Image Compression

cuu duong than cong . com

Instructor
LE Thanh Sach, Ph.D.

Ltsach@cse.hcmut.edu.vn

http://www.cse.hcmut.edu.vn/~ltsach/
Outline

❖ Introduction
❖ Lossless Compression
❖ Lossy Compression
Introduction

- The goal of image compression is to reduce the amount of data required to represent a digital image.
- Important for reducing storage

[Diagram of compression and decompression process]
Approaches

❖ **Lossless**
  - Information preserving
  - Low compression ratios
  - e.g., Huffman

❖ **Lossy**
  - Does not preserve information
  - High compression ratios
  - e.g., JPEG

❖ **Tradeoff:** image quality vs compression ratio
Data vs Information

- Data and information are not synonymous terms!

- **Data** is the means by which **information** is conveyed.

- Data compression aims to reduce the amount of data required to represent a given quantity of information while preserving as much information as possible.
Data Redundancy

- Data redundancy is a mathematically quantifiable entity!

Data Set 1
(image)

n1 carrying units
(e.g., bits)

compression

Data Set 2
(compressed image)

n2 carrying units
(e.g., bits)
Data Redundancy (cont’d)

- **Compression ratio:** \( C_R = \frac{n_1}{n_2} \)

- **Relative data redundancy:** \( R_D = 1 - \frac{1}{C_R} \)

Example:

If \( C_R = \frac{10}{1} \), then \( R_D = 1 - \frac{1}{10} = 0.9 \) (90% of the data in dataset 1 is redundant)

- if \( n_2 = n_1 \), then \( C_R = 1, R_D = 0 \)
- if \( n_2 \ll n_1 \), then \( C_R \to \infty, R_D \to 1 \)
- if \( n_2 \gg n_1 \), then \( C_R \to 0, R_D \to -\infty \)
Types of Data Redundancy

(1) Coding
(2) Interpixel
(3) Psychovisual

The role of compression is to reduce one or more of these redundancy types.
Coding Redundancy

Data compression can be achieved using an appropriate encoding scheme.

Example: binary encoding

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>4</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>5</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>6</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>7</td>
<td>111</td>
<td></td>
</tr>
</tbody>
</table>
Encoding Schemes

- **Elements of an encoding scheme:**
  - **Code:** a list of symbols (letters, numbers, bits etc.)
  - **Code word:** a sequence of symbols used to represent a piece of information or an event (e.g., gray levels)
  - **Code word length:** number of symbols in each code word

Example: (binary code, symbols: 0,1, length: 3)

- 0: 000
- 1: 001
- 2: 010
- 3: 011
- 4: 100
- 5: 101
- 6: 110
- 7: 111
Definitions

N x M image

$r_k$: k-th gray level

$P(r_k)$: probability of $r_k$

Expected value:

$$E(X) = \sum_x xP(X = x)$$

$l(r_k)$: # of bits for $r_k$

Average # of bits:

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

Total # of bits:

$$NML_{avg}$$
Constant Length Coding

\[ l(r_k) = c \quad \text{which implies that } L_{\text{avg}} = c \]

Example:

<table>
<thead>
<tr>
<th>( r_k )</th>
<th>( p_r(r_k) )</th>
<th>Code</th>
<th>( l_1(r_k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.19</td>
<td>000</td>
<td>3</td>
</tr>
<tr>
<td>1/7</td>
<td>0.25</td>
<td>001</td>
<td>3</td>
</tr>
<tr>
<td>2/7</td>
<td>0.21</td>
<td>010</td>
<td>3</td>
</tr>
<tr>
<td>3/7</td>
<td>0.16</td>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>4/7</td>
<td>0.08</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>5/7</td>
<td>0.06</td>
<td>101</td>
<td>3</td>
</tr>
<tr>
<td>6/7</td>
<td>0.03</td>
<td>110</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0.02</td>
<td>111</td>
<td>3</td>
</tr>
</tbody>
</table>

Assume an image with \( L = 8 \)

Assume \( l(r_k) = 3 \), \( L_{\text{avg}} = \sum_{k=0}^{7} 3 P(r_k) = 3 \sum_{k=0}^{7} P(r_k) = 3 \) bits

Total number of bits: \( 3NM \)
Avoiding Coding Redundancy

- To avoid coding redundancy, codes should be selected according to the probabilities of the events.

- **Variable Length Coding**
  - Assign fewer symbols (bits) to the more probable events (e.g., gray levels for images)
Variable Length Coding

- Consider the probability of the gray levels:

<table>
<thead>
<tr>
<th>$r_k$</th>
<th>$p_r(r_k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_0 = 0$</td>
<td>0.25</td>
</tr>
<tr>
<td>$r_1 = 1/7$</td>
<td>0.125</td>
</tr>
<tr>
<td>$r_2 = 2/7$</td>
<td>0.125</td>
</tr>
<tr>
<td>$r_3 = 3/7$</td>
<td>0.125</td>
</tr>
<tr>
<td>$r_4 = 4/7$</td>
<td>0.125</td>
</tr>
<tr>
<td>$r_5 = 5/7$</td>
<td>0.125</td>
</tr>
<tr>
<td>$r_6 = 6/7$</td>
<td>0.125</td>
</tr>
<tr>
<td>$r_7 = 1$</td>
<td>0.25</td>
</tr>
</tbody>
</table>

The entropy is 2.7 bits (about 10%).
Interpixel redundancy

- Interpixel redundancy implies that any pixel value can be reasonably predicted by its neighbors (i.e., correlated).

\[ \text{correlation: } f(x) \circ g(x) = \int_{-\infty}^{\infty} f^*(a)g(x + a)da \]

\[ \text{autocorrelation: } g(x) = f(x) \]
Interpixel redundancy (cont’d)

- To reduce interpixel redundancy, the data must be transformed in another format (i.e., through a transformation)
  - e.g., thresholding, or differences between adjacent pixels, or DFT

Example:

- original

- thresholded

Run-length encoding:

\[(1,63) \ (0,87) \ (1,37) \ (0,5) \ (1,4) \ (0,56) \ (1,62) \ (0,210)\]

Using 11 bits/pair: \((1+10)\)

88 bits are required (compared to 1024 !!)
To reduce interpixel redundancy, the data must be transformed in another format (i.e., through a transformation) such as thresholding, or differences between adjacent pixels, or DFT.

Example:

Original

Thresholded

Run-length encoding:

\[(1,63) (0,87) (1,37) (0,5) (1,4) (0,556) (1,62) (0,210)\]

Using 11 bits/pair:

\[(1+10)\]

88 bits are required (compared to 1024!!)
Psychovisual redundancy

- Takes into advantage the peculiarities of the human visual system.
- The eye does not respond with equal sensitivity to all visual information.
- Humans search for important features (e.g., edges, texture, etc.) and do not perform quantitative analysis of every pixel in the image.
Psychovisual redundancy (cont’d)
Example: Quantization

256 gray levels

16 gray levels

16 gray levels
improved gray-scale quantization

8/4
= 2:1

i.e., add to each pixel a pseudo-random number prior to quantization
How do we measure information?

- What is the information content of a message/image?

- What is the minimum amount of data that is sufficient to describe completely an image without loss of information?
Modeling the Information Generation Process

- Assume that information generation process is a probabilistic process.

- A random event $E$ which occurs with probability $P(E)$ contains:

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

Note that when $P(E) = 1$, then $I(E) = 0$: no information!
How much information does a pixel contain?

- Suppose that the gray level value of pixels is generated by a random variable, then $r_k$ contains

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

units of information
Average information of an image

- **Entropy**: the average information content of an image

\[ E = \sum_{k=0}^{L-1} I(r_k) \Pr(r_k) \]

using

\[ I(r_k) = - \log(P(r_k)) \]

we have:

\[ H = - \sum_{k=0}^{L-1} P(r_k) \log(P(r_k)) \]

**Assumption**: statistically independent random events
Modeling the Information Generation Process (cont’d)

- **Redundancy**

\[ R = L_{avg} - H \]

where:

\[ L_{avg} = E(l(r_k)) = \sum_{k=0}^{L-1} l(r_k)P(r_k) \]

note that if \( L_{avg} = H \), then \( R = 0 \) - no redundancy
Entropy Estimation

- Not easy!

```
image

<table>
<thead>
<tr>
<th></th>
<th>21</th>
<th>21</th>
<th>21</th>
<th>95</th>
<th>169</th>
<th>243</th>
<th>243</th>
<th>243</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>95</td>
<td>169</td>
<td>243</td>
<td>243</td>
<td>243</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>95</td>
<td>169</td>
<td>243</td>
<td>243</td>
<td>243</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>95</td>
<td>169</td>
<td>243</td>
<td>243</td>
<td>243</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gray Level</th>
<th>Count</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>12</td>
<td>3/8</td>
</tr>
<tr>
<td>95</td>
<td>4</td>
<td>1/8</td>
</tr>
<tr>
<td>169</td>
<td>4</td>
<td>1/8</td>
</tr>
<tr>
<td>243</td>
<td>12</td>
<td>3/8</td>
</tr>
</tbody>
</table>
```
Entropy Estimation

- **First order** estimate of $H$:

$$H = - \sum_{k=0}^{3} P(r_k) \log(P(r_k)) = 1.81 \text{ bits/pixel}$$

Total bits: $4 \times 8 \times 1.81 = 58 \text{ bits}$
Estimating Entropy (cont’d)

**Second order** estimate of $H$:

*Use relative frequencies of **pixel blocks**:

<table>
<thead>
<tr>
<th>Gray Level Pair</th>
<th>Count</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(21, 21)</td>
<td>8</td>
<td>1/4</td>
</tr>
<tr>
<td>(21, 95)</td>
<td>4</td>
<td>1/8</td>
</tr>
<tr>
<td>(95, 169)</td>
<td>4</td>
<td>1/8</td>
</tr>
<tr>
<td>(169, 243)</td>
<td>4</td>
<td>1/8</td>
</tr>
<tr>
<td>(243, 243)</td>
<td>8</td>
<td>1/4</td>
</tr>
<tr>
<td>(243, 21)</td>
<td>4</td>
<td>1/8</td>
</tr>
</tbody>
</table>

$H = 2.5/2 = 1.25$ bits/pixel
Estimating Entropy (cont’d)

- Comments on first and second order entropy estimates:

  - The first-order estimate gives only a lower-bound on the compression that can be achieved.

  - Differences between higher-order estimates of entropy and the first-order estimate indicate the presence of interpixel redundancies.
Question

❖ How do we deal with interpixel redundancy?

Apply a transformation!

Https://fb.com/tailieudientucntt
Estimating Entropy (cont’d)

- E.g., consider difference image:

<table>
<thead>
<tr>
<th>21 21 21 95 169 243 243 243</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 21 21 95 169 243 243 243</td>
</tr>
<tr>
<td>21 21 21 95 169 243 243 243</td>
</tr>
<tr>
<td>21 21 21 95 169 243 243 243</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>21 0 0 74 74 74 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 0 0 74 74 74 0 0</td>
</tr>
<tr>
<td>21 0 0 74 74 74 0 0</td>
</tr>
<tr>
<td>21 0 0 74 74 74 0 0</td>
</tr>
</tbody>
</table>

---

**Gray Level or Difference** | **Count** | **Probability**
---|---|---
0 | 12 | 1/2
21 | 4 | 1/8
74 | 12 | 3/8
Estimating Entropy (cont’d)

- Entropy of difference image:

\[ H = - \sum_{k=0}^{2} P(r_k) \log(P(r_k)) = 1.41 \text{ bits/pixel} \]

- Better than before (i.e., H=1.81 for original image), however, a better transformation could be found:

1.41 bits/pixel > 1.25 bits/pixel (from 2nd order estimate of \( H \))
Image Compression Model

Encoder

Source encoder

Channel encoder

Channel

Decoder

Source decoder

\( f(x,y) \)

\( \hat{f}(x,y) \)

**noise tolerant representation**

(additional bits are included to guarantee detection and correction of errors due to transmission over the channel - **Hamming coding**)

**compression (no redundancies)**
Image Compression Model (cont’d)

- **Mapper**: transforms the input data into a format that facilitates reduction of interpixel redundancies.
Image Compression Model (cont’d)

Quantizer: reduces the accuracy of the mapper’s output in accordance with some pre-established fidelity criteria.
Symbol encoder: assigns the shortest code to the most frequently occurring output values.
The inverse operations are performed.

But … quantization is **irreversible** in general.
Fidelity Criteria

How close is $f(x, y)$ to $\hat{f}(x, y)$?

Criteria

- **Subjective**: based on human observers
- **Objective**: mathematically defined criteria
Subjective Fidelity Criteria

<table>
<thead>
<tr>
<th>Value</th>
<th>Rating</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Excellent</td>
<td>An image of extremely high quality, as good as you could desire.</td>
</tr>
<tr>
<td>2</td>
<td>Fine</td>
<td>An image of high quality, providing enjoyable viewing. Interference is not objectionable.</td>
</tr>
<tr>
<td>3</td>
<td>Passable</td>
<td>An image of acceptable quality. Interference is not objectionable.</td>
</tr>
<tr>
<td>4</td>
<td>Marginal</td>
<td>An image of poor quality; you wish you could improve it. Interference is somewhat objectionable.</td>
</tr>
<tr>
<td>5</td>
<td>Inferior</td>
<td>A very poor image, but you could watch it. Objectionable interference is definitely present.</td>
</tr>
<tr>
<td>6</td>
<td>Unusable</td>
<td>An image so bad that you could not watch it.</td>
</tr>
</tbody>
</table>
Objective Fidelity Criteria

- **Root mean square error (RMS)**

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

- **Mean-square signal-to-noise ratio (SNR)**

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

Cuuduongthancong.com
https://fb.com/tailieudientucntt
Example

original

RMS=5.17  RMS=15.67  RMS=14.17
Lossless Compression

- Huffman, Golomb, Arithmetic ➔ coding redundancy

- LZW, Run-length, Symbol-based, Bit-plane ➔ interpixel redundancy
Huffman Coding (i.e., removes coding redundancy)

- It is a variable-length coding technique.
- It creates the optimal code for a set of source symbols.
- Assumption: symbols are encoded one at a time!
Huffman Coding (cont’d)

**Optimal code:** minimizes the number of code symbols per source symbol.

- **Forward Pass**
  1. Sort probabilities per symbol
  2. Combine the lowest two probabilities
  3. Repeat Step 2 until only two probabilities remain.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Source reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.06</td>
<td>0.1</td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.04</td>
<td></td>
</tr>
</tbody>
</table>
Huffman Coding (cont’d)

- **Backward Pass**

Assign code symbols going backwards

<table>
<thead>
<tr>
<th>Sym.</th>
<th>Prob.</th>
<th>Code</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>0.4</td>
<td>1</td>
<td>0.4 1</td>
<td>0.4 1</td>
<td>0.4 1</td>
<td>0.6 0</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.3</td>
<td>00</td>
<td>0.3 00</td>
<td>0.3 00</td>
<td>0.3 00</td>
<td>0.4 1</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.1</td>
<td>011</td>
<td>0.1 011</td>
<td>0.2 010</td>
<td>0.3 01</td>
<td>0.4 1</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.1</td>
<td>0100</td>
<td>0.1 0100</td>
<td>0.1 011</td>
<td>0.4 1</td>
<td></td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.06</td>
<td>01010</td>
<td>0.1 0101</td>
<td>0.4 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.04</td>
<td>01011</td>
<td>0.4 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Huffman Coding (cont’d)

- $L_{\text{avg}}$ using Huffman coding:

\[ L_{\text{avg}} = E(l(a_k)) = \sum_{k=1}^{6} l(a_k)P(a_k) = 3 \times 0.1 + 1 \times 0.4 + 5 \times 0.06 + 4 \times 0.1 + 5 \times 0.04 + 2 \times 0.3 = 2.2 \text{ bits/symbol} \]

- $L_{\text{avg}}$ assuming binary codes:

6 symbols, we need a 3-bit code

\[ (a_1: 000, a_2: 001, a_3: 010, a_4: 011, a_5: 100, a_6: 101) \]

\[ L_{\text{avg}} = \sum_{k=1}^{6} l(a_k)P(a_k) = \sum_{k=1}^{6} 3P(a_k) = 3 \times \sum_{k=1}^{6} P(a_k) = 3 \text{ bits/symbol} \]
Huffman Coding (cont’d)

Comments

- After the code has been created, *coding/decoding* can be implemented using a look-up table.
- Decoding can be done in an unambiguous way!!
Arithmetic (or Range) Coding (i.e., removes coding redundancy)

- No assumption on encoding symbols one at a time.
  - **No one-to-one correspondence between source and code words.**

- Slower than Huffman coding but typically achieves better compression.

- A sequence of source symbols is assigned a single arithmetic code word which corresponds to a sub-interval in $[0,1]$
Arithmetic Coding (cont’d)

- As the number of symbols in the message increases, the interval used to represent it becomes smaller.
  - Each symbol reduces the size of the interval according to its probability.

- Smaller intervals require more information units (i.e., bits) to be represented.
Arithmetic Coding (cont’d)

Encode message:  \(a_1 a_2 a_3 a_3 a_4\)

1) Assume message occupies \([0, 1)\)

\[
\begin{array}{cc}
0 & 1
\end{array}
\]

2) Subdivide \([0, 1)\) based on the probabilities of \(\alpha_i\)

3) Update interval by processing source symbols

<table>
<thead>
<tr>
<th>Source Symbol</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>0.2</td>
</tr>
<tr>
<td>(a_2)</td>
<td>0.2</td>
</tr>
<tr>
<td>(a_3)</td>
<td>0.4</td>
</tr>
<tr>
<td>(a_4)</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial Subinterval</th>
</tr>
</thead>
<tbody>
<tr>
<td>([0.0, 0.2))</td>
</tr>
<tr>
<td>([0.2, 0.4))</td>
</tr>
<tr>
<td>([0.4, 0.8))</td>
</tr>
<tr>
<td>([0.8, 1.0))</td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>Source Symbol</th>
<th>Probability</th>
<th>Initial Subinterval</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>0.2</td>
<td>[0.0, 0.2)</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.2</td>
<td>[0.2, 0.4)</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.4</td>
<td>[0.4, 0.8)</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.2</td>
<td>[0.8, 1.0)</td>
</tr>
</tbody>
</table>

\[
[0.06752, 0.0688) \quad \text{or,} \quad 0.068
\]
Example

- The message $a_1 \ a_2 \ a_3 \ a_4$ is encoded using 3 decimal digits or \textbf{0.6} decimal digits per source symbol.

- The entropy of this message is:

\[-(3 \times 0.2 \log_{10}(0.2) + 0.4 \log_{10}(0.4)) = 0.5786 \text{ digits/symbol}\]

\textbf{Note:} Finite precision arithmetic might cause problems due to truncations!
Arithmetic Coding (cont’d)

Decode 0.572

\[ a_3 a_3 a_1 a_2 a_4 \]
Arithmetic Coding (cont’d)

❖ **Encode**

❖ **Low** = 0
❖ **High** = 1
❖ **Loop. For all the symbols.**

✓ Range = high - low
✓ High = low + range * high_range of the symbol being coded
✓ Low = low + range * low_range of the symbol being coded

❖ **Where:**

❖ **Range,** keeps track of where the next range should be.
❖ **High and low,** specify the output number.
Arithmetic Coding (cont’d)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Range</th>
<th>Low value</th>
<th>High value</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
<td>0.5</td>
<td>0.75</td>
</tr>
<tr>
<td>a</td>
<td>0.25</td>
<td>0.5</td>
<td>0.625</td>
</tr>
<tr>
<td>c</td>
<td>0.125</td>
<td>0.59375</td>
<td>0.625</td>
</tr>
<tr>
<td>a</td>
<td>0.03125</td>
<td>0.59375</td>
<td>0.609375</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>[0.0, 0.5)</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>[0.5, 0.75)</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>[0.75, 1.0)</td>
</tr>
</tbody>
</table>
Arithmetic Coding (cont’d)

❖ Decode

❖ Loop. For all the symbols.

✓ Range = high_range of the symbol - low_range of the symbol
✓ Number = number - low_range of the symbol
✓ Number = number / range
Arithmetic Coding (cont’d)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Range</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0.25</td>
<td>0.59375</td>
</tr>
<tr>
<td>a</td>
<td>0.5</td>
<td>0.375</td>
</tr>
<tr>
<td>c</td>
<td>0.25</td>
<td>0.75</td>
</tr>
<tr>
<td>a</td>
<td>0.5</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>[0.0, 0.5)</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>[0.5, 0.75)</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>[0.75, 1.0)</td>
</tr>
</tbody>
</table>
LZW Coding
(i.e., removes inter-pixel redundancy)

- Requires **no priori knowledge** of probability distribution of pixels
- Assigns **fixed length** code words to **variable length** sequences
- Patented Algorithm US 4,558,302
- Included in GIF and TIFF and PDF file formats
LZW Coding

- A **codebook** or a **dictionary** has to be constructed.
  - **Single pixel values and blocks of pixel values**

- For an 8-bit image, the first 256 entries are assigned to the gray levels 0,1,2,...,255.

- As the encoder examines image pixels, gray level sequences (i.e., pixel combinations) that are not in the dictionary are assigned to a new entry.
Example

Consider the following 4 x 4 8 bit image

\[
\begin{array}{cccc}
39 & 39 & 126 & 126 \\
39 & 39 & 126 & 126 \\
39 & 39 & 126 & 126 \\
39 & 39 & 126 & 126 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Dictionary Location</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>256</td>
<td>-</td>
</tr>
<tr>
<td>511</td>
<td>-</td>
</tr>
</tbody>
</table>

Initial Dictionary
Example

- Is 39 in the dictionary …….. Yes
- What about 39-39 …………… No
- Then add 39-39 in entry 256

<table>
<thead>
<tr>
<th>Dictionary Location</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>256</td>
<td>39-39</td>
</tr>
<tr>
<td>511</td>
<td></td>
</tr>
</tbody>
</table>
### Example

**concatenated sequence (CS)**

<table>
<thead>
<tr>
<th>(CR)</th>
<th>(P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>126</td>
<td>126</td>
</tr>
<tr>
<td>126</td>
<td>126</td>
</tr>
</tbody>
</table>

If CS is found:
1. No Output
2. CR = CS

If CS is not found:
1. Output D(CR)
2. Add CS to D
3. CR = P

<table>
<thead>
<tr>
<th>Currently Recognized Sequence</th>
<th>Pixel Being Processed</th>
<th>Encoded Output</th>
<th>Dictionary Location (Code Word)</th>
<th>Dictionary Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>39</td>
<td>39</td>
<td>256</td>
<td>39-39</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
<td>39</td>
<td>257</td>
<td>39-126</td>
</tr>
<tr>
<td>126</td>
<td>126</td>
<td>126</td>
<td>258</td>
<td>126-126</td>
</tr>
<tr>
<td>126</td>
<td>39</td>
<td>126</td>
<td>259</td>
<td>126-39</td>
</tr>
<tr>
<td>126</td>
<td>39</td>
<td>258</td>
<td>261</td>
<td>126-126-39</td>
</tr>
<tr>
<td>126-126</td>
<td>39</td>
<td>259</td>
<td>263</td>
<td>126-39-39</td>
</tr>
<tr>
<td>126</td>
<td>39</td>
<td>257</td>
<td>264</td>
<td>39-126-126</td>
</tr>
<tr>
<td>126</td>
<td>126</td>
<td>126</td>
<td>126</td>
<td>39-126-126-126</td>
</tr>
</tbody>
</table>
Decoding LZW

- The dictionary which was used for encoding need not be sent with the image.
  
  cuu duong than cong . com

- A separate dictionary is built by the decoder, on the “fly”, as it reads the received code words.

  cuu duong than cong . com
Run-length coding (RLC) (i.e., removes interpixel redundancy)

- Used to reduce the size of a repeating string of characters (i.e., runs)

```
  a a a b b b b b b b c c  \rightarrow  (a, 3) (b, 6) (c, 2)
```

- Encodes a run of symbols into two bytes, a count and a symbol.
- Can compress any type of data but cannot achieve high compression ratios compared to other compression methods.
Run-length coding
(i.e., removes interpixel redundancy)

❖ Code each contiguous group of 0’s and 1’s, encountered in a left to right scan of a row, by its **length**.

```
1 1 1 1 1 0 0 0 0 0 0 1  
```

→ (1,5) (0, 6)

(1, 1)
Bit-plane coding
(i.e., removes inter pixel redundancy)

- An effective technique to reduce inter pixel redundancy is to process each bit plane individually.

- The image is decomposed into a series of binary images.

- Each binary image is compressed using one of well-known binary compression techniques.
  - *e.g.*, Huffman, Run-length, etc.
Combining Huffman Coding with Run-length Coding

- Once a message has been encoded using Huffman coding, additional compression can be achieved by encoding the lengths of the runs using variable-length coding!

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

e.g., (0,1)(1,1)(0,1)(1,1)(0,2)(1,4)(0,2)
Lossy Compression

- Transform the image into a domain where compression can be performed more efficiently.
- Note that the transformation itself does not compress the image!

\[ \sim (N/n)^2 \text{ subimages} \]
Lossy Compression (cont’d)

Example: Fourier Transform

\[ f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u, v)e^{\frac{j2\pi(ux+vy)}{N}}, \quad x, y = 0, 1, ..., N-1 \]

\[ \hat{f}(x, y) = \frac{1}{N} \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} F(u, v)e^{\frac{j2\pi(ux+vy)}{N}}, \quad x, y = 0, 1, ..., N-1 \]

The magnitude of the FT decreases, as \( u, v \) increase!

\[ \sum_{x,y} (\hat{f}(x, y) - f(x, y))^2 \text{ is very small!!} \]
Transform Selection

\[ f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} T(u, v)h(x, y, u, v) \]

- \( T(u,v) \) can be computed using various transformations, for example:
  - DFT
  - DCT (Discrete Cosine Transform)
  - KLT (Karhunen-Loeve Transformation)
DCT

forward

\[ C(u, v) = \alpha(u) \alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos\left(\frac{(2x + 1)u\pi}{2N}\right) \cos\left(\frac{(2y + 1)v\pi}{2N}\right), \]

inverse

\[ f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \alpha(u) \alpha(v) C(u, v) \cos\left(\frac{(2x + 1)u\pi}{2N}\right) \cos\left(\frac{(2y + 1)v\pi}{2N}\right), \]

\[ \alpha(u) = \begin{cases} \sqrt{1/N} & \text{if } u=0 \\ \sqrt{2/N} & \text{if } u>0 \end{cases} \quad \alpha(v) = \begin{cases} \sqrt{1/N} & \text{if } v=0 \\ \sqrt{2/N} & \text{if } v>0 \end{cases} \]
DCT (cont’d)

- Basis set of functions for a 4x4 image (i.e., cosines of different frequencies).
DCT (cont’d)

- DFT
- WHT
- DCT

8 x 8 subimages
64 coefficients per subimage
50% of the coefficients truncated

RMS error:
- 2.32
- 1.78
- 1.13
DCT (cont’d)

- DCT minimizes "blocking artifacts" (i.e., boundaries between subimages do not become very visible).

**DFT**

i.e., n-point periodicity gives rise to discontinuities!

**DCT**

i.e., 2n-point periodicity prevents discontinuities!
DCT (cont’d)

- Subimage size selection

![Subimage size selection comparison](image.png)
JPEG Compression

- JPEG uses DCT for handling interpixel redundancy.

- Modes of operation:
  1. Sequential DCT-based encoding
  2. Progressive DCT-based encoding
  3. Lossless encoding
  4. Hierarchical encoding
JPEG Compression
(Sequential DCT-based encoding)
JPEG Steps

1. Divide the image into 8x8 subimages;
   For each subimage do:
2. Shift the gray-levels in the range [-128, 127]
3. Apply DCT (64 coefficients will be obtained: 1 DC coefficient $F(0,0)$, 63 AC coefficients $F(u,v)$).
4. Quantize the coefficients (i.e., reduce the amplitude of coefficients that do not contribute a lot).

$$C_q(u, v) = \text{Round} \left[ \frac{C(u, v)}{Q(u, v)} \right]$$

Quantization Table

https://fb.com/tailieudientucntt
5. Order the coefficients using zig-zag ordering
   - Place non-zero coefficients first
   - Create long runs of zeros (i.e., good for run-length encoding)
   - See next slide


   **DC** coefficients are encoded using predictive encoding
   All coefficients are converted to a binary sequence:
   6.1 Form intermediate symbol sequence
   6.2 Apply Huffman (or arithmetic) coding (i.e., entropy coding)
JPEG Steps (cont’d)

✓ AC coefficients are arranged into a zig-zag sequence:

```
0  1  5  6  14  15  27  28
2  4  7 13 16 26 29 42
3  8 12 17 25 30 41 43
9 11 18 24 31 40 44 53
10 19 23 32 39 45 52 54
20 22 33 38 46 51 55 60
21 34 37 47 50 56 59 61
35 36 49 48 57 58 62 63
```

Horizontal frequency

Vertical frequency
Shifting and DCT

<table>
<thead>
<tr>
<th>(a) Original 8×8 block</th>
<th>(b) Shifted block</th>
<th>(c) Block after FDCT Eqn. (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>140 144 147 140 140 155 179 175</td>
<td>12 16 19 12 11 27 51 47</td>
<td>185 -17 14 -8 23 -9 -13 -18</td>
</tr>
<tr>
<td>144 152 140 147 140 148 167 179</td>
<td>16 24 12 19 12 20 39 51</td>
<td>20 -34 26 -9 -10 10 13 6</td>
</tr>
<tr>
<td>152 155 156 167 163 162 152 172</td>
<td>24 27 8 39 35 34 24 44</td>
<td>-10 -23 -1 6 -18 3 -20 0</td>
</tr>
<tr>
<td>168 145 156 160 152 155 136 160</td>
<td>40 17 28 32 24 27 8 32</td>
<td>-8 -5 14 -14 -8 -2 -3 8</td>
</tr>
<tr>
<td>162 148 156 148 140 136 147 162</td>
<td>34 20 28 20 12 8 19 34</td>
<td>-3 9 7 1 -11 17 18 15</td>
</tr>
<tr>
<td>147 167 140 155 155 140 136 162</td>
<td>19 39 12 27 27 12 8 34</td>
<td>3 -2 -18 8 8 -3 0 -6</td>
</tr>
<tr>
<td>136 156 123 167 162 144 140 147</td>
<td>8 28 -5 39 34 16 12 19</td>
<td>8 0 -2 3 -1 -7 -1 -1</td>
</tr>
<tr>
<td>148 155 136 155 152 147 147 136</td>
<td>20 27 8 27 24 19 19 8</td>
<td>0 -7 -2 1 1 4 -6 0</td>
</tr>
</tbody>
</table>

(non-centered spectrum)
Quantization

- Quantization Table Example

\[
\begin{align*}
\text{for } & i=0 \text{ to } n; \\
\text{for } & j=0 \text{ to } n; \\
Q[i,j] & = 1 + (1+i+j)\times\text{quality}; \\
\text{end } & j; \\
\text{end } & i; \\
1 & \leq \text{quality} \leq 25
\end{align*}
\]

(best - low compression) (worst - high compression)
Quantization (cont’d)

(c) Block after FDCT
Eqn. (5)

(d) Quantization table
(quality = 2)

(e) Block after quantization
Eqn. (6)
Zig-Zag Ordering (cont’d)

(e) Block after quantization
Eqn. (6)

(f) Zig-zag sequence

61, -3, 4, -1, -4, 2, 0, 2, -2, 0, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, -1, 0, 0, -1, 0, 0,
0, 0, -1, 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.
Intermediate Coding (cont’d)

(f) Zig-zag sequence
61,-3,4,-1,-4,2,0,2,-2,0,0,0,0,0,2,0,0,0,1,0,0,0,0,0,0,-1,0,0,-1,0,0,
0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

(g) Intermediate symbol sequence
(6)(61),(0,2)(-3),(0,3)(4),(0,1)(-1),(0,3)(-4),(0,2)(2),(1,2)(2),(0,2)(-2),
(5,2)(2),(3,1)(1),(6,1)(-1),(2,1)(-1),(4,1)(-1),(7,1)(-1),(0,0)

symbol_1, symbol_2

DC  (6)  (61)
AC  (0,2)  (-3)

End of Block
DC/AC Symbol Encoding

- **DC encoding**
  - symbol_1 (size)
  - symbol_2 (amplitude)

 predictive coding:

- **AC encoding**
  - symbol1 (RUN-LENGTH, SIZE) (AMPLITUDE)
  - symbol2

| 0 0 0 0 0 0 476 | (6,9)(476) |

If RUN-LENGTH > 15, then symbol (15,0) means RUN-LENGTH=16

[-2048, 2047] 11 bits

[-1023, 1024] 10 bits
Entropy Encoding (cont’d)

(a) Intermediate symbol sequence
(6)(61),(0,2)(-3),(0,3)(4),(0,1)(-1),(0,3)(-4),(0,2)(2),(1,2)(2),(0,2)(-2),
(5,2)(2),(3,1)(1),(6,1)(-1),(2,1)(-1),(4,1)(-1),(7,1)(-1),(0,0)

(e) Encoded bit sequence (total 98 bits)
1110111101001001000001000110110011011100101111111011
1101110101111101101110001110110111101001010

End of Block

See Table 8.17-8.19, page 500, 501, 501
Entropy Encoding

**Symbol 1**
(Variable Length Code (VLC))

<table>
<thead>
<tr>
<th>(Runlength, size)</th>
<th>Code word</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0) EOB</td>
<td>11010</td>
</tr>
<tr>
<td>(0,1)</td>
<td>00</td>
</tr>
<tr>
<td>(0,2)</td>
<td>01</td>
</tr>
<tr>
<td>(0,3)</td>
<td>100</td>
</tr>
<tr>
<td>(1,2)</td>
<td>11011</td>
</tr>
<tr>
<td>(2,1)</td>
<td>11100</td>
</tr>
<tr>
<td>(3,1)</td>
<td>111100</td>
</tr>
<tr>
<td>(4,1)</td>
<td>111011</td>
</tr>
<tr>
<td>(5,2)</td>
<td>11111011</td>
</tr>
<tr>
<td>(6,1)</td>
<td>1111011</td>
</tr>
<tr>
<td>(7,1)</td>
<td>11111010</td>
</tr>
</tbody>
</table>

**Symbol 2**
(Variable Length Integer (VLI))

<table>
<thead>
<tr>
<th>Size</th>
<th>Amplitude range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-1, 1)</td>
</tr>
<tr>
<td>2</td>
<td>(-3, -2) (2,3)</td>
</tr>
<tr>
<td>3</td>
<td>(-7, -4) (4, 7)</td>
</tr>
<tr>
<td>4</td>
<td>(-15, -8) (8,15)</td>
</tr>
<tr>
<td>5</td>
<td>(-31, -16) (16,31)</td>
</tr>
<tr>
<td>6</td>
<td>(-63, -32) (32,63)</td>
</tr>
<tr>
<td>7</td>
<td>(-127, -64) (64,127)</td>
</tr>
<tr>
<td>8</td>
<td>(-255, -128) (128,255)</td>
</tr>
<tr>
<td>9</td>
<td>(-511, -256) (256,511)</td>
</tr>
<tr>
<td>10</td>
<td>(-1023, -512) (512,1023)</td>
</tr>
</tbody>
</table>

See Table 8.17-8.19, page 500, 501, 501
JPEG Examples

- **10 (8k bytes)**
  - worst quality,
  - highest compression

- **50 (21k bytes)**

- **90 (58k bytes)**
  - best quality,
  - lowest compression
### Results

**Table 6. Results of JPEG Compression for Grayscale Image ‘Lisa’ (320 × 240 pixels)**

<table>
<thead>
<tr>
<th>Quality factors</th>
<th>Original number of bits</th>
<th>Compressed number of bits</th>
<th>Compression ratio (Cr)</th>
<th>Bits/pixel (Nb)</th>
<th>RMS error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>512,000</td>
<td>48,021</td>
<td>10.66</td>
<td>0.75</td>
<td>2.25</td>
</tr>
<tr>
<td>2</td>
<td>512,000</td>
<td>30,490</td>
<td>16.79</td>
<td>0.48</td>
<td>2.75</td>
</tr>
<tr>
<td>4</td>
<td>512,000</td>
<td>20,264</td>
<td>25.27</td>
<td>0.32</td>
<td>3.43</td>
</tr>
<tr>
<td>8</td>
<td>512,000</td>
<td>14,162</td>
<td>36.14</td>
<td>0.22</td>
<td>4.24</td>
</tr>
<tr>
<td>15</td>
<td>512,000</td>
<td>10,479</td>
<td>48.85</td>
<td>0.16</td>
<td>5.36</td>
</tr>
<tr>
<td>25</td>
<td>512,000</td>
<td>9,034</td>
<td>56.64</td>
<td>0.14</td>
<td>6.40</td>
</tr>
</tbody>
</table>
Progressive JPEG

- The image is encoded in multiple scans, in order to produce a quick, rough decoded image when transmission time is long.
Progressive JPEG (cont’d)

- Each scan, codes a subset of DCT coefficients.

- Let’s look at three methods:
  
  (1) Progressive spectral selection algorithm
  (2) Progressive successive approximation algorithm
  (3) Combined progressive algorithm
Progressive JPEG (cont’d)

(1) Progressive spectral selection algorithm

- Group DCT coefficients into several spectral bands
- Send low-frequency DCT coefficients first
- Send higher-frequency DCT coefficients next

Band 1: DC coefficient only
Band 2: $AC_1$ and $AC_2$ coefficients
Band 3: $AC_3$, $AC_4$, $AC_5$, $AC_6$, coefficients
Band 4: $AC_7$,...,$AC_{63}$, coefficients
Progressive JPEG (cont’d)

(2) Progressive successive approximation algorithm

- All DCT coefficients are sent first with lower precision
- Refine them in later scans

Band 1: All DCT coefficients (divided by four)
Band 2: All DCT coefficients (divided by two)
Band 3: All DCT coefficients (full resolution)
Progressive JPEG (cont’d)

(3) Combined progressive algorithm

- Combines spectral selection and successive approximation.
Results using spectral selection

<table>
<thead>
<tr>
<th>Spectral selection</th>
<th>Scan 1</th>
<th>DC, AC1, AC2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scan 2</td>
<td>AC3–AC9</td>
</tr>
<tr>
<td></td>
<td>Scan 3</td>
<td>AC10–AC35</td>
</tr>
<tr>
<td></td>
<td>Scan 4</td>
<td>AC 36–AC 63</td>
</tr>
</tbody>
</table>

Table 8. Progressive spectral selection JPEG. (Image ‘Cheetah’: 320 × 240 pixels → 512,000 bits)

<table>
<thead>
<tr>
<th>Scan number</th>
<th>Bits transmitted</th>
<th>Compression ratio</th>
<th>Bits/pixel</th>
<th>RMS error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29,005</td>
<td>17.65</td>
<td>0.45</td>
<td>19.97</td>
</tr>
<tr>
<td>2</td>
<td>37,237</td>
<td>7.73</td>
<td>1.04</td>
<td>13.67</td>
</tr>
<tr>
<td>3</td>
<td>71,259</td>
<td>3.72</td>
<td>2.15</td>
<td>7.90</td>
</tr>
<tr>
<td>4</td>
<td>32,489</td>
<td>3.01</td>
<td>2.66</td>
<td>4.59</td>
</tr>
<tr>
<td>Sequential JPEG</td>
<td>172,117</td>
<td>2.97</td>
<td>2.69</td>
<td>4.59</td>
</tr>
</tbody>
</table>
Results using successive approximation

**Table 9. Progressive successive approximation JPEG. (Image ‘Cheetah’: 320 x 240 pixels -> 512,000 bits)**

<table>
<thead>
<tr>
<th>Scan number</th>
<th>Bits transmitted</th>
<th>Compression ratio</th>
<th>Bits/pixel</th>
<th>RMS error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>26,215</td>
<td>19.53</td>
<td>0.41</td>
<td>22.48</td>
</tr>
<tr>
<td>2</td>
<td>34,506</td>
<td>8.43</td>
<td>0.95</td>
<td>12.75</td>
</tr>
<tr>
<td>3</td>
<td>63,792</td>
<td>4.11</td>
<td>1.95</td>
<td>7.56</td>
</tr>
<tr>
<td>4</td>
<td>95,267</td>
<td>2.33</td>
<td>2.43</td>
<td>4.59</td>
</tr>
<tr>
<td>Sequential JPEG</td>
<td>172,117</td>
<td>2.97</td>
<td>2.69</td>
<td>4.59</td>
</tr>
</tbody>
</table>
Example using successive approximation

after 0.9s

after 1.6s

after 3.6s

after 7.0s
Lossless JPEG

Use a predictive algorithm instead of DCT-based
Fingerprint Compression

- An image coding standard for digitized fingerprints, developed and maintained by:
  - FBI
  - Los Alamos National Lab (LANL)
  - National Institute for Standards and Technology (NIST).

- The standard employs a discrete wavelet transform-based algorithm (Wavelet/Scalar Quantization or WSQ).
Memory Requirements

- FBI is digitizing fingerprints at 500 dots per inch with 8 bits of grayscale resolution.
- A single fingerprint card turns into about 10 MB of data!

A sample fingerprint image
768 x 768 pixels = 589,824 bytes
Preserving Fingerprint Details

The "white" spots in the middle of the black ridges are *sweat pores*. They’re admissible points of identification in court, as are the little black flesh “islands” in the grooves between the ridges.

These details are just a couple pixels wide!
What compression scheme should be used?

- Better use a lossless method to preserve every pixel perfectly.

Unfortunately, in practice lossless methods haven’t done better than 2:1 on fingerprints!

- Does JPEG work well for fingerprint compression?
Results using JPEG compression

file size 45853 bytes
compression ratio: 12.9

The fine details are pretty much history, and the whole image has this artificial “blocky” pattern superimposed on it.

The **blocking artifacts** affect the performance of manual or automated systems!

[Image of fingerprint with JPEG compression artifacts]
Results using WSQ compression

file size 45621 bytes
compression ratio: 12.9

The fine details are preserved better
than they are with JPEG.

NO blocking artifacts!
WSQ Algorithm

![Diagram of WSQ Algorithm]

**WSQ Encoder:**
- Source Image
- DWT
- Scalar Quant.
- Huffman Coder
  - 1001 1101
  - Compressed Data

**WSQ Decoder:**
- 1001 1101
  - Compressed Data
- Huffman Decoder
- Scalar Dequant.
- IDWT
  - Reconstructed Image
Varying compression ratio

- FBI’s target bit rate is around 0.75 bits per pixel (bpp)
  - i.e., corresponds to a target compression ratio of 10.7 (assuming 8-bit images)

- This target bit rate is set via a "knob" on the WSQ algorithm.
  - i.e., similar to the "quality" parameter in many JPEG implementations.
Varying compression ratio (cont’d)

- In practice, the WSQ algorithm yields a higher compression ratio than the target because of unpredictable amounts of lossless entropy coding gain.
  - i.e., mostly due to variable amounts of blank space in the images.

- Fingerprints coded with WSQ at a target of 0.75 bpp will actually come in around 15:1
Varying compression ratio (cont’d)

Original image 768 x 768 pixels (589824 bytes)
Varying compression ratio (cont’d)
0.9 bpp compression

**WSQ** image, file size 47619 bytes, compression ratio 12.4

**JPEG** image, file size 49658 bytes, compression ratio 11.9
Varying compression ratio (cont’d)
0.75 bpp compression

WSQ image, file size 39270 bytes, compression ratio 15.0

JPEG image, file size 40780 bytes, compression ratio 14.5
Varying compression ratio (cont’d)

0.6 bpp compression

WSQ image, file size 30987 bytes, compression ratio 19.0

JPEG image, file size 30081 bytes, compression ratio 19.6