Advance Digital Signal Processing

**JPEG for Still Image Compression**

Tutorial

Po-Hong Wu, Ting-Yu Chen

Email: r98942124@ntu.edu.tw, ripomannn@gmail.com

Department of Electrical Engineering

National Taiwan University, Taipei, Taiwan, ROC

**Abstract**

This tutorial describes the popular JPEG still image coding format. The purpose is to compress images while maintaining acceptable image quality. This is achieved by dividing the image in blocks of 8×8 pixels and applying a discrete cosine transform (DCT) on the partitioned image. The resulting coefficients are quantized, less significant coefficients are cut off. After quantization, two encoding steps are made, zero run length encoding (RLE) followed by an entropy coding. Part of the JPEG encoding is *lossy*, and part is *lossless*. JPEG allows for some flexibility in the different stages, not every option is explored. Also the focus is on the DCT and quantization. The entropy coding will only be briefly discussed and the file structure definitions will not be considered.

**1. Introduction**

The amount of information required to store pictures on modern computers is quite large in relation to the amount of bandwidth commonly available to transmit them over the Internet and applications like video where many thousands of pictures are required would be prohibitively intensive for use on most systems if there wasn’t a way to reduce the storage requirements of these pictures [1].

A still image is a sensory signal that contains significant amount of redundant information which exists in their canonical forms. Image data compression is the technique of reducing the redundancies in image data required to maintain a given quantity of information. Therefore, data storage requirements and communication costs are decreased. In digital image compression, data redundancy is the main issue and better compression can be achieved by reducing more data redundancy with a degree of quality. There are three types of basic data redundancies: coding redundancy, inter-pixel redundancy, and perceptual redundancy.

Coding redundancy occurs when the codes assigned to a set of events such as the pixel values of an image have not been selected to take full advantage of the probabilities of the events [4]. Inter-pixel redundancy usually refers to the correlations between the structural or geometric relationships of the objects in an image. Due to the high correlation between the neighboring pixels, any given pixel can be easily predicted from the values of its neighboring pixels, so the information carried by individual pixels can be relatively small. Any information is said to be perceptually redundant if certain information simply has less relative importance than other information in terms of the human perceptual system. For instance, all the neighboring pixels in the smooth region of a natural image have a very high degree of similarity and this insignificant variation in the values of the neighboring pixels is not noticeable to the human eye. The data size of Fig. 1(a) is 83,261 bytes, and the data size of Fig. 1(b) that is compressed by JPEG is 15,138 bytes which is approximately 1/5 of the former one. As the result, it is hard to distinguish Fig. 1(a) and Fig. 1(b).



(a) (b)

Fig. 1: (a) Original image (83,261 bytes). (b) JPEG compressed image (15,138 bytes).

**2. Some basic ideas of still image**

* Pixel

In digital image, a pixel is a single point in a raster image. The pixel is the smallest addressable screen element shown in Fig. 2.1; it is the smallest unit of picture that can be controlled. Each pixel has its own address. The address of a pixel corresponds to its coordinates. Pixels are normally arranged in a 2-dimensional grid, and are often represented using dots or squares. Each pixel is a sample of an original image; more samples typically provide more accurate representations of the original. The intensity of each pixel is variable. In color image systems, a color is typically represented by three or four component intensities such as red, green, and blue.

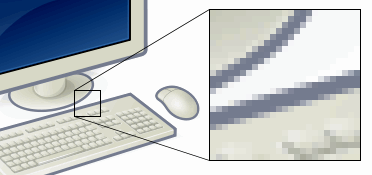


Fig. 2.1 Pixel is smallest element of an image.

* RGB Color Image and Grayscale Image



Fig. 2.3 A grayscale image.

|  |  |
| --- | --- |
| When the eye perceives an image on a computer monitor, it is in actually perceiving a large collection of finite color elements, or pixels [3]. Each of these pixels is in and of itself composed of three dots of light; a green dot, a blue dot, and a red dot. The color the eye perceives at each pixel is a result of varying intensities of green, red, and blue light emanating from that location. A color image can thus be represented as 3 matrixes of values, each corresponding to the brightness of a particular color in each pixel. Therefore, a full color image can be reconstructed by superimposing these three matrices of “RGB”. If an image is measured by an intensity matrix with the relative intensity being represented as a color between black and white, it would appear to be a grayscale image shown in Fig. 2.3. |  |
|  | Fig. 2.2 An color image is made up of three matrices. |

* Grayscale

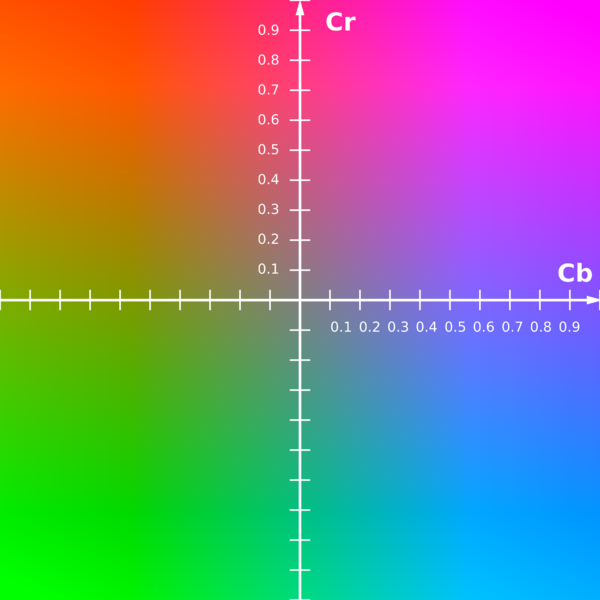
The intensity of a pixel is expressed within a given range between a minimum and a maximum, inclusive. This range is represented in an abstract way as a range from 0 (total absence, black) and 1 (total presence, white), with any fractional values in between. This notation is used in academic papers, but it must be noted that this does not define what "black" or "white" is in terms of colorimetry.

In computing, although the grayscale can be computed through rational numbers, image pixels are stored in binary. Some early grayscale monitors can only show up to sixteen (4-bit) different shades shown in Fig. 2.4, but today grayscale images intended for visual display are commonly stored with 8 bits per sampled pixel, which allows 256 different intensities to be recorded, typically on a non-linear scale. The precision provided by this format is barely sufficient to avoid visible banding artifacts, but very convenient for programming due to the fact that a single pixel then occupies a single byte.



Fig. 2.4 4-bit grayscale

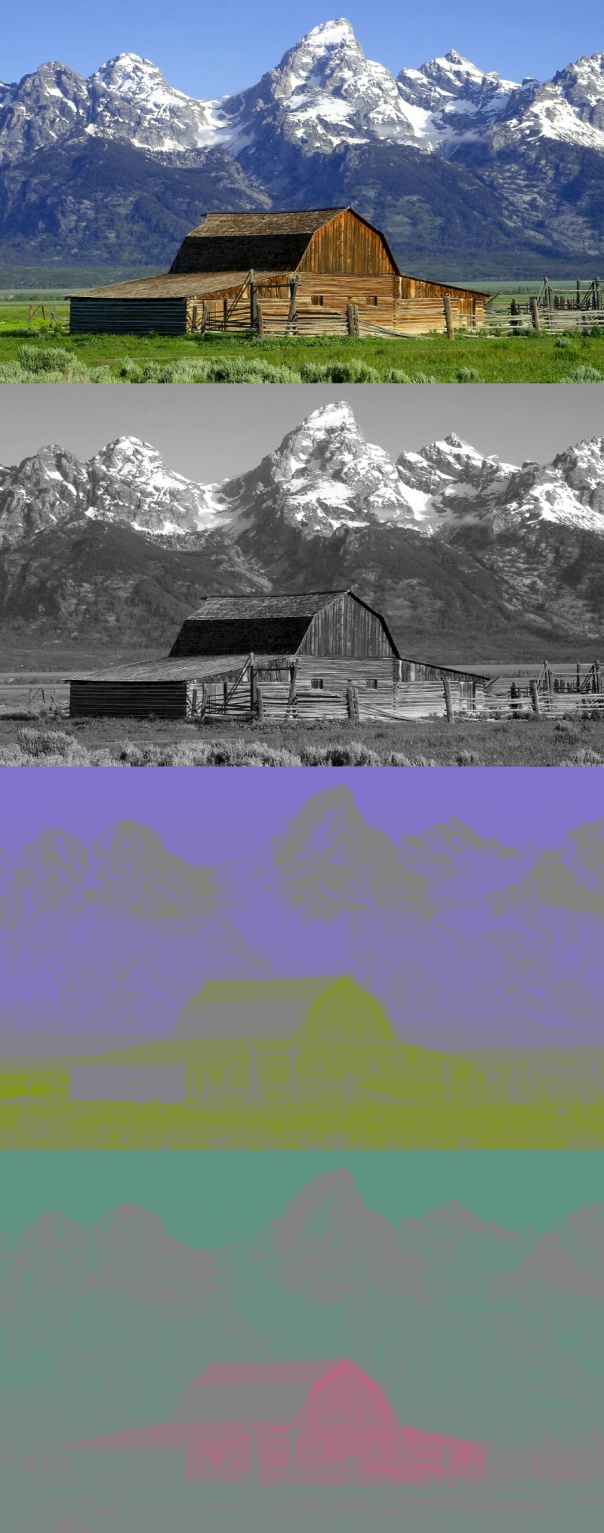
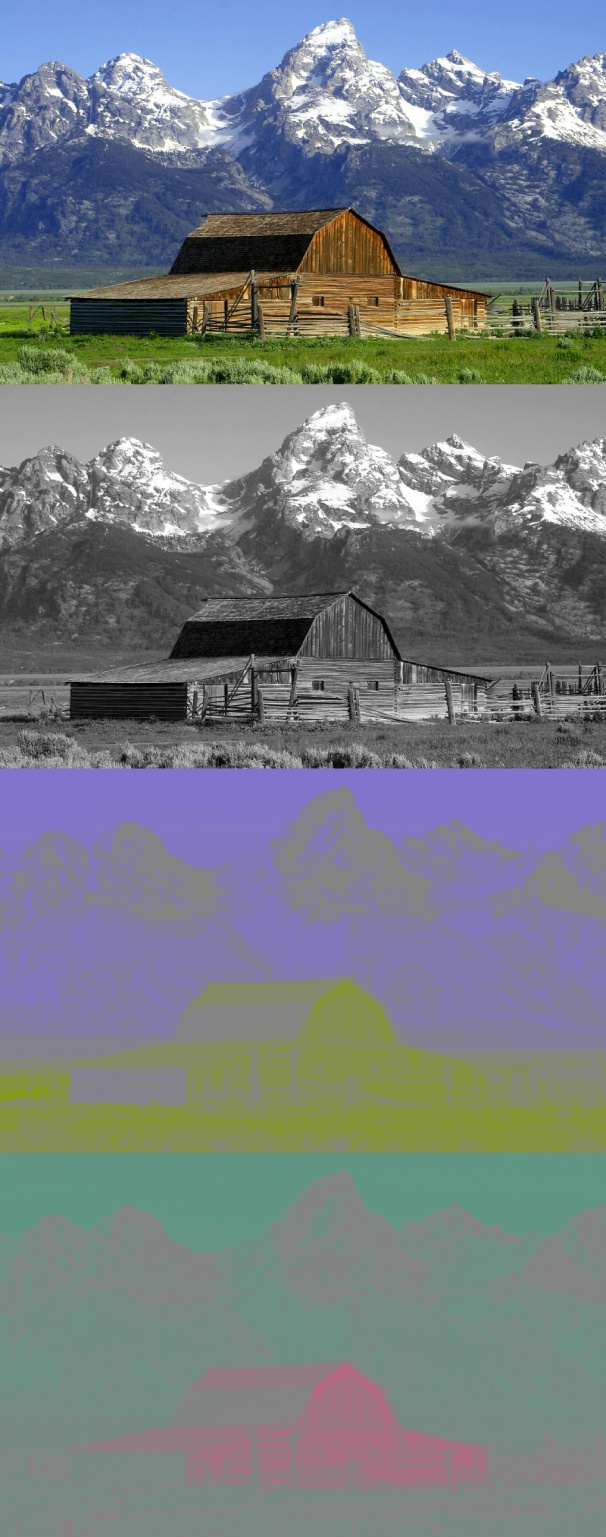
* YUV & YCbCr Color Space

The scope of the terms YUV and YCbCr is usually ambiguous and sometimes these two terms are used interchangeable. Originally, the term YUV were used for a specific analog encoding of in television systems, and YCbCr were used for digital encoding suited for video and still-image compression and transmission such as [MPEG](http://en.wikipedia.org/wiki/MPEG) and JPEG. Today, the term "YUV" mostly means "YCbCr" when referring to signals in video or digital form.

In the case of a color RGB picture a point-wise transform is made to the YCbCr (luminance, blue chrominance, red chrominance) color space. Since the YCbCr color can separate the luminance component, which humans are more sensitive to, from chrominance component, YCbCr color space in some sense is more efficient for subsampling in data compression than the RGB space and will allow for better quantization later. The transform is given by

Fig. 2.5 The CbCr plane at constant luma Y'=0.5

, (1)

and the inverse transform is

. (2)

(c)

(a)

Fig. 2.5 (a) The original image. (b) Y component of image (a). (c) Cb component of image (a). (d) Cr component of image (a).

(b)

(d)

**3. Basic Image Compressed Model**

The JPEG compression process contains three primary parts as shown in Fig. 3.1. To prepare for processing, the matrix representing the image is converted from RGB color space to YCbCr and undergoes the subsampling process. Then the partition process divides the matrix into 8x8 squares (the size was dependent on the balance between image quality and the processing power of the time) which is formally called “block” and passes them through the encoding process in chunks.

To reverse the compression and display a close approximation to the original image the compressed data is fed into the reverse process as shown in Fig. 3.2. These figures illustrate the special case of single-component (grayscale) image compression. Color image compression can then be approximately regarded as compression of multiple grayscale images, which are either compressed entirely one at a time, or are compressed by alternately interleaving 8x8 sample blocks from each in turn.



Fig. 3.1 JPEG encoding flow chart.



Fig. 3.2 JPEG decoding flow chart.

**3.1 Color Space Conversion**

Fig. 3.3 Color space conversion from RGB to YCbCr

The process of compression starts from the conversion of color space. As describe in section 2, we use the transform matrix

,

to convert the three dimensions color matrix of the image from RGB to YCbCr pixel by pixel.

**3.2 Subsampling**

Since the human vision is not sensitive to the chrominance domain, we can apply the subsampling filter to reduce the data size. The principal idea of subsampling is to encode images by implementing less resolution for chrominance [information](http://en.wikipedia.org/wiki/Information) than for luminance information, taking advantage of the human visual system's lower acuity for color differences than for luminance.

The type of subsampling can be determined through the encoding process. Some common subsampling types are introduced as follow.

* 4:4:4

For type 4:4:4 no subsampling is adopt in the encoding process. In the other word, full resolution of luminance and chrominance components are encoded; no loss are induce by subsampling process.

Y

Cb

Cr

* 4:2:0

The zero in 4:2:0 means that Cb and Cr are sampled at half the vertical resolution of Y.

Y

Cb

Cr

* 4:2:2

Cb and Cr are sampled at half the horizontal resolution of Y. This type of color sampling is widely used and considered very high quality.

Y

Cb

Cr

* 4:1:1

Cb and Cr are sampled at one quarter the horizontal resolution.

Y

Cb

Cr

The type of subsampling would affect the compression ratio of JPEG compression for still image and the loss due to image compression.

**3.3 Partition**

After subsampling process, we divide the luminance and two chrominance components into 8x8 pixel- block. In the following compression processes, “block” is taken as the basic unit during the transformation and encoded block by block.

8

**a pixel**

8

Fig. 3.4 A block of 8x8 pixels

**3.4 Image Coding Using Orthogonal Transform**

Suppose that we have a matrix *V*. If the transpose matrix *Vt* equals to the inverse matrix *V*-1, then the matrix *V* is called an orthogonal matrix. An orthogonal transform is easier to implement compared to conventional linear transforms because the computation of the inverse of matrix *V* is not required. One only needs to calculate the transposed matrix of *V*.

In most coding algorithms, the input image is divided into transform blocks, such as 8x8 blocks, and orthogonal transforms are then performed on each block. Neighboring image pixels have high correlations with each other. We need to remove such correlations in both horizontal and vertical directions. Therefore, an image with a size of *K*1 by *K*2 can be divided into blocks each size of *N*1 by *N*2, and each block is transformed into a 1-D vector by being extracted from left to right in each row starting from the top row all the way to the bottom row. The resulting vector is

 (3)

, where N=N1xN2. Then the number of multiplications required for performing orthogonal transformation on a block which is converted into 1-D vector format is

 (4)

The number of multiplications required for the orthogonal transformation on the whole image is

 (5)

It is obvious that dividing the overall image into smaller blocks reduces the computational complexity. However, this defeats the purpose of removing the correlations between the neighboring pixels. In signal processing, Parseval’s relation states that orthogonal transformation is only a matrix which rotates the vector and will not change the length of the vector. Therefore, the orthogonal matrix *V* will preserve length and angle of the vector x.

**3.5 Karhunen-Loeve Transform (KLT)**

Suppose M is the number of total blocks in an image and N is the number of total pixels within each block, pixels in each block can be expressed in the following 1-D vector form [2]:

 (6)

Then the equation of the orthogonal transform performed on each block is

. (7)

We use the optimal orthogonal transform to reduce the correlation among the N pixels in each block to the greatest extent. Here, we use covariance to represent correlation among *x* and *y* shown in Eq. (8) and Eq. (9).

, (8)

 (9)

, where

. (10)

To simplify the calculation, we assume that

 (11)

Therefore, the covariance measurement can be simplified and re-expressed as

 (12)

 . (13)

The covariance can be written as the following covariance matrices:

 (14)

 . . . (15)

Then the covariance matrices can also be defined in vector format

 . (16)

. (17)

By substituting Eq. (7) into Eq. (17), the following can be derived:

 (18)

For KLT, *Cyy* is a diagonal matrix  which is a result of the diagonalization of  by matrix *V*. The condition  must be satisfied for KLT.

KLT is the orthogonal transform that optimizes the transforming by minimizing the average difference of square *D*. Consequently, minimum *D* is obtained by

 . (19)

where ε is a correction constant determined by the image itself and the  represents the discrete degree. Theoretically, any arbitrary covariance matrix can be applied to the following condition:

 (20)

*Cyy* is a diagonal matrix for KLT; therefore we can substitute Eq. (18) into the left-hand side of Eq. (20):

 .(21)

Obviously, we can find that is independent of the transform matrix *V* and can be re-written according to this relation:

 (22)

Consequently,  is independent of the transformation matrix *V*. The best decorrelation is about signal X itself.

Suppose that we have a 3 by 2 transform block:

 . (23)

We can represent the *ij* element of *Cxx* as

 (24)

where *h* is the horizontal distance between *xi* and *xj*, and *v* is the vertical distance between *xi* and *xj*. In the case of (23), *Em*[*x*1*x*6] equals to *ρ*H2*ρ*V. Therefore, *Cxx* can be obtained as

 (25)

This above equation can also be re-written as

 (26)

Furthermore, we can further simplify Eq. (26) by using Kronecker Product and represent *Cxx* as

 .. (27)

As shown in Eq. (27), the auto-covariance function can be separated into vertical and horizontal components *C*V and *C*H such that the computational complexity of the KLT is greatly reduced.

Since KLT is dependent on the image data to be transformed, individual KLT must be computed for each image. This increases the computational complexity. The KLT matrix used by the encoder must be sent to the decoder. This also increases the overall decoding time. These problems can be solved by generalizing KLT.

**3.6 Discrete Cosine Transform (DCT)**

Based on the concept of KLT, DCT, a generic transform that does not need to be computed for each image can be derived. Based on the derivation in Eq. (25), the auto-covariance matrix of any transform block of size *N* can be represented by

 (28)

which is in a Toeplitz matrix form. This means that the KLT can be decided only by one parameter *ρ*, but the parameter *ρ* is still data dependent. Therefore, by setting the parameter *ρ* as 0.90 to 0.98 such that *ρ*→1, we can obtain the extreme condition of KLT. That is, as *ρ* approaches to 1, the KLT is no longer optimal, but fortunately the KLT at this extreme condition can still efficiently remove the correlation between pixels. The 2-D DCT is expressed as

 (29)

where 0≤*u, v*≤N-1

 (30)

The discrete cosine transform shown is closely related to the Discrete Fourier Transform (DFT). Both take a set of points from the spatial domain and transform them into an equivalent representation in the frequency domain. The difference is that while the DFT takes a discrete signal in one spatial dimension and transforms it into a set of points in one frequency dimension and the Discrete Cosine Transform (for an 8x8 block of values) takes a 64-point discrete signal, which can be thought of as a function of two spatial dimensions x and y, and turns them into 64 basis-signal amplitudes (also called DCT coefficients) which are in terms of the 64 unique orthogonal two-dimensional “spatial frequencies” or “spectrum” shown in Fig. 3.3. The DCT coefficient values are the relative amounts of the 64 spatial frequencies present in the original 64-point input. The element in the upper most left corresponding to zero frequency in both directions is the “DC coefficient” and the rest are called “AC coefficients.”



Fig. 3.3: 64 two-dimensional spatial frequencies

Calculating the coefficient matrix this way is rather inefficient. The two summations will require N2 calculations where N is the length (or width) of the matrix. A more efficient way to calculate the matrix is with matrix operations. Set C equal to

 (30)

Because pixel values typically change vary slowly from point to point across an image, the FDCT processing step lays the foundation for achieving data compression by concentrating most of the signal in the lower spatial frequencies. For a typical 8x8 sample block from a typical source image, most of the spatial frequencies have zero or near-zero amplitude and need not be encoded.

At the decoder the IDCT reverses this processing step. It takes the 64 DCT coefficients and reconstructs a 64-point output image signal by summing the basis signals. Mathematically, the DCT is one-to-one mapping for 64-point vectors between the image and the frequency domains. In principle, the DCT introduces no loss to the source image samples; it merely transforms them to a domain in which they can be more efficiently encoded.

**3.7 Quantization**

Since we always deal with 8x8 matrices when dealing with JPEG compression, the DCT matrix C can be calculated in Eq. (31). Using C, the FDCT can be found by

 (31)

where P is the matrix of values from the image being compressed. We should subtract 128 from every element in P before deriving FDCT coefficient.

 (32)

After output of FDCT, we haven’t really accomplished any compression. In fact the matrix FDCT takes up more space than P because it has the same number of elements but each element is a floating point number with a range of [−1023, 1023] instead of an integer with range [0, 255]. However, the human vision is more sensitive to low frequencies than high and most image data is relatively low frequency. Low frequency data carries more important information than the higher frequency. The data in the FDCT matrix is organized from lowest frequency in the upper left to highest frequency in the lower right.

Quantization is the step where we actually throw away data. The DCT is a lossless procedure. The data can be precisely recovered through the IDCT without considering computational implementing. During Quantization, every element in the 8x8 FDCT matrix is divided by a corresponding step size in a quantization matrix Q to yield a matrix QFDCT followed by rounding to the nearest integer shown in Eq. (33):

 (33)

This output value is normalized by the quantizer step size. Dequantization is the inverse function, which in this case means simply that the normalization is removed by multiplying by the step size, which returns the result to a representation appropriate for input to the IDCT:

 (34)

The goal of quantization is to reduce most of the less important high frequency coefficients to zero, the more zeros we can generate the better the image will compress. The matrix Q generally has lower numbers in the upper left that increase in magnitude as they get closer to the lower right. While Q could be any matrix, there are actually two quantization tables specified by the JPEG standard for reference: the luminance quantization matrix (Eq. 35) and the chrominance quantization matrix (Eq. 36). If the user wants better quality at the price of compression he can lower the values in the Q matrix. If he wants higher compression with less image quality he can raise the values in the matrix.

 (35)

 (36)

According to Eq. (33), those values that were reduced to zero are gone forever and cannot be reconstructed, and this is what we call “lossy”. However, the coefficients we have lost were the higher frequency, less important ones. We have succeeded in turning most of the higher frequency coefficients into zeros. Actually, it’s how many zeros we can get in a row that determines how much we can compress the data.

Fig 3.4(a) is an 8x8 block of 8-bit samples, arbitrarily extracted from a real image. The small variations from sample to sample indicate the predominance of low spatial frequencies. After subtracting 128 from each sample for the required level-shift, the 8x8 block is input to the FDCT, Eq. (31). Figure 3.4(b) shows the resulting DCT coefficients. Except for a few of the lowest frequency coefficients, the amplitudes are quite small. Fig. 3.4(c) is the example quantization table for luminance (grayscale) components. Fig. 3.4(d) shows the quantized DCT coefficients, normalized by their quantization table entries. At the decoder these numbers are “denormalized” according to Eq. (34), and input to the IDCT. Finally, Fig. 3.4(f) shows the reconstructed sample values, remarkably similar to the originals in Fig. 3.4(a). Of course, the numbers in Fig. 3.4(d) must be Huffman-encoded before transmission to the decoder.

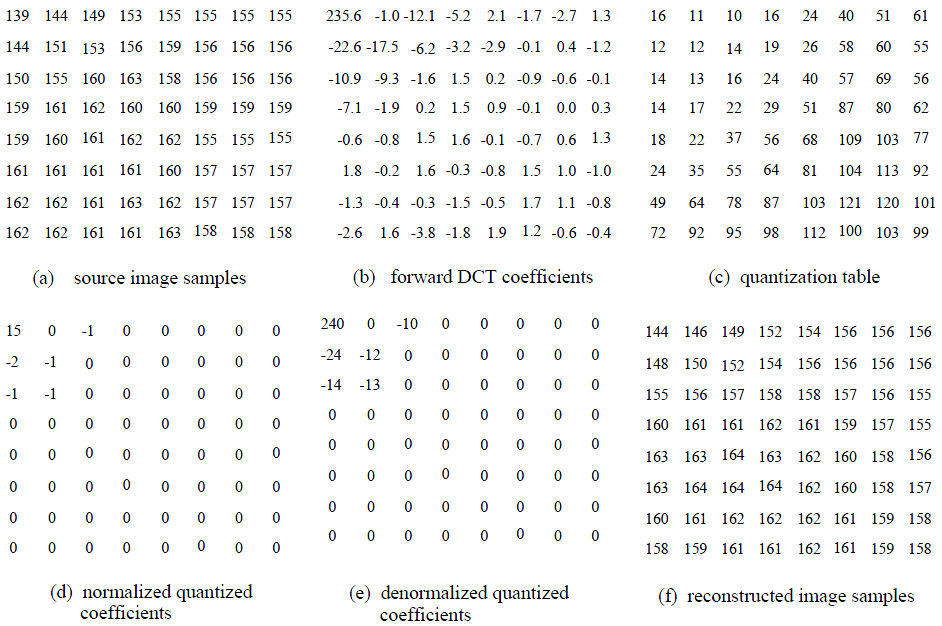
****

Fig. 3.4: DCT and Quantization Examples.

**3.5 Entropy Encoding**

After the quantization of the DCT coefficients, the DC coefficient and the 63 AC coefficients are treated differently. Since the DC coefficients usually contain a large part of the total energy in an image, the quantized DC coefficient is encoded by differential encoding which encode the DC coefficient as the difference from the previous quantized DC coefficient. Differential DC encoding is performed in a defined order as shown in Fig. 3.5.

Fig. 3.5 Differential DC encoding.

blocki-1

blocki

DCi-1

DCi

Difference = DCi - DCi-1

……….

……….

The AC coefficients are ordered into the “zig-zag” order, shown in Fig. 3.6, such that AC coefficients are placed from high frequency to low. By keeping high frequencies together, we can form long runs of zeros because high frequencies more likely to be zero after quantization.

Fig. 3.6: The zig-zag pattern of Entropy Encoding

Consequently, the nonzero AC coefficients are coded using a variable-length code. The coded coefficients can be expressed as the number of preceding zeros and the coefficient’s value that interrupts the zero run. For example, suppose that we have the following coefficient sequence after the zig-zag process:



The above sequence can be expressed as

*.*

As mention above, the DC and AC coefficients are encoded with different code table. In the standard JPEG compression process, Huffman code or arithmetic code are used for the entropy coding. Tables of both DC and AC coefficients are encoded and specified in the beginning of the image bit stream. Among Huffman code and arithmetic code, Huffman code is used for Baseline Sequential DCT-based mode of JPEG encoding process and is considered to be more common. The coding process of Huffman coding for DC and AC coefficients are discussed as follow.

* Huffman Coding

In computer science and information theory, Huffman coding is an entropy encoding algorithm used for lossless data compression. It encodes a source symbol with variable-length code which its code table is derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol.

1. The JPEG Code for DC Coefficient

The first coefficient of each block (top-left position in Fig. 3.6 in previous discussion) is the DC coefficient, which represents the mean value of the pels in the block.

Since the DC coefficients still exhibit significant local correlations, differential coding is used in which the value to be coded is the difference between the DC coefficient of the current block and that of the previous block - the blocks are scanned from left to right, row by row. The first block in each row is coded with respect to zero.

By adopting differential coding the entropy can be greatly reduced owing to the space correlation of image. This can be shown by Fig. 3.7 the entropy is reduced from 6.42 bits to 6.07 bits.

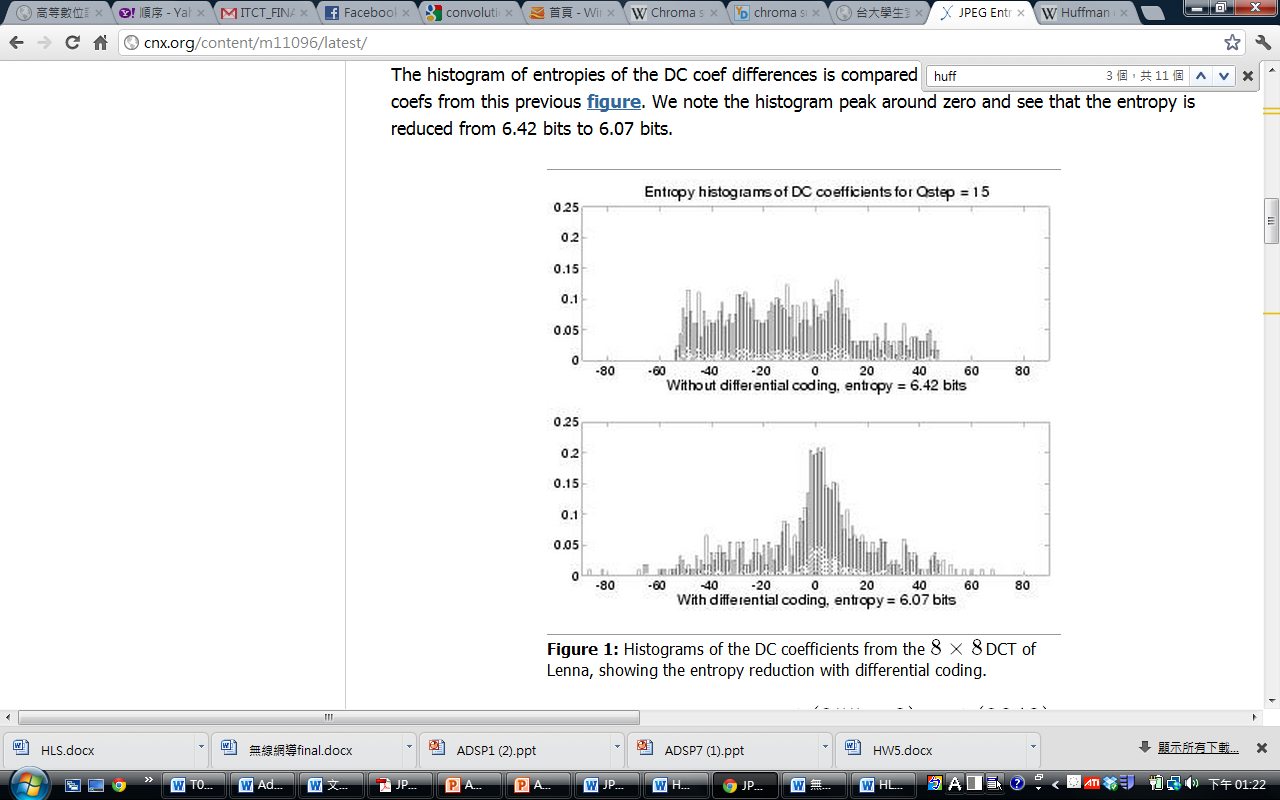


Fig. 3.7 (a) Entropy without differential coding. (b) Entropy with differential coding.

Another coding technique used by JPEG Huffman coding is floating-point representation. The size of the differences can in theory be up to ±(255×8)=±(2040) if the input pels occupy the range -128 to +(127). Hence the Huffman code table would have to be quite large. Taking advantages of the floating point representation, the code length and table can be reduced.

The encoding processes can be illustrated by an example:

blocki-1

blocki

DCi-1=56

DCi=61

Difference = DCi - DCi-1

……….

……….

blocki+1

blocki+2

DCi=52

DCi=49

DCi-1=55

Difference between blocks

= 1 5 -9 -3 ……

**Step 1:**

Look up the following table

|  |  |
| --- | --- |
| **DIFF Values** | **Code Word** |
| -1, 1 | 0, 1 |
| -3, -2, 2, 3 | 00, 01, 10, 11 |
| -7, -6, -5, -4, 4, 5, 6, 7 | 000, 001, 010, 011, 100, 101, 110, 111 |
| -15, -14, ……, 15 | 0000, 0001, ……, 1111 |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DIFF | 1 | 5 | -9 | -3 |
| Code | 1 | 101 | 0110 | 00 |
| Code Length | 1 | 3 | 4 | 2 |

**Step 2:**

Look up the following table

|  |  |
| --- | --- |
| **Additional Code Length**  **(Category)** | **Code Word** |
| 0 | 00 |
| 1 | 010 |
| 2 | 011 |
| 3 | 100 |
| 4 | 101 |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DIFF | 1 | 5 | -9 | -3 |
| Code | 1 | 101 | 0110 | 00 |
| Code Length | 1 | 3 | 4 | 2 |
| Code Word | 010 | 100 | 101 | 011 |
| Full Code Word | 010 1 | 100 101 | 101 0110 | 011 00 |

The DC difference can be encoded as the bit sequence:

......0101100101101011001100……

The decoding process is simply the inverse of the encoding process.

In JPEG all Huffman code tables are defined in the image header. Each table requires 16+n bytes, where n is the number of code words in the table. The first 16 bytes list the number of code words of each length from 1 to 16 bits (code words longer than 16 bits are forbidden). The remaining n bytes list the decoded output values of the n code words in ascending code word order ( n<256). Hence 16+12=28 bytes are needed to specify the code table for DC coefficients.

1. The JPEG Run-Amplitude Code for AC Coefficient

The remaining 63 coefficients (AC part) of the 8x8 block usually contain many zeros and so are coded with a combined run-amplitude Huffman code.

The code word represents the run-length of zeros before a non-zero coefficient and the Size of that coefficient. This is then followed by the Additional Bits which define the coefficient amplitude and sign precisely. Size and Additional Bits are defined just as for DC coefficients.

This 2-dimensional Huffman code (Run, Size) is efficient because there is a strong correlation between the Size of a coefficient and the expected Run of zeros which precedes it - small coefficients usually follow long runs; larger coefficients tend to follow shorter runs. No single 2-D event is so probable that the Huffman code becomes inefficient.

The encoding processes can be illustrated by an example:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DC | 0 | -1 | 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 |
| 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 |

AC sequence: 0, -2, -1, -1, -1, 0, 0, -1, 0, 0, …, 0

AC <run, amplitude> sequence:

<1, -2> <0, -1> <0, -1> <0, -1> <2, -1> <EOB>

**Step 1:**

Look up the following table

|  |  |  |
| --- | --- | --- |
| **Length** | **amplitude** | **code** |
| 1 | -1,1 | 0, 1 |
| 2 | -3,-2,2,3 | 00, 01, 10, 11 |
| 3 | -7..-4,4..7 | 000, 001, 010, 011, 100, 101, 110, 111 |
| 4 | -15..-8,8,15 | 0000, 0001, ……, 1111 |
| 5 | -31..-16,16..31 | 00000, ……, 11111 |
| 6 | -127..-64,64..127 | 000000, ……, 111111 |
|  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **<run, amplitude>** | <1, -2> | <0, -1> | <0, -1> | <0, -1> | <2, -1> | <EOB> |
| **code** | 01 | 0 | 0 | 0 | 0 |  |
| **length** | 2 | 1 | 1 | 1 | 1 |  |
| **<run, length>** | <1, 2> | <0, 1> | <0, 1> | <0, 1> | <2, 1> |  |

**Step 2:**

Look up the following table

|  |  |  |
| --- | --- | --- |
| Run/Length | Code Length | Code Word |
| 0 / 0 (EOB) | 4 | 1010 |
| 0 / 1 | 2 | 00 |
|  |  |  |
| 1/2 | 5 | 11011 |
|  |  |  |
| 2/1 | 5 | 11100 |
|  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **<run, amplitude>** | <1, -2> | <0, -1> | <0, -1> | <0, -1> | <2, -1> | <EOB> |
| **code** | 01 | 0 | 0 | 0 | 0 |  |
| **length** | 2 | 1 | 1 | 1 | 1 |  |
| **<run, length>** | <1, 2> | <0, 1> | <0, 1> | <0, 1> | <2, 1> |  |
| **Code word** | 11011 | 00 | 00 | 00 | 11100 | 1010 |

The AC run-amplitude can be encoded as the bit sequence:

......1101101 000 000 000 111000 1010……

**4. Conclusion**

Before the JPEG 2000, the JPEG lossy compression scheme is one of the most popular and versatile compression schemes in widespread use. The JPEG can efficiently compress image without severe distortion and cost less for implement. It has also been extended to work on moving pictures in the MPEG (motion jpeg) standards that are beginning to play a vital role in the online distribution of film. More research is also being done to incorporate wavelet technology into the standard as well. The JPEG standard has proven a versatile and effective tool in the compression of data.

**5. Reference**

1. JPEG International Standard (T.81E).
2. G. K. Wallace, "The JPEG Still Picture Compression Standard", *Communications of the ACM*, Vol. 34, Issue 4, pp.30-44, 1991.
3. 酒井善則、吉田俊之 共著，白執善 編譯，“*影像壓縮技術*”，全華，2004.
4. Tim, Jesse Trutna, “ An Introduction to JPEG”
5. Jian-Jiun Ding and Jiun-De Huang, "Image Compression by Segmentation and Boundary Description", Master’s Thesis, National Taiwan University, Taipei, 2007.
6. Jian-Jiun Ding and Tzu-Heng Lee, "Shape-Adaptive Image Compression", Master’s Thesis, National Taiwan University, Taipei, 2008.