

Chapter 8

How are digital images compressed in the web?

"In 2005, it was estimated that there were more than 1.6 billion images in the web¹."

Ferran Marqués([°]), Manuel Menezes(*), Javier Ruiz([°])

([°]) Universitat Politècnica de Catalunya, Spain

(*) Instituto Superior de Ciências do Trabalho e da Empresa, Portugal.

In 1992, a joint committee between the International Organization for Standardization (ISO) and the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T) known as the Joint Photographic Experts Group (JPEG) issued a standard that was approved in 1994 as ISO/IEC 10918-1 or ITU-T Recommendation T.81. This standard received the name of *Information technology – Digital compression and coding of continuous-tone still images requirements and guidelines* but it is commonly known as the *JPEG standard*.

The original goal of JPEG was to provide still image compression techniques for a large range of (i) types of images, (ii) image reconstructed qualities and (iii) compression ratios, while allowing software implementations. The quality of the proposed solution has made the standard so successful that, nowadays, JPEG encoded images are present in almost all areas and applications. Clear examples of this success are the constant use of JPEG

¹ From <http://hypertextbook.com/facts/2007/LorantLee.shtml>

encoded images in such universal environments as the web or the global acceptance in the digital camera industry as standard for storage.

In this Chapter, we are going to present and illustrate the main concepts behind the JPEG standard. We are not going to detail specificities of the standard but we will concentrate on the common, generic tools and concepts that it uses. Moreover, since these concepts are applied as well in video compression, they will be revisited in Chapter 9.

1.1 Background - JPEG

Fig. 8.1 presents the block diagram of a typical image compression and transmission system. In it, we can observe two different pairs of blocks: source encoder/decoder and channel encoder/decoder. The objective of the source encoder is to reduce the amount of data (average number of bits) necessary to represent the information in the signal. This data compression is mainly carried out by exploiting the redundancy in the image information. In turn, the objective of the channel encoder is to add redundancy to the output of the source encoder to enhance the reliability on the transmission. In this Chapter, we will concentrate on the problem of source coding.

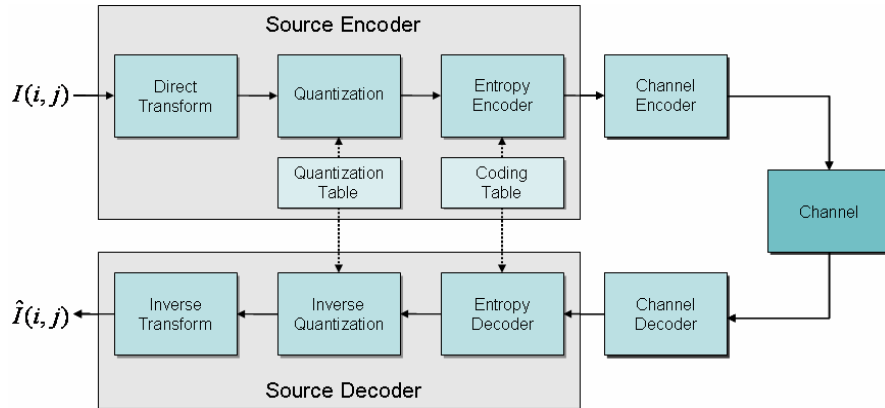


Fig. 8.1 Block diagram of an image compression and transmission system.

The *Direct Transform* block (Noll and Jayant 1984) changes the description of the image looking for a new, less correlated set of transformed samples describing the signal. The image can be exactly recovered from the transformed samples; that is, the transform step does not introduce losses in the description.

The *Quantization* step (Gersho and Gray 1993) represents a range of values of a given transformed sample by a single value in the range². The set of output values are the quantization indices. This step is responsible of the losses in the system and, therefore, determines the achieved compression.

The *Entropy Encoder* (Cover and Thomas 1991) produces the final bitstream. It exploits the non-uniformity of the probability distribution of quantization indices (symbols). This is carried out by assigning larger codewords to less likely symbols.

In order to correctly decode the bitstream produced by the coding system, the encoder has to share some information with the decoder. The decoder should know (i) how the different codewords have been stored in the bitstream (syntax definition) and (ii) which actions are associated to their different decoded values (decoder definition). This shared information defines a *standard* (Chiariglione 1999) and leads to, for example, fixing the specific transform to be used at a given stage of the decoding process or the range of possible values of a given quantization index.

Nevertheless, a standard does not fix the way in which the various parameters have to be computed, opening the door to improved implementations of standard compliant encoders (this concept is further discussed in Chapter 9 in the case of motion estimation). Moreover, a standard can introduce some additional flexibility, as some of the coding tools adopted by the standard can be adapted to the data being processed. This is the reason for explicitly representing the *Quantization* and *Coding Tables* in Fig. 8.1. Typically, these elements can be chosen at the encoder side to improve the coding performance and transmitted to the decoder in the bitstream.

In this section, we are going to concentrate on a very specific set of tools among all the possible ones in the context of transforming, quantizing and entropy encoding. These tools are the basic ones for understanding the principles of the JPEG standard (and a part of the video coding standards). This way, in the transforming context, we will address the basic color transform from RGB to YCbCr (1.1.1) and, as frequency transform, the Discrete Cosine Transform (1.1.2). Other transform techniques that are commonly used as well in image coding are covered in previous and subsequent chapters of this book; namely, linear prediction in Chapter 1, filter banks in Chapter 3 and wavelet transform in Chapter 10. Moreover, the concept of temporal prediction is addressed in the framework of video coding in Chapter 9. Regarding quantization, Chapter 2 already covers the concepts required in this chapter and, therefore, this topic is not further developed here. In turn, in the

² Here, we are assuming scalar quantization. Grouping a set of transformed samples into a vector and performing vector quantization is possible as well.

entropy coding area (1.1.3), we are going to present the Huffman codes, since these are the most common entropy coding tools in image coding³. This section is completed with a review of a few JPEG specifications (1.1.4) and a discussion on the most common quality measures (1.1.5).

1.1.1 Color Transform

In the context of image and video compression, a common way to represent color images is using the so-called YCbCr representation⁴. The relation between these components and the RGB ones is given by the following expressions. Given the weighting factors:

$$\alpha_R \triangleq 0.299 \quad \alpha_G \triangleq 0.587 \quad \alpha_B \triangleq 0.114$$

the relationship between components is given by

$$\begin{aligned} Y &= \alpha_R R + \alpha_G G + \alpha_B B \\ Cb &= \frac{0.5}{1 - \alpha_B} (B - Y) \\ Cr &= \frac{0.5}{1 - \alpha_R} (R - Y) \end{aligned} \tag{8.1}$$

The component Y is a weighted average of the red, green and blue components and, therefore, it is common referred to as the *luminance* component of the image. In turn, Cb (Cr) is a weighted difference between the blue (red) component and the luminance component. These color differences are commonly referred to as the *chrominance* components.

Due mainly to the different density of luminance and color photoreceptors (cones and rods, respectively) in the human eye, the human visual system is substantially less sensitive to the distortion in the chrominance components. This property is exploited in image and video encoders in different manners. For instance, chrominance components are usually sub-sampled by a factor of two in the horizontal and vertical directions.

³ Other entropy codes (e.g.: Golomb or Arithmetic codes) are currently used as well.

⁴ In the case of the JPEG standard, the use of this color representation is not normative but very common and most decoders expect the image to be represented this way.

1.1.2 Frequency Transform: the Discrete Cosine Transform

1-D Transforms

Finite linear transforms are commonly used in image and video compression. An efficient transform should take advantage of the statistical dependencies among the signal samples $x(n)$ ($n = 0, \dots, N-1$) (that is, their correlation) and obtain a set of transformed samples $y(k)$ ($k = 0, \dots, M-1$) presenting, at most, local dependencies; that is, a transform should decorrelate the original samples. Moreover, a transform should separate relevant from irrelevant information in order to identify the irrelevant samples. Last but not least, the transform should be invertible in order to recover (an approximation of) the signal from the quantized transformed samples. Note that the possibility of dropping some coefficients is included in the quantization process; that is, the quantization can set some transformed coefficients to zero.

It is usual to study a transform by analyzing first the role of the inverse transform. The inverse transform can be seen as a linear decomposition of a signal $x(n)$ in terms of a weighted sum of elementary functions $s_k(n)$, the weights being the transformed samples $y(k)$:

$$x(n) = \sum_{k=0}^{M-1} s_k(n) y(k) \quad \rightarrow \quad \mathbf{x} = \mathbf{S} \mathbf{y} \quad (8.2)$$

where

$$\begin{aligned} \mathbf{x}^T &= [x(0), x(1), \dots, x(N-1)] \\ \mathbf{y}^T &= [y(0), y(1), \dots, y(M-1)] \end{aligned} \quad (8.3)$$

The vector \mathbf{s}_k containing the samples of the elementary function $s_k(n)$

$$\mathbf{s}_k^T = [s_k(0), s_k(1), \dots, s_k(N-1)] \quad (8.4)$$

is commonly referred to as the k -th *synthesis vector*. The reason is that, by means of the linear combination of the synthesis vectors, the original signal is synthesized (reconstructed). In turn, the matrix \mathbf{S} is known as the synthesis matrix. Note that the synthesis vectors \mathbf{s}_k are the columns of the synthesis matrix \mathbf{S} :

$$\mathbf{S} = [\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_{M-1}] \quad (8.5)$$

The direct transform (and, therefore, the transformed samples) is obtained by comparing the signal $x(n)$ with a set of elementary functions $a_k(n)$:

$$y(k) = \sum_{n=0}^{N-1} a_k^*(n) x(n) \quad \rightarrow \quad \mathbf{y} = \mathbf{A}^H \mathbf{x} \quad (8.6)$$

In that case, the vector \mathbf{a}_k is formed by

$$\mathbf{a}_k^T = [a_k(0), a_k(1), \dots, a_k(N-1)] \quad (8.7)$$

which is commonly referred to as the k -th *analysis vector*. In an analogous manner, the matrix \mathbf{A} is known as the analysis matrix:

$$\mathbf{A} = [\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_{M-1}] \quad (8.8)$$

Note that the transform coefficients can be expressed as

$$y(k) = \mathbf{a}_k^H \mathbf{x} \quad (8.9)$$

and this can be seen as the analysis of signal \mathbf{x} with respect to vector \mathbf{a}_k .

In order to have a revertible transform, \mathbf{S} has to be the left-inverse of \mathbf{A}^H ; that is, $\mathbf{S}\mathbf{A}^H = \mathbf{I}$. Nevertheless, in this chapter, we are going to restrict the study to the case of non-expansive transforms; that is, we are assuming that the transform maps N samples of $x(n)$ into N samples of $y(k)$. In that case, $M = N$ and $\mathbf{S}^{-1} = \mathbf{A}^H$ and, if the transform is real-valued, $\mathbf{S}^{-1} = \mathbf{A}^T$.

A common property of the image transforms is to be *orthonormal*. A transform is said to be orthonormal if all analysis vectors are orthonormal and have unit norm:

$$\mathbf{A}^H \mathbf{A} = \mathbf{I} \rightarrow \begin{cases} \langle \mathbf{a}_i, \mathbf{a}_j \rangle = \mathbf{a}_i^H \mathbf{a}_j = 0 & \forall i \neq j \\ \langle \mathbf{a}_i, \mathbf{a}_i \rangle = \|\mathbf{a}_i\|^2 = \mathbf{a}_i^H \mathbf{a}_i = 1 \end{cases} \quad (8.10)$$

In that case, and given the previous condition of non-expansiveness, the transform is defined by a unitary matrix \mathbf{A} and it is said to be a *unitary transform*. For unitary transforms, the synthesis matrix \mathbf{S} coincides with the analysis matrix \mathbf{A} . As a matter of fact:

$$\left. \begin{array}{l} \mathbf{S}^{-1} = \mathbf{A}^H \rightarrow \mathbf{S} = \mathbf{A}^{-H} \\ \mathbf{A}^H \mathbf{A} = \mathbf{I} \rightarrow \mathbf{A}^{-H} = \mathbf{A} \end{array} \right\} \rightarrow \mathbf{S} = \mathbf{A} \quad (8.11)$$

Unitary transforms maintain the energy of the samples:

$$\|\mathbf{x}\|^2 = \mathbf{x}^H \mathbf{x} = [\mathbf{A}\mathbf{y}]^H \mathbf{A}\mathbf{y} = \mathbf{y}^H \mathbf{A}^H \mathbf{A}\mathbf{y} = \mathbf{y}^H \mathbf{y} = \|\mathbf{y}\|^2 \quad (8.12)$$

This characteristic is very useful when working with the transform samples since the actions taken over a coefficient have a clear interpretation in terms of the final energy of the signal. This is a very relevant aspect when quantizing the transform coefficients in order to approximate a signal.

Nevertheless, although the total energy is preserved, it is not equally distributed among the various transformed coefficients (i.e.: the components of vector \mathbf{y}). It can be demonstrated (Noll and Jayant 1984) that the transform leading to the optimum energy concentration is the so-called Karhunen-Loewe Transform (KLT). That is, the KLT places as much energy as possible in as few coefficients as possible. It therefore mostly “explains” the input signal in terms of a limited number of synthesis vectors (those for which \mathbf{y} has high synthesis coefficients), a property which is immediately turned into profit in the quantization step. However, the analysis matrix \mathbf{A} in the case of the KLT is not fixed but it depends on the statistical properties of the signal under analysis. Actually, its analysis vectors \mathbf{a}_k are the eigenvectors of the signal correlation matrix \mathbf{R}_x :

$$\mathbf{R}_x \mathbf{a}_k = \lambda_k \mathbf{a}_k \text{ with } \mathbf{R}_x = E\{\mathbf{x}\mathbf{x}^H\} \quad (8.13)$$

Therefore, the KLT is signal adapted and the analysis matrix that defines the transform must be estimated for each signal. Although fast algorithms have been proposed, the overall complexity of the KLT is significantly higher than the complexity of other possible transforms.

In the case of images, a good compromise between energy concentration and computational complexity is the Discrete Cosine Transform (DCT). A common definition of the components of the analysis/synthesis DCT vectors is given by the following expression⁵:

$$a_k(n) = \sqrt{\frac{s}{N}} \cos\left(\frac{(2n+1)k\pi}{2N}\right) \quad s = \begin{cases} 1 & k = 0 \\ 2 & k \neq 0 \end{cases} \quad (8.14)$$

The analysis/synthesis vectors consist of samples of cosine functions (Fig. 8.2). Observe that the first analysis vector (\mathbf{a}_0) has all samples equal and, thus, the first DCT coefficient is related to the average value of the samples in the signal. In the literature, this coefficient is referred to as the *DC coefficient*, whereas the rest of the transformed coefficients are named the *AC coefficients*⁶.

To further analyze the properties of the DCT, we are going to move first to the case of transforms in two dimensions. This way, we will be able to highlight some features of the DCT in the case of processing images.

2-D Transforms

To simplify the notation, we are going to assume that we have unitary transforms. The definition of a 2D transform is given by:

$$y(k, l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} a_{k,l}^*(m, n) x(m, n) \quad (8.15)$$

whereas the inverse transform is given by:

$$x(m, n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} a_{k,l}(m, n) y(k, l) \quad (8.16)$$

⁵ This is known as the Type II DCT transform

⁶ These names come from the historical use of DCT for analyzing electrical circuits with direct- and alternating currents.

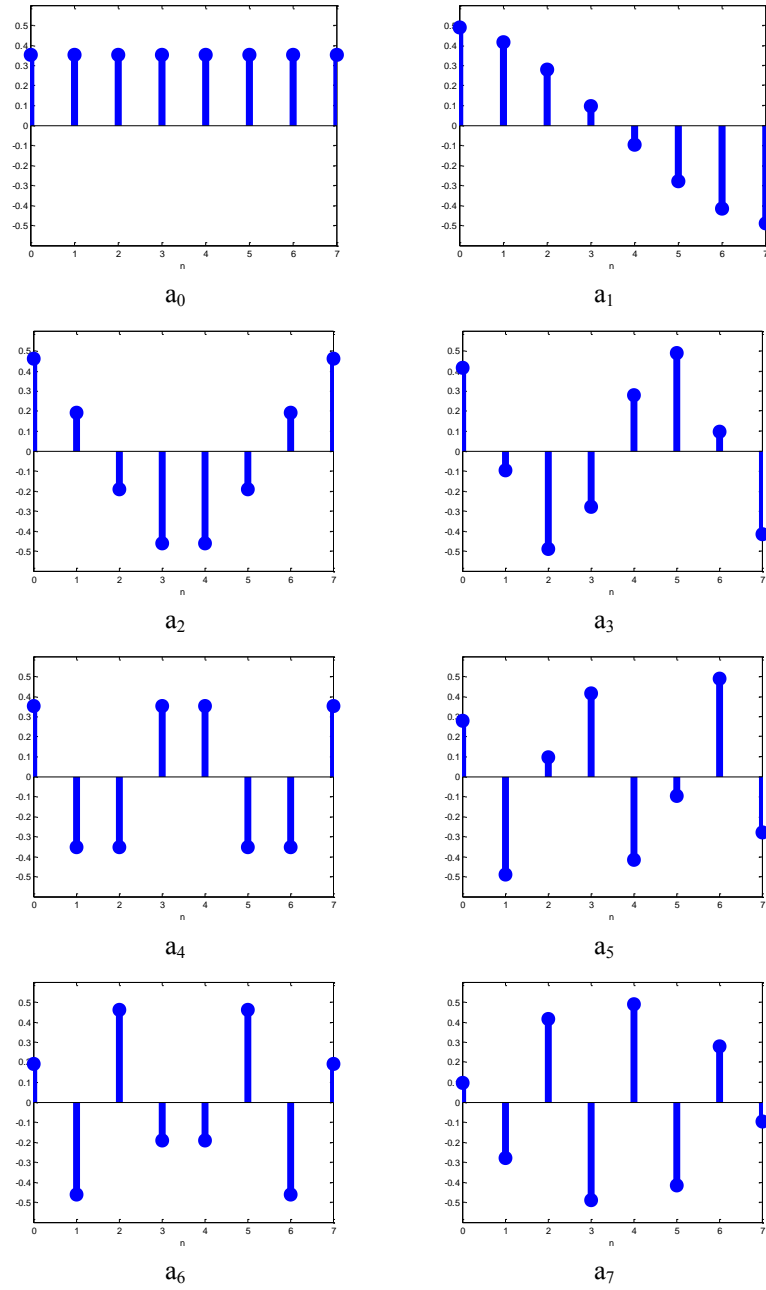


Fig. 8.2 Analysis vectors of the 1D DCT ($N = 8$ samples)

As previously, we can interpret the inverse transform as a linear decomposition of a signal $x(m,n)$ in terms of a weighted sum of elementary functions $a_{k,l}(m,n)$, which now can be seen as elementary images.

If these 2D elementary functions can be expressed as a product of two 1D elementary functions, each one dealing with one dimension in the 2D elementary function definition, the transform is said to be *separable*. Moreover, if the two 1D elementary functions are the same, the transform is said to be *symmetric*:

$$a_{k,l}(m,n) = a_k(m)b_l(n) = a_k(m)a_l(n) \quad (8.17)$$

In that case, the transform can be expressed in a different manner:

$$y(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} a_{k,l}^*(m,n) x(m,n) = \sum_{n=0}^{N-1} a_l^*(n) \left(\sum_{m=0}^{M-1} a_k^*(m) x(m,n) \right)$$

This expression allows a notation in terms of the analysis matrix \mathbf{A} associated to the 1D transform:

$$\mathbf{Y} = \mathbf{A}^* \mathbf{X} \mathbf{A}^H$$

where

$$\mathbf{X} = \begin{bmatrix} x(0,0) & \cdots & x(0,n) & \cdots & x(0,N-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x(m,0) & \cdots & x(m,n) & \cdots & x(m,N-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x(M-1,0) & \cdots & x(M-1,n) & \cdots & x(M-1,N-1) \end{bmatrix} \quad (8.18)$$

and

$$\mathbf{Y} = \begin{bmatrix} y(0,0) & \cdots & y(0,l) & \cdots & y(0,N-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y(k,0) & \cdots & y(k,l) & \cdots & y(k,N-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y(M-1,0) & \cdots & y(M-1,l) & \cdots & y(M-1,N-1) \end{bmatrix} \quad (8.19)$$

The separability property makes possible to compute the transform of an $N \times N$ image by means of two matrix multiplications of size $N \times N$. If the transform is not separable, we could represent the image by means of a single vector of size $1 \times N^2$ and, using expression (8.6), compute the transform by means of one multiplication of a vector of size $1 \times N^2$ with a matrix of size $N^2 \times N^2$. Therefore, the complexity is reduced⁷ from $O(N^4)$ to $O(N^3)$.

In the case of images, as we have previously commented, the Discrete Cosine Transform is commonly used. The DCT presents all the properties above commented (it is separable and symmetric) while, in addition, being real-valued. The expression of the 2D DCT is therefore given by:

$$y[k, l] = \sum_{n=0}^{N-1} \sqrt{\frac{s}{N}} \cos\left(\frac{(2n+1)l\pi}{2N}\right) \left(\sum_{m=0}^{M-1} \sqrt{\frac{s}{M}} \cos\left(\frac{(2m+1)k\pi}{2M}\right) x[m, n] \right)$$

with the parameter s defined as in expression (8.14).

In terms of energy concentration, the DCT reaches very high values, getting very close to the KLT performance. Actually, it can be demonstrated that, for the specific case of first order 2D Markov processes, the DCT is optimum as well (Noll and Jayant 1984). This energy concentration property is illustrated in Fig. 8.3:

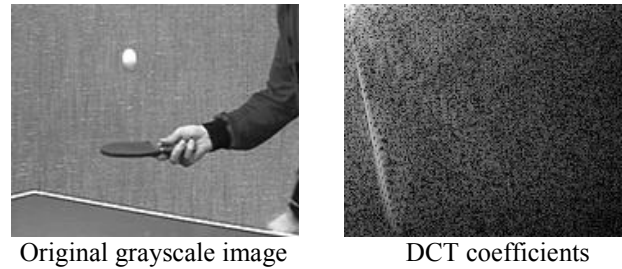


Fig. 8.3 An original image and its corresponding DCT transformation. Bright (Dark) values correspond to high (low) energy coefficients.

Fig. 8.3 shows an original image of 176×144 pixels and the corresponding DCT coefficients (values have been conveniently shifted and scaled for visualization). In Fig. 8.4, the percentage of energy of the image representation when using an increasing number of DCT coefficients is presented to illustrate how the DCT transform is able to compact the energy of the image pixels into fewer coefficients. In Fig. 8.4.a, the DCT coefficients have been

⁷ The KL transform is, in general, non-separable and, therefore, cannot profit from this complexity reduction.

sorted by descending order of energy, whereas in Fig. 8.4.b, a zigzag scan has been used to order the DCT coefficients.

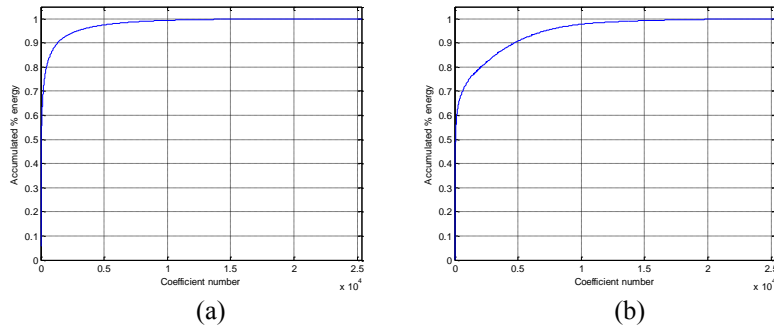


Fig. 8.4 Accumulated percentage of energy of (a) DCT coefficients sorted by descending order of energy and (b) zigzag scanned DCT coefficients

The so-called *zigzag scan* (see Fig. 8.5) is based on the observation that the transformed coefficient energy of most images tends to decrease with increasing spatial frequency. This way, the zigzag scan visits coefficients in order of roughly decreasing energy.

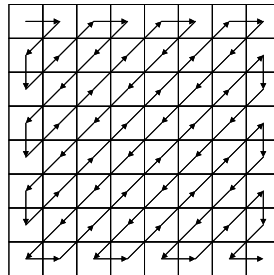


Fig. 8.5 Example of zigzag scan for the case of an image of 8x8 pixels

The following two Figures illustrate the effects of reconstructing the original image after discarding (zeroing) several DCT coefficients. In Fig. 8.6, reconstructed images are shown using the N coefficients with higher energy (all other DCT coefficients are zeroed). As the DCT can compact the energy, using a few coefficients leads to good reconstructions of the original image.

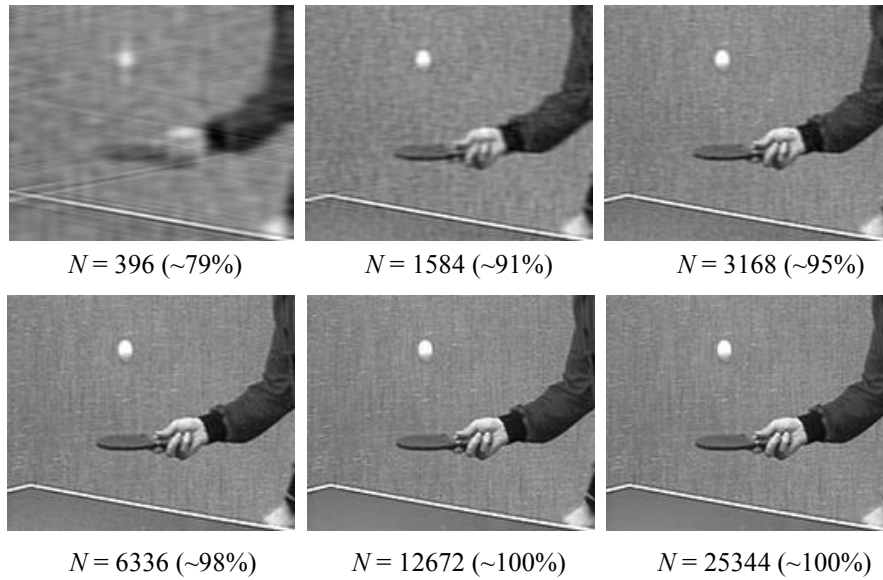


Fig. 8.6 Reconstructed images using the first N coefficients with higher energy (in brackets the % of energy corresponding to the N selected coefficients)

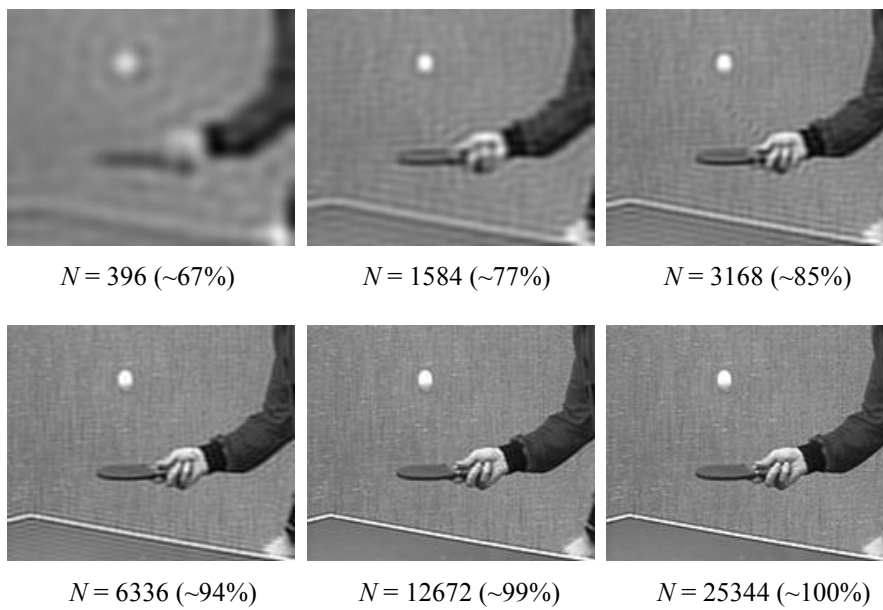


Fig. 8.7 Reconstructed images using the first zigzag scanned N coefficients (in brackets the % of energy corresponding to the N selected coefficients)

In Fig. 8.7, the same experiment is performed but, this time, the first N coefficients are selected using a zigzag scan. It can be noted that the zigzag scan is a good approximation for the selection of coefficients with higher energy.

In the previous example, we have observed that, although the DCT can compact the energy of the signal into a reduced number of coefficients, the fact that images are non-stationary signals prevents the transform from further exploiting the statistical homogeneities of the signal. The possibility of segmenting the image into its statistically homogeneous parts has not been envisaged in classical approaches since (i) the process of image segmentation is a very complicated task (segmentation is often named an ill-posed problem) and (ii) if a segmentation of the image is obtained, the use of the partition for coding purposes would require transmitting the shapes of the arbitrary regions and this boundary information is extremely difficult to compress.

This problem is commonly solved by partitioning the image into a set of square blocks and processing the information within each block separately. This leads to the use of the so-called *block transforms*. In this case, a fixed partition is used, which is known before hand by the receiver and does not require to be transmitted. Since this partition is independent of the image under analysis, the data contained in each block may not share the same statistical properties. However, given the small size of the used blocks (typically, 8x8 pixels), the information within blocks can be considered quite homogenous. When using block transforms (that is, when imposing an image independent partition for image compression), it is likely to observe the block structure in the reconstructed image. This artifact is known as *block effect*, and it is illustrated in Fig. 8.22.

In the case of the DCT, when using blocks of 8x8, the analysis/synthesis vectors correspond to the following expression:

$$a_{k,l}(m,n) = \sqrt{\frac{s}{8}} \cos\left(\frac{(2n+1)l\pi}{16}\right) \sqrt{\frac{s}{8}} \cos\left(\frac{(2m+1)k\pi}{16}\right) \quad (8.20)$$

which can be interpreted as a set of 64 elementary images on which the original signal is decomposed. In Fig. 8.8 we present these 64 images of 8x8 pixels each. Note that the analysis/synthesis vector values have been conveniently shifted and scaled in order to represent them as images.

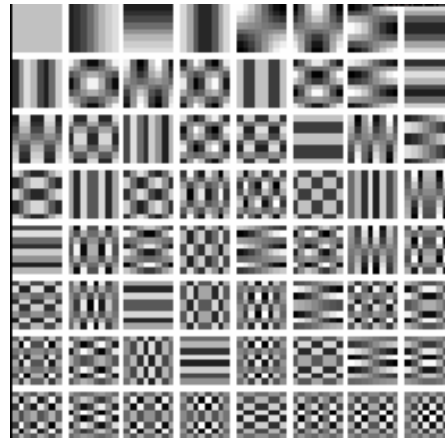


Fig. 8.8 Set of DCT analysis/synthesis vectors for the case of 8x8 pixel blocks

The study of the performance of the DCT used as a block transform for image compression is addressed in the next Section.

Regarding the computational complexity of the DCT, it has to be said that, in addition to the fact of being separable and symmetric, the DCT presents a structure close to that of the FFT, which allows for highly efficient implementations. For instance, in (Arai *et al.* 1988) a DCT implementation is presented that only requires, for 8x8 pixel blocks, 5 multiplications and 29 additions, instead of the basic 64 multiplications. Furthermore, most of the multiply operations can be performed in parallel, so that the proposed algorithm is ideal for a parallel implementation.

1.1.3 Entropy coding

The entropy encoder exploits the non-uniformity of the probability distribution of quantization indices (symbols) to generate the shortest bit stream representing a given sequence of symbols. This is carried out by assigning larger codewords to less likely symbols and the theoretical framework to do so is the so-called *Information Theory* (Cover and Thomas 1991).

The fundamental premise of Information Theory is that the generation of information can be modeled as a probabilistic process that can be measured so that the measure value agrees with our intuition. The information associated to a given event should thus fulfill the following conditions: (i) it should be a positive value, (ii) the probability of the event should be related to the amount of information contained in this event, and (iii) two indepen-

dent events should contain the same amount of information when they appear jointly or separately.

This way, a random event E with probability $P(E)$ is said to contain

$$I(E) = \log \frac{1}{P(E)} = -\log P(E) \quad (8.21)$$

units of information. If the base of the logarithm is 2, this information is measured in bits. Note that, if the $P(E) = 1$ or $P(E) = 0$, the event does not contain any information ($I(E) = 0$); that is, if the event is sure to happen or impossible, its associated information is null. Moreover, if $P(E) = 0.5$, the information conveyed by the event is maximum and equal to 1 bit.

Let us consider now a source X of statistically independent events (symbols) $\{x_1, x_2, \dots, x_M\}$ with associated probabilities $\{P(x_1), P(x_2), \dots, P(x_M)\}$. The average information of that source, named *entropy* and denoted by $H(X)$, is given by

$$H(X) = -\sum_{i=1}^M P(x_i) \log P(x_i) \quad (8.22)$$

The entropy measures the uncertainty of the source; that is, the higher the entropy, the more uncertain the source and, therefore, the more information it conveys. It can be proven that, for a source of equiprobable symbols, the entropy is maximal; that is, if all symbols have the same probability of occurrence, the uncertainty about which symbol will be produced by the source is maximal (Cover and Thomas 1991).

The goal of an entropy encoder is to assign a codeword c_x from a codebook C to every symbol $\{x_1, x_2, \dots, x_M\}$, where c_x is a string of $|c_x|$ bits. The assignment has to be performed so that the average length of the produced codewords R is minimal:

$$R = \sum_{m=1}^M P(x_m) |c_m| \quad (8.23)$$

Shannon's first theorem (the *noiseless coding theorem*) assures that the entropy of a source is the lowest bound of the average number of bits necessary to code the symbols produced by the source. Therefore, no coder can produce codewords whose average length is smaller than the entropy of the source. The difference between the attained average length and the entropy

of the source is called the redundancy of a code. It can be demonstrated (Cover and Thomas 1991) that, for a source that generates independent symbols (a so-called *zero-memory* source) with entropy $H(X)$, it is possible to find a code whose mean length R is

$$H(X) \leq R \leq H(X) + 1 \quad (8.24)$$

Moreover, such codes satisfy the so-called *prefix condition*; that is, no codeword is a prefix of any other codeword in the set. This is a very relevant property of a code, since no additional symbols are required in the stream to mark the separation between codes.

Several codes have been proposed in the literature to generate a set of codewords satisfying the previous condition. Among them, Huffman codes are the most popular ones. Huffman codes provide an optimal code, under the constraint that symbols are coded one at a time. Moreover, the prefix condition allows the use of a simple Look Up Table to implement the Huffman decoder (Cover and Thomas 1991).

Huffman codes require the previous knowledge of the probability density function of the source⁸. The algorithm for constructing a Huffman code has two main steps. In a first step, a series of source reductions is created by combining the two symbols with lowest probability into a single symbol. This compound symbol and its associated probability are used in a new iteration of the source reduction process, as illustrated in Fig. 8.9.

The process is repeated until a single compound symbol representing the complete source is obtained. In a second step, each reduced source is coded, starting by the smallest source (a source with only two symbols: in the example in Fig. 8.9, the *a1* symbol and the compound symbol representing the combination of the remaining original symbols: *a2 ... a8*). The codewords 0 and 1 are arbitrarily assigned to each one of the two symbols. These codewords propagate as prefixes to the two symbols that have been used to create the compound symbol, and the process is iterated at each branch of the tree. In the example of Fig. 8.9, the complete Huffman code for the given source is presented. Note that the entropy of the source is $H(X) = 2.48$ bits while the mean length is $R = 2.55$ bits (and, therefore, the redundancy is only 0.07 bits).

⁸ There exist extensions of the basic Huffman code that iteratively estimate the probability density function and can even adapt to its variations.

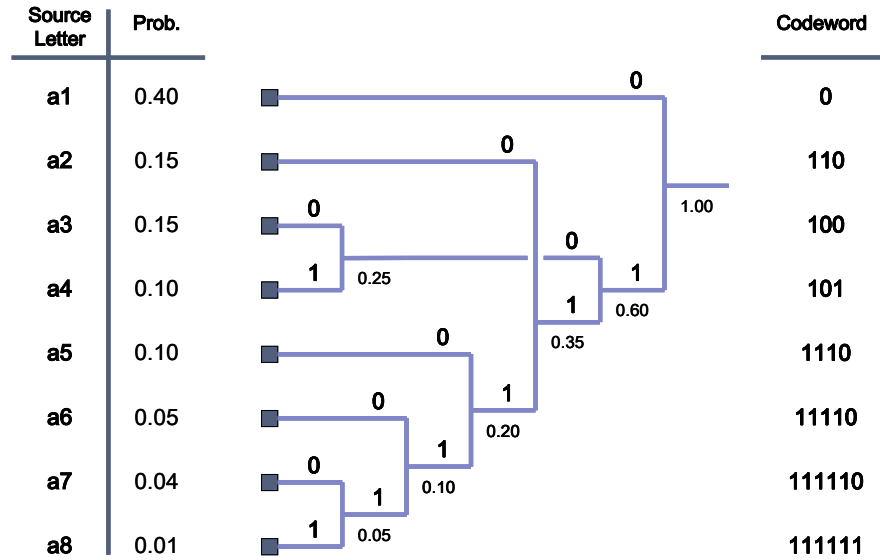


Fig. 8.9 Example of construction of a Huffman code.

1.1.4 A few specificities of the JPEG standard

Fig. 8.10 presents a block diagram of the main parts of the JPEG standard (Pennebaker and Mitchell 1993). Note that, the basic tools used in the standard are those that have been discussed above. Nevertheless, there are a few specificities of the standard that require a more concrete explanation.

Prior to the frequency transformation, a level offset is subtracted from every image sample. If the image is represented by B bits ($B = 8$ or $B = 12$ in the JPEG standard), the level offset is equal to 2^{B-1} to ensure that the output signal is a signed quantity in the range -2^{B-1} to $2^{B-1}-1$. This is done to ensure that all DCT coefficients are signed quantities with a similar dynamic range.

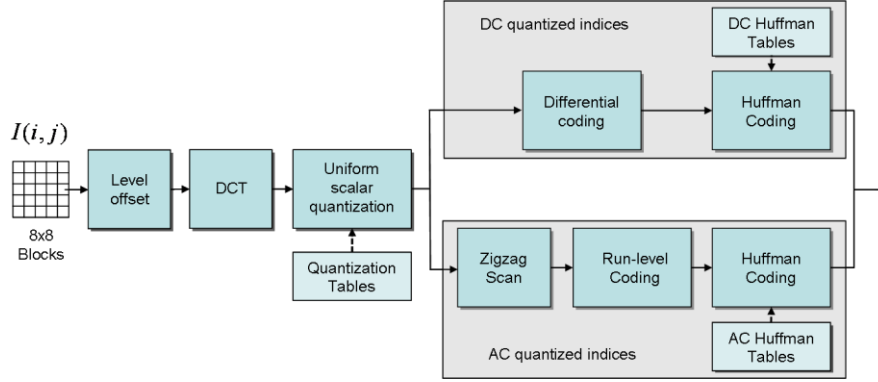


Fig. 8.10 Basic scheme of the JPEG encoder.

The scalar quantization in the JPEG standard is implemented by using a quantization matrix (the so-called *Q Table*: $Q(k,l)$). Quantization is defined as the division of each DCT coefficient by its corresponding quantizer step size. Each component of the Q Table contains its associated quantizer step size. The final quantized coefficient is obtained by rounding the value of the division to the nearest integer:

$$\hat{y}(k,l) = \text{round} \left[\frac{y(k,l)}{Q(k,l)} \right] \quad (8.25)$$

The standard does not specify the concrete values of the Q Table but proposes the use of Lohscheller Tables (Lohscheller 1984). These tables have been obtained by analyzing the relative relevancy of the different DCT coefficients for the human visual system.

Although the JPEG standard allows using Arithmetic coding (Cover and Thomas 1991), the most common implementations apply Huffman Tables. As shown in Fig. 8.10, the DC coefficient is handled separately from the AC coefficients. In the case of the DC coefficient, a differential coding is initially applied and the prediction errors (named DIFF values) are the symbols that are entropy coded. In turn, in the case of the AC coefficients, they are initially scanned following the zigzag pattern presented in Fig. 8.8. This sequence of AC coefficients is then described in a new manner: each nonzero AC coefficient is represented in combination with the “runlength” (consecutive number) of zero-valued AC coefficients which precede it in the zig-zag sequence.

This way, there are two different sources of symbols: the DIFF values for the DC coefficients and the combination runlength/nonzero-coefficient for the AC coefficients. In JPEG, the DIFF values and the nonzero AC coefficients are initially categorized in order to represent them by a pair of symbols: SIZE and AMPLITUDE. SIZE is the index of the category that gives a possible range of values and AMPLITUDE is the offset within this category. This allows reducing the sizes of the Huffman tables while still providing very efficient entropy encoding.

The category (SIZE) is encoded using variable-length codes (VLC) from a Huffman table. For the DC case, a simple Huffman table is created where the entries are the different categories. For the AC case, the entries are the pairs runlength/category. Finally, the AMPLITUDE information is encoded using a variable-length integer (VLI) code. VLI are variable length codes as well, but they are not Huffman codes (and, therefore, they do not share all their properties). An important distinction is that the length of a VLC (Huffman code) is not known until it is decoded, but the length of a VLI is stored in its preceding VLC. The complete JPEG tables are not presented in this text and the reader is referred to the standard definition in order to obtain them (JPEG 1994).

Table 8.1 Categorization tables.

SIZE	DIFF values	AC Coefficients
0	0	
1	-1,1	-1,1
2	-3,-2,2,3	-3,-2,2,3
3	-7,-4,4,7	-7,-4,4,7
4	-15,-8,8,15	-15,-8,8,15
5	-31,-16,16,31	-31,-16,16,31
6	-63,-32,32,63	-63,-32,32,63
7	-127,-64,64,127	-127,-64,64,127
8	-255,-128,128,255	-255,-128,128,255
9	-511,-256,256,511	-511,-256,256,511
10	-1 023,-512,512,1 023	-1 023,-512,512,1 023
11	-2 047,-1 024,1 024,2 047	

1.1.5 Quality measures

Finally, in order to conclude this theoretical part of the Chapter, let us introduce the main measures that are commonly used in image compression to assess the quality of the reconstructed images.

The most common measure of the distortion in image compression is the Mean Square Error (MSE), defined as:

$$MSE = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \|x(m,n) - \hat{x}(m,n)\|^2 \quad (8.26)$$

where $\hat{x}(m,n)$ is the reconstructed image. This measure is often given in terms of the equivalent reciprocal measure, the *Peak Signal-to-Noise Ratio* (PSNR):

$$PSNR = 10 \log_{10} \left(\frac{(2^B - 1)^2}{MSE} \right) dB \quad (8.27)$$

where B is the number of bits used to represent the pixel values in the image. The PSNR is expressed in decibels (dB) and good reconstructed images typically have PSNR values of 30 dB or more.

Although these measures do not take the human visual system characteristics into account, they are commonly used since they give an approximated idea of the visual quality while being easy to compute and tractable in linear optimization problems.

1.2 MATLAB proof of concept

In this Section we are going to illustrate the main concepts behind the JPEG standard. These concepts are applied as well in video compression in the so-called hybrid approach which is the basis for standards such as MPEG1 or MPEG2 and, therefore, these concepts will be revisited in Chapter 9. Note that, in these two Chapters, we will not concentrate on the specificities of any of these standards but on the common, generic tools and concepts that they use. Moreover, in order to illustrate these concepts, we will work mainly with grey level images and explain the extensions that should be introduced in the proposed systems to work with color images.

Let us start by reading a grey level image. In our case, this is the first image of a video sequence that, in this Chapter, we will use it as an example to study the case of still (isolated) image coding and of the exploitation of the spatial redundancy. Moreover, in Chapter 9, it will be used as the initial point for the study of video sequence coding and of the exploitation of both spatial and temporal redundancies.

First we will study how the spatial redundancy can be exploited to reduce the number of bits needed to represent the image. Let us load file 'table_000_g.bmp' from disk and show its pixel values (**Erreur ! Source du renvoi introuvable.**). This file contains a gray image corresponding to frame #000 of the *Table Tennis* sequence with size 176x144 pixels⁹ stored in non compressed form.

```
table = imread('seqs/table/table_000_g.bmp');  
imshow(table);
```



Fig. 8.11 Table Tennis frame #000

1.2.1 Block image transformation

Image block processing

Spatial redundancy is exploited by performing a local analysis of the image; that is, by dividing the image into non-overlapping square blocks. Then, the information contained in each of these blocks will be processed separately. Typically, images are partitioned into blocks of 8x8 pixels (Fig. 8.12).

```
addgriddtofigure(size(table),[8 8]);
```

⁹ This image format is usually referred to as QCIF (Quarter Common Intermediate Format).

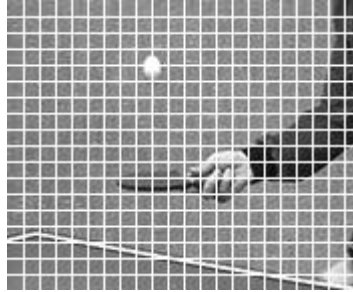


Fig. 8.12 Block processing for *Table Tennis* image

Mirror padding

Given that generic images (images of any size) may not allow an exact division in 8x8 pixel blocks, information in the right and low boundary blocks is usually padded before partitioning the image. A common way to pad this information is by mirroring it (which reduces the transitions introduced when padding with, for instance, zero values). Let us see this case with a new version of the previous *Table Tennis* image where 4 rows and 4 columns have been removed from the bottom and right sides of the original image (Fig. 8.13).

```
table_crop = table(1:end-4,1:end-4);  
table_padded = padarray(table_crop,[4,4],'symmetric','post');  
imshow(table_padded);
```



Fig. 8.13 Mirror padding for *Table Tennis* image

If we analyze one of the mirrored blocks, the effect of the mirroring can be easily seen (Fig. 8.14). For instance, the block in the first row and last column shows how the mirror padding is created along the vertical axis. Note as well how this padding preserves the original block contours.

```
imshow(table_padded(1:8,end-7:end));
```



Fig. 8.14 Mirror padded block

DCT block transformation

Now, we are going to analyze how our basic coding unit (that is, a generic 8x8 pixel block which, in the sequel, we will refer to as a *block*) is processed. Among all the different transforms that can be applied on an image block to obtain a set of less correlated coefficients in the transformed domain, the Discrete Cosine Transform (DCT) has been shown to present very good properties (see the discussion in the previous Section).

Let us analyze one block from the *Table Tennis* image to illustrate the whole transform process. First we select a given block from the image, for example, the block situated 12 blocks from the top and 13 from the left:

```
posr = 12; posc = 13;
f = table*0.3;
f(8*(posr-1)+1:8*(posr-1)+8,8*(posc-1)+1:8*(posc-1)+8) = ...
    table(8*(posr-1)+1:8*(posr-1)+8,...
        8*(posc-1)+1:8*(posc-1)+8);
imshow(f);
```



Fig. 8.15 Selected block (highlighted)

The following image shows a magnification of the selected block. Also, the corresponding matrix of pixel values is shown:


```
b = table(8*(posr-1)+1:8*(posr-1)+8,...
          8*(posc-1)+1:8*(posc-1)+8)
```



Fig. 8.16 Magnification of the selected block

```
b =
    73    89    90    93   179   241   227   186
   122   124    98   155   232   223   208   183
    83    69    92   198   199   183   181   153
    31    28   137   226   179   148   134   114
    15    50   184   211   180   140   110    81
    20    73   162   158   133   120   107    60
    32   102   151   135   112   107    87    47
    60   132   162   127   117   100    68    41
```

The MATLAB function `dct2` performs the DCT of a given image block. The transformed matrix (that is, the matrix containing the DCT coefficient values) for the block under study is

```
d = dct2(double(b)-128),
```

```
d =
 -16.25 -160.49 -254.84 -12.36 -24.00  26.22 -0.51  7.40
 181.21 -184.67  69.58 109.83  46.68 -4.86 -15.38 -6.91
  11.36  13.69 117.01  62.23 -62.06 -61.07 -11.17  3.44
 -24.09 -24.65 -1.88 -8.66 -49.69 -5.43 19.46 17.67
 -19.50 -5.18 -26.64 -23.48 -7.75  5.99 23.21  5.81
 -47.70 -26.86  5.86 -8.08 -6.08 19.95  3.30 -10.85
 -19.59 -11.74 10.82  1.97 -6.76  9.26 -0.01 -2.29
 -0.12 -3.55 -4.60 -0.43 -3.76 -0.46  1.19 -0.61
```

where as explained in the previous Section a value of 128 has been subtracted to all pixels of block `b` to create a signal with zero mean and obtain a lower energy DC coefficient.

We can observe the capability of the DCT to compact the energy of the signal with respect to the energy distribution in the original block. Fig. 8.17 shows the absolute value of the DCT coefficients (sorted in descending order):

```
v = sort(abs(d(:)),1,'descend');
plot(v);
```

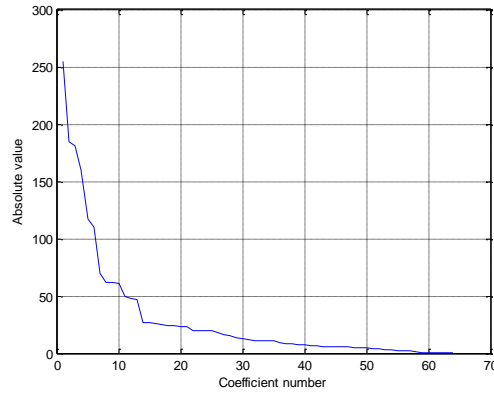


Fig. 8.17 Sorted DCT coefficients

If we plot the percentage of accumulated energy for these coefficients (see Fig. 8.18), it can be noted that, for this specific block, 95% of the energy of the signal in the transform domain is contained in the 13 coefficients with highest energy.

```
dct_energy_distribution = cumsum(v.*v)/sum(v.*v);
```

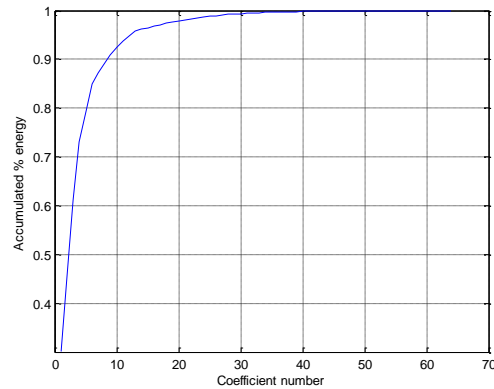


Fig. 8.18 Accumulated percentage of energy for the sorted DCT coefficients

However, in the original signal (the block b), the same percentage of energy is reached when adding up the contributions of the 47 pixels with the highest energy (as shown in Fig. 8.19).

```

bv = sort(double(b(:)),1,'descend');
ima_energy_distribution = cumsum(bv.*bv)/sum(bv.*bv);

```

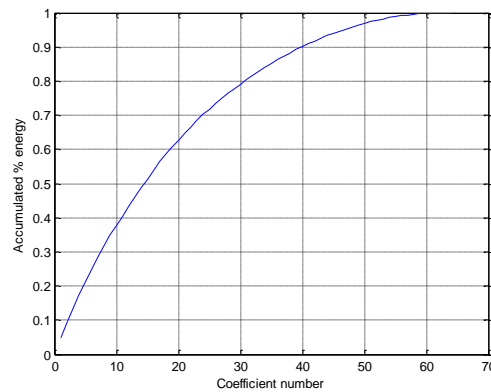


Fig. 8.19 Accumulated percentage of energy for the sorted pixels

Moreover note that, roughly speaking, coefficients in the DCT domain with higher energy are gathered around the left-top corner of matrix *d* (lower frequency coefficients). This is a common behavior for all natural image block DCT coefficient matrices, which will be further exploited in the coding process when including the quantization step.

The DCT transformation is invertible and we can recover the initial pixel values of the original block *b* using the inverse DCT transformation (function `idct2` in MATLAB):

```

br = uint8(idct2(d)+128);
b = br,

```

```

ans =
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

```

Elimination of DCT coefficients

Now, we can see the effect of zeroing some of the DCT coefficients in the reconstruction of the original block. In Fig. 8.20 we present the result of reconstructing the image block using the first $N = (1, 4, 8, 16, 32, 64)$ coeffi-

cients in the zigzag scan. This strategy to select coefficients is often referred to as “zonal coding”.

```
v = zigzag8(d);
for N=[1 4 8 16 32 48],
    ve = [v(1:N),zeros(1,64-N)];
    dr = izigzag8(ve);
    br = uint8(idct2(dr)+128);
    imshow(br);
end;
```

In turn, in Fig. 8.21 we present the result of using the same number of coefficients as in the example but, in this case, the N coefficients with highest energy are selected directly. This strategy to select coefficients is often referred to as “threshold coding”. Below each image, the percentage of energy corresponding to the N selected coefficients is presented.

```
d2 = d(:);
[v,i] = sort(abs(d2),1,'descend');
v = d2(i); ii(i) = 1:64;
dct_energy_distribution = cumsum(v.*v)/sum(v.*v);
for N=[1 4 8 16 32 48],
    ve = [v(1:N);zeros(64-N,1)];
    veu = ve(ii);
    dr2 = reshape(veu,8,8);
    br = uint8(idct2(dr2)+128);
    imshow(kron(br,uint8(ones(16))));
    title(['N=' int2str(N) ' (~'...
    int2str(round(dct_energy_distribution(N)*100)) '%)']);
end;
```

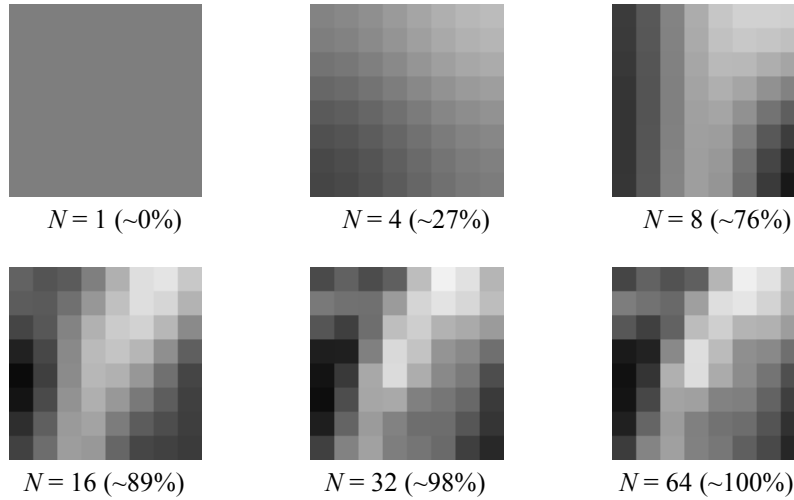


Fig. 8.20 Reconstructed blocks using the first N coefficients with a zig-zag scan (in brackets the % of energy corresponding to the N selected coefficients)

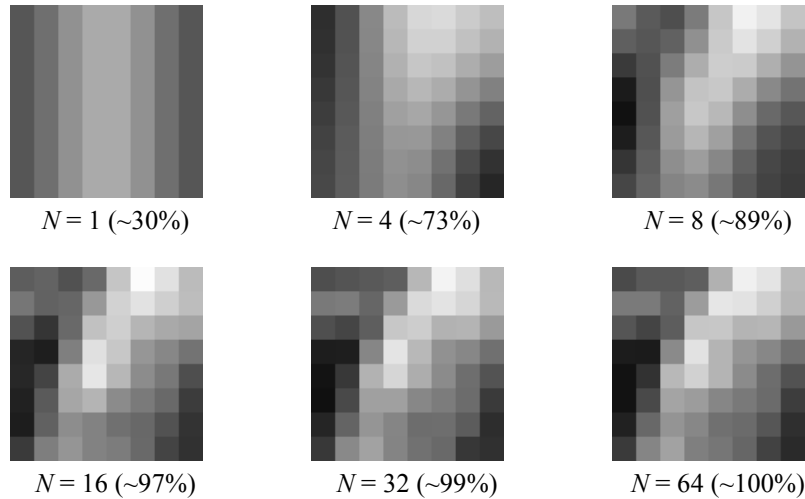


Fig. 8.21 Reconstructed blocks using the first N coefficients with higher energy (in brackets the % of energy corresponding to the N selected coefficients)

1.2.2 Complete image block coding

In order to draw more general conclusions, we are going to analyze some of these features in a whole image. Matrix E_d presents the energy distribution per DCT coefficient averaged among all the blocks in the image. As it can be seen, the previous behavior is preserved; that is, coefficients with largest energy are those of lowest frequency (top-left matrix corner).

```
D = blkproc(double(table)-128,[8 8],@dct2);
Ed = blkmean(D.*D,[8 8]);
Ed = round(Ed/sum(sum(Ed))*100),
```

```
Ed =
59    6    1    1    0    0    0    0
 7    2    1    0    0    0    0    0
 4    2    0    0    0    0    0    0
 4    1    0    0    0    0    0    0
 3    2    0    0    0    0    0    0
 2    1    0    0    0    0    0    0
 1    1    0    0    0    0    0    0
 0    0    0    0    0    0    0    0
```

The effect of dropping some DCT coefficients when reconstructing the complete image (the so-called *Block Effect*) can be observed in the following two Figures. Since an image independent block partition has been imposed and blocks have been separately coded, block boundaries are noticea-

ble in the decoded image. This effect is more remarkable when fewer coefficients are used. In Fig. 8.22, we present six different versions of the original *Table Tennis* image: keeping $N = (1, 4, 8, 16, 32, 64)$ coefficients at each block, respectively.

```
for N=[1 4 8 16 32 64],
    Dk = blkproc(D,[8 8],@coeffs_keep_zigzag,N);
    tabler = uint8(blkproc(Dk,[8 8],@idct2)+128);
    imshow(tabler);
end;
```

In Fig. 8.23 we present the result of using the same number of coefficients as in the previous example, but, in this case, those coefficients with higher energy are selected (“threshold coding”). For each number of coefficients N , the % of energy selected is shown.

```
dct_total_energy = sum(sum(D.^2));
for N=[1 4 8 16 32 64],
    Dk = blkproc(D,[8 8],@coeffs_keep_higher,N);
    dct_energy = sum(sum(Dk.^2));
    tabler = uint8(blkproc(Dk,[8 8],@idct2)+128);
    imshow(tabler);
    title(['N=' int2str(N) ' (~' int2str(round(dct_energy ...
        / dct_total_energy * 100)) '%')]);
end;
```

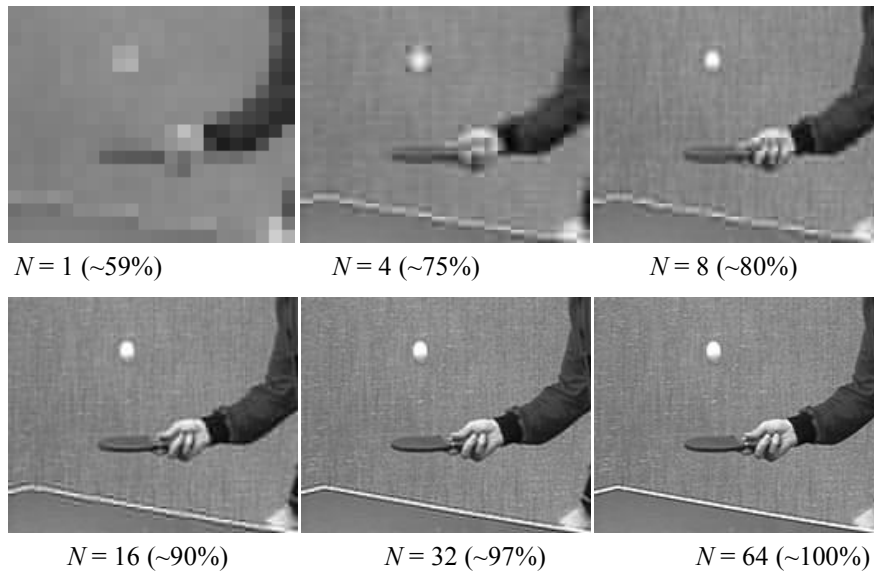


Fig. 8.22 Reconstructed images using the first N coefficients for each block with a zigzag scan (in brackets the % of energy corresponding to the N selected coefficients)

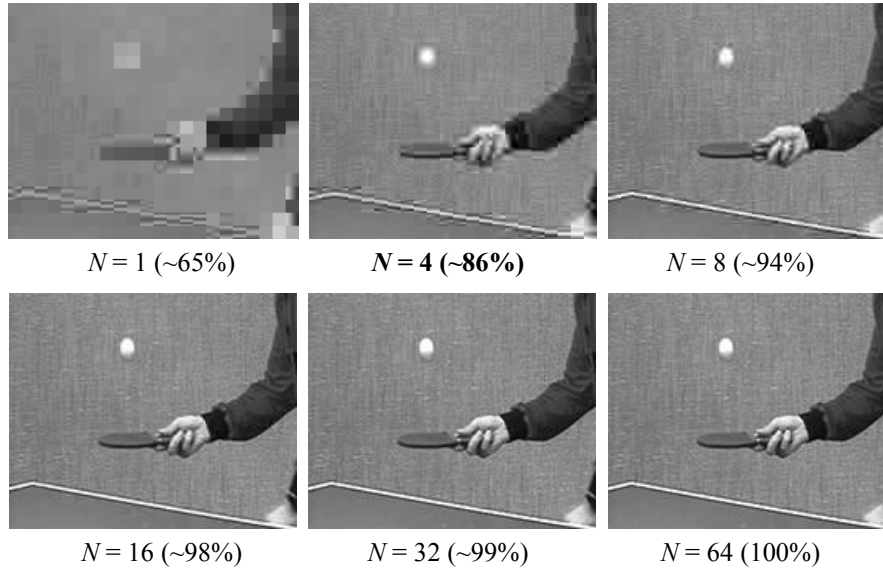


Fig. 8.23 Reconstructed images using the first N coefficients with higher energy (in brackets the % of energy corresponding to the N selected coefficients)

The analysis of the previous examples leads to the following comments:

- Increasing the same amount of coefficients/energy in all blocks does not translate into the same increase of visual quality for all blocks. This is due to the fact that not all DCT coefficients are equally relevant for the human visual system. This concept will be further exploited in the sequel to adapt the quantization of the DCT coefficients to the final human observer.
- In order to obtain a homogenous image quality (for instance, a given global PSNR), a different amount of coefficients/energy (and, eventually, bits) can be assigned to the various blocks. Therefore, given a bit budget for a specific image, those bits can be shared among the blocks in a content dependent manner. That is, more/fewer bits are used for those blocks containing more/less complex information. This concept is the basis for the *Rate-Distortion Theory* that will be further addressed in Chapter 10.

1.2.3 DCT quantization

As commented in the previous Section, DCT coefficients need to be quantized in order to generate a discrete source whose symbols can be afterwards

entropy encoded. In the current case, a separated scalar, uniform quantization is used for each DCT coefficient.

The different quantization of each coefficient is represented by the so-called Quantization Table (Q Table). This is a matrix containing at position (i, j) the value to which the DCT coefficient (i, j) should be compared to generate its associated quantization index¹⁰. A possible Q Table was proposed by Lohscheller¹¹ (Lohscheller 1984) and it is presented as matrix Q . As it can be seen, roughly speaking, the higher the DCT coefficient frequency, the larger the associated value in the Q Table. This table is the results of psychovisual experiments that have determined the relative relevance of the different DCT coefficients for the human visual system.

Q = jpegsteps,

$$Q = \begin{matrix} & \begin{matrix} 8 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \end{matrix} \\ \begin{matrix} 12 \\ 14 \\ 14 \\ 18 \\ 24 \\ 49 \\ 72 \end{matrix} & \begin{matrix} 12 \\ 13 \\ 17 \\ 22 \\ 35 \\ 64 \\ 92 \end{matrix} & \begin{matrix} 14 \\ 16 \\ 22 \\ 37 \\ 55 \\ 78 \\ 95 \end{matrix} & \begin{matrix} 19 \\ 24 \\ 29 \\ 56 \\ 64 \\ 87 \\ 98 \end{matrix} & \begin{matrix} 26 \\ 40 \\ 51 \\ 68 \\ 81 \\ 103 \\ 112 \end{matrix} & \begin{matrix} 58 \\ 57 \\ 87 \\ 109 \\ 104 \\ 121 \\ 100 \end{matrix} & \begin{matrix} 60 \\ 69 \\ 80 \\ 103 \\ 113 \\ 120 \\ 103 \end{matrix} & \begin{matrix} 55 \\ 56 \\ 62 \\ 77 \\ 92 \\ 101 \\ 99 \end{matrix} \end{matrix}$$

The comparison between a given DCT coefficient and its associated value in the Q Table is carried out as expressed in the following code in which, for illustration purposes, we are using the same block we have used in the previous subsection (see Fig. 8.16):

dq = round(d./Q),

$$dq = \begin{matrix} & \begin{matrix} -2 & -15 & -25 & -1 & -1 & 1 & 0 & 0 \end{matrix} \\ \begin{matrix} 15 \\ 1 \\ -2 \end{matrix} & \begin{matrix} -15 \\ 1 \\ -1 \end{matrix} & \begin{matrix} -25 \\ 5 \\ 7 \\ 0 \end{matrix} & \begin{matrix} -1 \\ 6 \\ 3 \\ 0 \end{matrix} & \begin{matrix} -1 \\ 2 \\ -2 \\ -1 \end{matrix} & \begin{matrix} 1 \\ 0 \\ -1 \\ 0 \end{matrix} & \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix} & \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix} \end{matrix}$$

¹⁰ There is a difference in the way in which DCT coefficients are commonly indexed and their MATLAB implementation. DCT coefficients of an 8x8 pixel block are usually indexed from 0 to 7 in both dimensions with the first index indicating the desired column and the second the desired row. In turn, the coordinates of a MATLAB matrix are indexed from 1 to 8 in both dimensions with the first index indicating the desired row and the second the desired column.

¹¹ Lohscheller Tables have been adopted in the JPEG and MPEG standards as default proposals, although, in these standards, different tables can be set by the encoder and included in the bit stream. In the case of JPEG, there is a slight variation in the $Q(1, 1)$ value since, due to implementation features, the implemented DCT transform is not unitary.

-1	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

For example, a coefficient $d(3,1)$ smaller than 5 implies that the coefficient is zeroed after quantization ($Q(3,1) = 10$) whereas coefficient $d(6,8)$ has to be smaller than 50 in order to be zeroed ($Q(6,8) = 100$).

Different coding qualities can be obtained by multiplying the values in the Q Table by a given constant, k , producing a new Q Table: kQ . The following MATLAB code presents the quantization of the block b using $k = 3$ and $k = 5$.

```
k=3;
k.*Q,round(d./k./Q),
```

```
ans =
24    33    30    48    72    120    153    183
36    36    42    57    78    174    180    165
42    39    48    72    120    171    207    168
42    51    66    87    153    261    240    186
54    66    111    168    204    327    309    231
72    105    165    192    243    312    339    276
147    192    234    261    309    363    360    303
216    276    285    294    336    300    309    297
```

```
ans =
-1    -5    -8     0     0     0     0     0
5     -5     2     2     1     0     0     0
0     0     2     1    -1     0     0     0
-1     0     0     0     0     0     0     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     0     0     0     0     0     0
```

```
k=5;
k.*Q,round(d./k./Q),
```

```
ans =
40    55    50    80    120    200    255    305
60    60    70    95    130    290    300    275
70    65    80    120    200    285    345    280
70    85    110    145    255    435    400    310
90    110    185    280    340    545    515    385
120    175    275    320    405    520    565    460
245    320    390    435    515    605    600    505
360    460    475    490    560    500    515    495
```

```
ans =
0    -3    -5     0     0     0     0     0
3    -3     1     1     0     0     0     0
0     0     1     1     0     0     0     0
0     0     0     0     0     0     0     0
0     0     0     0     0     0     0     0
0     0     0     0     0     0     0     0
0     0     0     0     0     0     0     0
```

0 0 0 0 0 0 0 0

In turn, Fig. 8.24 presents the reconstruction of the *Table Tennis* image when quantizing the DCT coefficients of its blocks using $k = 1$, $k = 3$ and $k = 5$. In our implementation, the MATLAB functions `quantizedct8` and `iquantizedct8` perform the (direct and inverse) quantization of the DCT coefficients of a given matrix. Moreover, they are responsible of clipping the DC value between $[0, 255]$ and AC values between $[-128, 127]$ ¹².

```
tabledct = blkproc(table, [8 8], @dct2);
for k=[1 3 5],
    tabledctq = blkproc(tabledct, [8 8], @quantizedct8, k.*Q);
    tabledctqi = blkproc(tabledctq, [8 8], @iquantizedct8, k.*Q);
    tabler = uint8(blkproc(tabledctqi, [8 8], @idct2));
    imshow(tabler);
end;
```

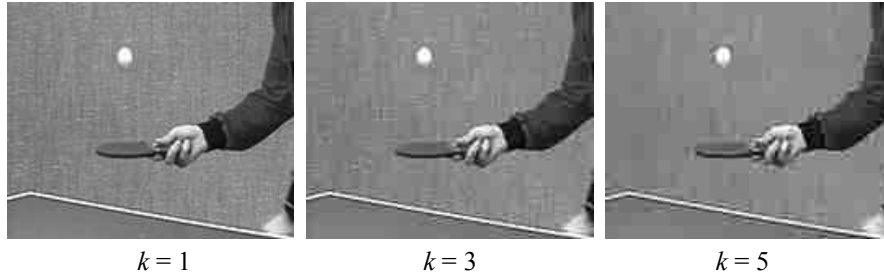


Fig. 8.24 Reconstructed images using various quantization tables

1.2.4 Spatial decorrelation between blocks

So far, we have decorrelated the information in the image only using the DCT and this has been done grouping the data into blocks of 8×8 pixels. Since this grouping is independent of the image content, it is likely that collocated transformed coefficients in neighbor blocks will still be correlated. If such a correlation is present, an additional transformation can be applied on the DCT coefficients to further reduce it. To illustrate this correlation, the next Figures present the 2-Dimensional distribution of collocated DCT coefficients in consecutive blocks. To ensure that the blocks that are compared are neighbors, blocks are ordered following the zigzag scan.

¹² Note that, in the proposed implementation, the value $Q(1, 1) = 8$ is used and there is no subtraction of 128 to the pixel values before computing the DCT transform. Therefore, the DC coefficient can be directly quantized in the range $[0, 255]$.

In order to create more meaningful 2-Dimensional histograms (that is, to have more data), we use a 352x288 pixels image. This image format is usual known as Common Intermediate Format (CIF). Moreover, we have selected the first image of the *Foreman* sequence instead of the *Table Tennis* one for this specific example. The *Table Tennis* image has a large number of blocks with similar DC values (all blocks corresponding to the wall in the background), which prevents from correctly illustrating the spatial decorrelation effect. On the contrary, the *Foreman* sequence (see Fig. 8.25) presents more variation of the DC values as blocks are less homogeneous. Fig. 8.26, Fig. 8.27, and Fig. 8.28 show the distribution of the values of DCT coefficients in positions (1,1), (2,1) and (4,4) for consecutive blocks in the zigzag scan, respectively.



Fig. 8.25 *Foreman* CIF original image

```
k = 1;
im = imread('seqs/foreman/foreman_cif_000_g.bmp');
imdct = blkproc(im,[8 8],@dct2);
imdctq = blkproc(imdct,[8 8],@quantizedct8,k*Q);
pos = {[1 1],[2 1],[1 3]};
for p = 1:length(pos),
    coeffsm = imdct(pos{p}(1):8:end,pos{p}(2):8:end);
    coeffs = zigzag(coeffsm);
    figure,hold on,grid
    for i=2:length(coeffs),
        plot(coeffs(i-1),coeffs(i),'o');
    end;
end;
```

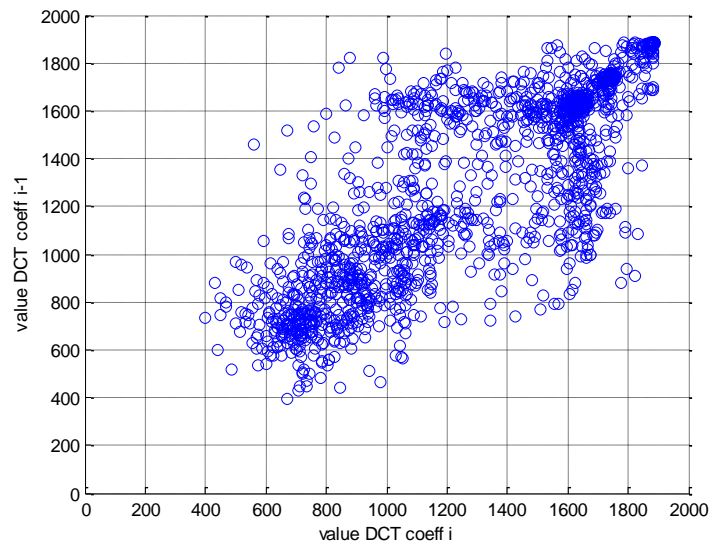


Fig. 8.26 2-Dimensional distribution of consecutive DC coefficients (zig-zag scan)

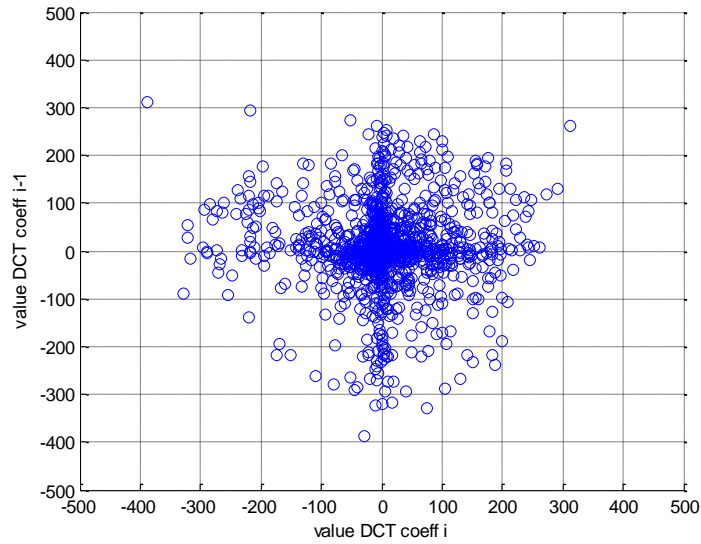


Fig. 8.27 2-Dimensional distribution of AC coefficients in position (2,1) in consecutive blocks (zig-zag scan)

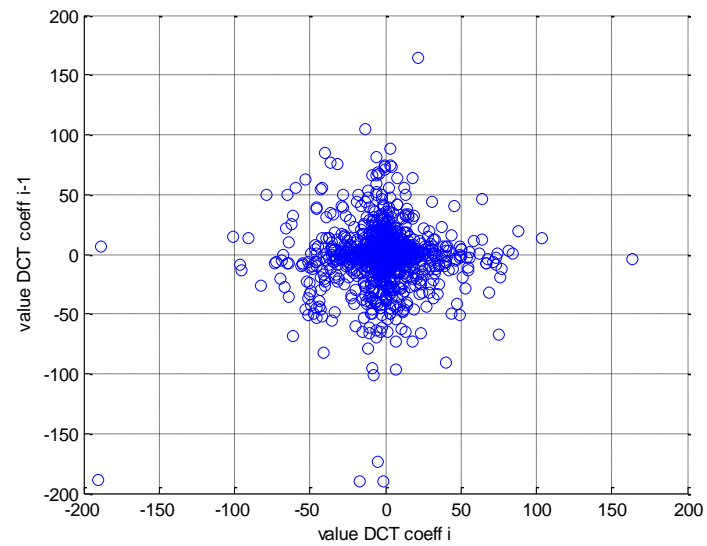


Fig. 8.28 2-Dimensional distribution of AC coefficients in position (1,3) in consecutive blocks (zig-zag scan)

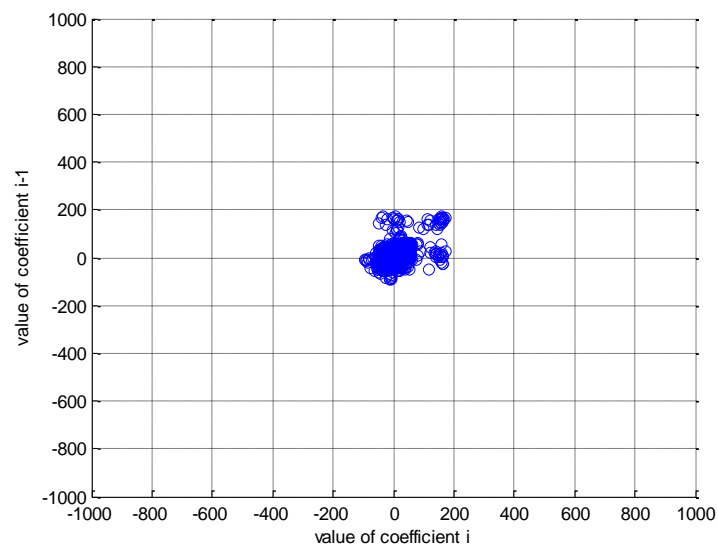


Fig. 8.29 2-Dimensional distribution of consecutive DC coefficients (zig-zag scan) after decorrelation

Note that the highest correlation appears in the case of the DC coefficient (at position (1,1)) since the points of the distribution are placed near the diagonal. This correlation is commonly exploited by using a prediction step to represent the DC values of consecutive blocks. This way, quantized DC coefficients are losslessly coded using a Differential Pulse Code Modulation (DPCM) technique. In the proposed implementation, the DC coefficient is predicted by the mean value of the DC coefficients of the 3 blocks above and the block on the left (Fig. 8.29).

```
coeffsdc = imdctq(1:8:end,1:8:end);
h = [-0.25 -0.25 -0.25; -0.25 1 0; 0 0 0];
coeffsdc_decorrelated = filter2(h, coeffsdc);
coeffs = zigzag(coeffsdc_decorrelated);
for i=2:length(coeffs),
    plot(coeffs(i-1),coeffs(i),'o');
end;
```

1.2.5 Entropy coding

At this moment, we have two different sets of quantization indices or symbols: those related to the prediction error of the decorrelated quantized DC coefficients and those associated to the zigzag ordered AC coefficients.

The JPEG standard specifies that these coefficients are entropy encoded through either Huffman or Arithmetic encoding. Commonly, Huffman encoding is used and, thus, this is the approach that we discuss in this Section.

Let us see the entropy coding in the proposed MATLAB implementation where Huffman coding is used. Let us start as an example with the quantized DCT coefficients of our selected block *b* (using $k = 1$).

```
k = 1;
dq = round(dct2(b)./k./Q),
```

```
dq =
126   -15   -25    -1    -1     1     0     0
 15   -15     5     6     2     0     0     0
  1     1     7     3    -2    -1     0     0
 -2    -1     0     0    -1     0     0     0
 -1     0    -1     0     0     0     0     0
 -2    -1     0     0     0     0     0     0
  0     0     0     0     0     0     0     0
  0     0     0     0     0     0     0     0
```

As explained earlier in this Chapter, the DC coefficient is decorrelated with previous DC values and encoded separately. In the case of the AC coefficients, as it can be seen in the example above, there are many AC

```
dc = dq(1,1);
ac = zigzag(dq); ac = ac(2:end);
[nz vv] = runlength(ac,63);
```

```
load('codes/imcodes.mat');
codeddcd = encodemessage(dc+256', dcdcode),
```

The runlength encoding of AC coefficients is (again, 2 is added to the runlength run values to create indexes as the dynamic range is $[-1, 62]$ and 129 is added to runlength values with dynamic range $[-128, 128]$):

```

codedvv =
001011111001101001100010111110011011110001001001100000010100100000011
100110000111110000010100000101

```

```

tabledct = blkproc(table, [8 8], @dct2);
tabledctq = blkproc(tabledct, [8 8], @quantizedct8, k.*Q);
zzdctq = double(blkproc(tabledctq, [8 8], @zigzag8));

% Separate DC from AC components of the DCT.
[zznrows zzncolumns] = size(zzdctq);
mask = true([zznrows zzncolumns]);
mask(:, 1 : 64 : end) = false;

```

```

dc = zzdctq(~mask);
ac = reshape(zzdctq(mask), ...
    [zznrows, zzncolumns - zzncolumns / 64])' - 128;

% Decorrelate DC (add 256 to directly create index)
h = [-0.25 -0.25 -0.25; -0.25 1 0; 0 0 0];
dc = reshape(dc, size(table) / 8);
dcdindex = floor(filter2(h, dc)) + 256;

% Get NZ and VV sequence of AC component.
[nz vv] = runlength(ac(:)', 63);

% Encode DC, ACNZ and ACVV components with the Huffman code.
codeddcd = encodemessage(dcdindex(:)', dcdcode);
codednz = encodemessage(nz(:)' + 2, nzcode);
codedvv = encodemessage(vv(:)' + 129, vvcode);

% Header with codedacnz and codedacvv length
% get the number of bits needed to store the
% maximum possible number of runlength values
% for this image
nbits = ceil(log2(63 * prod(size(table)) / 64));
header = strcat(dec2bin(length(nz), nbits), ...
    dec2bin(length(vv), nbits));

```

After entropy coding, the final bitstream is constructed by concatenating the Huffman coding of DC and AC coefficients. In the example we have also included the header information with the input image size:

```

% Final bitstream
bitstream = strcat(header, codeddcd, codednz, codedvv);

```

In the case of the *Table Tennis* image, the resulting size in bits of the final bitstream corresponds to:

```
final_bits = size(bitstream,2),
```

```
final_bits = 22551
```

which yields the following figure of bits per pixel used in the representation of the decoded grey image:

```
bitapixel = final_bits / (size(table,1)*size(table,2)),
```

```
bitapixel = 0.8898
```

If we compare the obtained bits per pixel with the case of using 8 bits per pixel (when there is no compression), the final compression ratio achieved by our JPEG oriented encoder for the *Table Tennis* image is:

```
compression_ratio = 8 / bitapixel
```

```
compression_ratio = 8.993
```


In order to compare the final compressed image with the original one, the produced bitstream must be decoded. The following code shows how the bitstream can be decoded back into quantized DCT coefficients.

```
% Extract header from codedmessage and calculate number of
% ACNZ and ACVV to decode.
zzsize = size(table) .* [1/8 8];
dcsz = [zzsize(1, 1) ceil(zzsize(1, 2) / 64)];
ndc = prod(dcsz); nac = ndc * 63;
nbits = ceil(log2(nac));
nzlen = bin2dec(bitstream(1 : nbits));
vvlen = bin2dec(bitstream(nbits + 1 : 2 * nbits));
bitstream2 = bitstream(2 * nbits + 1 : end);

load('decoders/im/dcddecoder.mat');
[dcdindex bitstream2] = decodemessage(bitstream2, dcddecoder,
ndc);
load('decoders/im/nzdecoder.mat');
[nzindex bitstream2] = decodemessage(bitstream2, nzdecoder,
nzlen);
load('decoders/im/vvdecoder.mat');
[vvindex bitstream2] = decodemessage(bitstream2, vvdecoder,
vvlen);

% Calculate DCT - DC components from DC differentiated.
dcd = reshape(dcdindex, dcsz) - 256;
dc = zeros(dcsz + [1 2]);
for i = 2 : size(dc,1)
    for j = 2 : size(dc,2) - 1
        dc(i,j) = ceil(dcd(i-1,j-1) + 1/4 * (dc(i, j-1) + dc(i-
1, j-1) + dc(i-1, j) + dc(i-1, j+1)));
    end
end
dc = dc(2 : end, 2 : end - 1);

% Calculate AC coefficients from AC number of zeros (nz) and AC
% values (vv).
ac = irunlength(nzindex - 2, vvindex - 129, 63, nac);
ac = reshape(ac, [nac/zzsize(1, 1) zzsize(1, 1)])' + 128;

% Join DC and AC components into a new DCT.
zzdctq = zeros(zzsize);
mask = true(zzsize);
mask(:, 1 : 64 : end) = false;
zzdctq(~mask) = dc;
zzdctq(mask) = ac(:);

% reconstruct the image
fun = @(x) idct2(iquantizedct8(izigzag8(x),Q));
tabler = uint8(blkproc(zzdctq, [1 64], fun));
```

The PSNR is a good measure of the visual quality of the resulting compressed image (see Section 1.1.5). In the case of the *Table Tennis* image that we have used in all our examples, the final PSNR (for $k = 1$) is:

```
psnr = mypsnr(table,tabler),
```

```
psnr = 32.5078
```

1.2.6 Still image coding

For completeness, the MATLAB function `stillimagegrayencode` implements the total encoding of a single image following the concepts explained in this Chapter. It accepts the matrix of the original image and the factor k to multiply the quantization table. The function returns the reconstructed image, the bitstream with the encoded DCT coefficients and the peak signal-to-noise ratio (PSNR) of the compressed image.

```
[Ir,bits,psnr] = stillimagegrayencode(table,k);
```

Fig. 8.30 presents the result of applying this function to the *Foreman* CIF and *Tennis Table* QCIF original images previously used.

```
im = imread('seqs/foreman/foreman_000_cif.ras');
for k = [1 3 5],
    [Ir,bits,p] = stillimagegrayencode(im,k);
    imshow(Ir);
end;
im = imread('seqs/table/table_000_g.bmp');
for k = [1 3 5],
    [Ir,bits,p] = stillimagegrayencode(im,k);
    imshow(Ir);
end;
```



$k = 1$; PSNR = 35.2 dB
bits/pixel = 0.72
compress. ratio = 11.2



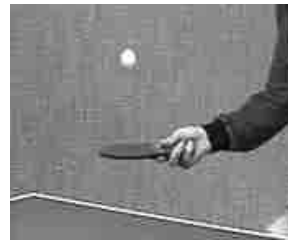
$k = 3$; PSNR = 31.8 dB
bits/pixel = 0.35
compress. ratio = 22.7



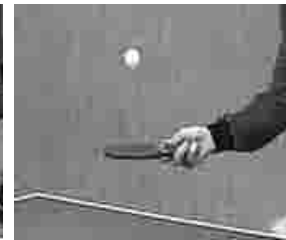
$k = 5$; PSNR = 30.0 dB
bits/pixel = 0.25
compress. ratio = 30.9



$k = 1$; PSNR = 32.5 dB
bits/pixel = 0.89
compress. ratio = 8.99



$k = 3$; PSNR = 29.1 dB
bits/pixel = 0.36
compress. ratio = 22.10



$k = 5$; PSNR = 27.8 dB
bits/pixel = 0.25
compress. ratio = 32.51

Fig. 8.30 Decoded gray scale images of *Foreman* and *Table Tennis* for various values of k

For color images, the same concepts apply to each of the three color channels (in the case of JPEG standard YCbCr color space is commonly used and the chrominance channels are decimated by factor of 2 in horizontal and vertical directions). The MATLAB function `stillimageencode` implements the encoder for color images. Fig. 8.31 shows the resulting compressed images for the same *Table Tennis* (QCIF) and *Foreman* images of previous examples (in QCIF and CIF formats), respectively:

```
table = imread('seqs/table/table_000.bmp');
foreman = imread('seqs/foreman/foreman_000.bmp');
for k=[1 3 5],
    [Ir,bits,p] = stillimageencode(table,k);
    imshow(Ir);
    [Ir,bits,p] = stillimageencode(foreman,k);
    imshow(Ir);
end;
```



$k = 1$; PSNR = 30.0 dB
bits/pixel = 0.95
compress. ratio = 25.1



$k = 3$; PSNR = 27.2 dB
bits/pixel = 0.41
compress. ratio = 59.1



$k = 5$; PSNR = 25.9 dB
bits/pixel = 0.30
compress. ratio = 78.8



$k = 1$; PSNR = 31.3 dB
bits/pixel = 1.12
compress. ratio = 41.4



$k = 3$; PSNR = 27.8 dB
bits/pixel = 0.56
compress. ratio = 42.5



$k = 5$; PSNR = 26.9 dB
bits/pixel = 0.42
compress. ratio = 57.3

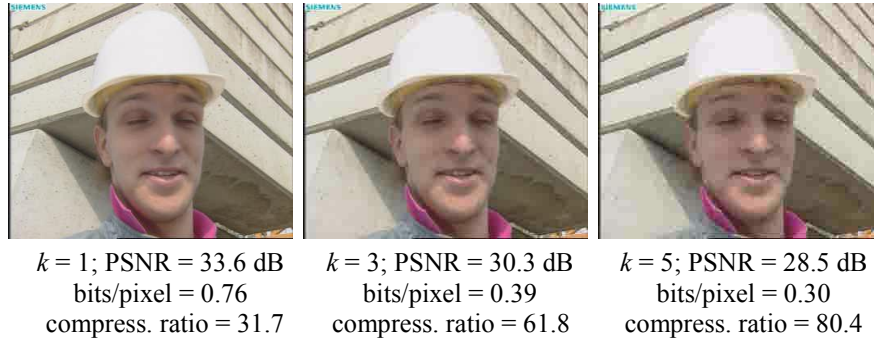


Fig. 8.31 Decoded color images of *Table Tennis* QCIF (first row), *Foreman* QCIF (second row) and *Foreman* CIF (third row) for various values of k

1.3 Going further

What we have presented in this Chapter represents the basic tools and concepts of the so-called baseline JPEG algorithm, known as well as the *JPEG sequential mode*. Nevertheless, the standard proposes some extensions that are worth commenting.

Progressive JPEG allows the compressed image to be incrementally refined through multiple scans as the compressed data arrives. It is a first version of a very useful functionality commonly known as *scalability*. There are two different implementations of progressive JPEG:

- *Successive approximation*, in which the DCT quantized coefficients are successively refined by transmitting initially their more significant bits and subsequently the less significant ones.
- *Spectral selection*, in which the DCT quantized coefficients are progressively sent, initially the DC coefficient and in successive scans the remaining AC coefficients.

In this Chapter, we have assumed that the same Q Table was used for all the blocks in an image. Commonly, this is not the case and, previous to quantization, there is a step in which the block under consideration is analyzed and classified. This way, quantization can be adapted to the characteristics of the block (for instance, blocks with more/less complex information), leading to the concept of *Adaptive Transform Coding*. If we have a given a bit budget for an image, those bits can be shared among the blocks in a content dependent manner. That is, more/fewer bits are used for those blocks containing more/less complex information. This concept is the basis

for the *Rate-Distortion Theory* that is paramount in image and video compression.

Finally, an extension of the baseline JPEG algorithm is the so-called *Motion JPEG* (usually referred to as M-JPEG) which has been for a long time a *de facto* standard for video coding. It is essentially identical to the baseline JPEG algorithm, with the exception that the quantization and Huffman tables cannot be dynamically selected in the encoding process but those originally given as examples in the JPEG standard have been fixed.

1.4 Conclusions

In this Chapter we have analyzed how most of digital images are compressed in the web. We have learned how spatial redundancy in images can be reduced using transforms and specifically the Discrete Cosine Transform (DCT) on consecutive blocks of the image. We have shown as well that DC coefficient information can be further decorrelated using prediction from the DC coefficients of neighbor blocks. The resulting DCT coefficients for each block are quantized and entropy encoded to form the final compressed image.

The techniques studied in this Chapter compose the basic tools used in the JPEG standard for coding gray and color images, which is the standard commonly used to compress images in the web and for storing digital photographs. For further details on the JPEG standard, the reader is referred to the standard description (JPEG 1994).

References

- ISO/IEC 14495-1 or ITU-T Recommendation T.81 (1994). Information technology – Digital compression and coding of continuous-tone still images requirements and guidelines
- ISO/IEC 10918-1 or ITU-T Recommendation T.87 (1999). Information technology – lossless and near-lossless compression of continuous-tone still images.
- Arai Y, Agui, Nakajima M (1988) A fast DCT-SQ scheme for images, Transactions of the IEICE, 71: 11, page 1095
- Chiariglione L (1999) Communication standards: Götterdämmerung? In: Advances in Multimedia, Standards and Networks. Puri, Chen Eds. NY:Marcel Dekker, Inc
- Cover T., Thomas A. (1991) Elements of Information Theory. New York, John Wiley & Sons, Inc.

- Gersho A, Gray RM (1993) Vector quantization and signal compression. Boston: Kluwer Academic Publishers
- Lohscheller H (1984) A subjectively adapted image communication system. IEEE Trans. Commun. COM-32, 1316-1322
- Noll NS, Jayant P (1984) Digital Coding of Waveforms: Principles and Applications to Speech and Video. Englewood Cliffs, NJ: Prentice Hall.
- Pennebaker W, Mitchell J (1993) JPEG Still Image Data Compression Standard. New York:Van Nostrand Reinhold