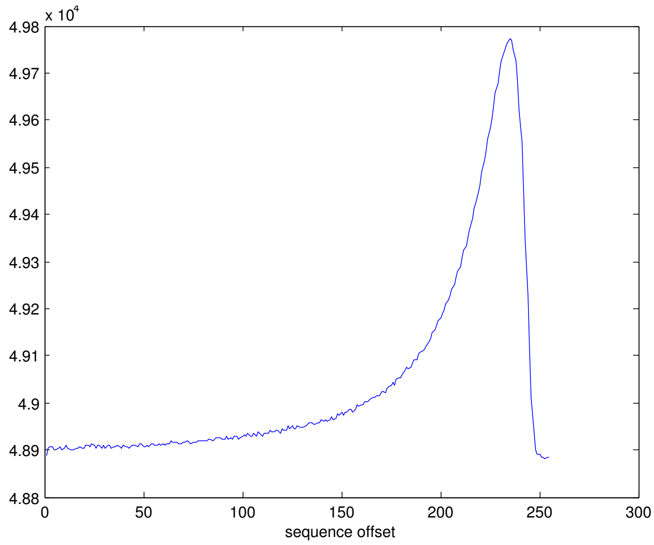
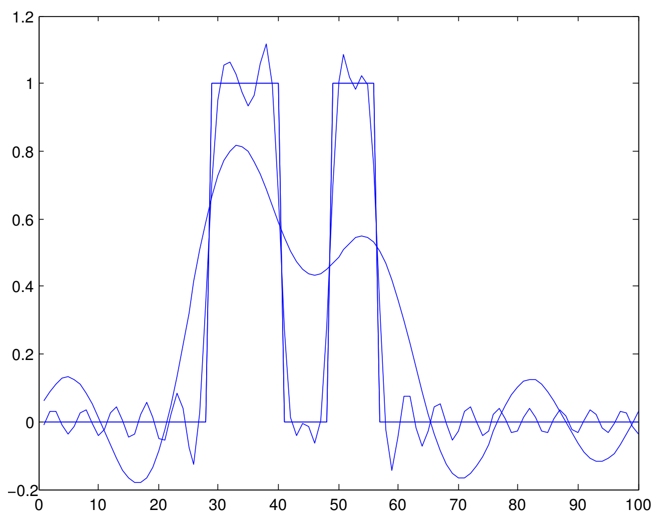
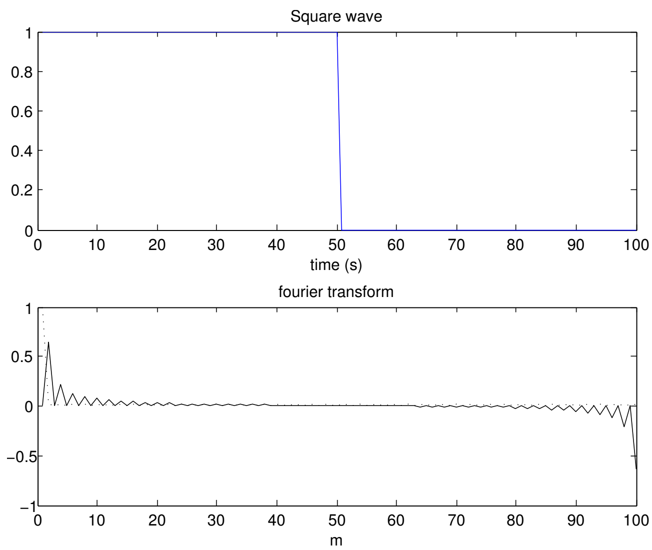


Figure 1: Correlation.



(a) Some Fourier representations of a simple signal, for $M = 5$ (smooth curve), $M = 20$ and $M = 50$ (square curve, same as original signal). The x-axis is time.



(b) The Fourier transform of a square wave. The b coefficient is drawn as a solid line, a is dotted. This is the expected result.

Figure 2: Some Fourier transforms.

A vectorized version is implemented in `DFT.m` [1] . The code in `cm2.m` then plots $\hat{x}$ for several different values of M (Figure 2a). We also plot the a_m and b_m coefficients of a square wave to ensure that they are as expected (Figure 2b).

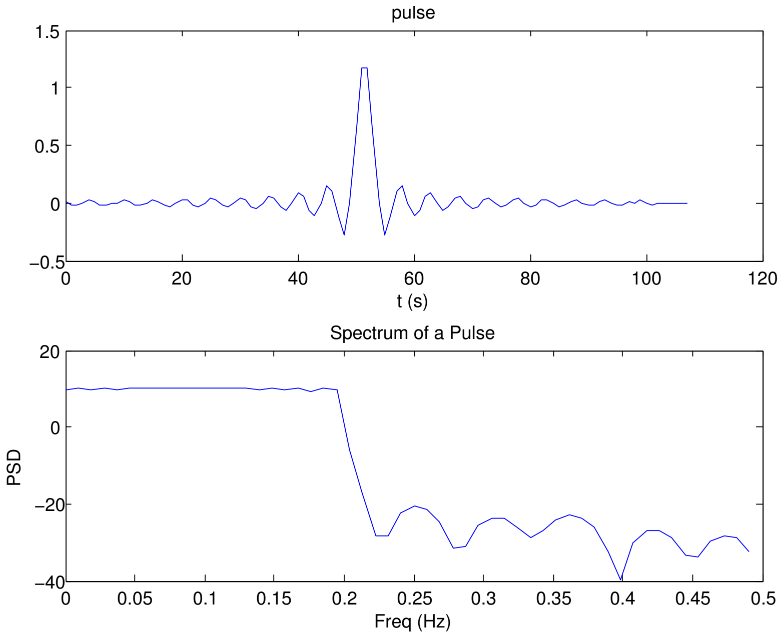
2.3 Zero Extension

Extending the signal with lots of zeros before we Fourier-transform it has the effect of giving us an interpolated spectrum (Figure 3b). See file `zero-extension.m`.

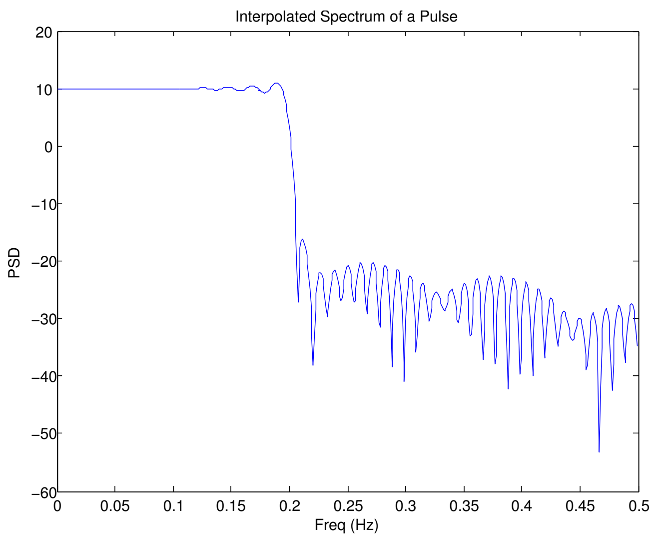
2.4 Zero Insertion

Extending the *spectrum* with lots of zeros has the effect of giving us an interpolated *signal* when we transform back to the time domain (Figure 3c). See file `zero-insertion.m`.

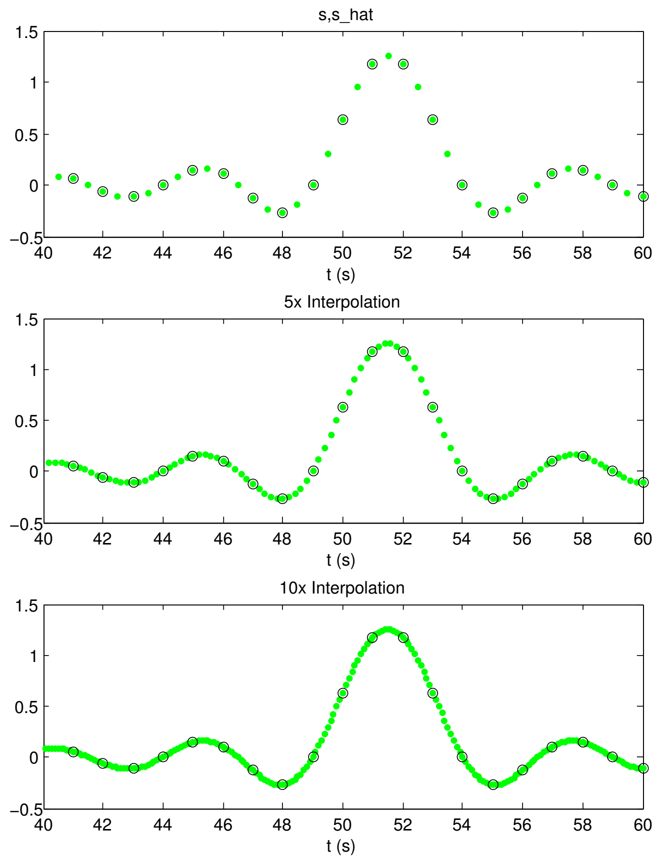
While these techniques don't actually add any information to the signal, they can sometimes make the signal easier to interpret, or allow smoothing of a signal if higher-frequency information is not wanted.



(a) The pulse to be interpolated.



(b) The spectrum resulting from zero extension.



(c) Zero insertion. The original signal is drawn as open circles.

Figure 3: Interpolation techniques.

3 Mathematical Background

The discrete cosine transform (DCT) X of an $N_1 \times N_2$ matrix A is

$$X_{m_1,m_2} = c_{m_1}\sqrt{\frac{2}{N_1}\frac{2}{N_2}}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}\cos\left(\frac{(2n_1+1)m_1\pi}{2N_1}\right)A_{n_1,n_2}\cos\left(\frac{(2n_2+1)m_2\pi}{2N_2}\right)$$

with

$$c_{m_1} = \begin{cases} \frac{1}{2} & \text{for } m_1 = 1 \\ 1 & \text{for } m_1 > 1 \end{cases}$$

The inverse transform is

$$A_{n_1,n_2} = c_{n_1}\sqrt{\frac{2}{N_1}\frac{2}{N_2}}\sum_{m_1=0}^{M_1-1}\sum_{m_2=0}^{M_2-1}\cos\left(\frac{(2m_1+1)n_1\pi}{2M_1}\right)X_{m_1,m_2}\cos\left(\frac{(2m_2+1)n_2\pi}{2M_2}\right)$$

This is implemented by the matlab code in the files `DCT.m` and `DCTinv.m`. [1]

When we are analyzing how effective our compression has been, we will use “Entropy” as a measure. This is calculated as

$$H = -\sum_{\{values\}} P(\text{value}) \times \log\left[P(\text{value})\right],$$

where $P(\text{value})$ is the probability of a value occurring. A lower entropy indicates that there is less information in a signal and hence that it can be more easily losslessly compressed, for example by Huffman coding.

4 What I Learned

The JPEG algorithm takes a block of 8x8 block of pixels, such as

$$A = \begin{pmatrix} 68 & 46 & 47 & 49 & 55 & 56 & 62 & 55 \\ 179 & 176 & 186 & 178 & 65 & 9 & 10 & 11 \\ 219 & 227 & 227 & 236 & 46 & 17 & 9 & 17 \\ 218 & 217 & 219 & 230 & 42 & 20 & 20 & 20 \\ 218 & 221 & 217 & 211 & 20 & 22 & 23 & 24 \\ 226 & 223 & 223 & 125 & 17 & 19 & 21 & 22 \\ 223 & 228 & 231 & 33 & 18 & 17 & 20 & 18 \\ 225 & 231 & 202 & 26 & 20 & 17 & 21 & 19 \end{pmatrix},$$

and uses the Discrete Cosine Transform to convert it into its frequency-space representation,

$$X = \begin{pmatrix} 827 & 607 & 66 & -161 & -44 & 62 & 22 & -25 \\ -39 & -130 & -113 & 4 & 86 & 43 & -35 & -50 \\ -140 & -134 & 61 & 83 & -37 & -71 & 17 & 78 \\ -72 & -106 & 16 & 32 & -7 & -4 & 9 & 12 \\ -42 & -84 & 3 & 28 & 9 & 4 & -9 & -12 \\ -28 & -47 & 15 & 19 & -14 & -11 & 10 & 16 \\ -6 & -24 & 6 & 8 & 3 & 3 & -2 & -13 \\ -1 & -8 & 4 & -2 & -6 & 1 & 2 & -7 \end{pmatrix},$$

with the lowest-frequency components in the upper-left corner. It turns out that most people don’t notice quite large changes to the high-frequency (bottom-right) components of an image. The JPEG scheme takes advantage of this by removing unnecessary high-frequency components using a QUANTIZATION MATRIX.

The step where all the compression happens is when X is divided by the quantization matrix and then rounded to the nearest integer. This means that large values in the quantization matrix reduce the entropy in components of X – a value of 128 in Q will reduce the 8-bit number (any value between 0..255) to a single bit of information, as the lower-order bits are discarded when the matrix is rounded off.

The suggested quantization matrix for a quality factor of 50 is

$$Q_{\text{JPEG50}} = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

which was determined using subjective tests where people would say how good they thought a particular image looked after being compressed. Other QUALITY FACTORS are determined by scaling this matrix up or down and clipping to the range $[1,255]$. For example, $Q_{\text{JPEG90}} = Q_{\text{JPEG50}} * (100-90)/50$.

It is quite clear that this matrix heavily suppresses the high-frequency components. For example, $X_{7,7}$ will get reduced to just over two bits of information from its previous 8. Interestingly, the lower-order components are also reduced quite a lot – for example, the constant term, $X_{1,1}$, has its lowest 4 bits rounded away. This is quite acceptable because the human eye mostly judges colour by comparison with nearby colours. It is hard-pressed to tell the difference between a pixel values of 40 or 50. This is why most components can be have their lowest-order bits discarded.

People tend not to notice if the high-frequency components of an image are less detailed, so most quantization matrices are larger in the lower-right components, resulting in more zeros in X when X is divided by Q. The reduced X is then normally further compressed using lossless Huffman encoding; we will use Entropy H and count zeros to get an estimate of the possible compression ratio.

These techniques can also be used to remove certain specific frequencies of noise from an image (by dividing or setting to zero the specific components of the DCT that are problematic). They can't do much about white noise as that is frequency-independent.

5 Results

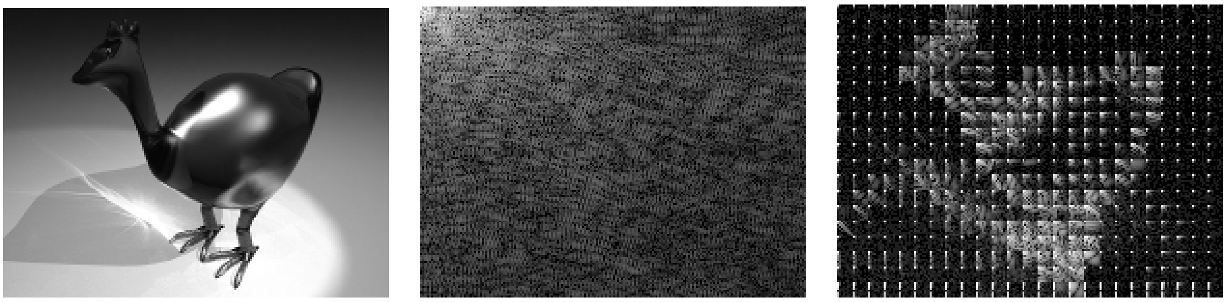
Code to transform images was implemented in `jpeg.m` [1]. I used Figure 4a as my sample image. It has regions where both high and low frequency are important.

5.1 Reducing the high frequencies

Figure 5 gives the results of a linear quantization matrix that is biased towards removing high-frequency components. This neatly erases the high-frequency components and results in a compression factor of $0.83/4 = 4.8$ using the entropy values of the DCTs. The image (Figure 5a) is still quite recognizable, although some small high-frequency noise is visible. The high-frequency components are more noisy because they have been rounded off more severely. A smaller maximum value for the linear compression scheme would result in a cleaner image but with a lower compression ratio. Trying to take the DCT of the full image instead of block-by-block is noticeably less successful. The compression ratio is not as good as well as the introduced noise being much more noticeable because it is of lower frequency.

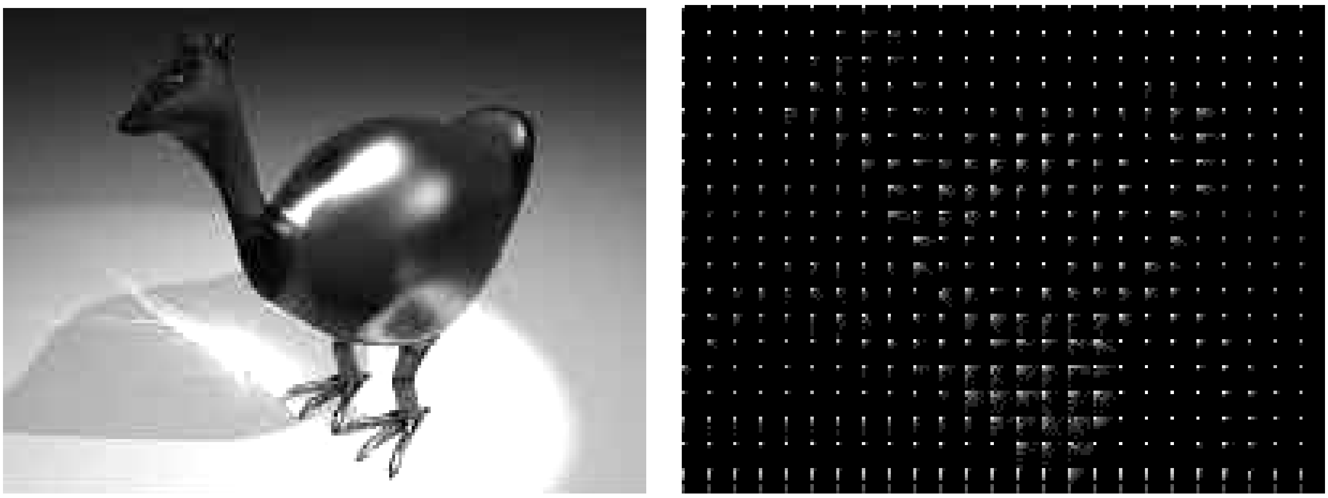
5.2 Reducing the low frequencies

Figure 6 gives the results of a linear quantization matrix that is biased towards removing low-frequency components. This results in a similar compression ratio but subjectively far worse image quality.



(a) Sample image (b) Full-image DCT. $H = 5.7$. (c) 8x8 DCT. 37% zeros, $H = 4.0$

Figure 4: The sample image and some DCTs of it. The full DCT contains more information (higher entropy H) than the 8x8 block DCT because the rounding was more drastic for the 8x8 blocks. In Figure 4c, observe how the low-frequency components are always strong, but the high-frequency components are only strong in complicated parts of the image.



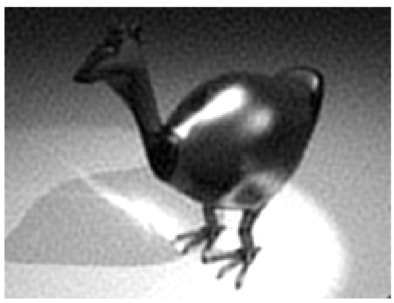
(a) The effect of reducing the amount of information in high-frequency components.

(b) The effect of removing high-frequency components. 91% zeros, $H = 0.83$

9	14	19	24	30	35	40	45
14	22	30	38	46	55	63	71
19	30	41	52	63	74	85	96
24	38	52	66	80	94	108	122
30	46	63	80	97	114	131	148
35	55	74	94	114	134	154	174
40	63	85	108	131	154	177	199
45	71	96	122	148	174	199	225

(c) A "linear" quantization matrix that removes high-frequency components.

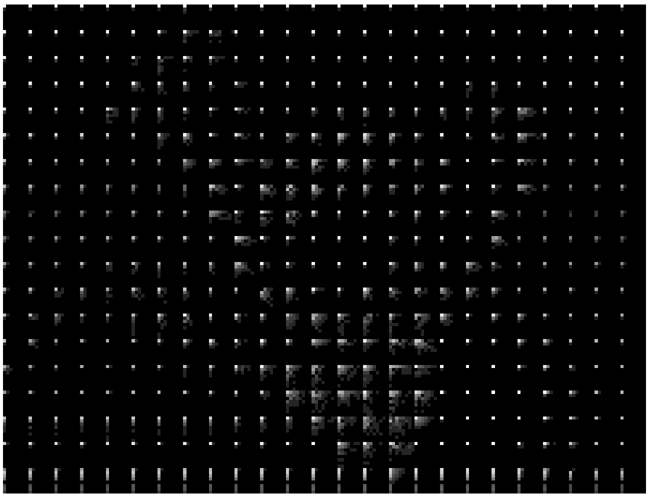
(d) A reduction of high frequencies in the full transform. 89% zeros, $H = 0.88$. Compare Figure 4b.



(e) The effect of a full linear reduction.

Figure 5: Results.

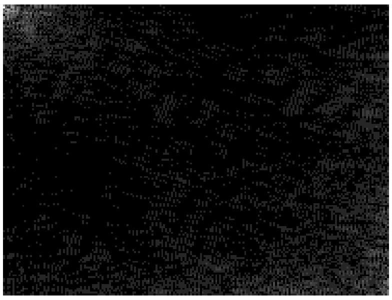
(a) The effect of reducing the information content in low-frequency components.



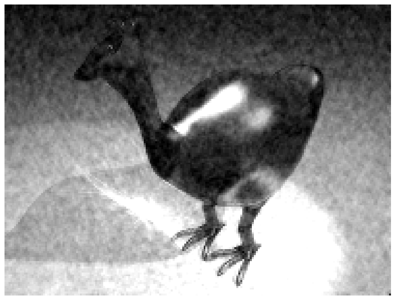
(b) The DCT: 95% zeros, $H = 0.38$

225	199	174	148	122	96	71	45
199	177	154	131	108	85	63	40
174	154	134	114	94	74	55	35
148	131	114	97	80	63	46	30
122	108	94	80	66	52	38	24
96	85	74	63	52	41	30	19
71	63	55	46	38	30	22	14
45	40	35	30	24	19	14	9

(c) A "linear" quantization matrix that removes low-frequency components.



(d) A reduction of low frequencies in the full transform. 75% zeros, $H = 1.2$. Compare Figure 4b.



(e) The effect of a full linear reduction of low frequencies.

Figure 6: Removing low-frequency components. The image quality is far worse than if high-frequency components are removed. Notice how fine detail is still visible.

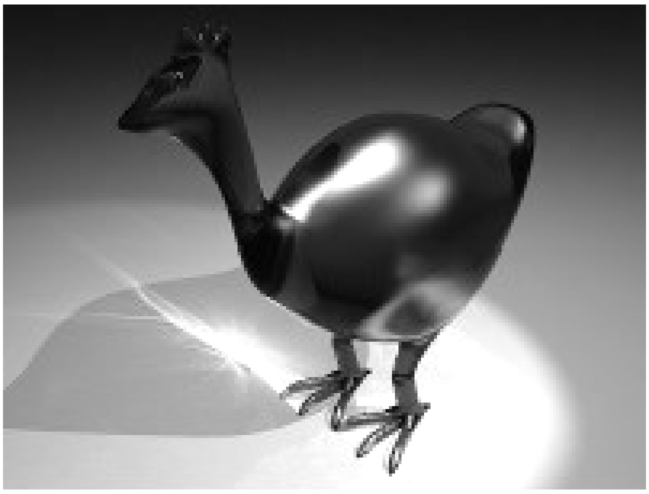
Preserving the high-frequency detail (observe the chicken's feet in Figure 6a) at the expense of low-frequency detail results in a very poor reconstructed image. Interestingly, when applied to the full DCT (6e), this approach resulted in the most reduction of the mid-range frequencies, as the low frequencies were too strong to be removed. The noise from this is quite noticeable but the image and its fine detail are still quite recognizable, with a decent entropy ratio of about 3.

5.3 JPEG

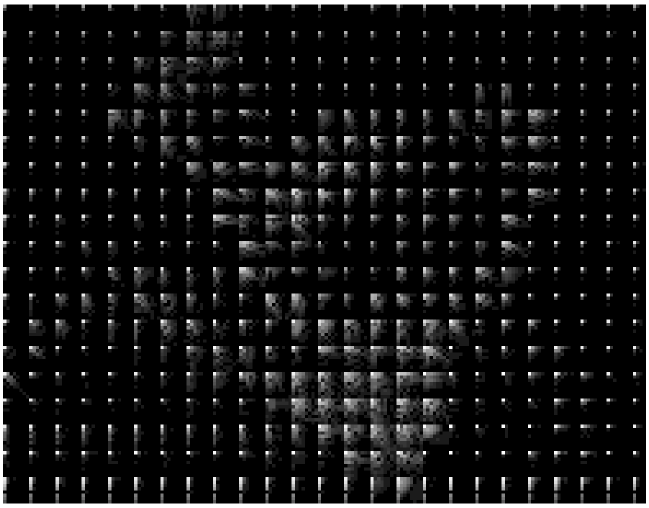
We now turn to the quantization matrix recommended in the JPEG standard (Figure 7). With a quality level of 90, the result of the compression is almost undetectable by visual inspection. This still gives a very decent entropy ratio of 2. With a quality level of 50, we just start to see some high-frequency noise creeping in, but with an excellent entropy ratio of 4. The image is still very recognizable, validating the recommended JPEG quantization matrix as better than the linear version. This is an excellent result. It is interesting that the JPEG Q-matrix (Q_{JPEG50}, page 6) reduces the almost-highest frequency components more than the very highest frequency components.

6 Conclusion

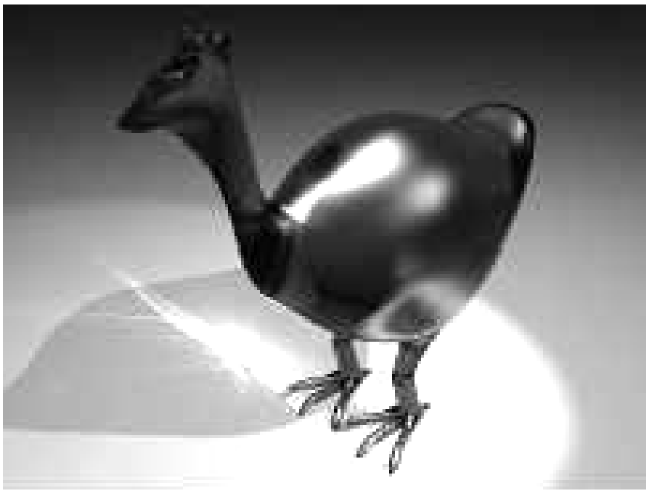
Quantization matrices which removed too many low-frequency components made the image appear blocky (see the Linear quantization matrix in Figure 5). Quantization matrices which removed high-frequency components resulted in high-frequency noise appearing in the image. Trying to perform a transform on the whole image instead of in blocks did not take advantage of local autocorrelation and



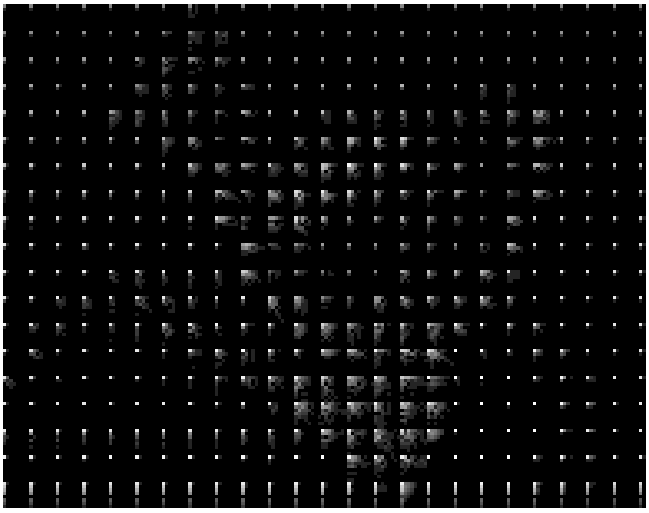
(a) JPEG-90.



(b) DCT of JPEG-90 quantization. 77% zeros, $H = 1.92$.

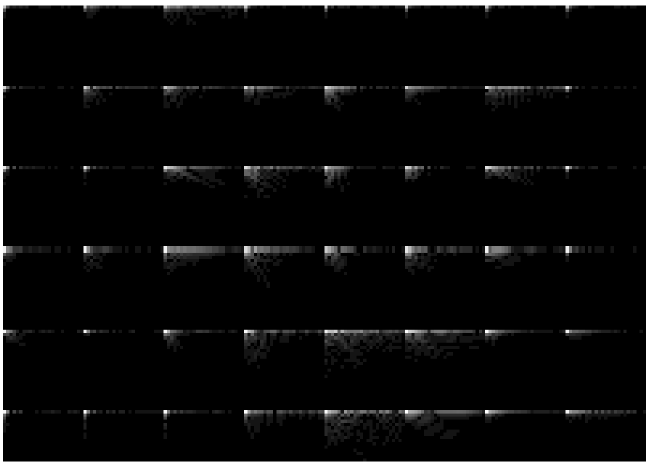


(c) JPEG-50.



(d) DCT of JPEG-50 quantization. 89% zeros, $H = 0.98$.

(e) Just for fun, an asymmetric quantization of the chicken image. Horizontal detail is mostly preserved, while vertical detail is reduced. Interestingly, the noise appears to be equal in both direction, even though (looking at the DCT) it is clear that there should be considerably more noise in the vertical direction.



(f) DCT after asymmetric quantization. Square size 25 pixels. 91% zeros.

Figure 7: JPEG compression.

so resulted in lower compression ratios (i.e. higher entropy, or lower numbers of zeros). Attempting to transform the whole image also introduced lower-frequency noise that was much more noticeable.

This form of compression is very suitable for image compression if the quantization matrix is chosen carefully. A JPEG quality factor of about 90% results in good compression without noticeably degrading the image.

7 Notes

1. The files referenced in this document can be found at `http://www.ph.unimelb.edu.au/~jnnewn/matlab`.