An Introduction to Lossy Image Compression

4C8: Digital Media Processing

Ussher Assistant Professor François Pitié

2021/2022

Department of Electronic & Electrical Engineering , Trinity College Dublin

adapted from original material written by Prof. Anil Kokaram.

Overview

- Entropy Coding
- Haar Transform
- Quantisation

Entropy Coding

Entropy Coding

It all starts with the entropy.

The entropy of a random variable X with a probability mass function $p(X)$ is defined by

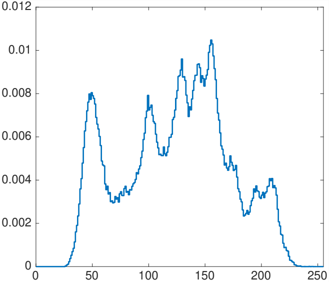
$$H(X) = -\sum_{x} p(x) \log_2 p(x)$$

Example

Suppose that we have a horse race with eight horses taking part. Assume that the probabilities of winning for the eight horses are $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64})$. We can calculate the entropy of the horse race as

$$\begin{aligned} H(X) &= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_4\frac{1}{2} - \frac{1}{8}\log_2\frac{1}{8} - \frac{1}{16}\log_2\frac{1}{16} - 4\frac{1}{64}\log_2\frac{1}{64} \\ &= 2 \text{ bits} \end{aligned}$$

Entropy of an Image



The entropy of Lenna is 7.57 bits/pixel.

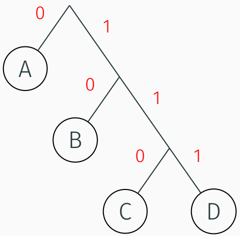
The maximum of the entropy could be 8 bits, so it doesn't look like much compression would be possible.

Huffman Coding

Huffman is the simplest entropy coding scheme. It achieves average code lengths no more than 1 bit/symbol of the entropy.

A binary tree is built by combining the two symbols with lowest probability into a dummy node.

The code length for each symbol is the number of branches between the root and respective leaf.



A:	$p(A) = 0.6$	code:	0
B:	$p(B) = 0.25$	code:	10
C:	$p(C) = 0.1$	code:	110
D:	$p(D) = 0.05$	code:	111

Huffman Coding of Lenna

Symbol	Code Length
0	42
1	42
2	41
3	17
4	14
...	...

The average code word length is

$$\sum_{k=0}^{255} p_k l_k = 7.59 \text{bits/pixel}$$

The code length is not much greater than the entropy.

Entropy Rate

Entropy is not the minimum average codeword length for a source with memory.

If the other pixel values are known we can predict the unknown pixel with much greater certainty and hence the effective (ie. conditional) entropy is much less.

The **Entropy Rate** is the minimum average codeword length for any source. It is defined as

$$H(X) = \lim_{n \to \infty} \frac{1}{n} H(X_1, X_2, \cdots, X_n)$$

It is the limit of the joint entropy for all pixel values taken in order.

It is very difficult to achieve codeword lengths close to the entropy rate. In fact it is difficult to calculate the entropy rate itself.

You looked at **LZW** in 3C5 as a practical coding algorithm. The average codeword length tends to the entropy rate if the file is large enough.

Efficiency is improved if we use Huffman to encode the output of LZW (see **Deflate**).

LZ algorithms used in lossless compression formats (eg. `.tiff`, `.png`, `.gif`, `.zip`, `.gz`, `.rar`, etc.)

Run-length encoding

It is a form of lossless data compression, where we indicate the number of times a symbol will be repeated. The encoded stream has the following form:

`[number of occurrences][symbol]...`

Example
aacaacabcabaaac

becomes

2a1c2a1c1a1b1c1a1b3a1c

It is a form of dictionary coding, where we point to previously decoded matches. The encoded stream has the following form:

```
(p, q, next symbol) ...
```

This tell the decoder that the next `q` symbols will be copied from previously decoded symbols at position `p` symbols back in the stream, and then insert new symbol `next symbol`.

Example
aacaacabcabaaac

becomes

(0,0,'a')(1,1,'c')(3,4,'b')(3,3,'a')(12,3,'$')

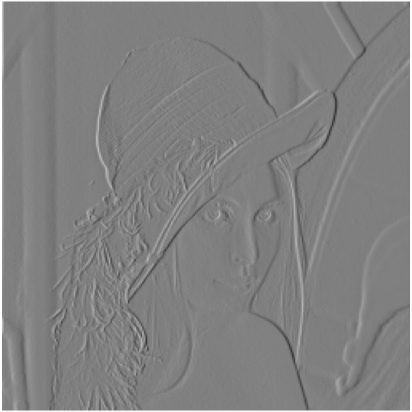
(0,0,'a') indicates that there is no previous match before `'a'`. `'$'` indicates the end of stream.

Huffman Coding

image size	256 × 256
uncompressed `tiff`	64.2 kB
LZW `tiff`	69.0 kB
Deflate (L77 + Huff) `tiff`	58 kB

image size	1920 × 720
uncompressed `tiff`	5.93 MB
LZW `tiff`	4.85 MB kB
Deflate (L77 + Huff) `tiff`	3.7 MB

Differential Coding

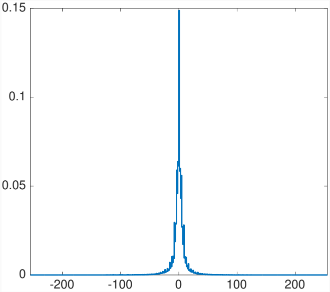


$G(x, y) = I(x, y) - I(x - 1, y)$

The key idea is to code the spatial differences in intensity.

Differential Coding

Here is the histogram of the spatial difference image:



The entropy is now 5.60 bits/pixel, which is much less than 7.57 bits/pixel we had before (despite having twice as many symbols as the range is now -255:255 instead of 0:255).

Differential Coding

It works because the entropy of a source is:

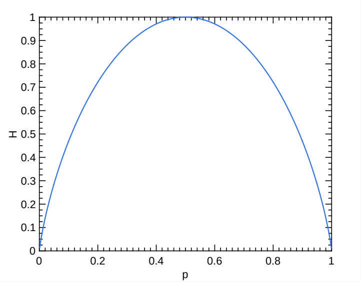
- maximised when all signals are equi-probable
- minimised when a few symbols are much more probable than the others

Differential Coding

To visualise this, consider a simple example with 2 symbols:

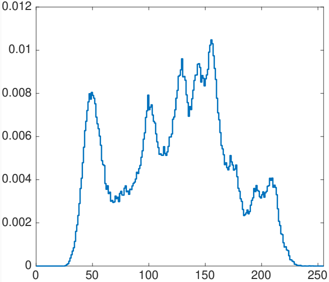
$$\begin{cases} X = 1 & \text{with probability } p \\ X = 0 & \text{with probability } 1 - p \end{cases}$$

The entropy is given by $H(X) = -p\log_2 p - (1-p)\log_2(1-p)$.

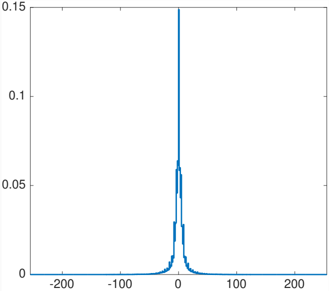


the maximum entropy is obtained at $p = 0.5$.

Differential Coding

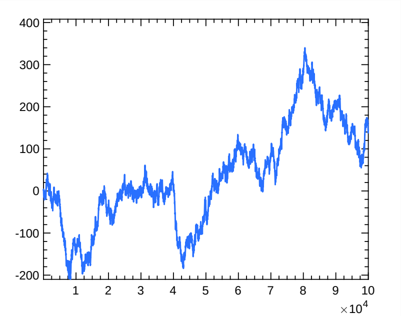


Histogram of original image.
Entropy = 7.57 bits/pixel

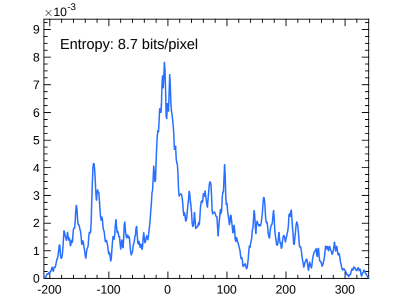


Histogram of difference image.
Entropy = 5.6 bits/pixel

Differential Coding (1D)



1D Brownian motion generated by random increments of ± 1.



Histogram & Entropy.

Now, we know that entropy is really 1 bit/pixel.

Lossy Compression

To go further, we need to take into account the human visual system and throw away the least perceptual details.

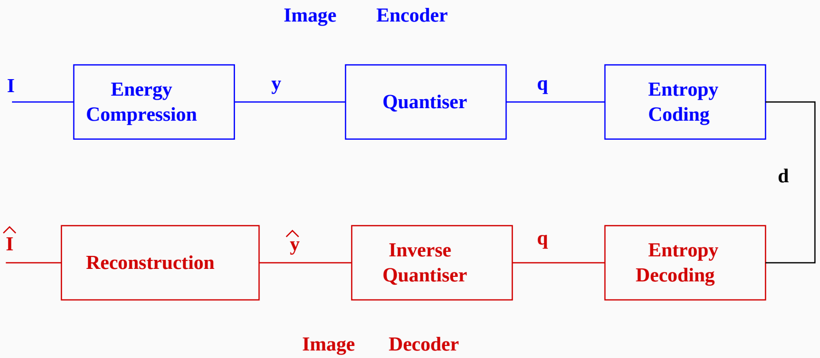
effective bit rate: 8 bits/pixel

effective bit rate: 1 bits/pixel

Haar Transform

Lossy Compression

Here are the main blocks of a compression system:



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:

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

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.

Energy Compaction: the Haar Transform

As T is orthogonal, the inverse Haar transform is simply:

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

The Haar Transform

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

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

The Haar Transform

To show more clearly what is happening:

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

These operations correspond to the following filtering processes:

The Haar Transform

To show more clearly what is happening:

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

These operations correspond to the following filtering processes:

Top left: $a+b+c+d$

It is a 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**.

The Haar Transform

To show more clearly what is happening:

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

These operations correspond to the following filtering processes:

Top right: $a - b + c - d$.

It is the 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**.

The Haar Transform

To show more clearly what is happening:

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

These operations correspond to the following filtering processes:

Lower left: $a+b-c-d = {\color{blue}a-c+}{\color{red}b-d}$.

Average vertical gradient or horizontal lowpass and vertical highpass. Lo-Hi filter.

The Haar Transform

To show more clearly what is happening:

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

These operations correspond to the following filtering processes:

Lower right: $a - b - c + d = a - b - (c - d)$.

Diagonal curvature or 2-D highpass: **Hi-Hi filter**.

The Haar Transform

Original

Haar Transform

Original

Haar Transform

We can change the brightness and contrast adjusted for visualisation: LoLo /2 and 128 is added to the LoHi, HiLo and HiHi bands.

The Haar Transform

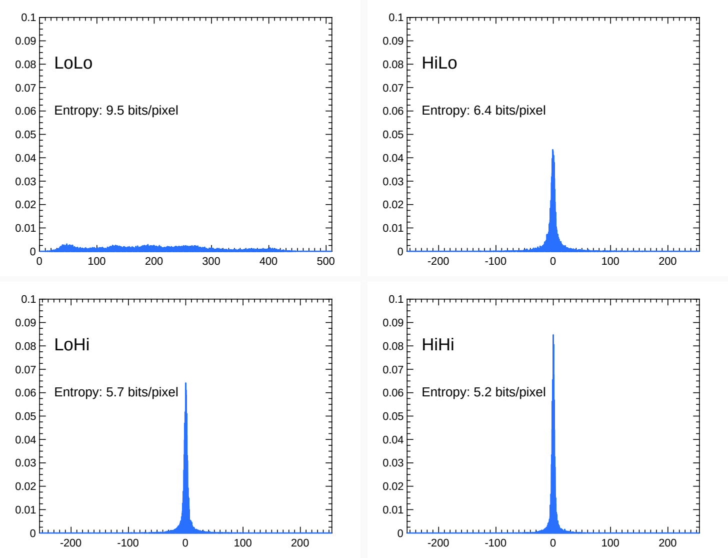


Figure: Histograms & Entropies for the 4 bands.

Calculating the overall entropy is trickier. Each coefficient in a band represents 4 pixel locations in the original image.

The entropy for Lenna is thus:

$$H(X) \approx \frac{9.4980}{4} + \frac{5.7141}{4} + \frac{6.3826}{4} + \frac{5.1978}{4} = 6.6981 \text{ bits/pixel}$$

The Haar Transform

It is better, but it is not great.

The transform generates a lot of new symbols. Also the 1D Haar transform will produce irrational numbers, eg. $\frac{1}{\sqrt{2}}(a + b)$.

Quantisation

The missing ingredient is Quantisation. It is applied to the transform coefficients:

$$Q(x) = Q_{step} \times \mathrm{round}\left(x/Q_{step}\right)$$

The step size Q_{step} should normally be decided by perceptual evaluation, and we should assign different step sizes to the different bands and we can use different step sizes for the different colour channels.

For now, we will consider a uniform step size, Q_{step}, for each band.

Quantisation

Original quantised ($Q_{step} = 15$)

Haar quantised ($Q_{step} = 15$)

The Haar Transform

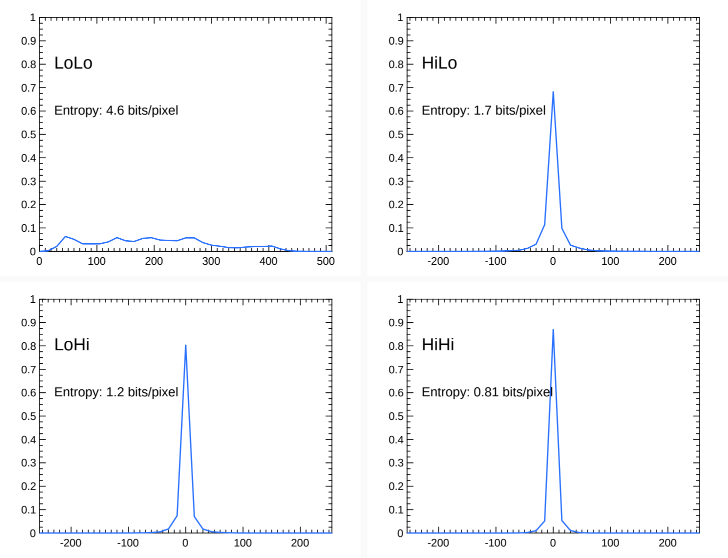


Figure: Histograms for $Q_{step} = 15$.

Quantisation

Original quantised ($Q_{step} = 15$).
Entropy: 3.7105 pixels/bit

Haar quantised ($Q_{step} = 15$).
Entropy: 2.078 pixels/bit

Multilevel Haar

Multi-Level Haar

Most of the entropy left is in the **LoLo** band.

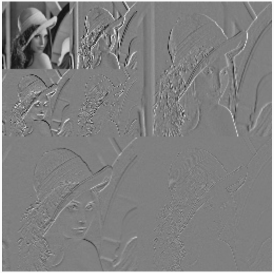
As the **LoLo** band basically a picture, we could try to apply the Haar transform to that smaller image and thus further reduce its entropy.

This is the idea behind the multi-level Haar Transform.

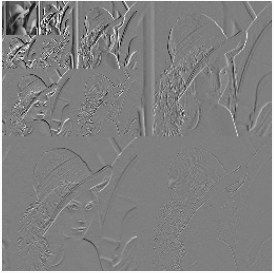
Multi-Level Haar

The multi-level Haar is obtained by iteratively applying the Haar transform to the **LoLo** band at each level.

1 level



2 levels



3 levels

note: the brightness and contrast have been changed for better visualisation.

Multi-Level Haar

Haar level 1, $Q_{step} = 15$
RMS error: 3.612

Haar level 4, $Q_{step} = 15$
RMS error: 3.534

Multi-Level Haar

Original quantised $Q_{step} = 30$
RMS error: 8.622

Haar level 1, $Q_{step} = 30$
RMS error: 6.356

Multi-Level Haar

Haar level 1, $Q_{step} = 30$
RMS error: 6.356

Haar level 4, $Q_{step} = 30$
RMS error: 5.878

Multi-Level Haar: Entropy

One Level 1 coefficient represents 4 pixels, one level 2 coefficient represents 16 pixels, etc.

	bands	Entropy/Coeff	multiplier	Entropy/pixel
Level 2	LoLo	5.58	1/16	0.34
	LoHi	2.22	1/16	0.14
	HiLo	2.99	1/16	0.19
	HiHi	1.75	1/16	0.11
Level 1	LoHi	1.15	1/4	0.29
	HiLo	1.70	1/4	0.43
	HiHi	0.80	1/4	0.29

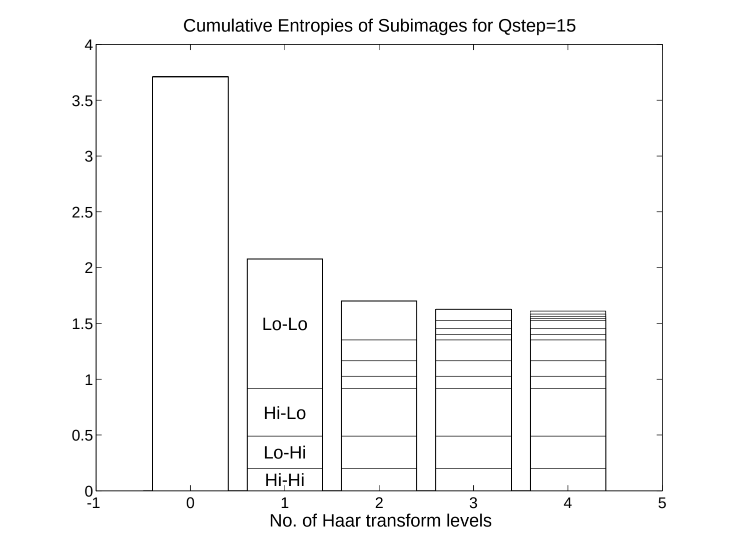
Thus for a level 2 transform at $Q_{step} = 15$, the total entropy is 1.70 bits/pixels.

Multi-Level Haar: Entropy

	bands	Entropy/Coeff	multiplier	Entropy/pixel
Level 3	LoLo	6.42	1/64	0.10
	LoHi	3.55	1/64	0.06
	HiLo	4.52	1/64	0.07
	HiHi	3.05	1/64	0.05
Level 2	LoHi	2.22	1/64	0.14
	HiLo	2.99	1/64	0.19
	HiHi	1.75	1/64	0.11
Level 1	LoHi	1.15	1/64	0.29
	HiLo	1.70	1/64	0.43
	HiHi	0.80	1/64	0.29

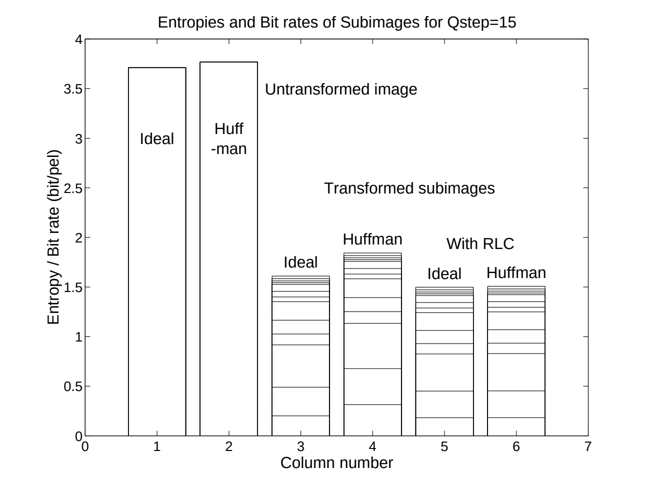
For a level 3 transform at $Q_{step} = 15$, the total entropy is 1.62 bits/pixels.

Multi-Level Haar: Entropy



Practical Entropy Encoding

Multi-Level Haar: Entropy



The ideal bit/pixel ratio given by the entropy is somewhat far from the actual performance of Huffman coding.

Multi-Level Haar: Entropy

Why is the code inefficient?

Remember that for a symbol k, we have:

$$\mathrm{length}_k = -\log_2(p_k)$$

The code is inefficient because after quantising the Haar tranform, the symbol **'0'** is very frequent, with a probability much bigger than 0.5 (0.8 approx). Hence the ideal symbol length for **'0'**:

$$\mathrm{length}_0 = -\log_2(0.8) = 0.32\mathrm{bits}$$

but with Huffman the minimum symbol length is 1 bit.

RLC

To achieve a bitrate that is lower than 1, we can use pre-process the symbols with Run-length codes (RLCs) on the 0 symbol.

That is, the pels of the subimage are scanned sequentially (usually in columns or rows) to form a long 1-dimensional vector.

Each non-zero sample is coded as a single symbol in the normal way.

Each run of consecutive zero samples (the most probable symbol) in the vector is coded as a single symbol with an associated length (as a power of 2).

Example

`-13, -5, 1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0`

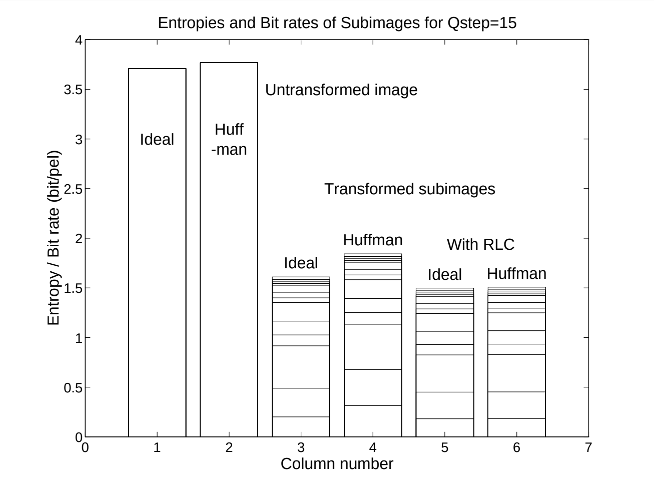
becomes

`-13, -5, 1, 4x0, 2x0, -1, 8x0, 2x0, 1x0`

where `1x0`, `2x0`, `4x0`, `8x0` are the symbols for 1, 2,4 and 8 consecutive zeros.

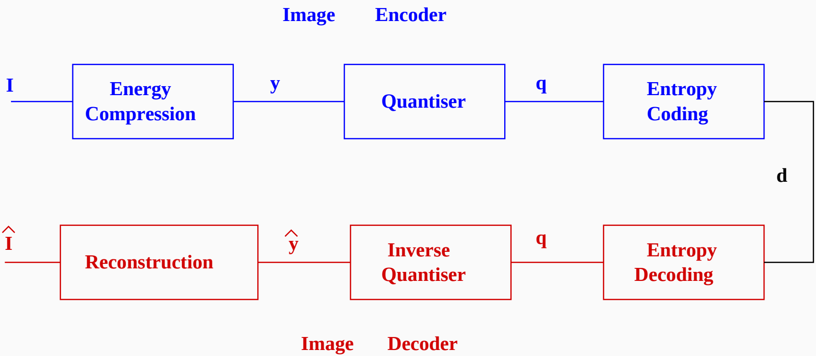
Multi-Level Haar: Entropy

by pre-processing the quantised transform by RLC, we are able to achieve near ideal entropy coding:



conclusion

Conclusion



This diagram is the core of image compression.

Conclusion

We have seen that the Haar Transform can be used to decrease the energy of the signal and thus achieve much better compression.

The transform leads to better picture quality. Note that we are able to achieve this **without** relying on the Human Visual System.

That is, we only looked so far at the spatial statistics of images and how they can be generated from fewer coefficients.

We are still to make better use of the Human Visual System.