Colour images

When coding images we usually don't use the RGB colour space. Instead the image is described in another colour space, where the pixel values are given using a *luminance* (or *luma*) component (called Y), that tells us how bright the pixel is (ie basically a grayscale signal) and two *chrominance* (or *chroma*) components (called Cb and Cr) that tells the actual colour of the pixel.

The chrominance components can often be downsampled to a lower resolution, without a human observer noticing any reduction in image quality.

There are many variants of luminance-chrominance colour spaces, but they are rather similar to each other.

RGB to YCbCr

Suppose that E_R, E_G and E_B are analog values between 0 and 1 that describe how much red, green and blue there is in a pixel (given eight bit quantization we have $E_R = R/255$, $E_G = G/255$ and $E_B = B/255$). A typical conversion to luminance-chrominance is then given by

$$\left\{ \begin{array}{lcr} E_Y & = & 0.299 \cdot E_R + 0.587 \cdot E_G + 0.114 \cdot E_B \\ E_{Cb} & = & -0.169 \cdot E_R - 0.331 \cdot E_G + 0.500 \cdot E_B \\ E_{Cr} & = & 0.500 \cdot E_R - 0.419 \cdot E_G - 0.081 \cdot E_B \end{array} \right.$$

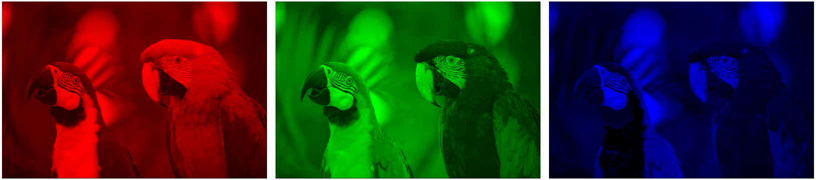
where E_Y is between 0 and 1 and E_{Cb} and E_{Cr} are between -0.5 and 0.5. Conversion to 8-bit integer values are then done by

$$\left\{ \begin{array}{lcl} Y & = & 219 \cdot E_Y + 16 \\ Cb & = & 224 \cdot E_{Cb} + 128 \\ Cr & = & 224 \cdot E_{Cr} + 128 \end{array} \right.$$

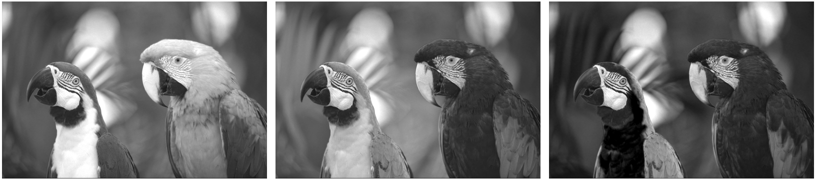
Example

Example, RGB

Colour components: R, G and B

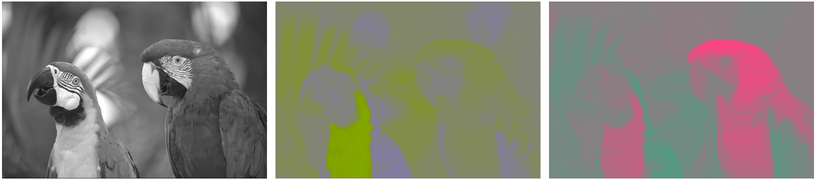


As grayscale images

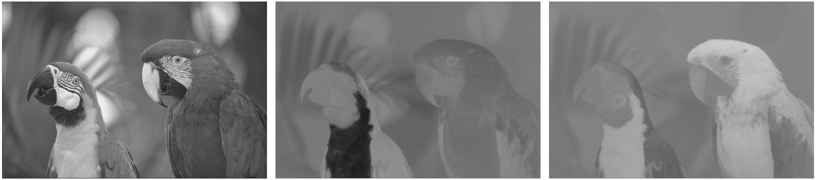


Example, YCbCr

Colour components: Y, Cb and Cr



As grayscale images



Example

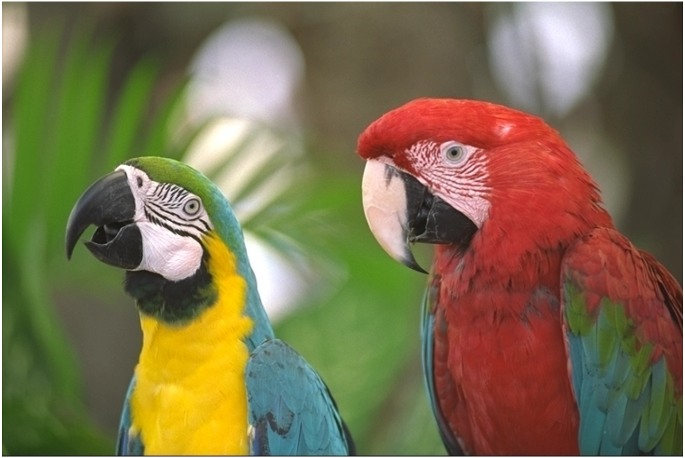


Image where Cb and Cr have been downsampled a factor 2 both horizontally and vertically, ie half of the image information has been removed.

Example

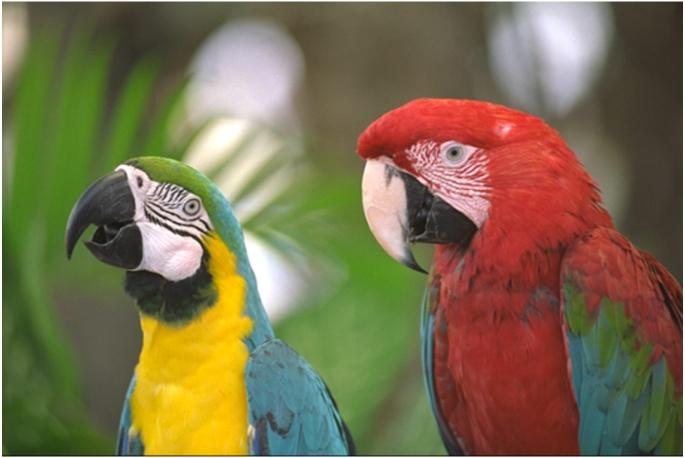
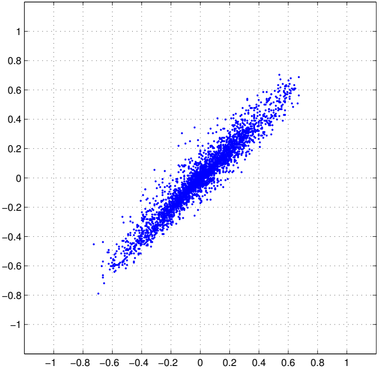


Image where Y, Cb and Cr have been downsampled a factor $\sqrt{2}$ both horizontally and vertically, ie half of the image information has been removed.

Transform coding, introduction

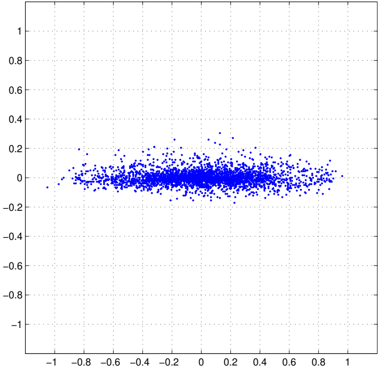
Consider pairs of samples from a speech signal.



Consecutive sampels are strongly correlated. If we quantize the samples scalarly, the quantizer for both sampels must be able to handle large variations in the signal values. If we instead describe the pairs in a new basis (another coordinate system) we remove the dependance between the samples and make it easier to do scalar quantization.

Transform coding, introduction

New basis vectors: $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $\frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \end{pmatrix}$.



Now we can use different quantizers for the different coordinates, and only one of the quantizers needs to handle large signal values. This will mean that we can get a more efficient coding (lower rate at the same distortion, or lower distortion at the same rate).

Transform coding

1. Split the signal into blocks of size N (or $N \times N$ if the signal is twodimensional). Transform the blocks using a suitable, reversible transform to a new sequence.
2. Quantize the transform components.
3. Use some kind of source coding on the quantized transform components (fixed length coding, Huffman, arithmetic coding et c.)

Linear transforms

Block of N samples from the signal $\{x_n\}_{n=0}^{N-1}$ are transformed to a block $\{\theta_n\}_{n=0}^{N-1}$

$$\theta_n = \sum_{i=0}^{N-1} a_{n,i} \cdot x_i$$

All the components of x have the same statistics (variance et c.) but the components of θ will have different statistics, depending on position n.

The inverse transform, that recreates $\{x_n\}$ from $\{\theta_n\}$ is given by

$$x_n = \sum_{i=0}^{N-1} b_{n,i} \cdot \theta_i$$

Matrix description

The transform and the inverse transform can be written in matrix form as

$$\bar{\theta} = \mathbf{A} \cdot \bar{\mathbf{x}} \quad ; \quad \bar{\mathbf{x}} = \mathbf{B} \cdot \bar{\theta}$$

where

$$\bar{\mathbf{x}} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix} \quad ; \quad \bar{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{N-1} \end{pmatrix}$$

and the matrix element at position (i, j) is given by

$$[\mathbf{A}]_{i,j} = a_{i,j} \quad ; \quad [\mathbf{B}]_{i,j} = b_{i,j}$$

The matrices $\mathbf{A}$ and $\mathbf{B}$ are the inverses of each other, ie $\mathbf{B} = \mathbf{A}^{-1}$.

Orthonormal transforms

We are usually only interested in *orthonormal* transforms, ie transforms where $\mathbf{B} = \mathbf{A}^{-1} = \mathbf{A}^T$.

Orthonormal transforms are energy preserving, ie the sum of the squares of the transformed signal is equal to the sum of the squares of the original signal

$$\begin{aligned}\sum_{i=0}^{N-1} \theta_i^2 &= \bar{\theta}^T \bar{\theta} \\ &= (\mathbf{A}\bar{\mathbf{x}})^T \mathbf{A}\bar{\mathbf{x}} \\ &= \bar{\mathbf{x}}^T \mathbf{A}^T \mathbf{A}\bar{\mathbf{x}} \\ &= \bar{\mathbf{x}}^T \bar{\mathbf{x}} = \sum_{i=0}^{N-1} x_i^2\end{aligned}$$

Parseval's identity

The transform as a basis change

The transform can be seen as describing the signal in another basis, ie as a linear combination of new basis vectors

$$\begin{aligned}
\bar{\mathbf{x}} &= \mathbf{A}^T \bar{\theta} \\
&= \begin{pmatrix} a_{00} & \cdots & a_{N-1,0} \\ \vdots & \ddots & \vdots \\ a_{0,N-1} & \cdots & a_{N-1,N-1} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \vdots \\ \theta_{N-1} \end{pmatrix} \\
&= \theta_0 \begin{pmatrix} a_{00} \\ \vdots \\ a_{0,N-1} \end{pmatrix} + \ldots + \theta_{N-1} \begin{pmatrix} a_{N-1,0} \\ \vdots \\ a_{N-1,N-1} \end{pmatrix}
\end{aligned}$$

The rows of the transform matrix (or the columns in the inverse transform matrix) are the basis vectors of the new basis.

Properties

Some desirable properties of the transform

- The transform should concentrate the signal energy to as few components as possible.
- The transform should decorrelate the transform components, ie if possible we want $E\{\theta_i \cdot \theta_j\} = 0, \; i \neq j$. This means that we remove all dependance (memory) between the transform components.
- The transform should be robust with respect to changes in source statistics.
- The transform should be simple and fast to calculate.

All of these properties can not be found in one transform.

The Karhunen-Loève-transform (KLT)

The KLT is a transform that will completely decorrelate the transform components and also give maximal energy concentration.

Assuming we have an input signal that is modelled as a stationary random process X_n with mean zero and auto correlation function $R_{XX}(k) = E\{X_n X_{n+k}\}$. Given a block size of N, we have signal vectors

$$\bar{\mathbf{x}} = \begin{pmatrix} X_n \\ X_{n+1} \\ \vdots \\ X_{n+N-1} \end{pmatrix}$$

The *correlation matrix* $\mathbf{R}_X$ is the matrix

$$\mathbf{R}_X = E\{\bar{\mathbf{x}}\bar{\mathbf{x}}^T\}$$

KLT, cont.

The correlation matrix can be expressed using the auto correlation function

$$\mathbf{R}_X = \left(\begin{array}{cccc} R_{XX}(0) & R_{XX}(1) & \cdots & R_{XX}(N-1) \\ R_{XX}(1) & R_{XX}(0) & \cdots & R_{XX}(N-2) \\ \vdots & \vdots & \ddots & \dots \\ R_{XX}(N-1) & R_{XX}(N-2) & \cdots & R_{XX}(0) \end{array} \right)$$

The correlation matrix $\mathbf{R}_\theta$ of the transformed signal, given a transform $\mathbf{A}$, is given by

$$\mathbf{R}_\theta = E\{\bar{\theta}\bar{\theta}^T\} = E\{\mathbf{A}\bar{\mathbf{x}}(\mathbf{A}\bar{\mathbf{x}})^T\} = \mathbf{A}\mathbf{R}_X\mathbf{A}^T$$

If we want the transform to decorrelate the signal, ie diagonalize $\mathbf{R}_\theta$ (all values zero except for the main diagonal), we should choose the basis vectors (rows of $\mathbf{A}$) as the normalized eigenvectors of $\mathbf{R}_X$.

KLT, cont.

For a KLT, the variances of the transform components will be equal to the eigenvalues of the signal correlation matrix.

In addition to decorrelating the source, the KLT will also be the transform that gives the maximum energy concentration to a few transform components. This is the same as saying that the KLT is the transform that minimizes the geometric mean of the transform component variances

$$(\prod_{i=0}^{N-1} \sigma_i^2)^{1/N}$$

A disadvantage of the KLT is that it is signal dependent, so it has to be transmitted as side information. There is usually also no fast way to perform the transform.

The discrete cosine transform (DCT)

The transform matrix **C** is given by

$$[\mathbf{C}]_{ij} = \begin{cases} \sqrt{\frac{1}{N}} & ;\ i = 0 \\ \sqrt{\frac{2}{N}} \cos \frac{(2j+1)i\pi}{2N} & ;\ i = 1, \ldots, N-1 \end{cases}$$

The DCT is a close relative of the discrete fourier transform (DFT). There are fast ways of doing a DCT, in the same way that there are fast fourier transforms (FFT).

The DCT will usually be very close to a KLT for sources where there is a high correlation between consecutive samples, which includes most natural audio and image sources.

The DCT is the most commonly used transform in image and video coding. For instance it is used in the JPEG and MPEG standards.

The discrete Walsh-Hadamard transform

A *Hadamard matrix* H_N of size $N = 2^k$ is given by

$$H_N = \left(\begin{array}{cc} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{array} \right)$$

where $H_1 = 1$.

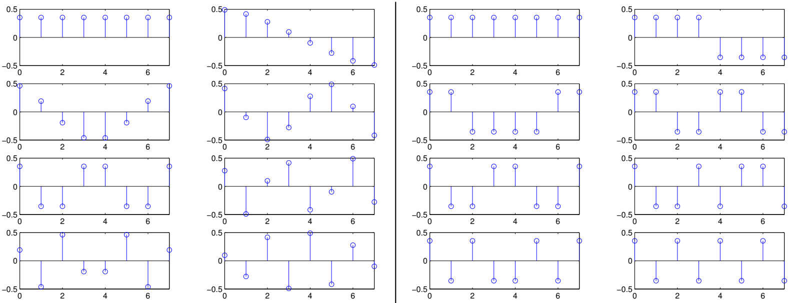
The transform matrix in DWHT is a Hadamard matrix, normalized with a factor $1/\sqrt{N}$. Usually the rows of the matrix are sorted in frequency order.

Since the transform matrix, apart from the normalizing factor, only contains ± 1, the transform is easy to calculate.

However, the DWHT does not give very good energy concentration, and since the basis vectors are very "square", any quantization errors will be very visible or audible.

DCT and DWHT

Basis vectors for 8-point DCT and DWHT

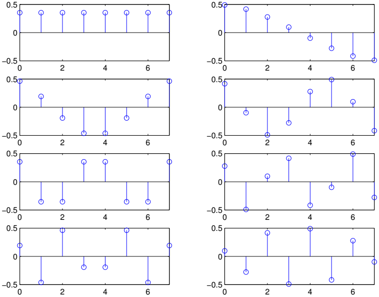


DCT

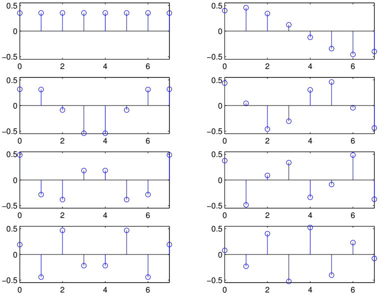
DWHT

DCT and KLT

Basis vectors for 8-point DCT and a KLT adapted to one of the audio signals used in lab 2.



DCT



KLT

Twodimensional signals

For a twodimensional signal (eg an image) we take blocks of size $N \times N$ to transform.

In general we can view this block as a vector of N^2 samples and use a transform matrix of size $N^2 \times N^2$.

Usually a *separable* transform is used. We then consider the block as a matrix $\mathbf{X}$ instead of a vector. A onedimensional transform is applied first to the rows of $\mathbf{X}$ and then to the columns (or the other way, the order will not matter). The resultat is a matrix $\mathbf{\Theta}$ of transform components

$$\mathbf{\Theta} = \mathbf{A}\mathbf{X}\mathbf{A}^T$$

The inverse transform is given by

$$\mathbf{X} = \mathbf{A}^T\mathbf{\Theta}\mathbf{A}$$

Twodimensional signals

We can view the block $\mathbf{X}$ as a linear combination of new basis matrices α_{ij} given by

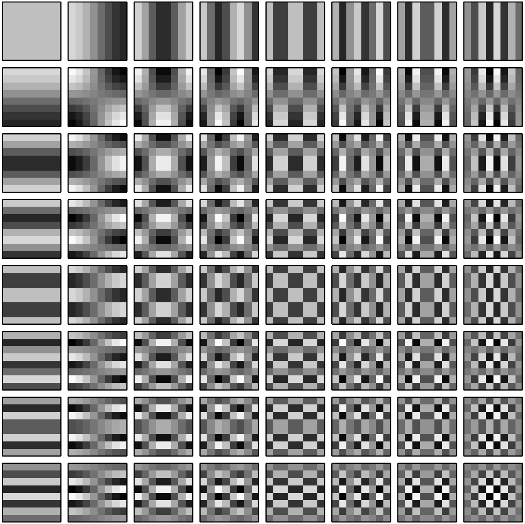
$$\alpha_{ij} = \bar{\mathbf{a}}_i^T \bar{\mathbf{a}}_j$$

where $\bar{\mathbf{a}}_i$ and $\bar{\mathbf{a}}_j$ are the i:th and j:th rows of $\mathbf{A}$.

$$\mathbf{X} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} [\mathbf{\Theta}]_{ij} \cdot \alpha_{ij}$$

A separable transform can always be written as a general transform applied to a vector of N^2 elements, but the reverse is not true.

Basis matrices for an 8 × 8 DCT



Block size

How should we choose the block size N?

A large N will give better concentration of the energy, but the transform will be more complicated to calculate. It will also be harder to adapt the coder if the source has different statistics in different parts (eg foreground and background in an image or different parts of a music signal). Large transforms can also give rise to more noticable quantization errors.

Typical block size for image coding is 8×8 pixels (JPEG, MPEG, DV)

Typical block sizes for audio coding are 256-2048 samples (Dolby Digital, MPEG AAC, Ogg Vorbis)

Distortion

For orthonormal transforms the distortion in the transform domain will be the same as the distortion in the signal domain.

Assume that we quantize and reconstruct the transform vector to $\hat{\theta}$ and inverse transform to the reconstructed vector $\hat{\mathbf{x}}$. The distortion is then

$$\begin{aligned}
D &= \frac{1}{N}||\bar{\mathbf{x}} - \hat{\mathbf{x}}||^2 = \frac{1}{N}(\bar{\mathbf{x}} - \hat{\mathbf{x}})^T(\bar{\mathbf{x}} - \hat{\mathbf{x}}) \\
&= \frac{1}{N}(\mathbf{A}^T\bar{\theta} - \mathbf{A}^T\hat{\theta})^T(\mathbf{A}^T\bar{\theta} - \mathbf{A}^T\hat{\theta}) \\
&= \frac{1}{N}(\bar{\theta} - \hat{\theta})^T\mathbf{A}\mathbf{A}^T(\bar{\theta} - \hat{\theta}) \\
&= \frac{1}{N}(\bar{\theta} - \hat{\theta})^T(\bar{\theta} - \hat{\theta}) = \frac{1}{N}||\bar{\theta} - \hat{\theta}||^2
\end{aligned}$$

The same reasoning also applies for random signals, with expectation.

Zonal coding

In zonal coding (or zonal sampling) we split the transformed vector (or block) into a number of parts (zones). All coefficients in the same zone are coded using the same quantizer and the same source coder.

If we have K zones and zone j has N_j coefficients, we of course have $N_1 + N_2 + \ldots + N_K = N$.

Given that zone j has the rate R_j bits/sample, the average rate R for the whole coder is

$$R = \frac{\sum_{j=1}^{K} N_j \cdot R_j}{N}$$

The zone division, quantization and source coding can be fixed for all blocks, or they can be changed when needed. This gives us a better possibility to adapt the coder to a varying signal, but it also means that we get more side information to transmit.

Quantization, zonal coding

From now on, assume that we let each transform coefficient be its own zone (ie all $N_j = 1$) and that we keep the coders fixed and don't switch coders between blocks.

Transform component k is quantized and coded to R_k bits, with a resulting distortion D_k. Assuming fine quantization, the distortion can be approximated by

$$D_k \approx c_k \cdot \sigma_k^2 \cdot 2^{-2R_k}$$

We want to find the bit allocation that minimizes the average distortion

$$D = \frac{1}{N} \sum_{k=0}^{N-1} D_k \approx \frac{1}{N} \sum_{k=0}^{N-1} c_k \cdot \sigma_k^2 \cdot 2^{-2R_k}$$

under the condition that the average rate is fixed

$$R = \frac{1}{N} \sum_{i=0}^{N-1} R_k$$

Bit allocation, zonal coding

For simplicity we assume that all transform components have the same type of distribution and that we use the same type of quantization and source coding. Then all c_k are equal. Lagrange optimization gives (see Sayood for details)

$$R_k = R + \frac{1}{2} \log_2 \frac{\sigma_k^2}{(\prod_{i=0}^{N-1} \sigma_i^2)^{1/N}}$$

Note that this can give some components a negative rate. In that case we set the rate for those components to 0, and redo the bit allocation for the other components, such that the average rate is still R.

For some types of quantization and coding (Lloyd-Max quantization, quantization followed by fixed length coding) we might have the condition that rates should be integers.

Distortion, zonal coding

For optimal bit allocation the distortion for each component (given that our fine quantization assumption still holds) is

$$\begin{aligned} D_k &\approx c \cdot \sigma_k^2 \cdot 2^{-2R_k} = \\ &= c \cdot \sigma_k^2 \cdot 2^{-2R - \log_2 \frac{\sigma_k^2}{(\prod_{i=0}^{N-1} \sigma_i^2)^{1/N}}} = \\ &= c \cdot \sigma_k^2 \cdot \frac{(\prod_{i=0}^{N-1} \sigma_i^2)^{1/N}}{\sigma_k^2} \cdot 2^{-2R} = \\ &= c \cdot (\prod_{i=0}^{N-1} \sigma_i^2)^{1/N} \cdot 2^{-2R} \end{aligned}$$

We will thus get the same distortion for each transform component. The average distortion will of course also take this value.

Bit allocation

Another variant is to iteratively give a bit at a time to the component that has the highest current distortion (calculate new distortion when we increase the rate for a component) until we reach the desired average rate.

For practical applications (especially at low rates) there is often no simple expression for how the distortion depends on the rate, and the bit allocation might be a little more complicated.

Transform coding gain

One way of measuring how good a certain transform is, is the *transform coding gain*. The transform gives the average distortion and signal to noise ratio

$$D_t = \frac{1}{N} \sum_{i=0}^{N-1} D_i, \quad \mathrm{SNR}_t = 10 \cdot \log_{10} \frac{\sigma_x^2}{D_t}$$

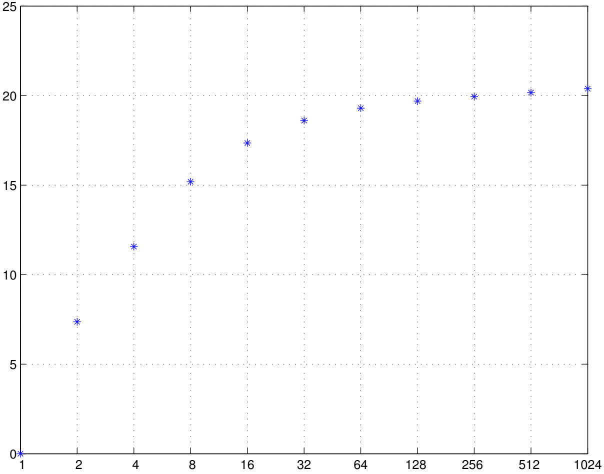
where σ_x^2 is the variance of the original signal.
Coding without transform to the same rate gives distortion D_o and signal to noise ratio SNR_o. The transform coding gain is the difference

$$\mathrm{SNR}_t - \mathrm{SNR}_o = 10 \cdot \log_{10} \frac{D_o}{D_t} \approx 10 \cdot \log_{10} \frac{\sigma_x^2}{(\prod_{i=0}^{N-1} \sigma_i^2)^{1/N}}$$

The final approximation holds when we have fine quantization and optimal bit allocation.

Transform coding gain, example

Theoretical transform coding gain for a mono version of `heyhey04.wav` as a function of the block size of the transform (DCT).



Example

Code a mono version of `heyhey04.wav` from the lab.

The signal has the variance $\sigma^2 \approx 0.04555$

If we construct a 4-bit (16 levels) scalar quantizer for the signal using the LBG algorithm we get the distortion $D \approx 0.0003472$ and a SNR of 21.18 dB.

We will now code the signal using a 4-point DCT. The variances of the four transform components are

$$\begin{aligned} \sigma_0^2 &\approx 0.1776 \\ \sigma_1^2 &\approx 0.003307 \\ \sigma_2^2 &\approx 0.001189 \\ \sigma_3^2 &\approx 0.0001455 \end{aligned}$$

Check: The average of the variances should be equal to the signal variance.

Example, cont.

We still want an average rate of $R = 4$ bits/sample. We use the formula

$$R_k = R + \frac{1}{2} \log_2 \frac{\sigma_k^2}{(\prod_{i=0}^{N-1} \sigma_i^2)^{1/N}}$$

to determine how many bits we should use in each quantizer.

$$\begin{aligned} R_0 &\approx 6.90 \approx 7 \\ R_1 &\approx 4.03 \approx 4 \\ R_2 &\approx 3.29 \approx 3 \\ R_3 &\approx 1.78 \approx 2 \end{aligned}$$

Example, cont.

We train quantizers for each transform component using the LBG algorithm with the given rates and get the resulting distortions

$$\begin{aligned} D_0 &\approx 0.00003662 \\ D_1 &\approx 0.00007431 \\ D_2 &\approx 0.0001012 \\ D_3 &\approx 0.00002551 \end{aligned}$$

The average distortion is

$$D = \frac{1}{4} \sum_{i=0}^{3} D_i \approx 0.00005941$$

which gives a SNR of 28.85 dB.
By using the transform we have gained 7.67 dB. The reasons that we didn't reach the theoretical gain (11.57 dB) are that the quantization is not fine and that we could not use the exact optimal rates.

Threshold coding

For each transform block we tell which transform components that have a magnitude over a threshold value. Only these components are quantized and coded, the rest are set to zero. Which components that are above the threshold needs to be transmitted as side information for every block.

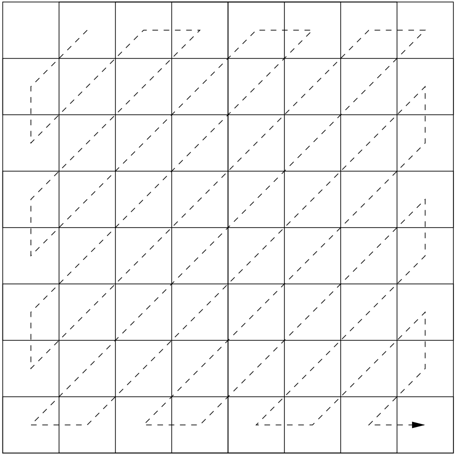
Often runlength coding of the zeros are used for this side information.

For twodimensional transforms a zigzag scanning of the components are usually performed, to get a onedimensional signal, before the runlength coding.

In practice, usually no separate thresholding is done. Instead, the components that are quantized to zero are the ones that are considered to be below the threshold.

Zigzag scanning

Zigzag scanning for 8 × 8 transform. The DC level in the upper left corner is usually treated separately.



JPEG

ISO standard (1990) for still image coding.

Uses DCT of size 8×8 pixels.

1-4 colour components.

Either 8 or 12 bits per colour components. The common file formats JFIF and EXIF only allow 8 bits per component.

No explicit thresholding, uniform quantization. The step size can be choosen freely for each of the 64 transform components. Typically the high frequency components are quantized harder than the low frequency components.

The source coding is either runlength coding of zeros followed by Huffman coding, or arithmetic coding. Since the arithmetic coder in the standard was protected by several patents, only Huffman coding is used in practice.

JPEG

Image quality is controlled by the choice of the step sizes of the 64 quantizers. Since we can choose them freely and independently of each other, it might be hard to find the best choice of step sizes for a given average rate or a given average distortion.

In order to simplify, most JPEG coders (eg digital cameras) only let the user choose one quality parameter. Each quality parameter will correspond to a pre-chosen matrix of step sizes. A quantization matrix might look like this

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

JPEG, coding of the DC level

The difference d from the DC level in the previous block is coded. The Huffman coding is not done directly on the difference values. Instead a category is formed according to

$$k = \lceil \log(|d| + 1) \rceil$$

Stastics are gathered for all categories and a Huffman code is constructed.
The codeword for a difference d consists of the Huffman codeword for k followed by k extra bits to exactly specify d.

k	d	extra bits
0	0	$-$
1	$-1, 1$	$0, 1$
2	$-3, -2, 2, 3$	$00, 01, 10, 11$
3	$-7, \ldots, -4, 4, \ldots, 7$	$000, \ldots, 011, 100, \ldots, 111$
$\vdots$	$\vdots$	$\vdots$

JPEG, coding of other components

The components are ordered in zigzag order. All runs of zeros are replaced by the length of the run (min 0, max 15). Just as for the DC component, we form the category for each non-zero component l as

$$k = \lceil \log(|l| + 1) \rceil$$

A new symbol alphabet is constructed, consisting of pairs (runlength, category). We gather statistics for the pairs and build a Huffman code for the new alphabet. Just as for the DC level, the codeword for each pair is followed by k bits that exactly tells us what value the non-zero component has.

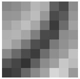
JPEG

In the Huffman code we also have two special symbols, (End Of Block) which is used when all the remaining components in a block are zero and ZRL (Zero Run Length) which is used when we have to code a run of zeros that is longer than 15. ZRL means 16 zeros. For example, a run of 19 zeros followed by category 5 is described as (ZRL)(3,5).

Example image

768 × 512 pixels, 8 bits/pixel

JPEG, example

One block from the image 

Pixel values:

6	13	26	54	45	−33	−56	12
21	23	46	60	24	−53	−38	22
32	47	62	39	−15	−69	−20	33
48	52	37	−1	−54	−57	12	20
51	30	−7	−49	−61	−6	17	31
9	−22	−52	−68	−18	14	29	65
−42	−58	−77	−32	12	31	71	59
−72	−63	−25	9	35	86	74	25

128 has been removed from all pixel calues, so black is -128 and white is 127.

JPEG, example

After DCT (rounded to integers for clarity)

$$\begin{matrix}
42 & -36 & 68 & -50 & 33 & 0 & -8 & 6 \\
37 & 213 & -35 & -116 & 65 & -20 & -3 & 6 \\
12 & -95 & -143 & 25 & 36 & -11 & 1 & -2 \\
-37 & -25 & 43 & 50 & 20 & -31 & 12 & -3 \\
8 & 24 & -14 & 12 & -33 & 11 & 8 & -11 \\
-12 & -12 & 9 & -7 & 9 & 2 & -21 & 5 \\
4 & 3 & -5 & -5 & -2 & -14 & 15 & 3 \\
0 & 0 & 4 & 1 & -1 & 0 & 0 & -5
\end{matrix}$$

JPEG, example

After quantization with step size 30 for all components (divide with the step size and round to integer)

$$\begin{matrix}
1 & -1 & 2 & -2 & 1 & 0 & 0 & 0 \\
1 & 7 & -1 & -4 & 2 & -1 & 0 & 0 \\
0 & -3 & -5 & 1 & 1 & 0 & 0 & 0 \\
-1 & -1 & 1 & 2 & 1 & -1 & 0 & 0 \\
0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}$$

Order in zigzag order (DC component removed)

-1 1 0 7 2 -2 -1 -3 -1 0 -1 -5 -4 1 0 2 1 1 1 0 0 0 0 2 1 -1 0 0 0 0 1 0 0 0
0 0 0 0 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0

JPEG, example

Gather statistics for the DC categories and construct a Huffman code. For the example test image we get the codeword lengths:

category	codeword length
0	2
1	2
2	2
3	3
4	4
5	5
6	5

The DC level in our quantized block is 1, which is category 1. Thus we will used 2+1 to code it.

JPEG, example

-1 1 0 7 2 -2 -1 -3 -1 0 -1 -5 -4 1 0 2 1 1 1 0 0 0 0 2 1 -1 0 0 0 0 1 0 0 0
0 0 0 0 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0

Code as pairs (runlength, non-zero component)

(0,-1) (0,1) (1,7) (0,2) (0,-2) (0,-1) (0,-3) (0,-1) (1,-1) (0,-5) (0,-4) (0,1) (1,2) (0,1) (0,1) (0,1) (4,2) (0,1) (0,-1) (4,1) (7,-1) (0,-1) (14,-1) (EOB)

Code the non-zero components as category plus extra bits

(0,1) 0 (0,1) 1 (1,3) 111 (0,2) 10 (0,2) 01 (0,1) 0 (0,2) 00 (0,1) 0 (1,1) 0 (0,3) 010 (0,3) 011 (0,1) 1 (1,2) 10 (0,1) 1 (0,1) 1 (0,1) 1 (4,2) 10 (0,1) 1 (0,1) 1 (4,1) 1 (7,1) 1 (0, 1) 0 (14,1) 0 (EOB)

Do the same for all blocks in the image, gather statistics for the pairs (runlength, category) and construct a Huffman code for them.

JPEG, example

Lengths of Huffman codewords (rows categories 1-7, columns runlengths 0-15):

2	5	6	7	6	6	5	4	7	8	10	11	11	10	9	8
4	7	8	9	8	8	6	7	12	-	-	-	-	14	14	14
6	10	10	9	10	9	8	9	-	-	-	-	-	-	-	-
5	12	12	12	12	12	10	12	-	-	-	-	-	-	-	-
4	-	-	-	-	15	15	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

EOB is coded with 2 bits, ZRL with 8 bits.

For our block we will need in total 3 bits for the DC level and 125 bits for the AC components.

JPEG, example

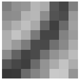
The decoder recreates the following transform block (multiply decoded components with the step sizes)

$$\begin{array}{rrrrrrrr}
30 & -30 & 60 & -60 & 30 & 0 & 0 & 0 \\
30 & 210 & -30 & -120 & 60 & -30 & 0 & 0 \\
0 & -90 & -150 & 30 & 30 & 0 & 0 & 0 \\
-30 & -30 & 30 & 60 & 30 & -30 & 0 & 0 \\
0 & 30 & 0 & 0 & -30 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -30 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

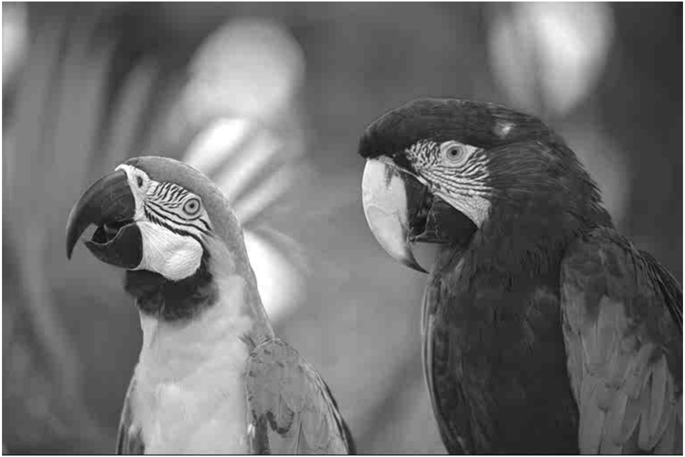
JPEG, example

The block is then inverse transformed to the following block

$$\begin{matrix}
-1 & 17 & 22 & 54 & 44 & -41 & -59 & 0 \\
12 & 14 & 51 & 55 & 13 & -45 & -52 & 26 \\
28 & 47 & 63 & 41 & -28 & -65 & -22 & 32 \\
46 & 67 & 44 & -6 & -59 & -59 & 2 & 27 \\
47 & 26 & -4 & -53 & -55 & -16 & 9 & 43 \\
10 & -29 & -61 & -54 & -14 & 18 & 36 & 59 \\
-43 & -57 & -66 & -21 & 19 & 43 & 67 & 45 \\
-73 & -71 & -22 & 4 & 22 & 73 & 71 & 20
\end{matrix}$$

which looks like 

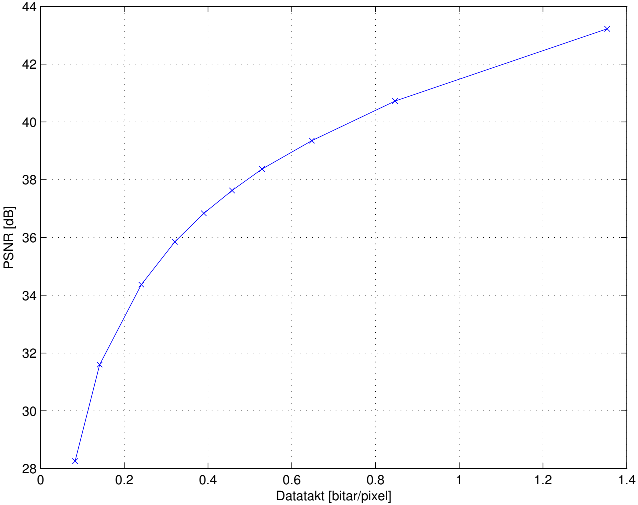
Decoded image



0.35 bits/pixel, PSNR 36.5 dB

JPEG coding

PSNR as a function of the rate by JPEG-coding of the grayscale test image.



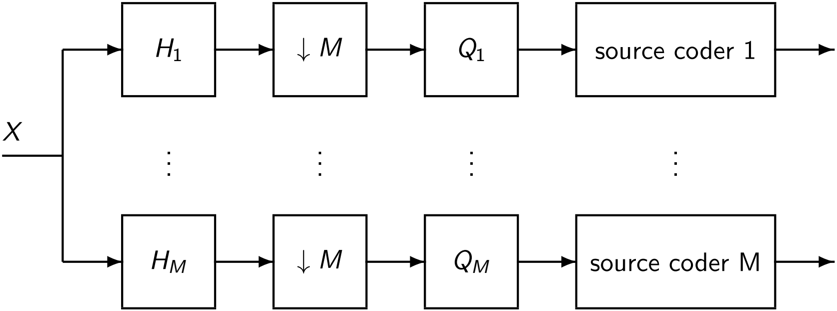
Subband coding

Split the signal into different frequency bands using a number of bandpass filters. The different frequency signals can (in theory) be downsampled without destroying any information, since they have a smaller bandwith than the original signal.

Quantize and source code the different frequency signals.

Transform coders and subband coders are closely related.

Subband coder (M bands)

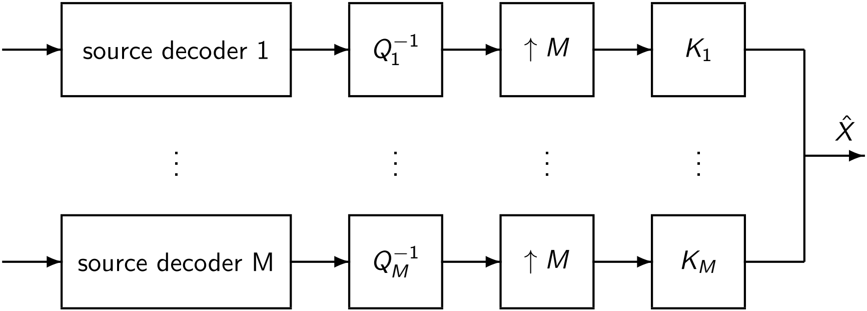


H_i ; $i = 1 \ldots M$ are called *analysis filters*.

$\downarrow M$ denotes downsampling with a factor M, ie we only keep every M:th sample in each subband.

The source coders and quantizers can of course also depend on each other.

Subband decoder (M bands)



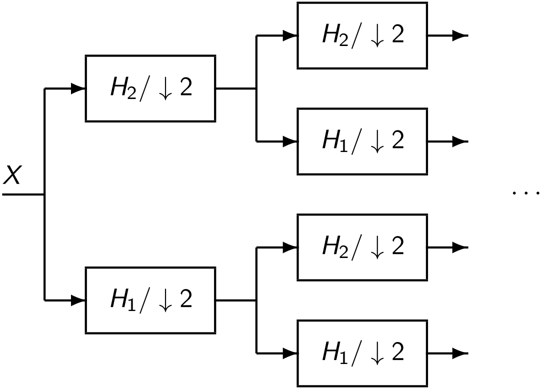
K_i ; $i = 1 \ldots M$ are called *synthesis filters*.

$\uparrow M$ denotes upsampling with a factor M, ie $M - 1$ zeros are inserted after each sample.

The source decoders and reconstructions can of course depend on each other.

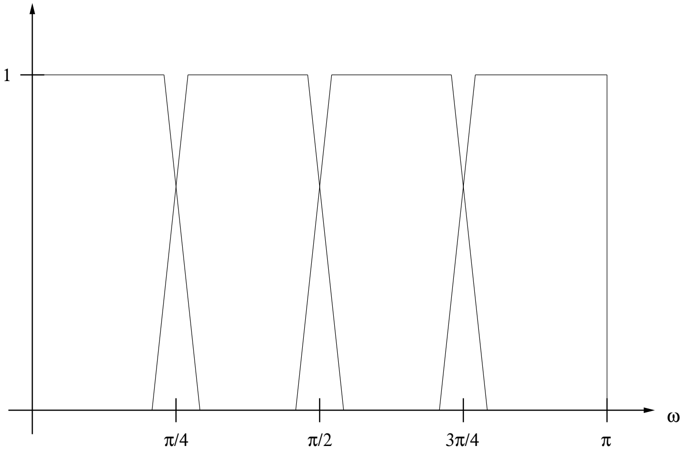
Recursive filtering

Either we have M actual filters, or we use only two filters (one highpass filter and one lowpass filter) and then apply the filters recursively to divide the signal into narrow bands. (The filtering and downsampling have been combined.)



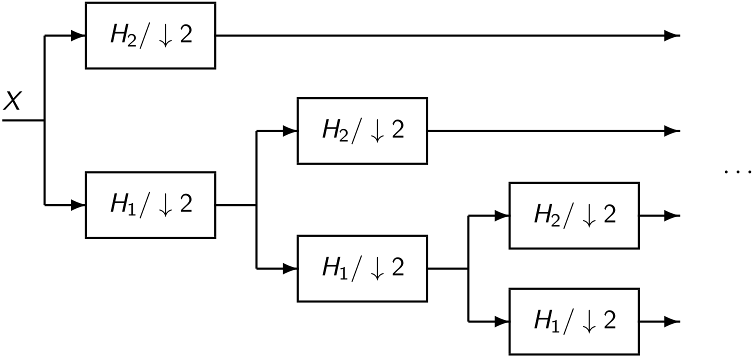
Flat filter bank

Division of the frequency axis using a flat filter bank with 4 bands:



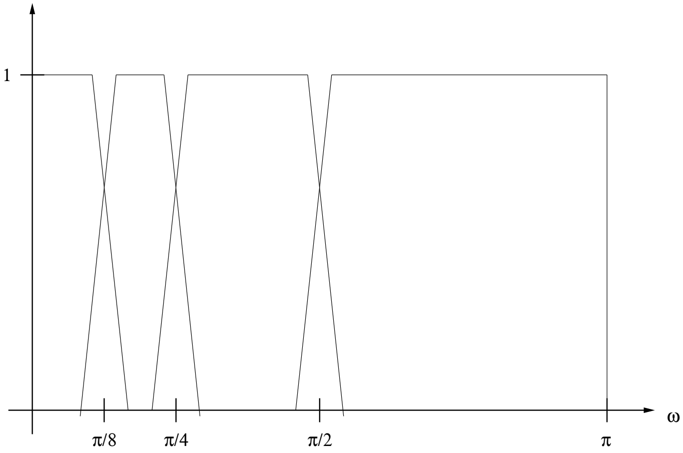
Recursive filtering, cont.

Another way is to only keep splitting the lowpass branch. Such a division is called *dyadic*.



Dyadic filter bank

Division of the frequency axis using a dyadic filter bank with 4 bands:



Filter properties

If we consider the system coder-decoder for a 2-band subband coder without quantization, we can show (see Sayood) that the reconstructed signal, expressed in the z-transform, looks like

$$\begin{aligned}\hat{X}(z) &= \frac{1}{2}[H_1(z)K_1(z) + H_2(z)K_2(z)]X(z) + \\ &+ \frac{1}{2}[H_1(-z)K_1(z) + H_2(-z)K_2(z)]X(-z)\end{aligned}$$

We're usually interested in filters that give perfect reconstruction, ie filters where the reconstructed signal is equal to the original signal, apart from a constant gain and/or a time delay

$$\hat{X}(z) = c \cdot z^{-n_0} \cdot X(z)$$

Another common demand is that we only want to use filters with a finite impulse response (FIR).

Filter choices

There are several ways of finding suitable filters for subband coding, eg QMF, power symmetric filters, wavelets. A few examples:

Haar filter

$$\begin{array}{ll} H_1(z) = \frac{1}{\sqrt{2}}[1 + z^{-1}] & H_2(z) = \frac{1}{\sqrt{2}}[1 - z^{-1}] \\ K_1(z) = H_2(-z) & K_2(z) = -H_1(-z) \end{array}$$

LeGall filter

$$\begin{aligned} H_1(z) &= \frac{1}{4\sqrt{2}}[-z^2 + 2z + 6 + 2z^{-1} - z^{-2}] \\ H_2(z) &= \frac{1}{2\sqrt{2}}[-1 + 2z^{-1} - z^{-2}] \\ K_1(z) &= H_2(-z) \\ K_2(z) &= -H_1(-z) \end{aligned}$$

Transform coding as subband coding

Suppose that we have an N-point transform $\mathbf{A}$. The signal to be coded is split into small blocks of N samples that are transformed. The transform components are calculated as

$$\theta_n = \sum_{i=0}^{N-1} a_{n,i} \cdot x_i$$

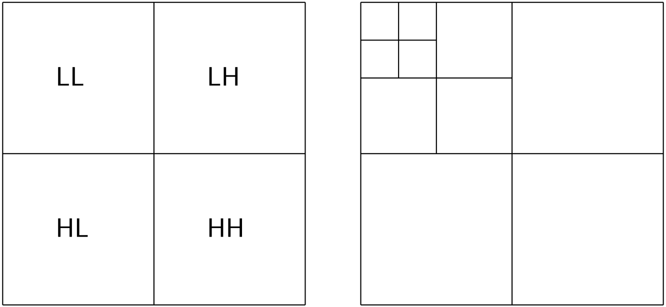
This is equivalent to filtering the signal with N filters with impulse responses

$$H_n(z) = \sum_{i=0}^{N-1} a_{n,i} \cdot z^{-i}$$

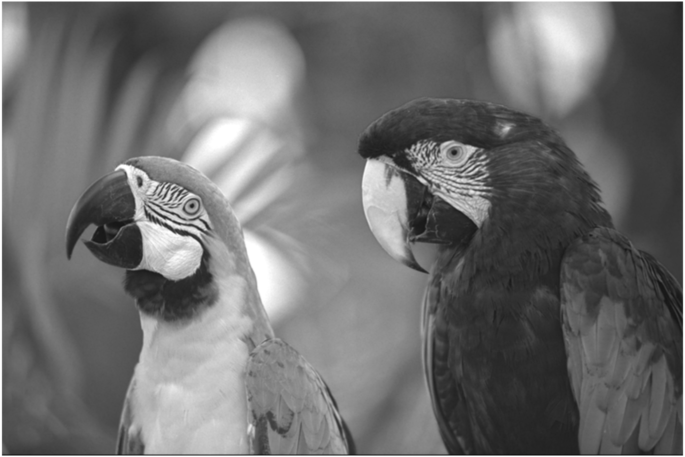
and then subsampling with a factor N.

Twodimensional signals

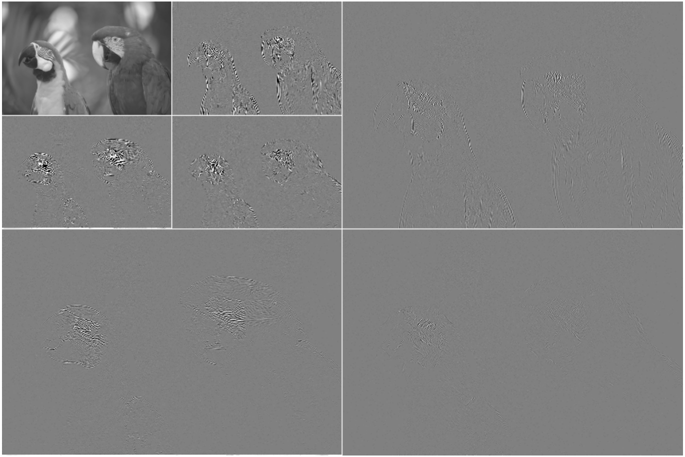
Usually only two filters (lowpass and highpass) are used. The image is filtered horizontally and then vertically with the filter pair so that we get four different frequency bands. Traditionally we only keep splitting the lowpass-lowpass part. Typically this is done for a few steps, depending on the size of the image.



Original image



Subband transformed image



The high frequency bands have been amplified to show the results more clearly.

Quantization and source coding

In principle we can use the same kinds of methods that are used in transform coding when we do quantization and source coding.

The most important part is to find an efficient way to do source coding.

In the high frequency bands most of the components will be quantized to 0, and there is a strong correlation between adjacent components in the same subband. There is also a correlation between components in different subbands at the same position in the image (eg an edge in the image will give large values in several subbands). This can be utilized in the source coding.

Bit allocation

If we have a flat filter bank we can do bit allocation in exactly the same way as in in transform coding (zonal coding).

If we dont have uniform frequency bands, eg from using a dyadic filter bank, we have to take into account the different sample rates of the different bands. This is because we have performed different number of subsamplings for the different bands.

JPEG 2000

ISO standard for coding of still images.

The image is first split into a number of rectangular parts (*tiles*). Normally we will only have one tile covering the whole image.

Each tile is transformed using a dyadic subband transform (wavelet transform), using 0-32 divisions.

The transformed image is divided into small rectangular blocks of $2^k \times 2^l$ ($2 \leq k, l \leq 10;\ k + l \leq 12$) coefficients for quantization and source coding.

The quantization is uniform.

Binary arithmetic coding. The coefficents are coded one bitplane at a time. The coefficients are coded conditioned on surrounding coefficents in the same block. The similarity between different subbands is not used.

JPEG 2000

Gives a progressive bitstream, ie its possible to decode just the beginning of the stream and still get a whole image, but with lower quality.

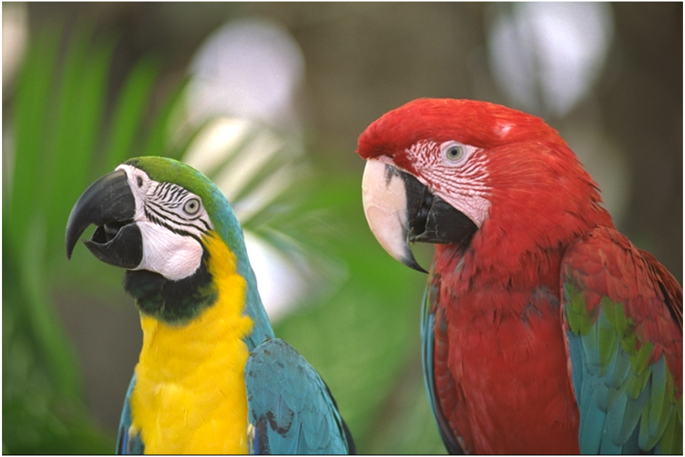
It's possible to specify a *region of interest*, ie a part of the image can be coded using higher quality than the rest of the image.

1-16384 colour components in the image. Can thus be used for multispectral and hyperspectral images.

The input image can have up to 38 bits per colour component.

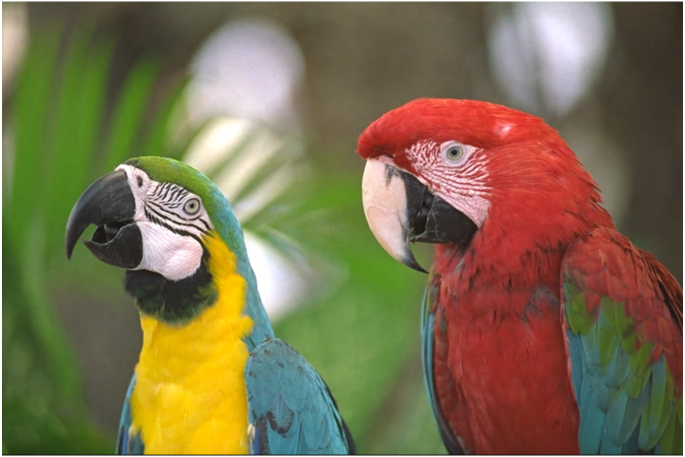
There is also a lossless coding mode, giving slightly worse results than for instance JPEG-LS.

Original image



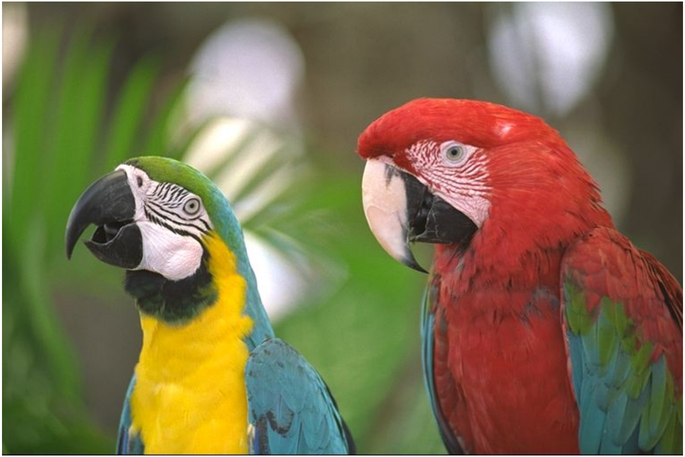
24 bits/pixel

JPEG 2000



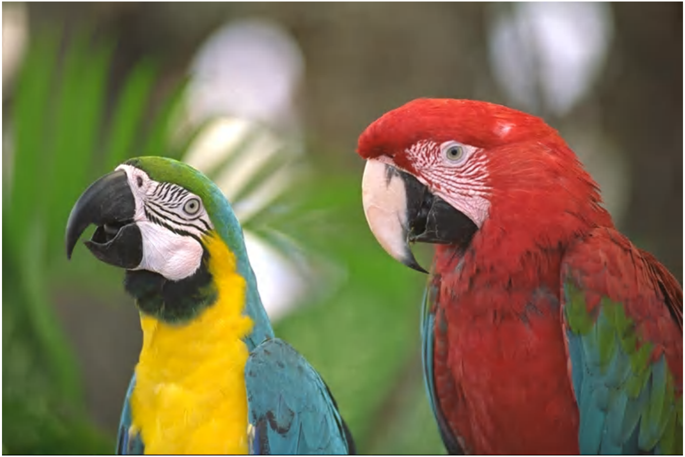
0.96 bits/pixel (compression ratio 25)

JPEG



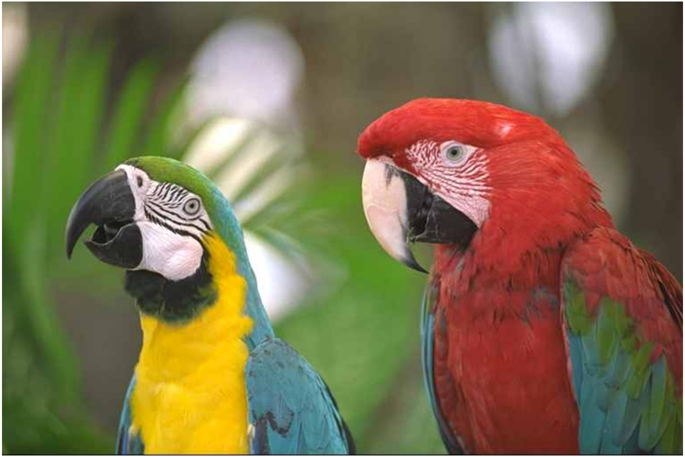
0.96 bits/pixel (compression ratio 25)

JPEG 2000



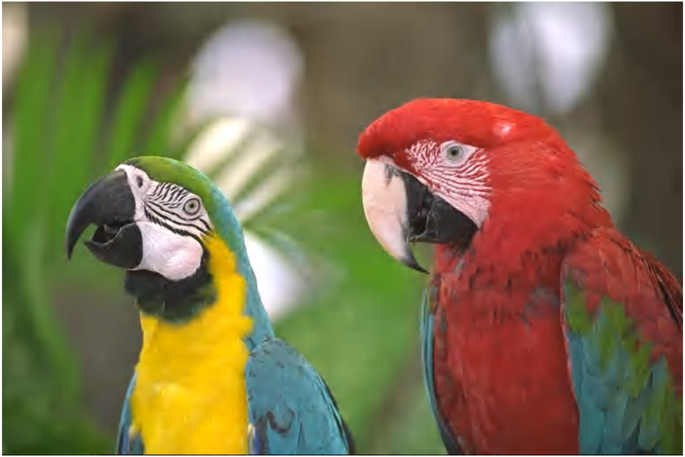
0.48 bits/pixel (compression ratio 50)

JPEG



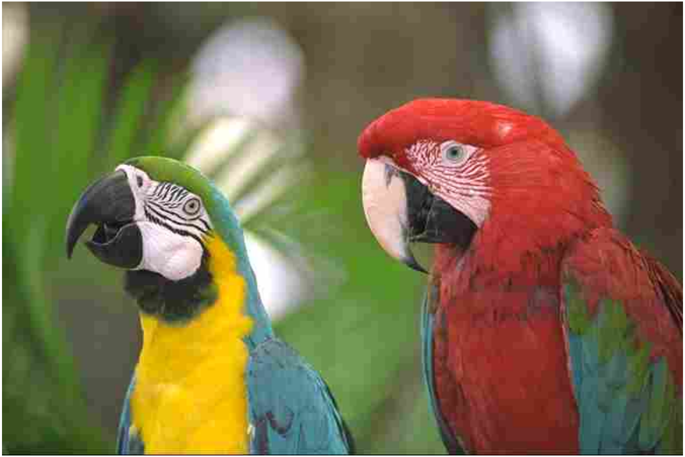
0.48 bits/pixel (compression ratio 50)

JPEG 2000



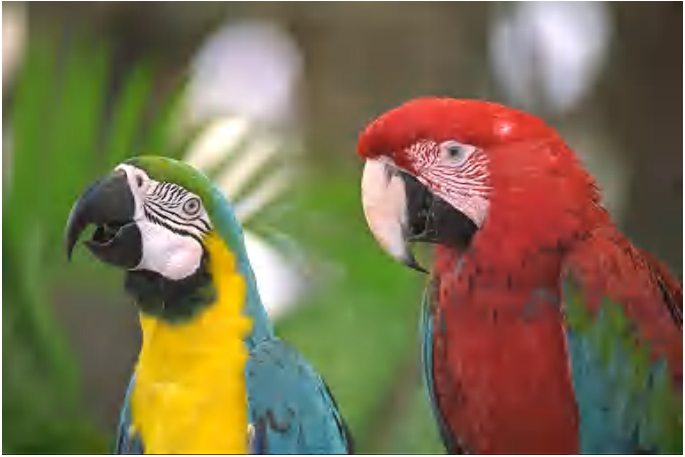
0.24 bits/pixel (compression ratio 100)

JPEG



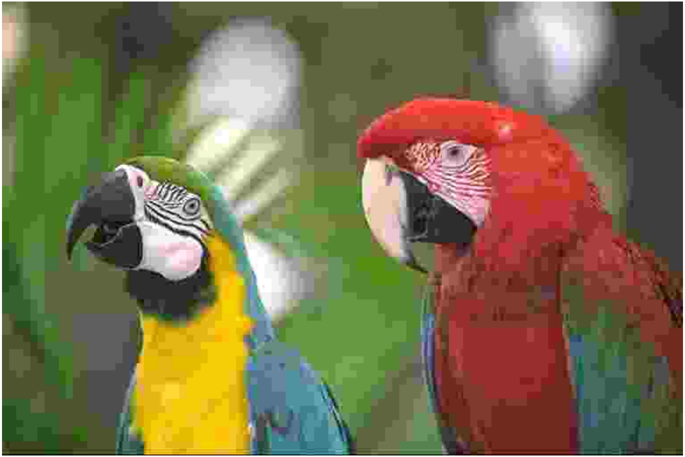
0.24 bits/pixel (compression ratio 100)

JPEG 2000



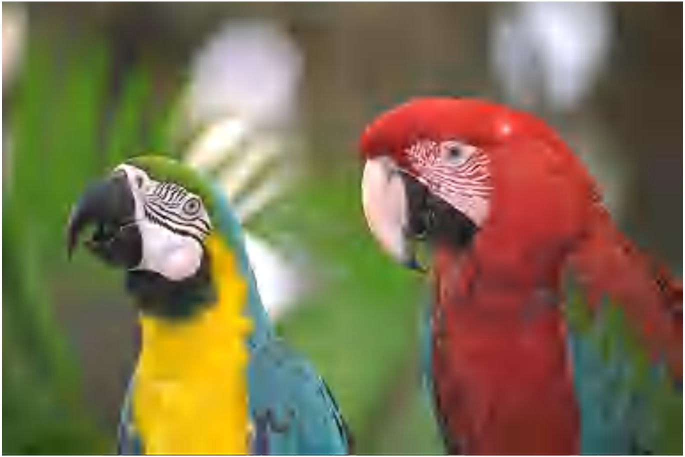
0.12 bits/pixel (compression ratio 200)

JPEG



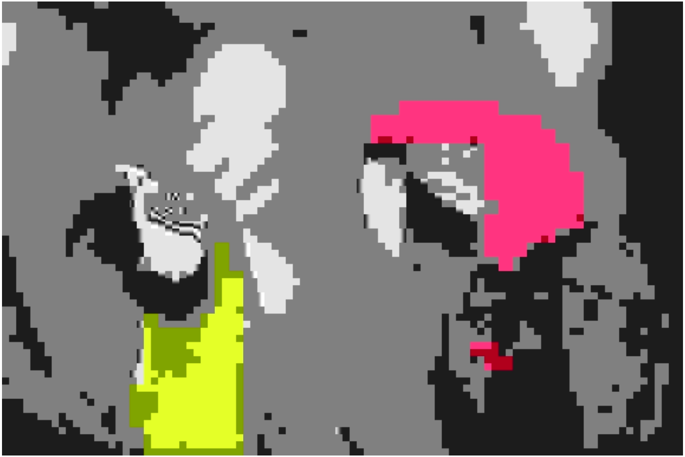
0.12 bits/pixel (compression ratio 200)

JPEG 2000



0.06 bits/pixel (compression ratio 400)

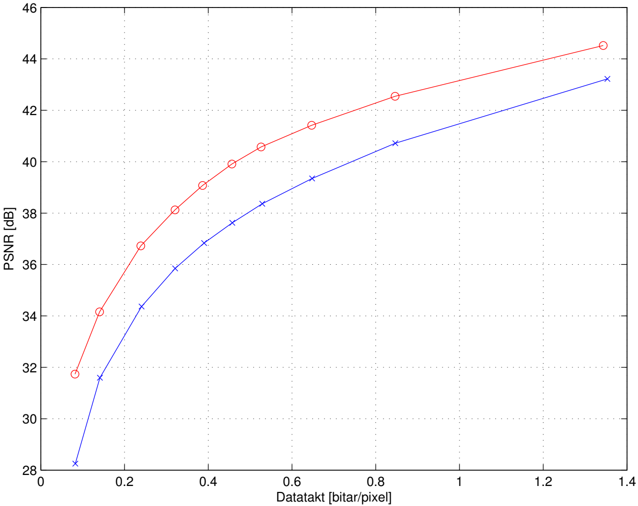
JPEG



0.06 bits/pixel (compression ratio 400)

Comparison JPEG/JPEG 2000

PSNR as a function of the rate when coding the grayscale test image using JPEG (blue curve) and JPEG 2000 (red curve).


PSNR [dB]
Datatakt [bitar/pixel]