IMAGE PROCESSING

COMPRESSION AND CODING

Chakravarthy Bhagvati
Dept. of Computer and Information Sciences
University of Hyderabad

16 September 2006

OVERVIEW

- Introduction
  - need for compression
  - data redundancy
  - measuring redundancy
  - information theory
- Lossless Compression Techniques
  - Huffman encoding and variants
  - Arithmetic encoding
  - Binary image encoding
  - Run-length encoding
  - Predictive encoding
- Lossy Compression Techniques

IMAGE COMPRESSION

- Images require enormous storage, especially when colour
  - A4 page at 300 dpi $\approx$ 7 MB (grayscale), $\approx$ 25 MB (colour)
- There is a great need to reduce image data size
- To understand compression, we need to distinguish between *data* and *information*
  - *data* is the set of symbols for conveying *information*, e.g.,
    - for a time I stood pondering on circles (38B)
    - 31415926 / 10000000 (17B)
    - SYMBOL FONT p (12B)
    - π (1B)
    - pi (2B)
  - The information is π to 7 decimal places, but data varies

REDUNDANCY

- Data has three types of redundancies
  - *coding* redundancy: data representation can be made more efficient
    - Huffman coding, Arithmetic coding, etc.
  - *inter-pixel* redundancy: correlations between adjacent pixels
    - run-length coding, left pixel subtraction coding, etc.
  - *psycho-visual* redundancy: humans do not necessarily perceive all the information
    - reduction in number of colours, quantization, etc.
- Compression aims to reduce all three redundancies, but psycho-visual redundancies normally result in *lossy* compression

MEASURING CODING REDUNDANCY

- *Relative Data Redundancy, R_D*

$$R_D = 1 - \frac{1}{C_R}$$
$$C_R = \frac{n_1}{n_2}$$

C_R is the *compression ratio*

- Key concept: *average number of bits* needed to represent a pixel, L_{avg},

$$\text{Let } p_r(r_k) = \frac{n_k}{n}, k = 0, 1, 2, \ldots, L-1$$
$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

- For an uncompressed image, $l(r_k) = 8, k = 0, 1, 2, \ldots, L-1$, and $L_{avg} = l(r_k) = 8$

MEASURING INTERPIXEL REDUNDANCY

- Interpixel redundancy measured as correlation between adjacent pixels. Correlation, $\gamma(\Delta n)$, between two pixels separated by a distance Δn,

$$\gamma(\Delta n) = \frac{A(\Delta n)}{A(0)}$$

$$A(\Delta n) = \frac{1}{N - \Delta n} \sum_{y=0}^{N-1-\Delta n} f(x,y) f(x, y+\Delta n)$$

  $\gamma(\Delta n)$ should be ≈ 0 if there is no correlation between pixels that are Δn apart

- Interpixel redundancy is also referred to as *geometric, spatial* or *inter-frame* redundancy

FIDELITY CRITERIA

- How do we know that compression did not lose any information?
- Error analysis

$$
\begin{aligned}
e(x, y) &= \hat{f}(x, y) - f(x, y) \\
e &= \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)] \\
e_{rms} &= \sqrt{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]}
\end{aligned}
$$

- Sometimes, *signal-to-noise ratio, SNR* is used instead of e_{rms}

$$
SNR_{rms} = \sqrt{\frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x, y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]^2}}
$$

DEALING WITH CODING REDUNDANCY

- How do we know if **there is** coding redundancy?

  Answer: Find the *minimum* number of bits needed per pixel. If it is smaller than L_{avg} then there is coding redundancy

- How do we find minimum number of bits needed for representing a pixel?

  Answer: *information theory*!

- Shannon's theory: *self-information* of a message E, $I(E)$, is

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

  where $P(E)$ is the probability of occurrence of E

A WEE BIT OF INFORMATION THEORY

The minimum number of bits needed is equal to the *entropy* per pixel.
Let $A = \{a_1, a_2, \ldots, a_J\}$ and the probability that a_j occurs is $P(a_j)$.
Then

$$I(a_j) = -\log P(a_j)$$

If k source symbols are generated, then from the law of large numbers, the symbol a_j will be output $kP(a_j)$ times.
The average information from k source symbols is

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

The average information per pixel is

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

This is called the *entropy* of the source (in this case, image)
If $L_{avg} >$ *entropy*, then there is redundancy

HUFFMAN CODING

- *Huffman coding* assigns short codewords to the most probable values and long codewords to rarely occurring symbols
- Is the most popular coding redundacy minimization technique

Huffman Coding Algorithm

STEP 1: Create a series of gray scale reductions

- Sort gray scales according to probabilities of occurrences
- Take the two least occurring gray scales and combine them to create a new *reduced gray scale*
- Repeat the above steps until only two gray scales remain

ORIGINAL		REDUCED					
a_j	$P(a_j)$	1	2	3	4	5	6
0	0.4	0.4	0.4	0.4	0.4	0.4	0.6
1	0.3	0.3	0.3	0.3	0.3	0.3	0.4
2	0.1	0.1	0.1	0.1	0.2	0.3	
3	0.1	0.1	0.1	0.1	0.1		
4	0.04	0.04	0.06	0.1			
5	0.03	0.03	0.04				
6	0.02	0.03					
7	0.01						

HUFFMAN CODING . . .

STEP 2: Assign codewords to each gray scale

- Assign codes 0 and 1 arbitrarily to the two gray scales obtained at the end of the first step.

  Note that one is real while the other is a reduced gray scale.
- Assign the code for the reduced gray scale to both its constituents
- Append 0 to one and 1 to the other constituent
- Repeat the above process until all gray scales are assigned a code

ORIGINAL		REDUCED					
a_j	$P(a_j)$	1	2	3	4	5	6
0	0.4	0.4	0.4	0.4	0.4	0.4	0.6
1	0.3	0.3	0.3	0.3	0.3	0.3	0.4
2	0.1	0.1	0.1	0.1	0.2	0.3	
3	0.1	0.1	0.1	0.1	0.1		
4	0.04	0.04	0.06	0.1			
5	0.03	0.03	0.04				
6	0.02	0.03					
7	0.01						

Assigned Codes

ORIG.	CODE	ORIG.	CODE
0	1	4	01011
1	00	5	010100
2	011	6	0101010
3	0100	7	0101011

COMPRESSION RATIO CALCULATIONS

- How good is Huffman encoding?
  - Entropy = 2.01
  - Uncompressed Image: $L_{avg} = 3$
  - Huffman Encoded:

$$1\times0.4+2\times0.3+3\times0.1+4\times0.1+5\times0.04+6\times0.03+7\times0.02+7\times0.01 = 2.29$$

  - Compression Ratio: 3.0 / 2.29 = 1.31
- Decompression is *very easy*
  - Scan encoded string from left to right
  - Whenever a codeword is seen, output it

Such simplicity possible because *no codeword is a prefix of another*

Decode: 1010101111001011

ORIG.	CODE	ORIG.	CODE
0	1	4	01011
1	00	5	010100
2	011	6	0101010
3	0100	7	0101011

HUFFMAN CODING VARIANTS

- Huffman code is theoretically the best
  - It comes nearest to entropy except that it is restricted to integer length codewords
  - Very slow to compute as it computes one codeword at a time
  - For a 24-bit colour image it is prohibitively slow
  - Length of codeword becomes large for small probabilities
- Usually variants on Huffman coding are used in practice
  - Truncated Huffman – use standard binary representation for all the low probability symbols
  - Huffman Shift — divide symbols into blocks; Huffman within a block; and use a *shift* symbol to travel from block to block
- Other codes
  - B_2 code – for symbols obeying exponential distributions
  - Binary shift code

ARITHMETIC CODING

- Huffman is an integer code, i.e., length of codeword is an integer
- Entire sequence of source symbols is mapped into a single real number in the interval $[0, 1)$
- The width of the interval depends on the probability of occurrence of the source symbol

a_j	$P(a_j)$	*sub-interval*	*example*
0	0.2	[0.0, 0.2)	Msg: 31415
1	0.3	[0.2, 0.5)	
2	0.05	[0.5, 0.55)	
3	0.15	[0.55, 0.7)	
4	0.2	[0.7, 0.9)	
5	0.1	[0.9, 1.0)	

LZW CODING

- Combines coding efficiency with interpixel redundancy
- Assigns fixed-length code words to variable-length source sequences
- It must be licensed under US Patent No. 4,558,302
- LZW coding is used in GIF, TIFF and PDF
- How does LZW work?
  - constructs a codebook or *dictionary*
  - pixel values $0, 1, \ldots, 255$ are unchanged
  - sequences of gray levels are assigned values from 256
    - For example, a sequence of $180, 190$ may be assigned 256
  - dictionary used in encoding and decoding the image
- Remarkable fact: dictionary can be reconstructed while decoding!
- Dictionary management is the big issue

LZW EXAMPLE

Currently Recog. Sequence	Current Pixel	Encoded Output	Code Word	Code 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
⋮	⋮	⋮	⋮	⋮

Image Segment

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

LOSSLESS PREDICTIVE CODING

- Minimizes *interpixel* redundancy
- Idea is to predict the next pixel and then store only the difference between actual and predicted values
- The difference is encoded using any of the earlier methods for greater compression
- 1-D Linear Predictive Coding scheme

$$\text{Let } e = \hat{f}(x,y) - f(x,y)$$
$$\hat{f}(x,y) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(x, y-i)\right]$$

- Other functions may be used to model $\hat{f}(x,y)$ such as *exponential smoothing*, *moving average*, etc.

LOSSY COMPRESSION

- Lossless compression schemes give very low compression ratios in general – usually between 2:1 and 5:1
- Compromise accuracy for increased compression ratios. Psycho-visual redundancy allows such compromises to be tolerated
- Compression ratios in excess of 30:1 are common. At ratios of 20:1, images are virtually indistinguishable from originals
- Common techniques
  - Lossy predictive coding
    - delta modulation
    - diffential pulse code modulation
  - Transform coding – DFT, DCT, etc.
  - Zonal coding
  - Threshold coding
- Image Compression Standards – JPEG

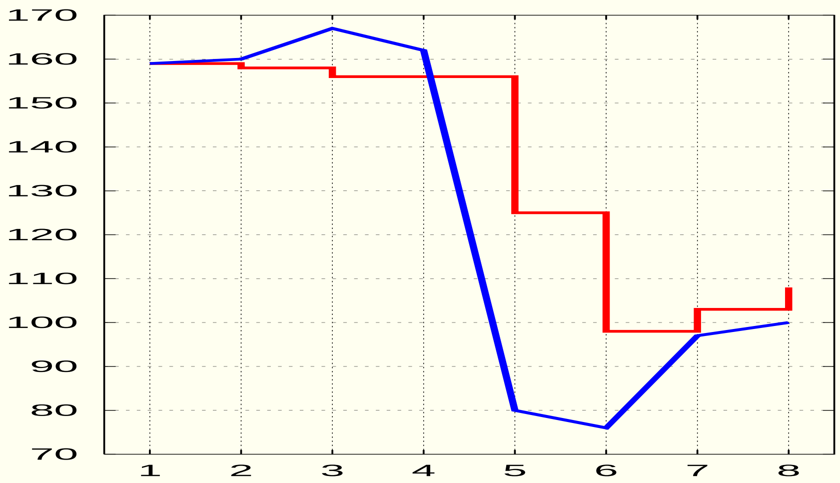
LOSSY PREDICTIVE CODING

- One of the simplest schemes is *delta modulation*
- As ζ is fixed, we need only to transmit the *sign*, i.e., 1 bit

$$\begin{aligned}\hat{f}_1 &= f_1\\ \hat{f}_n &= \alpha \hat{f}_{n-1} + e_n\\ e_n &= \begin{cases} +\zeta & \text{if } f_n - \alpha \hat{f}_{n-1} > 0\\ -\zeta & \text{otherwise}\end{cases}\end{aligned}$$

Example with $\alpha = 0.9$ and $\zeta = 15$

Input gray scales:
159 160 167 162 80 76 97 100



$$\begin{aligned}\hat{f}_1 &= 159\\ \hat{f}_2 &= 0.9 \times 159 + 15 = 158\\ \hat{f}_3 &= 0.9 \times 158 + 15 = 157\\ \hat{f}_4 &= 0.9 \times 157 + 15 = 156\\ \hat{f}_5 &= 0.9 \times 156 - 15 = 125\end{aligned}$$

$$\begin{aligned}\hat{f}_6 &= 0.9 \times 125 - 15 = 98\\ \hat{f}_7 &= 0.9 \times 98 + 15 = 103\\ \hat{f}_8 &= 0.9 \times 103 + 15 = 108\end{aligned}$$

OPTIMAL ENCODING

- Several variants of delta modulation exist
- Optimal encoding and *differential pulse code modulator (DPCM)*
  - minimize the encoder's mean-sqared-error

$$E\{e_n^2\} = E\{[f_n - \hat{f}_n]^2\}$$

- Assuming prediction is constrained to linear combination of m previous pixels

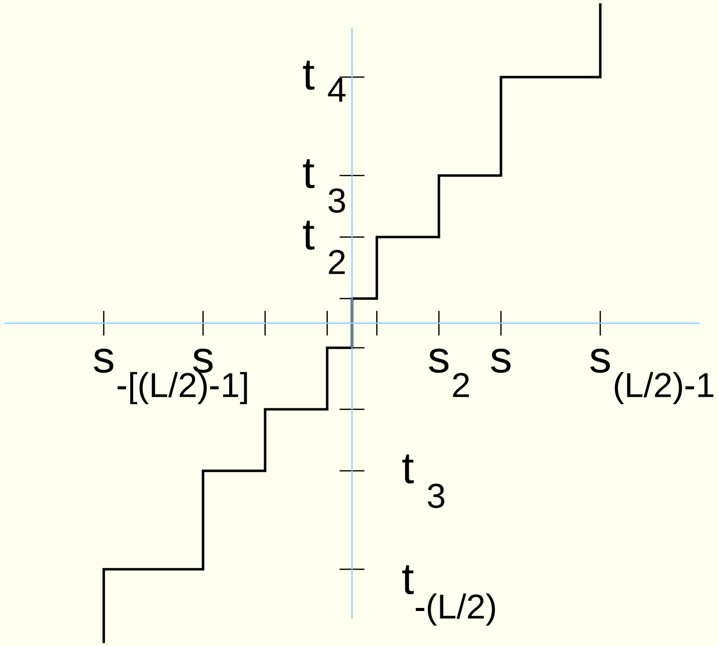
$$\hat{f}_n = \sum_{i=1}^{m} \alpha_i f_{n-i}$$

$$\text{Minimize } E\{e_n^2\} = E\{[f_n - \sum_{i=1}^{m} \alpha_i f_{n-i}]^2\}$$

- The above system may be solved as a system of simultaneous equations. Normally, $m \leq 3$

OPTIMAL QUANTISER

- Generic quantiser is a *staircase* function
- s are the *decision* points
- t are the *reconstruction* points
- We chose optimal s and t



Optimisation Procedure

- *Key Idea:* The number of pixels should be uniformly distributed over each quantisation interval

Optimisation Conditions

$$\int_{s_{i-1}}^{s_i} (s - t_i)p(s)ds = 0, \; i = 1, 2, \ldots, \frac{L}{2}$$

$$s_i = \begin{cases} 0 & i = 0 \\ \dfrac{t_i + t_{i+1}}{2} & i = 1, 2, \ldots, \frac{L}{2} - 1 \\ \infty & i = \frac{L}{2} \end{cases}$$

and

$$s_{-i} = -s_i, \quad t_{-i} = -t_i$$

TRANSFORM CODING: DCT

- Instead of the spatial domain, use a transform domain obtained from FFT, DFT, DCT, WHT, KLT, etc.
- Many coefficients in transform domain may be close to 0 and can be ignored
- We get high compression ratios with good image quality
- Some issues in transform coding
  - non-sinusoidals, e.g., WHT, are easy to implement
  - image independent basis functions are computationally better
  - sinusoids pack information better (i.e., they approximate original images better)
- Given the above, a good choice is *discrete cosine transform (DCT)*
  - sinusoid and independent basis functions
- JPEG is based on DCT

DISCRETE COSINE TRANSFORM

- The forward transform of an image $i(x,y)$ is given by

$$T(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} i(x,y)g(x,y,u,v)$$

- Given $T(u,v)$, the image $i(x,y)$ is given by the inverse transform

$$i(x,y) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} T(u,v)h(x,y,u,v)$$

- $g(x,y,u,v)$ and $h(x,y,u,v)$ are called the forward and inverse transformaiton kernels respectively
- *Discrete Cosine Transform* is defined by the following kernel pair

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

where

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

DCT MASKING FUNCTIONS

- In DCT, $g(x, y, u, v)$ and $h(x, y, u, v)$ are independent of the values of $i(x, y)$ or $T(u, v)$. Therefore, we rewrite the tranform equation as

$$\mathbf{I} = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} T(u, v)\mathbf{H}_{u,v}, \quad \text{where}$$

$$\mathbf{H}_{uv} = \begin{bmatrix} h(0,0,u,v) & h(0,1,u,v) & \cdots & h(0,n-1,u,v) \\ h(1,0,u,v) & h(1,1,u,v) & \cdots & h(1,n-1,u,v) \\ \vdots & \vdots & \vdots & \vdots \\ h(n-1,0,u,v) & h(n-1,1,u,v) & \cdots & h(n-1,n-1,u,v) \end{bmatrix}$$

- A coefficient *masking* function $\gamma(u, v)$ may be defined as

$$\gamma(u, v) = \begin{cases} 0 & \text{if } T(u, v) \text{ satisfies a truncation condition} \\ 1 & \text{otherwise} \end{cases}$$

- An approximation of the image $\hat{\mathbf{I}}$ is

$$\hat{\mathbf{I}} = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \gamma(u, v) T(u, v)\mathbf{H}_{u,v}$$

ENCODING THE MASKING FUNCTIONS

- $\gamma(u, v)$ is the key to compression
- *Truncation condition* is one of
  - Zonal coding
  - Threshold coding
- Truncating, quantising and encoding is called *bit allocation*

Zonal Coding

- Divide the image into $k \times k$ sub-blocks
- Compute $k \times k$ DCT coefficients
- Retain the coefficients that show maximum *variance*

Zonal Coding Mask

1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	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
0	0	0	0	0	0	0	0

8	7	6	4	3	2	1	0
7	6	5	4	3	2	1	0
6	5	4	4	3	1	1	0
4	4	3	3	2	1	0	0
3	3	3	2	1	1	0	0
2	2	1	1	1	0	0	0
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Zonal Bit Allocation

THRESHOLD CODING

- Zonal coding is fixed for all sub-blocks
- Threshold coding varies adaptively for different sub-blocks
- Retain *largest valued* coefficients

- Three ways to do threshold coding
  - *Global* threshold
  - *Local* threshold for each sub-block
  - *Function* that varies with each sub-block

- Global threshold gives variable compression ratios
- Local threshold always retains a fixed number of coefficients in each block and gives fixed compression ratio
  - also called *N-largest coding*
- The third scheme has maximum flexibility
- Define a *normalisation matrix*

$$\hat{T}(u,v) = \text{round}\left[\frac{T(u,v)}{Z(u,v)}\right]$$

- The matrix Z combines a delta-modulation scheme with coefficient selection

JPEG COMPRESSION SCHEME

- JPEG (Joint Photographic Experts Group) standardises DCT based compression scheme
- JPEG is lossy DCT scheme based on 8×8 sub-blocks using a standard normalization matrix

Normalization Matrix

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

25%
(21869 B)

Orig
(204624 B)

EXAMPLES (BALLOON)

Orig (1.28MB)

JPG (92KB)

75% (16KB)

50% (11KB)

25% (8.6KB)

5% (6KB)

EXAMPLES (PARROTS)

Orig (1.06MB)

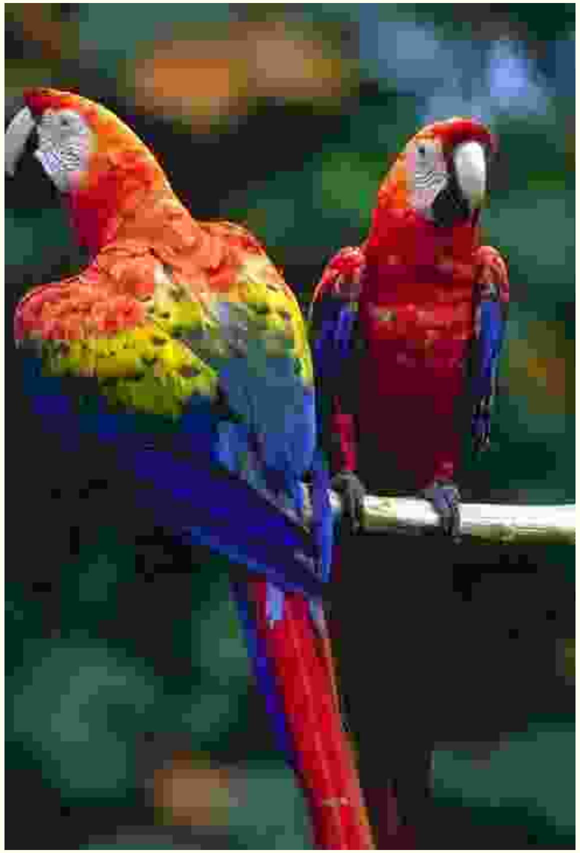
JPG (162KB)

75% (45KB)

PARROTS (CONTD.)

50% (34KB)

25% (20KB)



5% (9KB)

SUMMARY

- Image compression is possible because of redundancies in images
  - coding redundancy
  - interpixel redundancy
  - psychovisual redundancy
- Compression can be lossless or lossy
  - lossless compression ratios are quite small
  - lossy compression gives higher ratios
- DCT is the most popular compression technique today
- JPEG standardises DCT
- Wavelet based compression is the rage in research - latest version of JPG permits wavelets for compression