Image Compression 1
Image and Video Processing

Dr. Anil Kokaram* anil.kokaram@tcd.ie

Contents

1 Image Compression with the Haar Transform **2**

2 Introduction **2**

3 The Haar transform **4**

4 Entropy **7**

5 The Multi-level Haar Transform **9**

5.1 Use of Laplacian PDFs in Image Compression 16

6 Practical Entropy Coding Techniques **20**

7 Summary and Reference material **24**

A Huffman Coding **26**

*Abridged version of material from Nick Kingsbury, Signal Processing Group, Cambridge University

1 Image Compression with the Haar Transform

This section will begin to connect portions of this course with ideas from the study of Digital Communications in examining elements of a simple compression system for Images. The goals here are

- To illustrate the importance of Energy Compression, Quantisation and Entropy Coding in image compression
- To introduce the notion of transform compression of images
- To introduce the use of Laplacian p.d.f's for modelling image distributions
- To illustrate the importance of directional filters
- To introduce initial multi-scale image processing concepts

2 Introduction

As usual, the image is represented by a matrix $\mathbf{I}$, with elements $\mathbf{I}(\mathbf{x})$, with $\mathbf{x} = [n_1, n_2]$ being a position vector of row and column indices. Define the energy of $\mathbf{I}$ as

$$E(\mathbf{I}) = \sum_{\mathbf{x}} I(\mathbf{x})^2 \tag{1}$$

Thus the energy of an image is the sum of the squares of the pixel intensities at all sites in the image.

Figure 1 shows the main blocks of any compression system. The decoder performs the inverse operation to the encoder. The blocks perform the following tasks

Energy compression – This is usually a transformation or filtering process which aims to concentrate a high proportion of the energy of the image $\mathbf{I}$ into as few samples (coefficients) of $\mathbf{y}$ as possible while preserving the total energy of $\mathbf{I}$ in $\mathbf{y}$. This minimizes the number of non-zero samples of $\mathbf{y}$ which need to be transmitted for a given level of distortion in the reconstructed image $\hat{\mathbf{I}}$.

Quantisation – This represents the samples of $\mathbf{y}$ to a given level of accuracy in the integer matrix $\mathbf{q}$. The quantiser step size controls the tradeoff between distortion and bit rate and may be adapted to take account of human visual sensitivities. The inverse quantiser reconstructs $\hat{\mathbf{y}}$, the best estimate of $\mathbf{y}$ from $\mathbf{q}$.

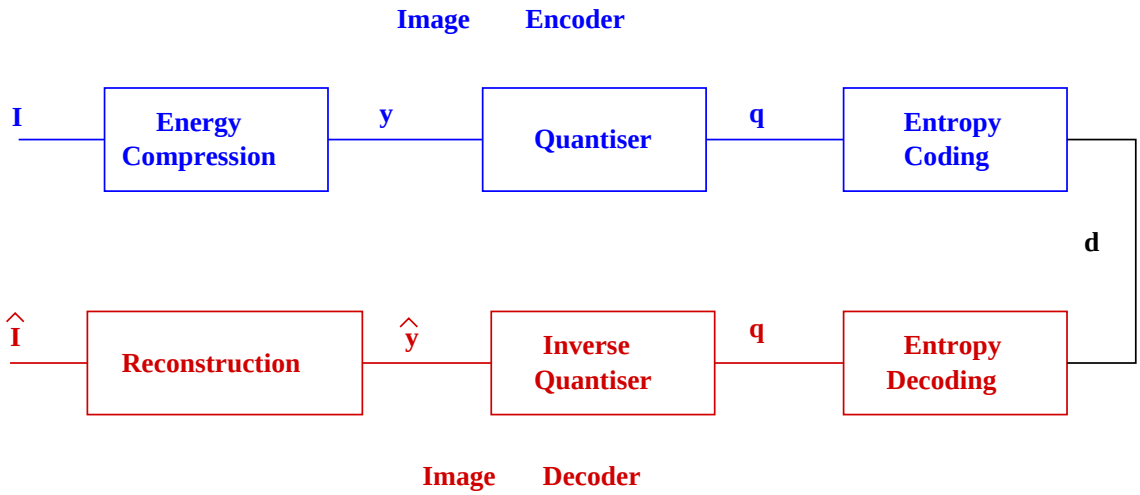

Figure 1: The Main blocks of a compression system.

Entropy coding – This encodes the integers in $\mathbf{q}$ into a serial bit stream $\mathbf{d}$, using variable-length entropy codes which attempt to minimise the total number of bits in $\mathbf{d}$, based on the statistics (PDFs) of various classes of samples in $\mathbf{q}$.

The energy compression / reconstruction and the entropy coding / decoding processes are normally all lossless. Only the quantiser introduces loss and distortion: $\hat{\mathbf{y}}$ is a distorted version of $\mathbf{y}$, and hence $\hat{\mathbf{I}}$ is a distorted version of $\mathbf{I}$. In the absence of quantisation, if $\hat{\mathbf{y}} = \mathbf{y}$, then $\hat{\mathbf{I}} = \mathbf{I}$.

Figure 2: Original (a) and the Level 1 Haar transform (b) of the 'Lenna' image.

3 The Haar transform

Probably the simplest useful energy compression process is the Haar transform. In 1-dimension, this transforms a 2-element vector $[x(1), x(2)]^T$ into $[y(1), y(2)]^T$ using:

$$\begin{bmatrix} y(1) \\ y(2) \end{bmatrix} = \mathbf{T} \begin{bmatrix} x(1) \\ x(2) \end{bmatrix} \qquad \text{where} \quad \mathbf{T} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2}$$

Thus $y(1)$ and $y(2)$ are simply the sum and difference of $x(1)$ and $x(2)$, scaled by $1/\sqrt{2}$ to preserve energy.

Note that $\mathbf{T}$ is an orthonormal matrix because its rows are orthogonal to each other (their dot products are zero) and they are normalised to unit magnitude. Therefore $\mathbf{T}^{-1} = \mathbf{T}^T$. (In this case $\mathbf{T}$ is symmetric so $\mathbf{T}^T = \mathbf{T}$.) Hence we may recover $\mathbf{x}$ from $\mathbf{y}$ using:

$$\begin{bmatrix} x(1) \\ x(2) \end{bmatrix} = \mathbf{T}^T \begin{bmatrix} y(1) \\ y(2) \end{bmatrix} \tag{3}$$

In 2-dimensions $\mathbf{x}$ and $\mathbf{y}$ become 2×2 matrices. We may transform first the columns of $\mathbf{x}$, by premultiplying by $\mathbf{T}$, and then the rows of the result by postmultiplying by $\mathbf{T}^T$. Hence:

$$\mathbf{y} = \mathbf{T}\ \mathbf{x}\ \mathbf{T}^T \qquad \text{and to invert:} \qquad \mathbf{x} = \mathbf{T}^T\ \mathbf{y}\ \mathbf{T} \tag{4}$$

To show more clearly what is happening:

$$\text{If } \mathbf{x} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \text{then } \mathbf{y} = \frac{1}{2}\begin{bmatrix} a+b+c+d & a-b+c-d \\ a+b-c-d & a-b-c+d \end{bmatrix}$$

These operations correspond to the following filtering processes:

Top left: $a+b+c+d$ = 4-point average or 2-D lowpass filter. It is separable: $H(z_1, z_2) = (z_1^{-1} + 1)(z_2^{-1} + 1)$, Low pass filter along rows then low pass filter along columns. It is called a Lo-Lo filter.

Top right: $a-b+c-d$ = Average horizontal gradient. Again separable: $H(z_1, z_2) = (1+z_1^{-1})(1-z_2^{-1})$, Average along columns and difference (gradient) along rows. Horizontal highpass and vertical lowpass (Hi-Lo) filter.

Lower left: $a+b-c-d = a-c+b-d$. Average vertical gradient or horizontal lowpass and vertical highpass (Lo-Hi) filter.

Lower right: $a-b-c+d = a-b-(c-d)$ Diagonal curvature or 2-D highpass (Hi-Hi) filter.

To apply this transform to a complete image, the pels are grouped into 2×2 blocks and apply (4) to each block. The result (after reordering) is shown in figure 2. To view the result sensibly, all the top left components of the 2×2 blocks in $\mathbf{y}$ have been grouped together to form the top left subimage in fig 2. The same has been for the components in the other 3 positions to form the corresponding other 3 subimages.

Fig 2 shows that most of the energy is contained in the top left (Lo-Lo) subimage and the least energy is in the lower right (Hi-Hi) subimage. Note how the top right (Hi-Lo) subimage contains the near-vertical edges and the lower left (Lo-Hi) subimage contains the near-horizontal edges.

The energies of the subimages and their percentages of the total are:

Lo-Lo:	**Hi-Lo:**
810×10^6	0.56×10^6
99.1%	0.56%

Lo-Hi:	**Hi-Hi:**
1.89×10^6	$.82 \times 10^6$
0.23%	0.1%

Total energy in fig 2 = 817×10^6

The table shows that a significant compression of energy into the Lo-Lo subimage has been achieved. However the energy measurements do not directly indicate how much data compression this gives.

A much more useful measure than energy is the *entropy* of the subimages after a given amount of quantisation. This gives the minimum number of bits per pel needed to represent the quantised data for each subimage, to a given accuracy, assuming that an ideal entropy code is used. By comparing the total entropy of the 4 subimages with that of the original image, an estimate can be made of the compression that one level of the Haar transform can provide.

4 Entropy

Entropy of source information was discussed in Peter Cullen's Digital Communications course. For an image $\mathbf{I}$, quantised to M levels, the entropy $H_{\mathbf{I}}$ is defined as:

$$H_{\mathbf{I}} = \sum_{i=0}^{M-1} p_n \log_2 \left(\frac{1}{p_n} \right) = - \sum_{n=0}^{M-1} p_n \log_2(p_n) \tag{5}$$

where p_n, $n = 0$ to $M-1$, is the probability of the n^{th} quantiser level being used (often obtained from a histogram of the pel intensities).

$H_{\mathbf{I}}$ represents the mean number of bits per pel with which the quantised image $\mathbf{I}$ can be represented using an ideal variable-length entropy code. A Huffman code usually approximates this bit-rate quite closely.

To obtain the number of bits to code an image (or subimage) $\mathbf{I}$ containing N pels:

- A histogram of $\mathbf{x}$ is measured using M bins corresponding to the M quantiser levels.
- The M histogram counts are each divided by N to give the probabilities p_n, which are then converted into entropies $h_n = -p_n \log_2(p_n)$. This conversion law is illustrated in fig 3 and shows that probabilities close to zero or one produce low entropy and intermediate values produce entropies near 0.5.
- The entropies h_n of the separate quantiser levels are summed to give the total entropy $H_{\mathbf{I}}$ for the subimage.
- Multiplying $H_{\mathbf{I}}$ by N gives the estimated total number of bits needed to code $\mathbf{I}$, assuming an ideal entropy code is available which is matched to the histogram of $\mathbf{I}$.

Fig 3 shows the probabilities p_n and entropies h_n for the original Lenna image and fig 4 shows these for each of the four subimages in 2, assuming a uniform quantiser with a step-size $Q_{step} = 15$ in each case. The original Lenna image contained pel values from 3 to 238 and a mean level of 120 was subtracted from each pel value before the image was analysed or transformed in order that all samples would be approximately evenly distributed about zero (a natural feature of highpass subimages).

The Haar transform preserves energy and so the expected distortion energy from quantising the transformed image $\mathbf{y}$ with a given step size Q_{step} will be approximately the same as that from quantising the input image $\mathbf{x}$ with the same step size. This is because quantising errors can usually be modelled as independent random processes with variance (energy) $= Q_{step}^2/12$ and the total squared quantising error (distortion) will tend to the sum of the variances over all pels. This applies whether the error energies are summed before or after the inverse transform (reconstruction) in the decoder.

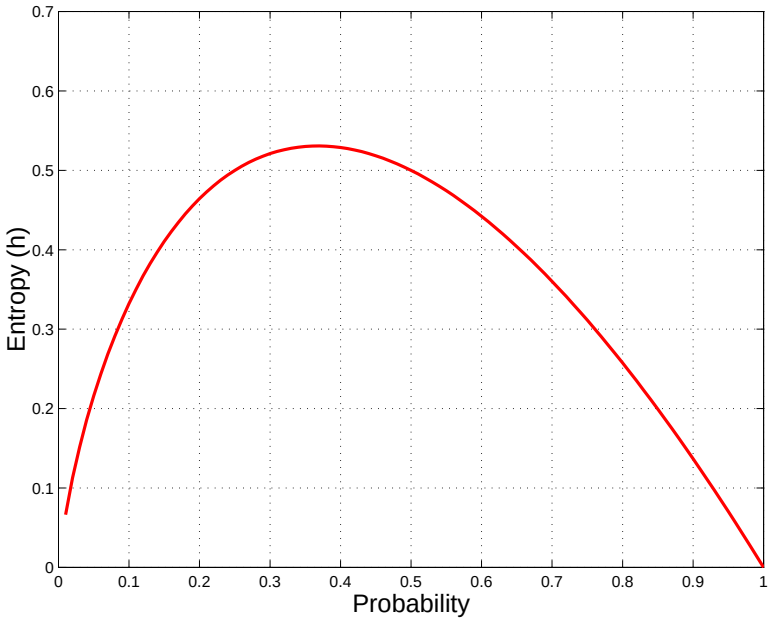

Figure 3: Conversion from probability p_n to entropy $h_n = -p_n \log_2(p_n)$.

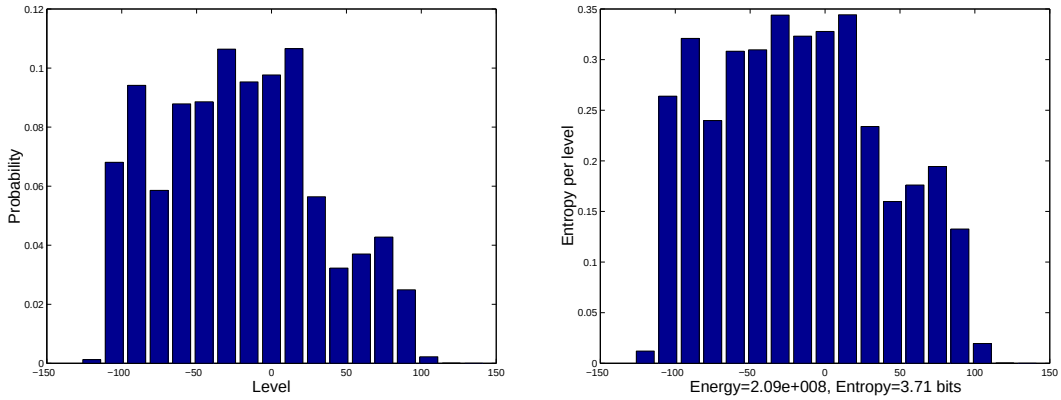

Figure 4: Probability histogram (left) and entropies (right) of the Lenna image

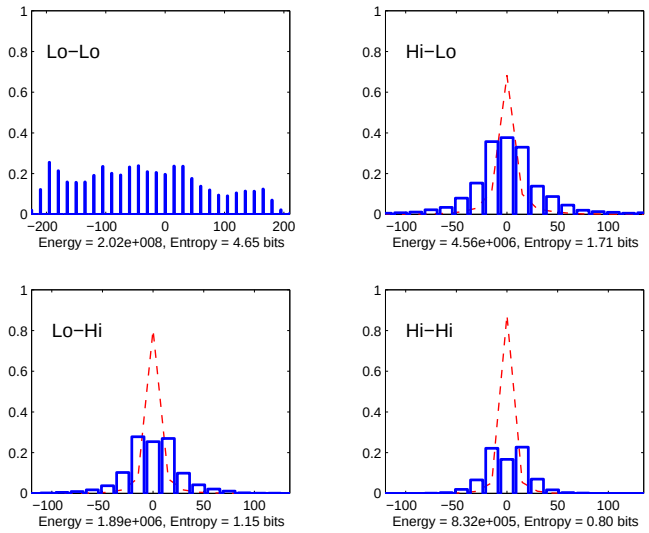

Figure 5: Probability histograms (blue) and entropies (dashed) of the four subimages of the Level 1 Haar transform of Lenna. **Energies are those of the original, 8 bit Lenna image.**

Hence equal quantiser step sizes before and after an energy-preserving transformation should generate equivalent quantising distortions and provide a fair estimate of the compression achieved by the transformation.

The first two columns of Fig 6 (*original* and *level 1*) compare the entropy (mean bit rate) per pel for the original image (3.71 bit / pel) with that of the Haar transformed image of fig 2 (2.08 bit / pel), using $Q_{step} = 15$. Notice that the entropy of the original image is almost as great as the 4 bit / pel that would be needed to code the 16 levels using a simple fixed-length code, because the histogram is relatively uniform.

The level 1 column of fig 6 shows the contributions of each of the four subimages of fig 2 to the total entropy per pel. Recall that the entropy is the **mean** number of bits per pel that would be required to code the image. Therefore, in order to make a fair comparison between the level 1 Haar decomposition and the original image entropy, the entropies from fig 2 have been divided by 4 since each subimage has one quarter of the total number of pels. The Lo-Lo subimage contributes 56% to the total entropy (bit rate) and has similar spatial correlations to the original image. Hence it is a logical step to apply the Haar transform again to this subimage.

5 The Multi-level Haar Transform

Figure 7 shows the result of applying the Haar transform to the Lo-Lo subimage of fig 2 (giving the level 2 subimages) and fig 10 shows the probabilities p_i and entropies h_i for the 4 new subimages.

The level 2 column of fig 6 shows how the total bit rate can be reduced by transforming the level 1 Lo-Lo subimage into four level 2 subimages. The process can be repeated by transforming

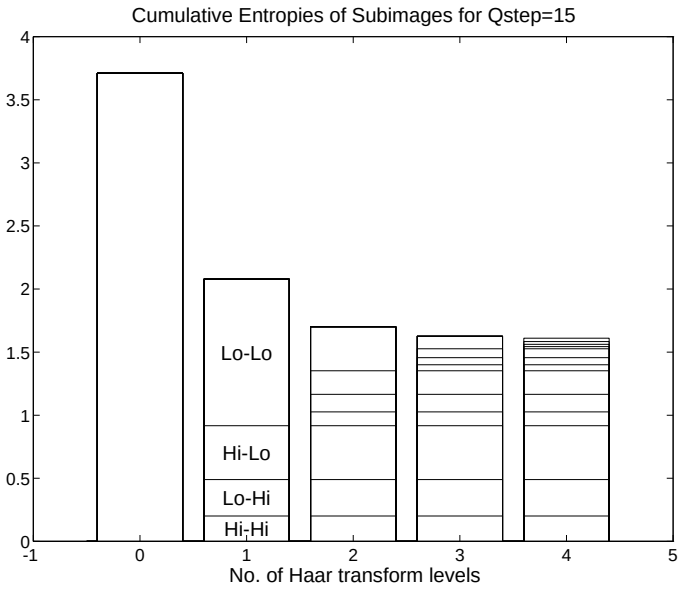

Figure 6: Mean bit rate for the original Lenna image and for the Haar transforms of the image after 1 to 4 levels, using a quantiser step size $Q_{step} = 15$.

the final Lo-Lo subimage again and again, giving the subimages in figs 8, 9 and the histograms in figs 11 and 12. The levels 3 and 4 columns of fig 2.6 show that little is gained by transforming to more than 4 levels.

However a total compression ratio of 4 bit/pel : 1.61 bit/pel = 2.45 : 1 has been achieved (in theory).

Note that as further levels of the Haar Xform are generated, the *range* of the pixel data will change. Therefore, although the quantisation step will remain the same, the number of levels that may need to be used to represent the image will differ from subimage to subimage.

Further, note that in all histogram/entropy figures shown *after* figure 5 the histograms do NOT show the *actual* intensity of the quantised image along the horizontal axis. The indexes along the horizontal axis just refer to the quantisation step index only.

Finally, the energies shown are always energies *before* quantisation.

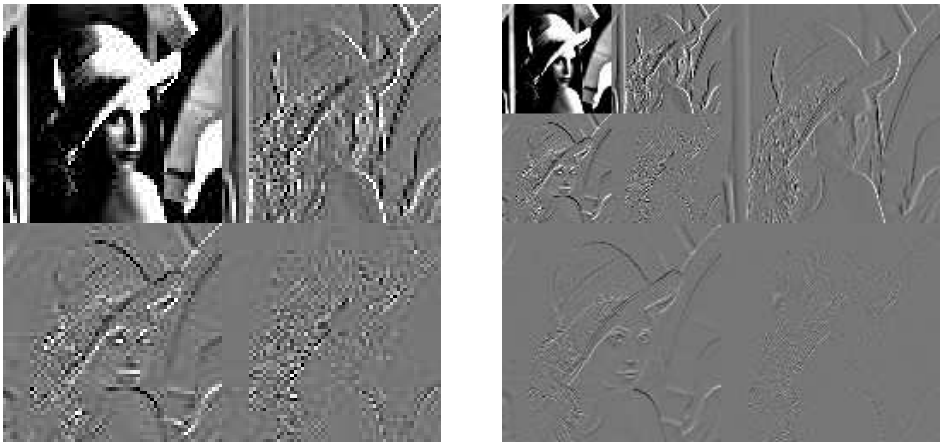

Figure 7: Left: Level 2 Haar Transform of Lenna. Right: Level 1 and 2 Xforms shown together.

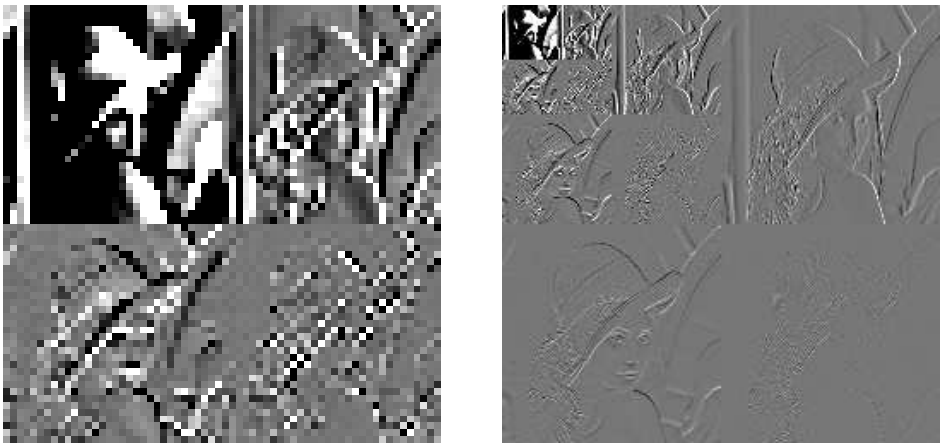

Figure 8: Left: Level 3 Haar Transform of Lenna. Right: Level 1, 2 and 3 Xforms shown together.

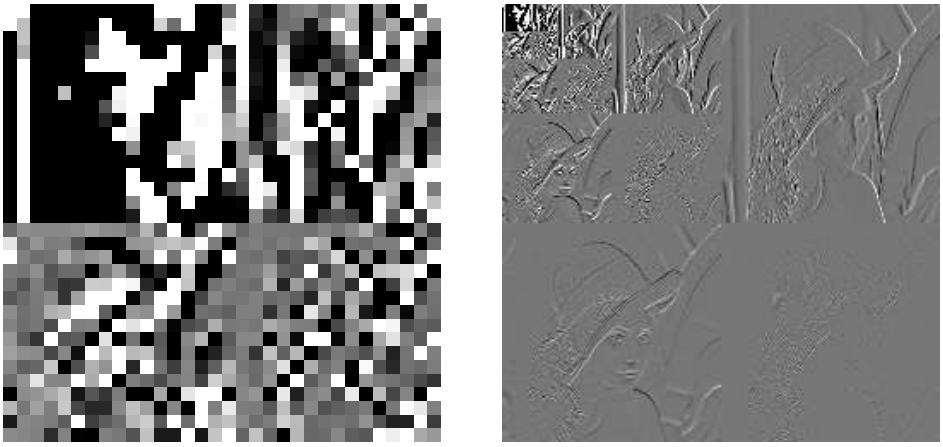

Figure 9: Left: Level 4 Haar Transform of Lenna. Right: Level 1,2,3,4 Xforms shown together.

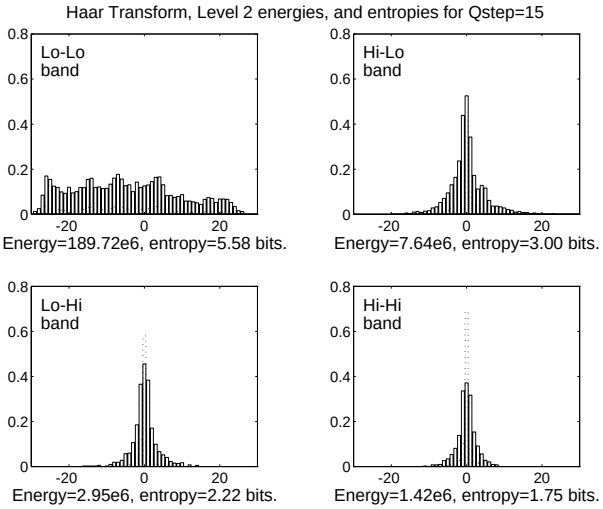

Figure 10: The probabilities p_i and entropies h_i for the 4 subimages at level 2.

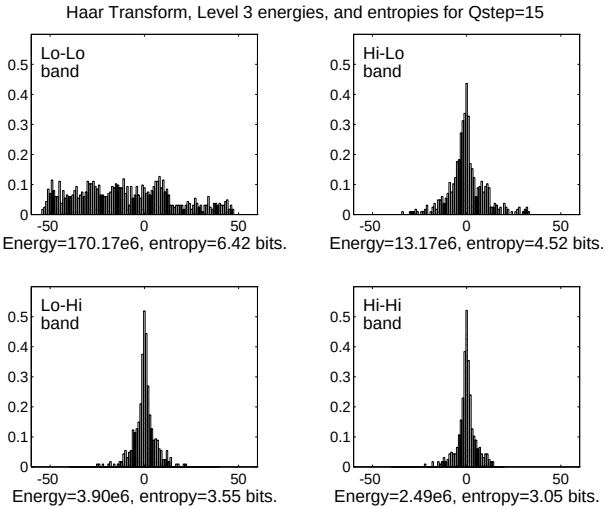

Figure 11: The probabilities p_i and entropies h_i for the 4 subimages at level 3.

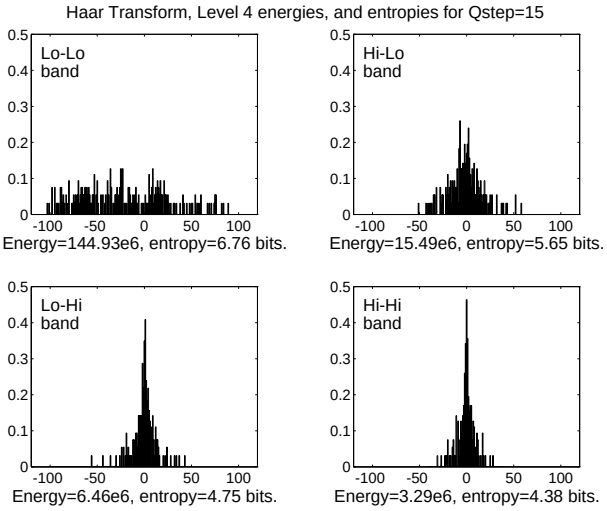

Figure 12: The probabilities p_i and entropies h_i for the 4 subimages at level 4.

Figure 13: Images reconstructed from (left) the original Lenna, and (right) the 4-level Haar transform, each quantised with $Q_{step} = 15$. The rms error of (left) = 4.3513, and of (right) = 3.5343.

Note the following features of the 4-level Haar transform:

- Figs 7 to 9 show the subimages from all 4 levels of the transform and illustrate the transform's *multi-scale* nature. The set of figures also shows that all the subimages occupy the same total area as the original and hence that the total number of transform output samples (coefficients) equals the number of input pels – there is *no redundancy*.
- From the Lo-Lo subimage histograms of figs 5, 10,11 and 12, we see the magnitudes of the Lo-Lo subimage samples increasing with transform level. This is because energy is being conserved and most of it is being concentrated in fewer and fewer Lo-Lo samples. (The DC gain of the Lo-Lo filter of equation (4) is 2.)
- We may reconstruct the image from the transform samples (fig 9), quantised to $Q_{step} = 15$, by inverting the transform, using the right-hand part of (4). We then get the image in fig 13 (right). Contrast this with 13 (left), obtained by quantising the pels of the original directly to $Q_{step} = 15$, in which contour artifacts are much more visible. Thus the transform provides improved subjective quality as well as significant data compression. The improved quality arises mainly from the high amplitude of the low frequency transform samples, which means that they are quantised to many more levels than the basic pels would be for a given Q_{step}.
- If Q_{step} is doubled to 30, then the entropies of all the subimages are reduced as shown in fig 14 (compare this with fig 6 in which $Q_{step} = 15$). The mean bit rate with the 4-level Haar transform drops from 1.61 to 0.97 bit/pel. However the reconstructed image quality drops to that shown in fig 15 (right). For comparison, fig 15 (left) shows the quality if $\mathbf{x}$ is directly quantised with $Q_{step} = 30$.

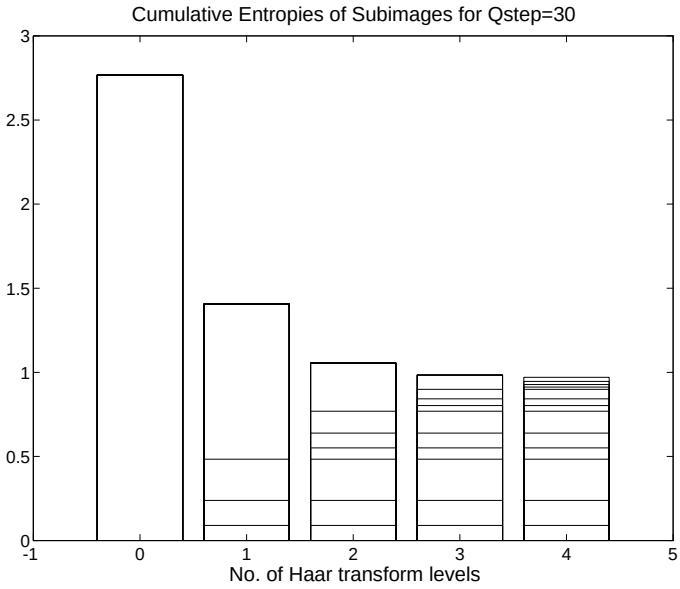

Figure 14: Mean bit rate for the original Lenna image and for the Haar transforms of the image after 1 to 4 levels, using a quantiser step size $Q_{step} = 30$.

Figure 15: Images reconstructed from (left) the original Lenna, and (right) the 4-level Haar transform, each quantised with $Q_{step} = 30$. The rms error of (left) = 8.6219, and of (right) = 5.8781.

5.1 Use of Laplacian PDFs in Image Compression

It is found to be appropriate and convenient to model the distribution of many types of transformed image coefficients by Laplacian distributions. It is appropriate because much real data is approximately modelled by the Laplacian probability density function (PDF), and it is convenient because the mathematical form of the Laplacian PDF is simple enough to allow some useful analytical results to be derived.

A Laplacian PDF is a back-to-back pair of exponential decays and is given by:

$$p(x) = \frac{1}{2x_0}\ e^{-|x|/x_0} \tag{6}$$

where x_0 is the equivalent of a *time constant* which defines the *width* of the PDF from the centre to the $1/e$ points. The initial scaling factor ensures that the area under $p(x)$ is unity, so that it is a valid PDF. Fig 16 shows the shape of $p(x)$.

The mean of this PDF is zero and the variance is given by:

$$v(x_0) = \int_{-\infty}^{\infty} x^2\ p(x)\ dx = 2\int_0^{\infty} \frac{x^2}{2x_0}\ e^{-x/x_0}\ dx = 2x_0^2 \tag{7}$$

(using integration by parts twice).

Hence the standard deviation is:

$$\sigma(x_0) = \sqrt{v(x_0)} = \sqrt{2}\ x_0 \tag{8}$$

Given the variance (power) of a subimage of transformed pels, we may calculate x_0 and hence determine the PDF of the subimage, assuming a Laplacian shape. We now show that, if we quantise the subimage using a uniform quantiser with step size Q, we can calculate the entropy of the quantised samples and thus estimate the bit rate needed to encode the subimage in bits/pel. This is a powerful analytical tool as it shows how the compressed bit rate relates directly to the energy of a subimage. The vertical dashed lines in fig 16 show the decision thresholds for a typical quantiser for the case when $Q = 2x_0$.

First we analyse the probability of a pel being quantised to each step of the quantiser. This is given by the area under $p(x)$ between each adjacent pair of quantiser thresholds.

Probability of being at step 0, $p_0 = \text{prob}(-\frac{1}{2}Q < x < \frac{1}{2}Q) = 2\ \text{prob}(0 < x < \frac{1}{2}Q)$

Probability of being at step k, $p_k = \text{prob}((k - \frac{1}{2})Q < x < (k + \frac{1}{2})Q)$

First, for $x_2 \geq x_1 \geq 0$, we calculate:

$$\text{prob}(x_1 < x < x_2) = \int_{x_1}^{x_2} p(x)\ dx = \left[-\frac{1}{2}\ e^{-x/x_0}\right]_{x_1}^{x_2} = \frac{1}{2}(e^{-x_1/x_0} - e^{-x_2/x_0})$$

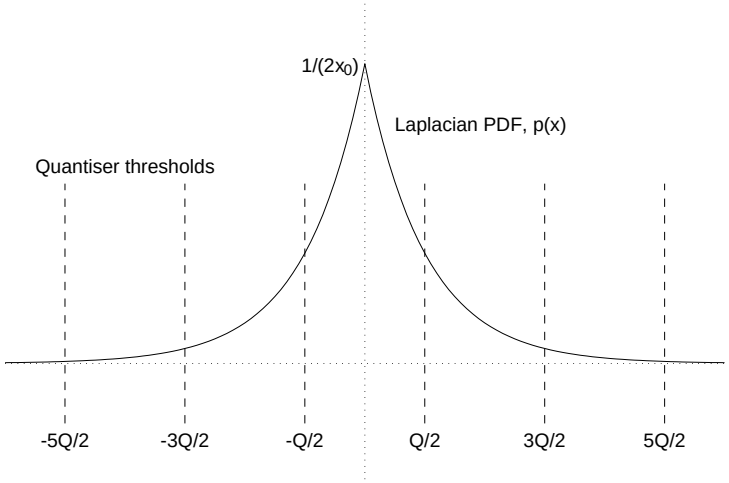

Figure 16: Laplacian PDF, $p(x)$, and typical quantiser decision thresholds, shown for the case when the quantiser step size $Q = 2x_0$.

$$\therefore\ p_0 = (1 - e^{-Q/(2x_0)}) \tag{9}$$

$$\text{and, for } k \geq 1, p_k = \frac{1}{2}(e^{-(k-\frac{1}{2})Q/x_0} - e^{-(k+\frac{1}{2})Q/x_0}) = \sinh\left(\frac{Q}{2x_0}\right)\ e^{-kQ/x_0} \tag{10}$$

By symmetry, if k is nonzero, $p_{-k} = p_k = \sinh\left(\frac{Q}{2x_0}\right)\ e^{-|k|Q/x_0}$

Now we can calculate the entropy of the subimage:

$$H = -\sum_{k=-\infty}^{\infty} p_k\ \log_2 p_k = -p_0\ \log_2 p_0 - 2\sum_{k=1}^{\infty} p_k\ \log_2 p_k \tag{11}$$

To make the evaluation of the summation easier when we substitute for p_k, we let

$$p_k = \alpha r^k \quad \text{where}\ \ \alpha = \sinh\left(\frac{Q}{2x_0}\right)\ \ \text{and}\ \ r = e^{-Q/x_0}$$

$$\therefore\ \sum_{k=1}^{\infty} p_k\ \log_2 p_k = \sum_{k=1}^{\infty} \alpha r^k\ \log_2(\alpha r^k) = \sum_{k=1}^{\infty} \alpha r^k(\log_2 \alpha + k\ \log_2 r)$$

$$= \alpha\ (\log_2 \alpha)\sum_{k=1}^{\infty} r^k + \alpha\ (\log_2 r)\sum_{k=1}^{\infty} k\ r^k$$

$$\text{Now}\ \sum_{k=1}^{\infty} r^k = \frac{r}{1-r} \qquad \text{and, differentiating by } r:\ \sum_{k=1}^{\infty} k\ r^{k-1} = \frac{1}{(1-r)^2}$$

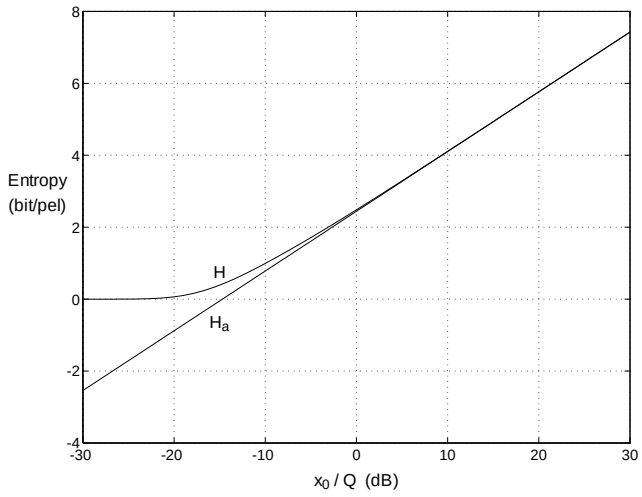

Figure 17: Entropy H and approximate entropy H_a of a quantised subimage with Laplacian PDF, as a function of x_0/Q in dB.

$$\begin{aligned}
\therefore \sum_{k=1}^{\infty} p_k \log_2 p_k &= \alpha \, (\log_2 \alpha) \, \frac{r}{1-r} + \alpha \, (\log_2 r) \, \frac{r}{(1-r)^2} \\
&= \frac{\alpha r}{1-r} \left[\log_2 \alpha + \frac{\log_2 r}{1-r} \right] \\
\text{and} \quad p_0 \log_2 p_0 &= (1-\sqrt{r}) \, \log_2 (1-\sqrt{r})
\end{aligned}$$

Hence the entropy is given by:

$$H = -(1-\sqrt{r}) \, \log_2 (1-\sqrt{r}) - \frac{2\alpha r}{1-r} \left[\log_2 \alpha + \frac{\log_2 r}{1-r} \right] \tag{12}$$

Because both α and r are functions of Q/x_0, then H is a function of just Q/x_0 too. We expect that, for constant Q, as the energy of the subimage increases, the entropy will also increase approximately logarithmically, so we plot H against x_0/Q in dB in fig 17. This shows that our expectations are born out.

We can show this in theory by considering the case when $x_0/Q \gg 1$, when we find that:

$$\alpha \approx \frac{Q}{2x_0} \qquad r \approx \left(1 - \frac{Q}{x_0}\right) \approx 1 - 2\alpha \qquad \sqrt{r} \approx 1 - \alpha$$

Using the approximation $\log_2(1-\epsilon) \approx -\epsilon / \log_e 2$ for small ϵ, it is then fairly straightforward to show that

$$H \approx -\log_2 \alpha + \frac{1}{\log_e 2} \approx \log_2 \left(\frac{2e \; x_0}{Q} \right)$$

We denote this approximation as H_a in fig 17, which shows how close to H the approximation is, for $x_0 > Q$ (i.e. for $x_0/Q > 0$ dB).

We can compare the entropies calculated using equation (12) with those that were calculated from the bandpass subimage histograms, as given in figs 2.5, 2.8, 2.9 and 2.10. (The Lo-Lo subimages have PDFs which are more uniform and do not fit the Laplacian model well.) The values of x_0 are calculated from:

$$x_0 = \frac{\text{std. dev.}}{\sqrt{2}} = \sqrt{\frac{\text{subimage energy}}{2\ (\text{no of pels in subimage})}}$$

The following table shows this comparison:

Transform level	Subimage type	Energy ($\times 10^6$)	No of pels	x_0	Laplacian entropy	Measured entropy
1	Hi-Lo	4.56	16384	11.80	2.16	1.71
1	Lo-Hi	1.89	16384	7.59	1.58	1.15
1	Hi-Hi	0.82	16384	5.09	1.08	0.80
2	Hi-Lo	7.64	4096	30.54	3.48	3.00
2	Lo-Hi	2.95	4096	18.98	2.81	2.22
2	Hi-Hi	1.42	4096	13.17	2.31	1.75
3	Hi-Lo	13.17	1024	80.19	4.86	4.52
3	Lo-Hi	3.90	1024	43.64	3.99	3.55
3	Hi-Hi	2.49	1024	34.87	3.67	3.05
4	Hi-Lo	15.49	256	173.9	5.98	5.65
4	Lo-Hi	6.46	256	112.3	5.35	4.75
4	Hi-Hi	3.29	256	80.2	4.86	4.38

We see that the entropies calculated from the energy via the Laplacian PDF method (second column from the right) are approximately 0.5 bit/pel greater than the entropies measured from the Lenna subimage histograms. This is due to the heavier tails of the actual PDFs compared with the Laplacian exponentially decreasing tails. More accurate entropies can be obtained if x_0 is obtained from the mean absolute values of the pels in each subimage. For a Laplacian PDF we can show that

$$\text{Mean absolute value} = \int_{-\infty}^{\infty} |x|\ p(x)\ dx = 2\int_{0}^{\infty} \frac{x}{2x_0}\ e^{-x/x_0}\ dx = x_0 \tag{13}$$

This gives values of x_0 that are about 20% lower than those calculated from the energies and the calculated entropies are then within approximately 0.2 bit/pel of the measured entropies.

6 Practical Entropy Coding Techniques

In the previous sections we have assumed that ideal entropy coding has been used in order to calculate the bit rates for the coded data. In practise we must use real codes and we shall now see how this affects the compression performance.

There are three main techniques for achieving entropy coding:

Huffman Coding – one of the simplest variable length coding schemes.

Run-length Coding (RLC) – very useful for binary data containing long runs of ones or zeros.

Arithmetic Coding – a relatively new variable length coding scheme that can combine the best features of Huffman and run-length coding, and also adapt to data with non-stationary statistics.

We shall concentrate on the Huffman and RLC methods for simplicity. Interested readers may find out more about Arithmetic Coding in chapters 12 and 13 of the JPEG Book.

First we consider the change in compression performance if simple Huffman Coding is used to code the subimages of the 4-level Haar transform.

The calculation of entropy in equation (5) assumed that each message with probability p_i could be represented by a word of length $w_i = -\log_2(p_i)$ bits. Huffman codes require the w_i to be integers and assume that the p_i are adjusted to become:

$$\hat{p}_i = 2^{-w_i} \tag{14}$$

where the w_i are integers, chosen subject to the constraint that $\sum \hat{p}_i \leq 1$ (to guarantee that sufficient uniquely decodable code words are available) and such that the mean Huffman word length (Huffman entropy), $\hat{H} = \sum p_i w_i$, is minimised.

We can use the probability histograms which generated the entropy plots of figs 5, 10, 11 and 12 to calculate the Huffman entropies $\hat{H}$ for each subimage and compare these with the true entropies to see the loss in performance caused by using real Huffman codes.

An algorithm for finding the optimum codesizes w_i is recommended in the JPEG specification [The JPEG Book, Appendix A, Annex K.2, fig K.1]; and a Matlab M-file to implement it is given in fig 21.

Fig 18 shows the results of applying this algorithm to the probability histograms and fig 19 lists the same results numerically for ease of analysis. Columns 1 and 2 compare the ideal entropy with the mean word length or bit rate from using a Huffman code (the Huffman entropy) for the case of the untransformed image where the original pels are quantised with $Q_{step} = 15$. We see that the increase in bit rate from using the real code is:

$$\frac{3.7676}{3.7106} - 1 = 1.5\%$$

But when we do the same for the 4-level transformed subimages, we get columns 3 and 4. Here we see that real Huffman codes require an increase in bit rate of:

$$\frac{1.8425}{1.6103} - 1 = 14.4\%$$

Comparing the results for each subimage in columns 3 and 4, we see that most of the increase in bit rate arises in the three level-1 subimages at the bottom of the columns. This is because each of the probability histograms for these subimages (fig 5) contain one probability that is greater than 0.5. Huffman codes cannot allocate a word length of less than 1 bit to a given event, and so they start to lose efficiency rapidly when $-\log_2(p_i)$ becomes less than 1, ie when $p_i > 0.5$.

Run-length codes (RLCs) are a simple and effective way of improving the efficiency of Huffman coding when one event is much more probable than all of the others combined. They operate as follows:

- The pels of the subimage are scanned sequentially (usually in columns or rows) to form a long 1-dimensional vector.
- Each run of consecutive zero samples (the most probable events) in the vector is coded as a single event.
- Each non-zero sample is coded as a single event in the normal way.
- The two types of event (runs-of-zeros and non-zero samples) are allocated separate sets of codewords in the same Huffman code, which may be designed from a histogram showing the frequencies of all events.
- To limit the number of run events, the maximum run length may be limited to a certain value (we have used 128) and runs longer than this may be represented by two or more run codes in sequence, with negligible loss of efficiency.

Hence RLC may be added before Huffman coding as an extra processing step, which converts the most probable event into many separate events, each of which has $p_i < 0.5$ and may therefore be coded efficiently. Fig 20 shows the new probability histograms and entropies for level 1 of the Haar transform when RLC is applied to the zero event of the three bandpass subimages. Comparing this

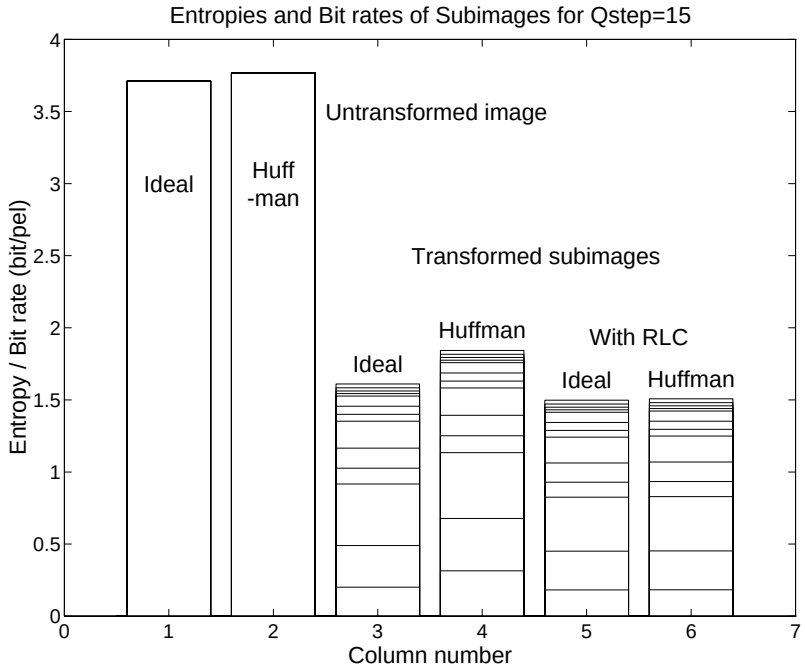

Figure 18: Comparison of entropies (columns 1,3,5) and Huffman coded bit rates (columns 2,4,6) for the original (columns 1 and 2) and transformed (columns 3 to 6) Lenna images. In columns 5 and 6, the zero amplitude state is run-length encoded to produce many states with probabilities < 0.5.

Column:	1	2	3	4	5	6	_
			0.0264	0.0265	0.0264	0.0266	
			0.0220	0.0222	0.0221	0.0221	Level 4
			0.0186	0.0187	0.0185	0.0186	
			0.0171	0.0172	0.0171	0.0173	_
			0.0706	0.0713	0.0701	0.0705	
			0.0556	0.0561	0.0557	0.0560	Level 3
	3.7106	3.7676	0.0476	0.0482	0.0466	0.0471	_
			0.1872	0.1897	0.1785	0.1796	
			0.1389	0.1413	0.1340	0.1353	Level 2
			0.1096	0.1170	0.1038	0.1048	_
			0.4269	0.4566	0.3739	0.3762	
			0.2886	0.3634	0.2691	0.2702	Level 1
			0.2012	0.3143	0.1819	0.1828	_
Totals:	3.7106	3.7676	1.6103	1.8425	1.4977	1.5071	

Figure 19: Numerical results used in figure above.

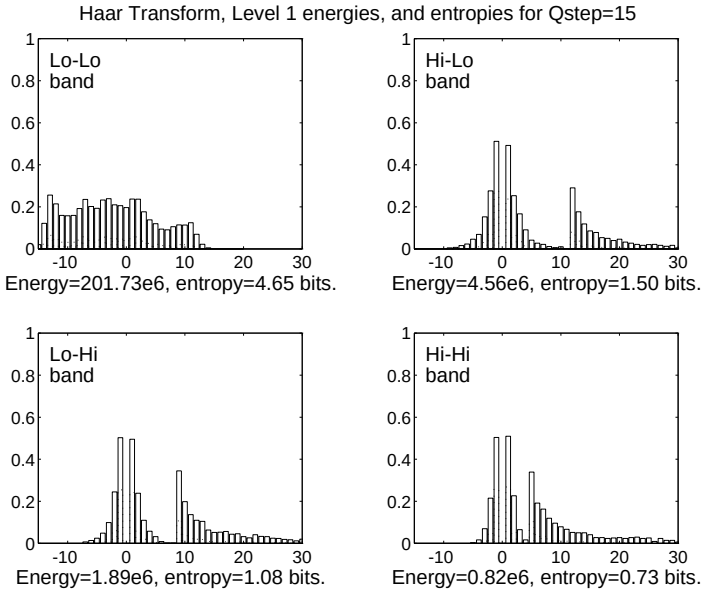

Figure 20: Probability histograms (dashed) and entropies (solid) of the four subimages of the Level 1 Haar transform of Lenna (fig 2) after RLC.

with fig 5, note the absence of the high probability zero events and the new states to the right of the original histograms corresponding to the run lengths.

The total entropy *per event* for an RLC subimage is calculated as before from the entropy histogram. However to get the entropy *per pel* we scale the entropy by the ratio of the number of events (runs and non-zero samples) in the subimage to the number of pels in the subimage (note that with RLC this ratio will no longer equal one – it will hopefully be much less).

Fig 20 gives the entropies per pel after RLC for each subimage, which are now *less* than the entropies in fig 5. This is because RLC takes advantage of spatial clustering of the zero samples in a subimage, rather than just depending on the histogram of amplitudes.

Clearly if all the zeros were clustered into a single run, this could be coded much more efficiently than if they are distributed into many runs. The entropy of the zero event tells us the mean number of bits to code each zero pel *if the zero pels are distributed randomly*, ie if the probability of a given pel being zero does not depend on the amplitudes of any nearby pels.

In typical bandpass subimages, non-zero samples tend to be clustered around key features such as object boundaries and areas of high texture. Hence RLC usually reduces the entropy of the data to be coded. There are many other ways to take advantage of clustering (correlation) of the data – RLC is just one of the simplest.

In fig 18, comparing column 5 with column 3, we see the modest (7%) reduction in entropy per pel achieved by RLC, due clustering in the Lenna image. The main advantage of RLC is apparent in column 6, which shows the mean bit rate per pel when we use a real Huffman code on the RLC

histograms of fig 20. The increase in bit rate over the RLC entropy is only

$$\frac{1.5071}{1.4977} - 1 = 0.63\%$$

compared with 14.4% when RLC is not used (columns 3 and 4).

Finally, comparing column 6 with column 3, we see that, relative to the simple entropy measure, combined RLC and Huffman coding can *reduce* the bit rate by

$$1 - \frac{1.5071}{1.6103} = 6.4\%$$

The closeness of this ratio to unity justifies our use of simple entropy as a tool for assessing the information compression properties of the Haar transform – and of other energy compression techniques as we meet them.

7 Summary and Reference material

This handout has covered an introduction to the basic elements of an image compression system. The use of a simple multi scale transform was explored, the Haar Transform. The basic operation of the Haar Transform was seen to be the application of a limited set of separable filters. The effect of Huffman and RLC encoding on the compression factor was also examined. A relationship between entropy and energy was observed.

For reference material see the following (all from Lim)

- Entropy and variable length codes: pages 612–616
- Huffman coding : pages 614-616. (Note that Arithmetic methods *can* approach more closely the Entropy limit than Huffman codes, but these are more 'recent' coding schemes).
- Transform image coding : pages 642–652

```
% Find Huffman code sizes: JPEG fig K.1, procedure Code_size.
% huffhist contains the histogram of event counts (frequencies).
freq = huffhist(:);
codesize = zeros(size(freq));
others = -ones(size(freq)); % Pointers to next symbols in code tree.

% Find non-zero entries in freq, and loop until only 1 entry left.
nz = find(freq > 0);
while length(nz) > 1,
% Find v1 for least value of freq(v1) > 0.
  [y,i] = min(freq(nz));
  v1 = nz(i);
% Find v2 for next least value of freq(v2) > 0.
  nz = nz([1:(i-1)  (i+1):length(nz)]);  % Remove v1 from nz.
  [y,i] = min(freq(nz));
  v2 = nz(i);
% Combine frequency values.
  freq(v1) = freq(v1) + freq(v2);
  freq(v2) = 0;
  codesize(v1) = codesize(v1) + 1;
% Increment code sizes for all codewords in this tree branch.
  while others(v1) > -1,
    v1 = others(v1);
    codesize(v1) = codesize(v1) + 1;
  end
  others(v1) = v2;
  codesize(v2) = codesize(v2) + 1;
  while others(v2) > -1,
    v2 = others(v2);
    codesize(v2) = codesize(v2) + 1;
  end
  nz = find(freq > 0);
end

% Generate Huffman entropies by multiplying probabilities by code sizes.
huffent = (huffhist(:)/sum(huffhist(:))) .* codesize;
```

Figure 21: Listing of the M-file to calculate the Huffman entropy from a given histogram.

A Huffman Coding

The basic requirement to effect communication between a sender and receiver of digital data is to agree on some alphabet for communiations. In effect this means assigning a symbol or a number to each type of data to be sent. With datum like audio and image samples they typically appear as 16bit or 8bit numbers respectively. Say we want to transmit some image data. The simplest idea is to say right, we have 8bits per pixel so each pixel can be repesented as a number between 0 and 255, hence my datum are 256 numbers starting with 0 and ending with 255. So I will always need an 8 bit number to reprsenet a pixel.

But thinking about it a little more allows us to realise that if someone told us that not all the values between 0 and 255 are equally likely then I can be on to a much better scheme. If pixel values that occur most often are assigned short *codewords* or *symbols* and those that occur less often are assigned long codewords or symbols, then on average your bitstreams may be way shorter for any typical signal. This is because the longer codewords need more bits, but you will be using this relatively rarely. This is called *Variable Length Encoding*. Because of VLB synchronisation of the decoder with the start and end of codewords is a major issue. Thus if an error occurs, one codeword could suddenly become another. VLB is great but tricky to use in a practical system. ALL compression schemes in use today for natural media (video and audio) use VLB schemes.

Huffman encoding is a very clever algorithm for assigning codewords to datum depending on their probabilty in the signal. So the most likely datum is assigned the shortest symbol, e.g. 0 or 1 and the next likely is assigned 10 , and so on. The scheme is therefore roughly as follows. The best way to understand it is to do an example. Consider that there are 6 messages to be transmitted a_k, $k = 1 \ldots 6$ and the probability of each occuring is $p_k = (5/8), (3/32), (3/32), (1/32), (1/8), (1/32)$. The Huffman encoding algorithm is as follows.

1. Draw a column showing the messages in order vertically, $a_1, \ldots a_6$, with their asociated probabilities.

$$\begin{array}{ll} a_1 & \frac{5}{8} \\ a_2 & \frac{3}{32} \\ a_3 & \frac{3}{32} \\ a_4 & \frac{1}{32} \\ a_5 & \frac{1}{8} \\ a_6 & \frac{1}{32} \end{array}$$

2. Combine the two messages having the lowest two probabilities into one message having the sum combined probability. Write the new message set as a new column. (You may wish to rank the new set of messages in order from Maximum probabilities at the top of the column to the lowest one before writing this column). Draw a line linking the new messages with the

messages that they originated from in the previous column.

$$\begin{array}{ll} a_1 & \frac{5}{8} \\ a_2 & \frac{3}{32} \\ a_3 & \frac{3}{32} \\ a_4 & \frac{1}{32} \\ a_5 & \frac{1}{8} \\ a_6 & \frac{1}{32} \end{array} \qquad \begin{array}{l} \frac{5}{8} \\ \frac{3}{32} \\ \frac{3}{32} \\ \frac{1}{8} \\ a_7 = a_4 + a_6 = \frac{1}{16} \end{array}$$

3. Repeat this until you have just one message left of probability 1.0

$$\begin{array}{llllll} a_1 & \frac{5}{8} & \frac{5}{8} & \frac{5}{8} & \frac{5}{8} & \frac{5}{8} \\ a_2 & \frac{3}{32} & \frac{3}{32} & \frac{1}{8} & a_9 = \frac{1}{8} + \frac{3}{32} = \frac{7}{32} & a_{10} = \frac{7}{32} + \frac{5}{32} = \frac{12}{32} \\ a_3 & \frac{3}{32} & \frac{3}{32} & \frac{3}{32} & a_8 = \frac{5}{32} & \\ a_4 & \frac{1}{32} & \frac{1}{8} & a_8 = \frac{3}{32} + a_7 = \frac{5}{32} & & \\ a_5 & \frac{1}{8} & a_7 = a_4 + a_6 = \frac{1}{16} & & & \\ a_6 & \frac{1}{32} & & & & \end{array} \qquad 1.0$$

4. Assign to each of the links the symbols $0, 1$ starting from trhe last node and introducing uniwue 0/1Cat each link split as appropriate.

5. Now the Huffman code for each symbol is read out startig from the last node and reading off the 0/1 symbols until the root nodes are reached. Thus the code for a_6 is 1111, and the code for a_5 is 101 and so on. THe codes from $a_1 to a_6$ are as follows 0, 100, 110, 1110, 101, 1111. Note how the resulting Variable Length Code has assigned long words to low probability messages and short words to high probability messages.

The average bit rate can now be calculated for the transmission of this code. The expected number of bits used for transmitting message a_3 for instance is $p_3 n_3$ where n_3 is the number of bits used for coding the associated symbol a_3. In this case the code for a_3 is 110 which means 3 bits are needed to transmit it. Thus the expected number of bits used for sending a_3 is $(3/32) \times 3$. Thus the average number of bits used to transmit this data is $1 \times (5/8) + 3 \times (3/32) + 3 \times (3/32) + 4 \times (1/32) + 3 \times (1/8) + 4 \times (1/32) = 1.813$ bits/message.

In theory, the minimum number of bits/message that could possibly be achieved is given by the Entropy of the stream. We can calculate this as well using $H = (5/8) \times (5/8) + (3/32) \times (3/32) + (3/32) \times (3/32) + (1/32) \times (1/32) + (1/8) \times (1/8) + (1/32) \times (1/32) = 1.752$ bits/message.

Thus the Huffman code gets close to the theoretical limit. But not exactly dead on. This is because its not easy to code a message with a *fractional* number of bits. Recently a new set of codes caled TURBOcodes, have achieved this limit. An amazing advance that occurred only in the last 20 years.