

Computer Vision

Course 8

Pseudo-color Image Processing

Pseudo-color (also called *false color*) image processing consists of assigning colors to gray values based on a specified criterion. The main use of pseudo-color is for human visualization and interpretation of gray-scale events in an image or sequence of images.

Intensity (Density) Slicing

If an image is viewed as a 3-D function, the method can be described as one of placing planes parallel to the coordinate plane of the image; each plane then “slices” the function in the area of intersection.

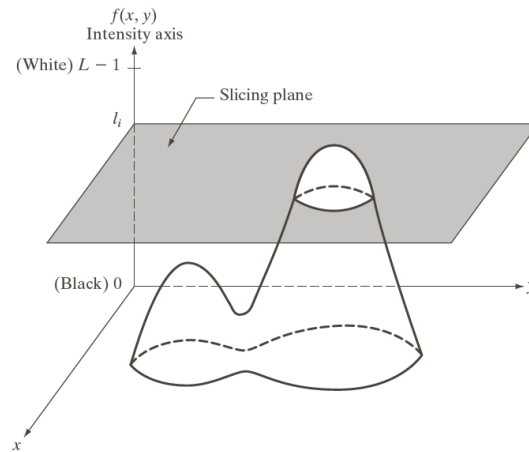


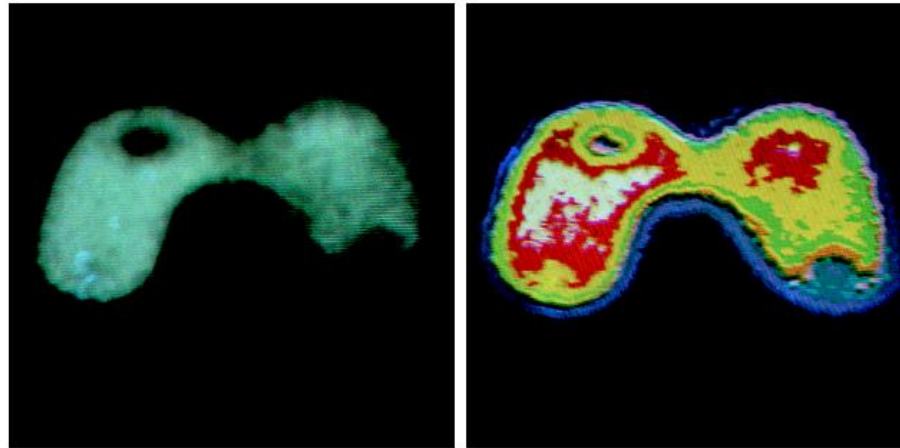
FIGURE 6.18
Geometric
interpretation of
the intensity-
slicing technique.

The plane at $f(x, y) = l_i$ slices the image function into two levels. If a different color is assigned to each side of the plane, any pixel whose intensity level is above the plane will be coded with one color and any pixel below the plane will be coded with other color. Levels that lie on the plane itself may be arbitrarily assigned one of the two colors. The result is a two color image whose

relative appearance can be controlled by moving the slicing plane up and down the intensity axis.

Let $[0, L-1]$ represent the gray scale, let l_0 represent black ($f(x,y)=0$) and level l_{L-1} represent white ($f(x,y)=L-1$). Suppose that P planes perpendicular to the intensity axis are defined at levels $l_1, l_2, \dots, l_P, 0 < P < L-1$. The P planes partition the gray scale into $P+1$ intervals, V_1, V_2, \dots, V_{P+1} . Intensity to color assignments are made according to the relation:

$$f(x, y) = c_k \quad \text{if } f(x, y) \in V_k.$$



a b

FIGURE 6.20 (a) Monochrome image of the Picker Thyroid Phantom. (b) Result of density slicing into eight colors. (Courtesy of Dr. J. L. Blankenship, Instrumentation and Controls Division, Oak Ridge National Laboratory.)

Measurements of rainfall levels with ground-based sensors are difficult and expensive, and total rainfall figures are even more difficult to obtain because a significant portion of precipitations occurs over the ocean. One way to obtain these figures is to use a satellite. The TRMM (Tropical Rainfall Measuring

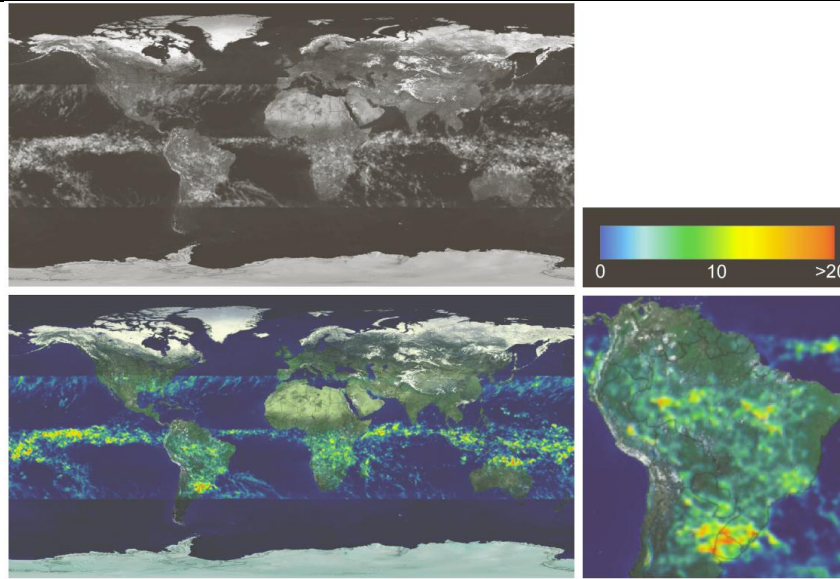
Computer Vision

Course 8

Mission) satellite utilizes, among others, three sensors specially designed to detect rain: a precipitation radar, a microwave imager, and a visible and infrared scanner. The results from the various rain sensors are processed, resulting in estimates of average rainfall over a given time period in the area monitored by the sensors. From these estimates, it is not difficult to generate gray-scale images whose intensity values correspond directly to rainfall, with each pixel representing a physical land area whose size depends on the resolution of the sensors.

Computer Vision

Course 8



a b
c d

FIGURE 6.22 (a) Gray-scale image in which intensity (in the lighter horizontal band shown) corresponds to average monthly rainfall. (b) Colors assigned to intensity values. (c) Color-coded image. (d) Zoom of the South American region. (Courtesy of NASA.)

Color Slicing

Highlighting a specific range of colors in an image is useful for separating objects from their surroundings. The basic idea is either to:

- display the colors of interest so they stand out from the background
- use the region defined by the colors as a mask for further processing.

One of the simplest ways to “slice” a color image is to map the colors outside some range of interest to a neutral color. If the colors of interest are enclosed by a cube (or hypercube, if $n > 3$) of width W and centered at a prototypical (e.g. average) color with components (a_1, a_2, \dots, a_n) the set of transformations is:

$$s_i = \begin{cases} \mathbf{0.5} & \text{if } |r_j - a_j| > \frac{W}{2}, 1 \leq j \leq n \\ r_i & \text{otherwise} \end{cases}, i = 1, 2, \dots, n$$

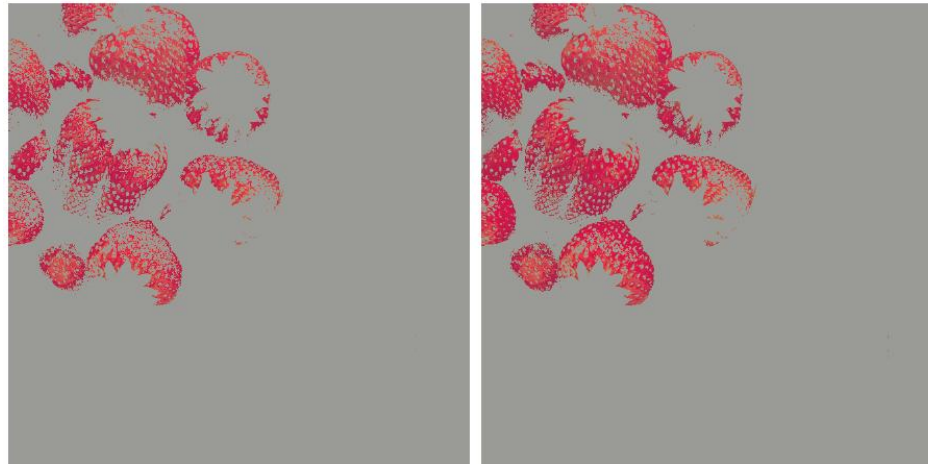
These transformations highlight the colors around the prototype by forcing all other colors to the midpoint of the reference color space (an arbitrarily chosen neutral point).

For the RGB color space, for example, a suitable neutral point is middle gray or color $(0.5, 0.5, 0.5)$.

If a sphere is used to specify the colors of interest, the transformations are:

$$s_i = \begin{cases} \mathbf{0.5} & \text{if } \sum_{j=1}^n (r_j - a_j)^2 > R_0^2 \\ r_i & \text{otherwise} \end{cases}, i = 1, 2, \dots, n$$

where R_0 is the radius of the enclosing sphere and (a_1, a_2, \dots, a_n) are the components of its center.



a b

FIGURE 6.34 Color-slicing transformations that detect (a) reds within an RGB cube of width $W = 0.2549$ centered at $(0.6863, 0.1608, 0.1922)$, and (b) reds within an RGB sphere of radius 0.1765 centered at the same point. Pixels outside the cube and sphere were replaced by color $(0.5, 0.5, 0.5)$.

Image Compression

Image compression is the art and science of reducing the amount of data required to represent an image.

Consider a two-hour standard definition (SD) television movie using $720 \times 480 \times 24$ bit pixel arrays. A digital movie is a sequence of video frames in which each frame is a full-color still image. Because video players must display the frames sequentially at rates 30 fps (frames per second), SD digital video must be accessed at:

$$30 \frac{\text{frames}}{\text{sec}} \times (720 \times 480) \frac{\text{pixels}}{\text{frame}} \times 3 \frac{\text{bytes}}{\text{pixel}} = 31.104.000 \text{ bytes/sec}$$

and a two-hour movie consist of

$$31.104.000 \frac{\text{bytes}}{\text{sec}} \times (60^2) \frac{\text{sec}}{\text{hour}} \times 2 \text{hours} \approx 2.24 \times 10^{11} \text{ bytes (224GB)}$$

To put a two-hour movie on a DVD, each frame must be compressed by a factor of **26.3**(on average). The compression must be even higher for *high definition* (HD) *television* where image resolution reach **1920×1080×24** bit/image.

Fundamentals

The term *data compression* refers to the process of reducing the amount of data required to represent a given quantity of information. *Data* and *information* are not the same thing; data are the means by which information is expressed. Because various amounts of data can be used to represent the same amount of information, representations that contain irrelevant or repeated information are said to contain *redundant data*.

Let b and b' denote the number of bits in two representations of the same information, the *relative data redundancy* R of the representation with b bits is:

$$R = 1 - \frac{1}{C}$$

Where C , commonly called the *compression ratio* is

$$C = \frac{b}{b'}$$

If $C=10$ - the larger representation has 10 bits of data for every 1 bit of data in the smaller representation. The

corresponding relative data redundancy of the larger representation is 0.9 ($R=0.9$), indicating that **90%** of its data is redundant.

In the context of digital image compression, b usually is the number of bits needed to represent an image as a 2-D array of intensity values. Two-dimensional intensity arrays (far from optimal) suffer from three principal types of data redundancies:

1. *Coding redundancy.* A *code* is a system of symbols (letters, numbers, bits...) used to represent a body of information or set of events. Each piece of information or event is assigned a sequence of *code symbols*, called a *code word*. The number of symbols in each code word is its *length*. The *8-bit* codes that are used to represent the intensities in most 2-D intensity arrays contain more bits than are needed to represent the intensities.

2. *Spatial and temporal redundancy.* Because the pixels of most 2-D intensity arrays are correlated spatially (i.e. each pixel is similar to or dependent on neighboring pixels), information is unnecessarily replicated in the representations of correlated pixels. In a video sequence, temporally correlated pixels (i.e., those similar to or dependent on pixels in nearby frames) also duplicate information.

3. *Irrelevant information.* Most 2-D intensity arrays contain information that is ignored by the human eye. This information is redundant in the sense that it is not used.

Compression is achieved when one or more redundancy is reduced or eliminated.

Computer Vision

Course 8



a b c

FIGURE 8.1 Computer generated $256 \times 256 \times 8$ bit images with (a) coding redundancy, (b) spatial redundancy, and (c) irrelevant information. (Each was designed to demonstrate one principal redundancy but may exhibit others as well.)

Code Redundancy

Assume that a discrete random variable r_k in the interval $[0, L-1]$ is used to represent the intensities of an $M \times N$ image and that each r_k occurs with probability $p(r_k)$.

$$p(r_k) = \frac{n_k}{M \cdot N}, \quad k = 0, 1, 2, \dots, L-1$$

Where L is the number of intensity values, and n_k is the number of times that the k -th intensity appears in the image. If the number of bits used to represent each value of r_k is $l(r_k)$,

then the average number of bits required to represent each pixel is:

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p(r_k)$$

The total number of bits required to represent an $M \times N$ image is MNL_{avg} . If the intensities are represented using a natural m -bit fixed-length code, $L_{avg} = m$.

Consider image in Figure 8.1 (a) ($M=N=256$) and the coding Table 8.1.

Computer Vision

Course 8

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

TABLE 8.1
Example of
variable-length
coding.

For Code 1 $L_{avg} = 8$. For Code 2 we have:

$$L_{avg} = 0.25 \cdot 2 + 0.47 \cdot 1 + 0.25 \cdot 3 + 0.03 \cdot 3 = 1.81 \text{ bits}$$

The total number of bits needed to represent the entire image

is $MNL_{avg} = 256 \times 256 \times 1.81 = 118.621$.

$$C = \frac{256 \times 256 \times 8}{256 \times 256 \times 1.81} = \frac{8}{1.81} \approx 4.42$$

$$R = 1 - \frac{1}{C} = 1 - \frac{1}{4.42} = 0.774$$

Thus, **77.4%** of the data in the original 8-bit 2-D intensity array is redundant.

The compression achieved by code 2 results from assigning fewer bits to the more probable intensity values than to the less probable ones, thus resulting a *variable-length code*. The best *fixed-length code* that can be assigned to the intensities

of the image in Figure 8.1(a) is the natural 2-bit counting sequence $\{00, 01, 10, 11\}$ but the resulting compression is only $C=8/2=4:1$ which is about *10%* less than the *4.42:1* compression of the variable-length code.

Coding redundancy is present when the codes assigned to a set of events (such as intensity values) do not take full advantage of the probabilities of the events. Coding redundancy is almost always present when the intensities of an image are represented using a natural binary code. Most

images are composed of objects that have a regular and somewhat predictable morphology (shape) and reflectance, and are sampled so that the objects being depicted are much larger than the picture elements. For most images, certain intensities are more probable than others (that is, the histograms of most images are not uniform). A natural binary encoding assigns the same number of bits to both the most and least probable values, failing to minimize the value of L_{avg} and resulting in coding redundancy.



a b c

FIGURE 8.1 Computer generated $256 \times 256 \times 8$ bit images with (a) coding redundancy, (b) spatial redundancy, and (c) irrelevant information. (Each was designed to demonstrate one principal redundancy but may exhibit others as well.)

Spatial and Temporal Redundancy

Consider the computer-generated image in Figure 8.1(b). In the corresponding 2-D intensity array:

- All 256 intensities are equally probable
- Because the intensity of each line was selected randomly, its pixels are independent of one another in the vertical direction

- Because the pixels along each line are identical, they are maximally correlated (completely dependent on one another) in the horizontal direction.

The first observation tells us that this image cannot be compressed by variable-length coding alone. Observation 2 and 3 reveal a significant spatial redundancy that can be eliminated, for instance, by representing this image as a sequence of *run-length pairs*, where each run-length pair specifies the start of a new intensity and the number of

consecutive pixels that have that intensity. A run-length based representation compresses the original 2-D, 8-bit intensity array by

$$C = \frac{256 \times 256 \times 8}{256 \times 2 \times 8} = 128 : 1$$

Each 256-pixel line of the original representation is replaced by a single 8-bit intensity value and length 256 in the run-length representation.

In most images, pixels are correlated spatially (in both x and y) and in time (in case of video sequences). Because most

pixels' intensities can be predicted reasonably well from neighboring intensities, the information carried by a single pixel is small. Much of its visual contribution is redundant in the sense that can be inferred from its neighbors. To reduce the redundancy associated with spatially and temporally correlated pixels, a 2-D intensity array must be transformed into a more efficient but usually 'non-visual' representation. Transformations of this type are called *mappings*. A mapping is said to be *reversible* if the pixels of the original 2-D

intensity array can be reconstructed without error from the transformed data set; otherwise the mapping is said to be *irreversible*.

Irrelevant Information

One of the simplest ways to compress a set of data is to remove superfluous data from the set. In the context of DIP, information ignored by the system which uses the image (human eye, computer programs) are obvious candidates for omission. Thus, the computer-generated image in Figure

8.1(c), because it appears to be a homogeneous field of grey, can be represented by its average intensity alone – a single 8-bit value. The original $256 \times 256 \times 8$ bit intensity array is reduced to a single byte; the resulting compression is ***65.536:1***.

Figure 8.3(a) shows the histogram of the image in Figure 8.1(c) – there are some intensity values (125 through 131) actually present. The human visual system averages these intensities, perceives only the average value, and ignores the small changes in intensity that are present in this case.

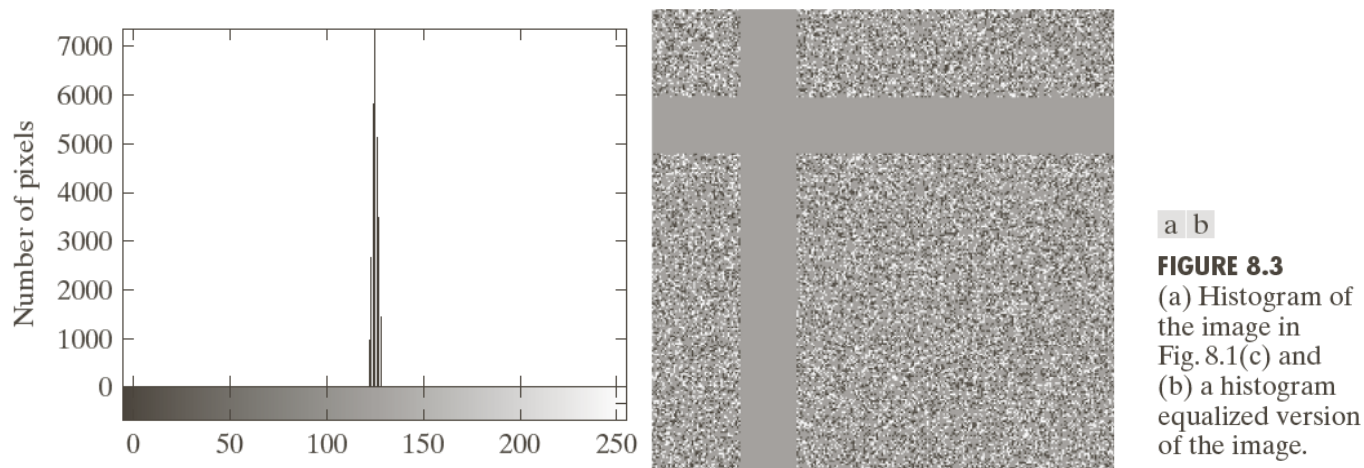


Figure 8.3(b), a histogram equalized version of the image in Figure 8.1 (c), makes the intensity changes visible and reveals two previously undetected regions of constant intensity.

If the image in Figure 8.1 (c) is represented by its average value alone, this ‘invisible’ structure is lost.

The kind of redundancy can be eliminated because the information itself is not essential for normal visual processing and/or the intended use of the image. Because its omission results in a loss of quantitative information, its removal is

commonly referred to as *quantization* (mapping of a broad range of input values to a limited number of output values). Because information is lost, quantization is an irreversible operation.

Measuring Image Information

Problem: is there a minimum amount of data that is sufficient to describe an image without losing information? The answer is given by *information theory*.

The generation of information can be modelled as a probabilistic process. A random event E with probability $P(E)$ is said to contain:

$$I(E) = \log \frac{1}{P(E)} = -\log P(E) \text{ units of information.}$$

If $P(E)=1$ (the event always occurs), $I(E)=0$ and no information is attributed to it (because no uncertainty is associated to the event, no information would be transferred by communicating that the event has occurred).

If the base m logarithm is used, the measurement is said to be in m -ary units. For $m=2$ the unit of information is the *bit*.

$P(E) = \frac{1}{2} \rightarrow I(E) = -\log \frac{1}{2} = 1$ (when only one of two possible equally likely events occur, the amount of information is 1 bit)

Given a source of statistically independent random events from a discrete set of possible events $\{a_1, a_2, \dots, a_J\}$ with associated probabilities $\{P(a_1), P(a_2), \dots, P(a_J)\}$, the average information per source output, called the *entropy* of the source is:

$$H = -\sum_{j=1}^J P(a_j) \log P(a_j)$$

The a_j are called *source symbols*. Because they are statistically independent, the source itself is called *zero-memory source*.

If an image is considered to be the output of an imaginary zero-memory “intensity source”, we can use the histogram of the observed image to estimate the symbol probabilities of the source. The intensity source’s entropy becomes:

$$\tilde{H} = -\sum_{k=0}^{L-1} p_r(r_k) \log_2 p_r(r_k) \quad (\text{H})$$

Formula (H) is the average information per intensity output of the imaginary intensity source in bits. It is not possible to code the *intensity values* of the imaginary source (and thus the sample image) with fewer than \tilde{H} bits/pixel.

The entropy of the image in Figure 8.1(a) can be estimated as follows:

$$\tilde{H} = -\left[0.25 \log_2 0.25 + 0.47 \log_2 0.47 + 0.25 \log_2 0.25 + 0.03 \log_2 0.03\right]$$

$$\tilde{H} \approx 1.6614 \text{ bits/pixel}$$

In a similar manner, the entropy of the images in Figure 8.1(b) and (c) are estimated to be **8** bits/pixel and **1.566** bits/pixel, respectively. Note that the image in Figure 8.1 (a) appears to have the most visual information, but has almost the lowest computed entropy. The obvious conclusion is that the amount of entropy and thus the information in an image is not an intuitive notion.

Fidelity Criteria

When “irrelevant information” is removed, information is lost. In this case, we need a means of quantifying the nature of the lost. Two types of criteria can be used:

- (1) objective fidelity criteria
- (2) subjective fidelity criteria

When information loss can be expressed as a mathematical function of the input and output of a compression process, it is said to be based on an *objective fidelity criterion*. An

example is the root-mean-square (*rms*) error between two images. Let $f(x,y)$ be an input image and $\bar{f}(x,y)$, be an approximation of $f(x,y)$ obtained from compressing-decompressing the input. The error is:

$$e(x, y) = \bar{f}(x, y) - f(x, y)$$

$$\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} e(x, y) \quad \text{total error between the images}$$

The *root-mean-square*, e_{rms} , between $f(x,y)$ and $\bar{f}(x,y)$ is

$$e_{rms} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\bar{f}(x,y) - f(x,y)]^2 \right]^{\frac{1}{2}}$$

If $\bar{f}(x,y)$ is considered to be the sum of the original image $f(x,y)$ and an error or “noise” signal $e(x,y)$, the *mean-square signal-to-noise* of the output image can be defined by:

$$\text{SNR}_{rms} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \bar{f}(x,y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\bar{f}(x,y) - f(x,y)]^2}$$

Decompressed images are ultimately viewed by humans. So, measuring image quality by the subjective evaluations of people is often more appropriate. This can be done by presenting a decompressed image to a cross section of viewers and averaging their evaluations.

Image Compression Models

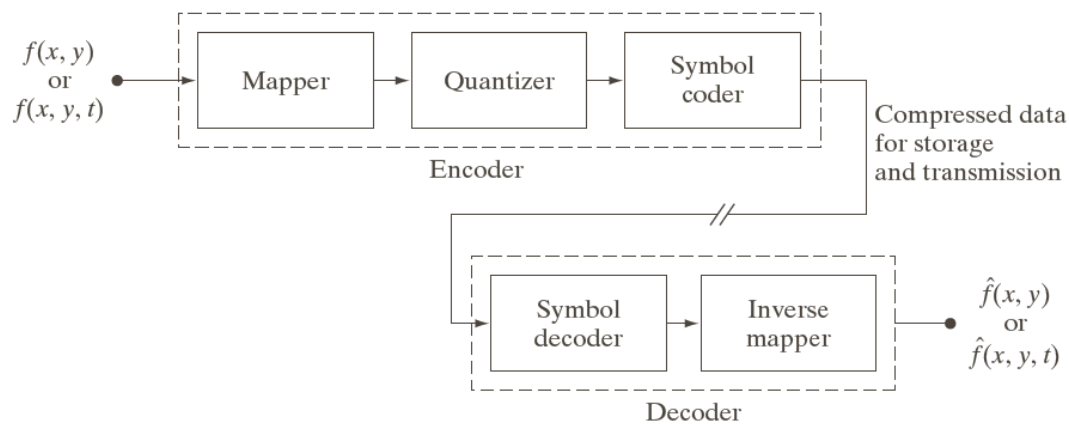


FIGURE 8.5
Functional block diagram of a general image compression system.

An image compression system is composed by two distinct functional components: an *encoder* and a *decoder*. Both operations can be performed in software, as is the case in

Web browsers and many commercial image editing programs, or in a combination of hardware and firmware, as in commercial DVD players. A *codec* is a device or program that is capable of both encoding and decoding.

Input image $f(x, \dots)$ is fed into the encoder, which creates a compressed representation of the input. When the compressed representation is presented to its complementary decoder, a reconstructed output image $\bar{f}(x, \dots)$ is generated. In still-image applications, the encoded input and decoded

output are $f(x,y)$ and $\bar{f}(x,y)$, respectively; in video applications, they are $f(x,y,t)$ and $\bar{f}(x,y,t)$. In general, $\bar{f}(x,\dots)$ may or may not be an exact replica of $f(x,\dots)$. If the two images are identical, the compression system is called *error free, lossless, or information preserving*. In the two images are different, the compression system is referred to as *lossy*.

The encoding and compression process

The encoder is designed to remove the three redundancies described before. In the first stage of the encoding process, a *mapper* transforms $f(x, \dots)$ into a (usually nonvisual) format designed to reduce spatial and temporal redundancy. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image.

The *quantizer* reduces the accuracy of the mapper's output in accordance with a pre-established fidelity criterion. The

goal is to keep irrelevant information out of the compressed representation. This operation is irreversible. It must be omitted when error-free compression is desired. In video applications, the *bit rate* of the encoded output is often measured (in bits/second) and used to adjust the operation of the quantizer so that a predetermined average output rate is maintained. Thus, the visual quality of the output can vary from frame to frame as a function of image content.

The *symbol coder* generates a fixed- or variable-length code to represent the quantizer output and maps the output in accordance with the code. In many cases, a variable-length code is used. The shortest code words are assigned to the most frequently occurring quantizer output values – thus minimizing coding redundancy. This operation is reversible.

The decoding or decompression process

The decoder contains only two components: a *symbol decoder* and an *inverse mapper*. They perform, in reverse order, the inverse operations of the encoder's symbol encoder and mapper.

Image Formats, Containers, and Compression Standards

An *image file format* is a standard way to organize and store image data. It defines how the data is arranged and the type of compression that is used. An *image container* is similar to a

file format but handles multiple types of image data. Image *compression standards* define procedures for compressing and decompressing images (for reducing the amount of data needed to represent an image).

ISO – *International Standards Organization*

IEC – *International Electrotechnical Commission*

ITU-T – *International Telecommunications Union*

CCITT – *Consultative Committee of the International Telephone and Telegraph*

Computer Vision

Course 8

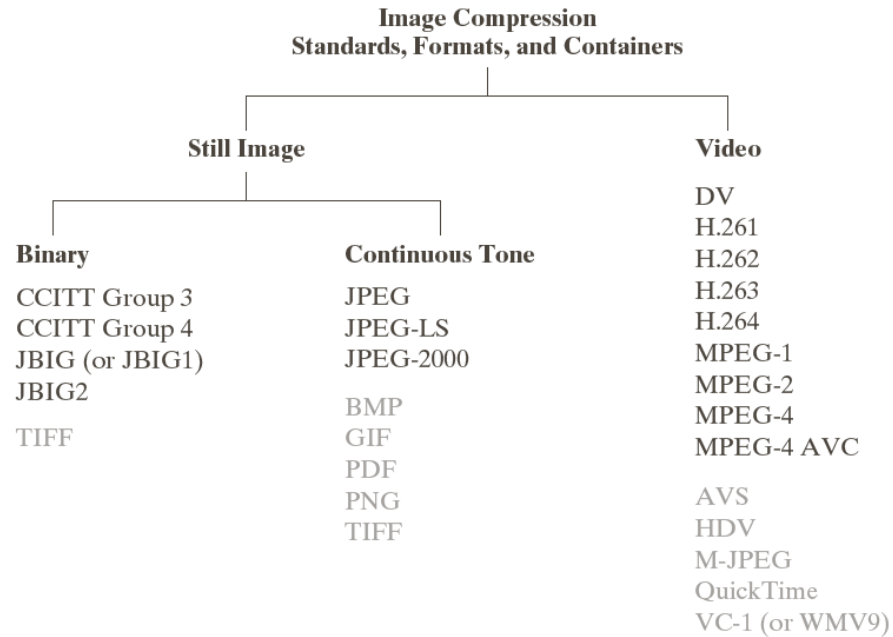


FIGURE 8.6 Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in black; all others are grayed.

TABLE 8.3
Internationally sanctioned image compression standards. The numbers in brackets refer to sections in this chapter.

Computer Vision

Course 8

Name	Organization	Description
<i>Bi-Level Still Images</i>		
CCITT Group 3	ITU-T	Designed as a facsimile (FAX) method for transmitting binary documents over telephone lines. Supports 1-D and 2-D run-length [8.2.5] and Huffman [8.2.1] coding.
CCITT Group 4	ITU-T	A simplified and streamlined version of the CCITT Group 3 standard supporting 2-D run-length coding only.
JBIG or JBIG1	ISO/IEC/ ITU-T	A <i>Joint Bi-level Image Experts Group</i> standard for progressive, lossless compression of bi-level images. Continuous-tone images of up to 6 bits/pixel can be coded on a bit-plane basis [8.2.7]. Context sensitive arithmetic coding [8.2.3] is used and an initial low resolution version of the image can be gradually enhanced with additional compressed data.
JBIG2	ISO/IEC/ ITU-T	A follow-on to JBIG1 for bi-level images in desktop, Internet, and FAX applications. The compression method used is content based, with dictionary based methods [8.2.6] for text and halftone regions, and Huffman [8.2.1] or arithmetic coding [8.2.3] for other image content. It can be lossy or lossless.
<i>Continuous-Tone Still Images</i>		
JPEG	ISO/IEC/ ITU-T	A <i>Joint Photographic Experts Group</i> standard for images of photographic quality. Its lossy <i>baseline coding system</i> (most commonly implemented) uses quantized discrete cosine transforms (DCT) on 8×8 image blocks [8.2.8], Huffman [8.2.1], and run-length [8.2.5] coding. It is one of the most popular methods for compressing images on the Internet.
JPEG-LS	ISO/IEC/ ITU-T	A lossless to near-lossless standard for continuous tone images based on adaptive prediction [8.2.9], context modeling [8.2.3], and Golomb coding [8.2.2].
JPEG-2000	ISO/IEC/ ITU-T	A follow-on to JPEG for increased compression of photographic quality images. Arithmetic coding [8.2.3] and quantized discrete wavelet transforms (DWT) [8.2.10] are used. The compression can be lossy or lossless.

(Continues)

Computer Vision

Course 8

Name	Organization	Description
<i>Video</i>		
DV	IEC	<i>Digital Video</i> . A video standard tailored to home and semiprofessional video production applications and equipment—like electronic news gathering and camcorders. Frames are compressed independently for uncomplicated editing using a DCT-based approach [8.2.8] similar to JPEG.
H.261	ITU-T	A two-way videoconferencing standard for ISDN (<i>integrated services digital network</i>) lines. It supports non-interlaced 352×288 and 176×144 resolution images, called CIF (<i>Common Intermediate Format</i>) and QCIF (<i>Quarter CIF</i>), respectively. A DCT-based compression approach [8.2.8] similar to JPEG is used, with frame-to-frame prediction differencing [8.2.9] to reduce temporal redundancy. A block-based technique is used to compensate for motion between frames.
H.262	ITU-T	See MPEG-2 below.
H.263	ITU-T	An enhanced version of H.261 designed for ordinary telephone modems (i.e., 28.8 Kb/s) with additional resolutions: SQCIF (<i>Sub-Quarter CIF</i> 128×96), 4CIF (704×576), and 16CIF (1408×512).
H.264	ITU-T	An extension of H.261–H.263 for videoconferencing, Internet streaming, and television broadcasting. It supports prediction differences within frames [8.2.9], variable block size integer transforms (rather than the DCT), and context adaptive arithmetic coding [8.2.3].
MPEG-1	ISO/IEC	A <i>Motion Pictures Expert Group</i> standard for CD-ROM applications with non-interlaced video at up to 1.5 Mb/s. It is similar to H.261 but frame predictions can be based on the previous frame, next frame, or an interpolation of both. It is supported by almost all computers and DVD players.
MPEG-2	ISO/IEC	An extension of MPEG-1 designed for DVDs with transfer rates to 15 Mb/s. Supports interlaced video and HDTV. It is the most successful video standard to date.
MPEG-4	ISO/IEC	An extension of MPEG-2 that supports variable block sizes and prediction differencing [8.2.9] within frames.
MPEG-4 AVC	ISO/IEC	MPEG-4 Part 10 <i>Advanced Video Coding</i> (AVC). Identical to H.264 above.

Computer Vision

Course 8

Name	Organization	Description
<i>Continuous-Tone Still Images</i>		
BMP	Microsoft	<i>Windows Bitmap</i> . A file format used mainly for simple uncompressed images.
GIF	CompuServe	<i>Graphic Interchange Format</i> . A file format that uses lossless LZW coding [8.2.4] for 1- through 8-bit images. It is frequently used to make small animations and short low resolution films for the World Wide Web.
PDF	Adobe Systems	<i>Portable Document Format</i> . A format for representing 2-D documents in a device and resolution independent way. It can function as a container for JPEG, JPEG 2000, CCITT, and other compressed images. Some PDF versions have become ISO standards.
PNG	<i>World Wide Web Consortium (W3C)</i>	<i>Portable Network Graphics</i> . A file format that losslessly compresses full color images with transparency (up to 48 bits/pixel) by coding the difference between each pixel's value and a predicted value based on past pixels [8.2.9].
TIFF	Aldus	<i>Tagged Image File Format</i> . A flexible file format supporting a variety of image compression standards, including JPEG, JPEG-LS, JPEG-2000, JBIG2, and others.
<i>Video</i>		
AVS	MII	<i>Audio-Video Standard</i> . Similar to H.264 but uses exponential Golomb coding [8.2.2]. Developed in China.
HDV	Company consortium	<i>High Definition Video</i> . An extension of DV for HD television that uses MPEG-2 like compression, including temporal redundancy removal by prediction differencing [8.2.9].
M-JPEG	Various companies	<i>Motion JPEG</i> . A compression format in which each frame is compressed independently using JPEG.
Quick-Time	Apple Computer	A media container supporting DV, H.261, H.262, H.264, MPEG-1, MPEG-2, MPEG-4, and other video compression formats.
VC-1 WMV9	SMPTE Microsoft	The most used video format on the Internet. Adopted for HD and <i>Blu-ray</i> high-definition DVDs. It is similar to H.264/AVC, using an integer DCT with varying block sizes [8.2.8 and 8.2.9] and context dependent variable-length code tables [8.2.1]—but no predictions within frames.

Some Basic Compression Methods

Huffman Coding (1952)

When coding the symbols of an information source individually, *Huffman coding* yields the smallest possible number of code symbols per source symbol. In the terms of Shannon's first theorem, the resulting code is optimal for a fixed value of n , subject to the constraint that the source symbols be coded *one at a time*. In practice, the source symbols may be either the intensities of an image or the

output of an intensity mapping operation (pixel differences, run length, ...).

The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. Fig. 8.7 illustrates this process for binary coding.

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6 0.4
a_6	0.3	0.3	0.3	0.3	
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1			
a_3	0.06	0.1	0.1		
a_5	0.04				

FIGURE 8.7
Huffman source
reductions.

Course 8

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source.

Original source			Source reduction			
Symbol	Probability	Code	1	2	3	4
a_2	0.4	1	0.4	1	0.4	1
a_6	0.3	00	0.3	00	0.3	00
a_1	0.1	011	0.1	011	0.2	010
a_4	0.1	0100	0.1	0100	0.1	011
a_3	0.06	01010	0.1	0101	0.3	01
a_5	0.04	01011				

FIGURE 8.8
Huffman code assignment procedure.

The average length of this code:

$$L_{avg} = 0.4 \cdot 1 + 0.3 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 4 + 0.06 \cdot 5 + 0.06 \cdot 5 = 2.2 \text{ bits/pixel}$$

$$H(p) = 2.14353$$

When a large number of symbols are to be coded, the construction of an optimal Huffman code is a nontrivial task. For the general case of J source symbols, J symbol probabilities, $J-2$ source reductions and $J-2$ code assignments are required. When source symbol probability can be estimated in advance, “near optimal” coding can be achieved with pre-computed Huffman codes. JPEG and MPEG standards specify default Huffman coding tables that have been pre-computed based on experimental data.

Arithmetic Coding

Arithmetic coding generates non-block codes, there is no one-to-one correspondence between source symbols and code words. An entire sequence of source symbols (or message) is assigned a single arithmetic code word. The code word itself defines an interval of real numbers in $[0,1]$. As the number of symbols in the message increases, the interval used to represent it becomes smaller and the number of bits required to represent the interval becomes larger. Each symbol of the

message reduces the size of the interval in accordance with its probability of occurrence.

Figure 8.12 and Table 8.6 illustrate the basic arithmetic coding process, for the sequence $a_1 a_2 a_3 a_3 a_4$ from a four-symbol source. At the end of the coding process we get the interval $[0.06752, 0.0688]$.

Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

TABLE 8.6
Arithmetic coding
example.

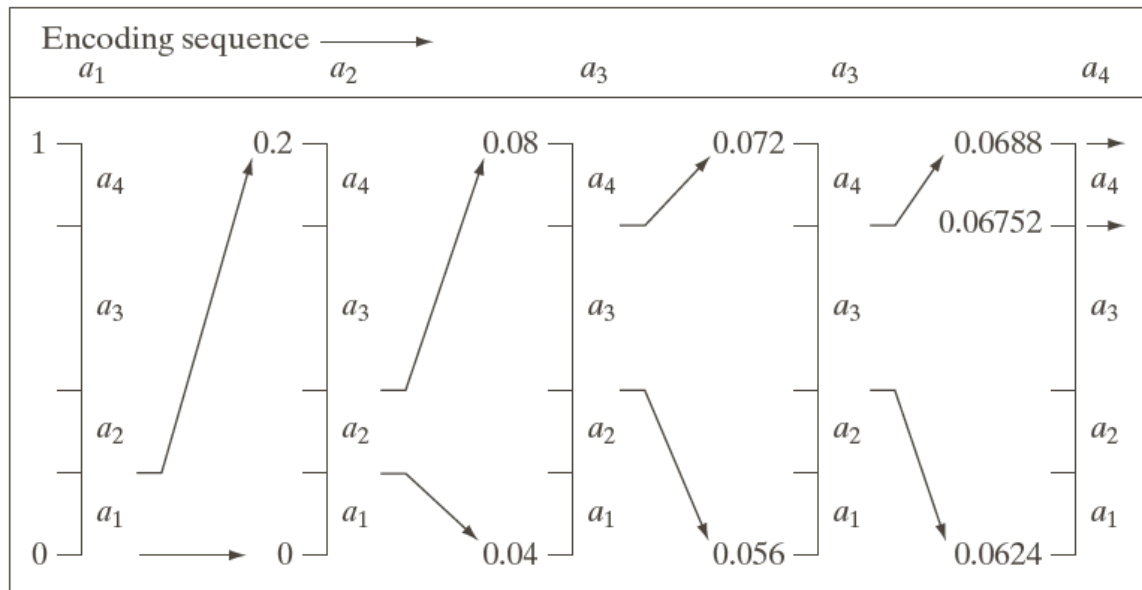


FIGURE 8.12
Arithmetic coding
procedure.

Any number within the interval found can be used to represent the message.

Practical implementations of arithmetic coding take into account a scaling strategy and a rounding strategy. The scaling strategy renormalizes each subinterval to the $[0,1)$ range before subdividing it in accordance with the symbol probabilities. The rounding strategy guarantees that the truncations associated with finite precision arithmetic do not prevent the coding subintervals from being represented accurately.

Adaptive context dependent probability estimates

With accurate input symbol probability models, that is, models that provide the true probabilities of the symbols being coded, arithmetic coders are near optimal in the sense of minimizing the average number of code symbols required to represent the symbols being coded.

Adaptive probability models update symbol probabilities as symbols are being coded or become known. Thus, the probabilities adapt to the local statistics of the symbols being

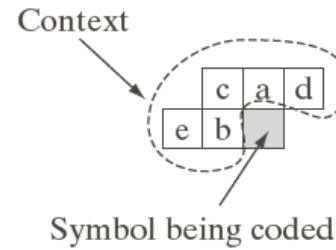
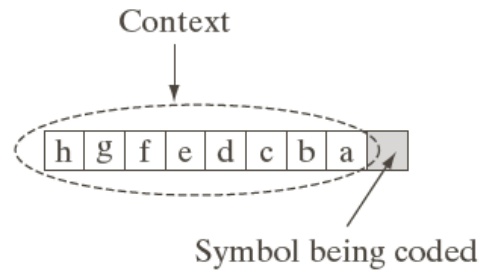
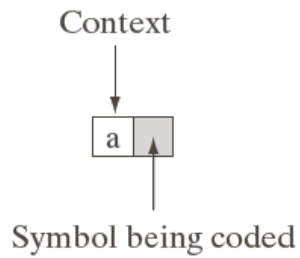
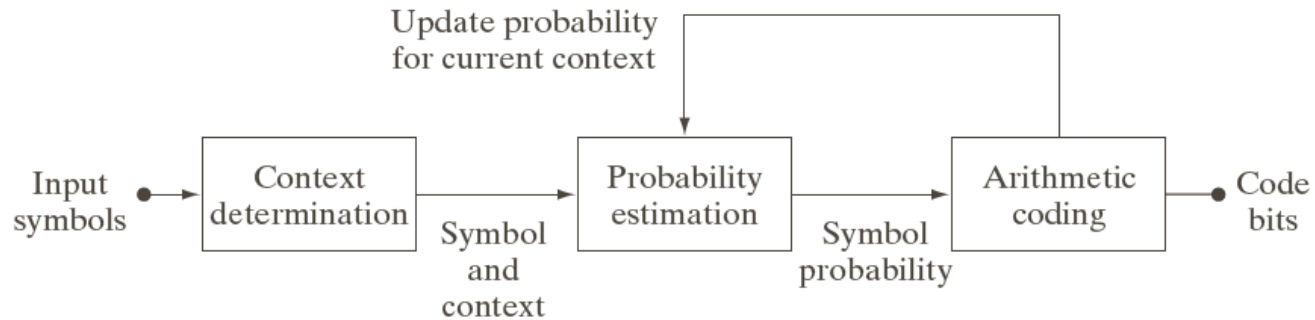
coded. *Context dependent* models provide probabilities that are based on a predefined neighborhood of pixels – called the *context* – around the symbols being coded. A *causal context*, one limited to symbols that have already been coded, is used. Both the Q-coder and MQ-coder, two arithmetic coding techniques that have been incorporated into the JBIG, JPEG-2000, use probability models that are both adaptive and context dependent. The Q-coder dynamically updates symbol

probabilities during the interval renormalizations that are part of the arithmetic coding process.

Arithmetic coding often is used when binary symbols are to be coded.

Computer Vision

Course 8



a
b c d

FIGURE 8.13
(a) An adaptive, context-based arithmetic coding approach (often used for binary source symbols). (b)–(d) Three possible context models.

LZW Coding

Lempel-Ziv-Welch (LZW) coding assigns fixed-length code words to variable length sequences of source symbols. A key feature of LZW coding is that it requires no a priori knowledge of the probability of occurrence of the symbols to be encoded. LZW compression has been integrated into a variety of mainstream imaging formats, including GIF, TIFF, and PDF. The PNG format was created to get around LZW licensing requirements.

LZW coding is conceptually very simple: a codebook or *dictionary* containing the source symbols to be coded is constructed. For 8-bit monochrome images, the first 256 words of the dictionary are assigned to intensities 0,1,..., 255. As the encoder sequentially examines image pixels, intensity sequences that are not in the dictionary are placed in algorithmically determined (e.g. the next unused) locations. If the first two pixels of the image are white, for instance, sequence “255-255” might be assign to location 256. The

next time that two consecutive white pixels are encountered, code word 256 is used to represent them. If a 9-bit, 512-word dictionary is employed in the coding process, the original (8+8) bits that were used to represent the two pixels are replaced by a single 9-bit code word.

Clearly, the size of the dictionary is an important system parameter. If it is too small, the detection of matching intensity-level sequences will be less likely; if it is too large,

Computer Vision

Course 8

the size of the code words will adversely affect compression performance.

Consider the image

39 39 126 126
39 39 126 126
39 39 126 126
39 39 126 126

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

EXAMPLE 8.7:
LZW coding.

Computer Vision

Course 8

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

TABLE 8.7
LZW coding
example.

Note that a unique feature of the LZW coding is that the coding dictionary is created while the data are being encoded. An LZW decoder builds an identical decompression dictionary as it decodes simultaneously the encoded data stream. Most practical applications require a strategy for handling dictionary overflow. A simple solution is to flush or reinitialize the dictionary when it becomes full and continue coding with a new initialized dictionary. A more complex option is to monitor compression performance and flush the

dictionary when it becomes poor or unacceptable. Alternatively, the least used dictionary entries can be tracked and replaced when necessary.

Run-Length Coding

Images with repeating intensities along their rows (or columns) can often be compressed by representing runs of identical intensities as *run-length pairs*, where each run-length pair specifies the start of a new intensity and the number of pixels that have that intensity. *Run-length*

encoding (RLE) was developed in the '50s and became the standard compression approach in facsimile (FAX) coding. Compression is achieved by eliminating a simple form of spatial redundancy – groups of identical intensities.

The BMP file format uses a form of run-length encoding in which image data is represented in two different modes: encoded and absolute – and either mode can occur anywhere in the image. In *encoded* mode, a two byte RLE representation is used. The first byte specifies the number of

consecutive pixels that have the colour index contained in the second byte. The 8-bit colour index selects the run's intensity (colour or grey value) from a table of 256 possible intensities. In *absolute* mode, the first byte is 0 and the second byte signals one of four possible conditions as shown in Table 8.8.

Second Byte Value	Condition
0	End of line
1	End of image
2	Move to a new position
3–255	Specify pixels individually

TABLE 8.8
BMP absolute coding mode options. In this mode, the first byte of the BMP pair is 0.

If the second byte is 2, the next two bytes contain unsigned horizontal and vertical offsets to a new spatial position (and pixel) in the image. If the second byte is between 3 and 255, it specifies the number of uncompressed pixels that follow – with each subsequent byte containing the colour index of one pixel. The total number of bytes must be aligned on a 16-bit word boundary.

Run-length encoding is particularly effective when compressing binary images. There are only two possible

intensities (black and white), adjacent pixels are more likely to be identical. Each image row can be represented by a sequence of length only – rather than length-intensity pairs. The basic idea is to code each contiguous group (i.e. run) of 0s and 1s encountered in a left to right scan of a row by its length and to established a convention for determining the value of the run. The most common conventions are: (1) to specify the value of the first run on each row or (2) to assume that each row begin with a white run, whose length may be 0.

Additional compression can be achieved by variable-length coding the run length themselves. The black and white run length can be coded separately using variable length codes that are specifically made to their own statistics.

Symbol-Based Coding

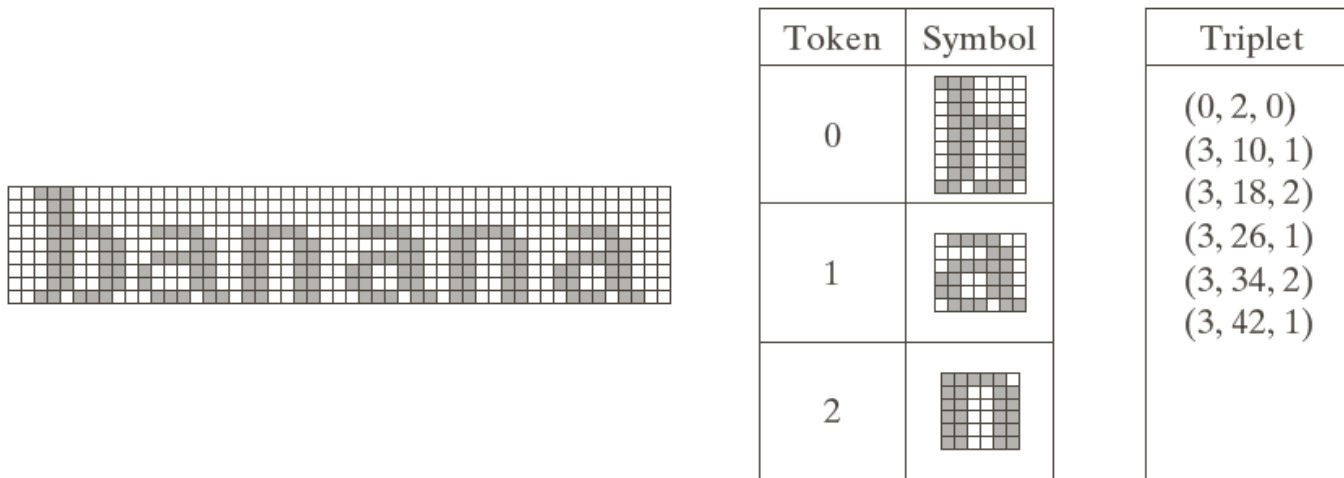
In *symbol- or token-based* coding, an image is represented as a collection of frequently occurring sub-images, called *symbols*. Each such symbol is stored in a *symbol dictionary* and the image is coded as a set of triplets $(x_i y_i, t_i)$ where each

(x_i, y_i) pair specifies the location of a symbol in the image and the token t_i is the address of the symbol or sub-image in the dictionary. Each triplet represents an instance of a dictionary symbol in the image. Storing repeated symbols only once can compress images significantly – particularly in document storage and retrieval applications, where the symbols are often character bitmaps that are repeated many times.

$$\{(x_1, y_1, t_1), (x_2, y_2, t_2), \dots\}$$

Computer Vision

Course 8



a b c

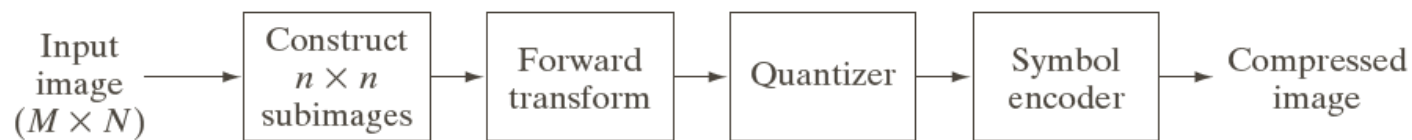
FIGURE 8.17

(a) A bi-level document,
(b) symbol dictionary, and
(c) the triplets used to locate the symbols in the document.

Block Transform Coding

Consider a compression technique that divides an image into small non-overlapping blocks of equal size (e.g. 8×8) and processes the blocks independently using a 2-D transform. In *block transform coding*, a reversible, linear transform (such as the Fourier transform) is used to map each *block* or *subimage* into a set of transform coefficients, which are then quantized and coded. For most images, a significant number of

coefficients have small magnitudes and can be quantized (or omitted entirely) with little image distortion.



a
b

FIGURE 8.21
A block transform coding system:
(a) encoder;
(b) decoder.

An $M \times N$ input image is subdivided first into subimages of size $n \times n$, which are then transformed to generate MN/n^2 subimage transform arrays, each of size $n \times n$. The goal of the

transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into the smallest number of transform coefficients. The quantization stage then selectively eliminates or more coarsely quantizes the coefficients that carry the least amount of information in a predefined sense. These coefficients have the smallest impact on reconstructed subimage quality. The encoding process terminates by coding (normally using a variable-length code) the quantized coefficients. Any or all of the transform

encoding steps can be adapted to local image content, called *adaptive transform coding*, or fixed for all subimages, called *non-adaptive transform coding*.

Transform selection

The choice of a particular transform in a given application depends on the amount of reconstruction error that can be tolerated and the computational resources available. Compression is achieved during the quantization of the transformed coefficients (not during the transformation steps).

Consider a subimage $g(x,y)$ of size $n \times n$ whose forward, discrete transform, $T(u,v)$, can be expressed in terms of the general relation:

$$T(u,v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x,y)r(x,y,u,v) \quad , \quad u,v = 0,1,\dots,n-1. \quad (1)$$

Given $T(u,v)$, $g(x,y)$ similarly can be obtained using the generalized inverse discrete transform:

$$g(x,y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v)s(x,y,u,v) \quad , \quad x,y = 0,1,\dots,n-1 \quad (2)$$

$r(x,y,u,v)$ and $s(x,y,u,v)$ are called the *forward* and *inverse transformation kernels*, respectively (also called *basis functions* or *basis images*). The $T(u,v)$ for $u,v=0,1,\dots,n-1$ are called *transform coefficients*.

The kernel in (1) is *separable* if:

$$r(x,y,u,v) = r_1(x,u)r_2(y,v)$$

The kernel is *symmetric* if $r_1 = r_2$

$$r(x,y,u,v) = r_1(x,u)r_1(y,v)$$

For the discrete Fourier transform:

$$r(x, y, u, v) = e^{-i2\pi \frac{(ux+vy)}{n}}$$

$$s(x, y, u, v) = \frac{1}{n^2} e^{i2\pi \frac{(ux+vy)}{n}}$$

Walsh-Hadamard transform (WHT) – $n = 2^m$

$$r(x, y, u, v) = s(x, y, u, v) = \frac{1}{n} (-1)^{\sum_{i=0}^{m-1} [b_i(x)p_i(u) + b_i(y)p_i(v)]}$$

The summation in the exponent of (-1) in the above expression is performed in modulo 2 arithmetic and $b_k(z)$ is

the k th (from right to left) in the binary representation of z .

The $p_i(u)$ are computed using:

$$p_0(u) = b_{m-1}(u)$$

$$p_1(u) = b_{m-1}(u) + b_{m-2}(u)$$

$$p_2(u) = b_{m-2}(u) + b_{m-3}(u)$$

\vdots

$$p_{m-1}(u) = b_1(u) + b_0(u)$$

where the sums are performed in modulo 2 arithmetic. The

Walsh-Hadamard kernels consist of alternating $+1$ and -1

arranged in a checkboard pattern. Figure 8.22 shows the kernel for $n=4$. Each block consists of $4 \times 4 = 16$ elements.

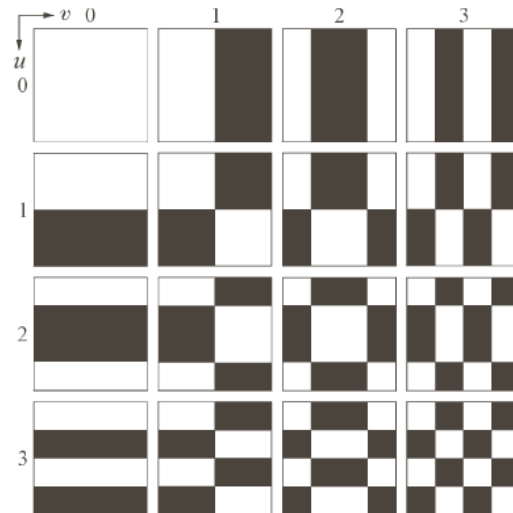


FIGURE 8.22
Walsh-Hadamard
basis functions for
 $n = 4$. The origin
of each block is at
its top left.

Discrete cosine transform (DCT)

$$\begin{aligned} r(x, y, u, v) &= s(x, y, u, v) = \\ &= \alpha(u)\alpha(v) \cos\left[\frac{(2x+1)u\pi}{2n}\right] \cos\left[\frac{(2y+1)v\pi}{2n}\right] \end{aligned}$$

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{n}} & \text{for } u = 0 \\ \sqrt{\frac{2}{n}} & \text{for } u = 1, 2, \dots, n-1 \end{cases}$$

Computer Vision

Course 8

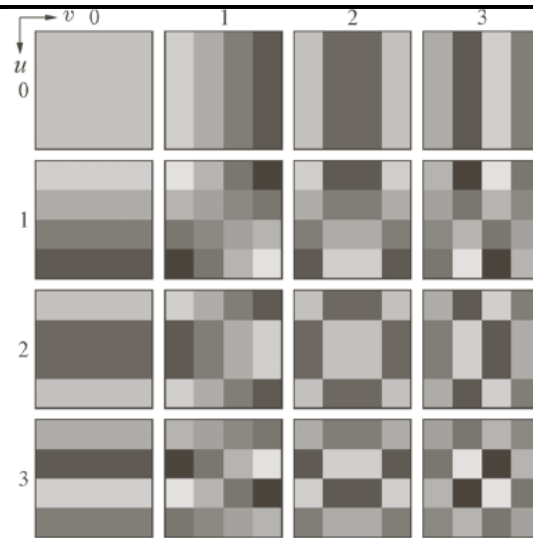


FIGURE 8.23

Discrete-cosine basis functions for $n = 4$. The origin of each block is at its top left.



FIGURE 8.9(a)



a	b	c
d	e	f

FIGURE 8.24 Approximations of Fig. 8.9(a) using the (a) Fourier, (b) Walsh-Hadamard, and (c) cosine transforms, together with the corresponding scaled error images in (d)–(f).

Figure 8.24(a) show three approximations of the 512×512 monochrome image in Figure 8.9(a). These pictures were obtained by dividing the original image into subimages of size 8×8 , representing each subimage using the transforms DFT, WHT and DCT, truncating 50% of the resulting coefficients, and taking the inverse transform of the truncated coefficient arrays. In each case, the 32 retained coefficients were selected on the basis of maximum magnitude.

This example shows that the information packing ability of the DCT is superior to that of DFT and WHT - this observation usually holds for most images. The Karhunen-Loève transform is the optimal transform in an information packing sense – it is a data dependent transform and its computational task is significant.

Most transform coding systems are based on the DCT which provides a good compromise between information packing ability and computational complexity.

Subimage size selection

In most applications, images are subdivided so that the correlation (redundancy) between adjacent subimages is reduced to some acceptable level and so that n is an integer power of 2. The most popular subimage sizes are 8×8 and 16×16 .

Image Segmentation

Segmentation subdivides an image into its constituent regions and objects. The level of detail to which the subdivision is carried depends on the problem being solved. Segmentation should stop when the objects or regions of interest in an application have been detected. For example, in the automated inspection of electronic assemblies, interest lies in analysing images of products with the objective of determining the presence or absence of specific anomalies,

such as missing components or broken connection paths. There is no point in carrying segmentation past the level of detail required to identify those elements.

Segmentation of nontrivial images is one of the most difficult tasks in image processing. Segmentation accuracy determines the eventual success or failure of computerized analysis procedure.

https://darwin-public.s3.eu-west-1.amazonaws.com/splash_page/v7-vs-other-tools.mp4

Image segmentation tasks:

(<https://www.v7labs.com/blog/image-segmentation-guide>)

1. **Semantic segmentation** segments out a broad boundary of objects belonging to a particular class. It classifies pixels in an image into semantic classes. Pixels belonging to a particular class are simply classified to that class with no other information or context taken into consideration.

2. **Instance segmentation** provides a segment map for each object it views in the image, without any idea of the class the object belongs to. Pixels are classified into categories on the basis

of “instances” rather than classes. Overlapping or very similar object regions are distinguished on the basis of their boundaries.

3. **Panoptic segmentation** is the conjugation of instance and semantic segmentation tasks. Panoptic segmentation gives us the segment maps of all the objects of any particular class present in the image.

**Semantic Segmentation vs. Instance Segmentation
vs. Panoptic Segmentation**



(a) Image



(b) Semantic Segmentation



(c) Instance Segmentation



(d) Panoptic Segmentation

