Digital Image Processing

Image Compression

• Goal:
  – Reducing the amount of data required to represent a digital image.
    • Transmission
    • Archiving
  – Mathematical Definition:
    • Transforming a 2-D pixel array into a statistically uncorrelated data set.
Digital Image Processing

Image Compression

- Two Main Category:
  - Lossless: TIFF (LZW,ZIP)
  - Lossy: JPEG and JPEG2000
Example: Original vs. JPEG2000

\[ C_R = 13.31 \]
\[ R_D = 0.925 \]

Best Compressor: 510,576 bytes

I am 1,085,496 Byte

I am 81,503 Byte
Image Compression

• Fundamentals:
  – Raw image: A set of $n_1$ bits
  – Compressed image: A set of $n_2$ bits.
  
  – Compression ratio: $C_R = \frac{n_1}{n_2}$
  
  – Relative Data Redundancy in A: $R_D = 1 - \frac{1}{C_R}$
  
  – Example: $n_1 = 100$KB and $n_2 = 10$Kb, then $C_0 = 10$, and $R_D = 90$

• Special cases: 1) $n_1 >> n_2 \rightarrow C_R \approx \infty, R_D \approx 1$
  2) $n_1 \approx n_2 \rightarrow C_R \approx 1, R_D \approx 0$
  3) $n_1 << n_2 \rightarrow C_R \approx 0, R_D \approx -\infty$
Three basic data redundancies:

- Coding redundancy
- Spatial/Temporal (Inter-Pixel) redundancy,
- Irrelevant/Psycho-Visual redundancy.

**FIGURE 8.1** Computer generated 256 × 256 × 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.)
Digital Image Processing

Image Compression

• Coding redundancy:
  – Type of coding (# of bits for each gray level)
  – Image histogram:
    • \( r_k \): Represents the gray levels of an image
    • \( p_r(r_k) \): Probability of occurrence of \( r_k \)
      \[
      p_r(r_k) = \frac{n_k}{n} \quad k = 0, 1, 2, ..., L - 1
      \]
    • \( l(r_k) \): Number of bits used to represent each \( r_k \).
    • \( L_{avg} \): Average # of bits required to represent each pixel:
      \[
      L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)
      \]
Digital Image Processing

Image Compression

• Example (1): Variable Length Coding

<table>
<thead>
<tr>
<th>$r_k$</th>
<th>$p_i(r_k)$</th>
<th>Code 1</th>
<th>$l_1(r_k)$</th>
<th>Code 2</th>
<th>$l_2(r_k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{87} = 87$</td>
<td>0.25</td>
<td>01010111</td>
<td>8</td>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>$r_{128} = 128$</td>
<td>0.47</td>
<td>10000000</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$r_{186} = 186$</td>
<td>0.25</td>
<td>11000100</td>
<td>8</td>
<td>000</td>
<td>3</td>
</tr>
<tr>
<td>$r_{255} = 255$</td>
<td>0.03</td>
<td>11111111</td>
<td>8</td>
<td>001</td>
<td>3</td>
</tr>
<tr>
<td>$r_k$ for $k \neq 87, 128, 186, 255$</td>
<td>0</td>
<td>—</td>
<td>8</td>
<td>—</td>
<td>0</td>
</tr>
</tbody>
</table>

$\text{Code 2:}$

$\left\{ \begin{array}{l}
L_{\text{avg}} = 0.25(2)+0.47(1)+0.25(3)+0.03(3)=1.81 \text{ bits} \\
C_R = \frac{n_1}{n_2} = \frac{256 \times 256 \times 8}{256 \times 256 \times 1.81} = \frac{8}{1.81} \approx 4.42 \Rightarrow R_D = 1 - \frac{1}{4.42} = 0.774 \Rightarrow 77.4\%
\end{array} \right.$
• Example (2): Spatial Redundancy
  – Uniform intensity
  – Highly correlated in horizontal direction
  – No correlation in vertical direction
  – Coding using run-length pairs: \((g_1, RL_1), (g_2, RL_2), \ldots\)
Image Compression

- Irrelevant/Psychovisual Redundancy:
  - Human perception of images:
    - Normally does NOT involve quantitative analysis of every pixel value.
  - Quantization eliminates psychovisual redundancy
Image Compression

- Image Compression using Psychovisual Redundancy:
  - A Simple method: Discard lower order bits (4)
  - IGS (Improved Gray Scale) Quantization:
    - Eye sensitivity to edges.
    - Add a pseudo random number (lower order bits of neighbors) to central pixel and then discard lower order bits.

\[
\begin{align*}
169 \div 16 &= 10.56 \quad \rightarrow \quad 11 \\
167 \div 16 &= 10.43 \quad \rightarrow \quad 10
\end{align*}
\]

\[
\begin{align*}
11 \times 16 &= 176 \\
10 \times 16 &= 160
\end{align*}
\]

Rounded values
## Image Compression

**IGS Example:**

<table>
<thead>
<tr>
<th>Pixel</th>
<th>Gray Level</th>
<th>SUM</th>
<th>IGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-1</td>
<td>N/A</td>
<td>0000 0000</td>
<td>N/A</td>
</tr>
<tr>
<td>i</td>
<td>0110 1100</td>
<td>0110 1100</td>
<td>0110</td>
</tr>
<tr>
<td>i+1</td>
<td>1000 1011</td>
<td>1001 0111</td>
<td>1001</td>
</tr>
<tr>
<td>i+2</td>
<td>1000 0111</td>
<td>1000 1110</td>
<td>1000</td>
</tr>
<tr>
<td>i+3</td>
<td>1111 0100</td>
<td>1111 0100</td>
<td>1111</td>
</tr>
</tbody>
</table>
## Digital Image Processing

**Image Compression**

- **Sample Results:**

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Simple Discard</th>
<th>IGS</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="Original" /></td>
<td><img src="image2.jpg" alt="Simple Discard" /></td>
<td><img src="image3.jpg" alt="IGS" /></td>
<td></td>
</tr>
</tbody>
</table>
ImageCompression

- Image information Measure:
  - Self Information

\[
I(E) = \log_m \frac{1}{P(E)} = - \log P(E): m\text{-ary units}
\]

\[
m = 2 \text{ for bit expression } (P(E) = 0.5 \Rightarrow I(E) = 1)
\]
• **Information Channel:**
  
  – A Source of statistically independent random events
  
  – Discrete possible events \(\{a_1, a_2, \ldots, a_J\}\) with associated probability: \(\{P(a_1), P(a_2), \ldots, P(a_J)\}\)
  
  – Average information per source output (Entropy):
    \[
    H = -\sum_{j=1}^{J} P(a_j) \log P(a_j)
    \]
  
  – For “zero-memory” (intensity source) imaging system:
    
    • Using Histogram
    \[
    \tilde{H} = -\sum_{k=0}^{L-1} p_r(r_k) \log_2 p_k(r_k) \text{ bits / pixel}
    \]
    
    • This lower band of bits-length
Image Compression

- Example:

<table>
<thead>
<tr>
<th>$r_k$</th>
<th>$p_i(r_k)$</th>
<th>Code 1</th>
<th>$l_1(r_k)$</th>
<th>Code 2</th>
<th>$l_2(r_k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{87} = 87$</td>
<td>0.25</td>
<td>01010111</td>
<td>8</td>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>$r_{128} = 128$</td>
<td>0.47</td>
<td>10000000</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$r_{186} = 186$</td>
<td>0.25</td>
<td>11000100</td>
<td>8</td>
<td>000</td>
<td>3</td>
</tr>
<tr>
<td>$r_{255} = 255$</td>
<td>0.03</td>
<td>11111111</td>
<td>8</td>
<td>001</td>
<td>3</td>
</tr>
<tr>
<td>$r_k$ for $k \neq 87, 128, 186, 255$</td>
<td>0</td>
<td>—</td>
<td>8</td>
<td>—</td>
<td>0</td>
</tr>
</tbody>
</table>

$\tilde{H} = 1.6614$ bits/pixel
Digital Image Processing

Image Compression

- Shannon First Theorem (Noiseless Coding Theorem):
  - For a $n$-symbol group with $L_{avg, n}$ average number of code symbol:

\[
\lim_{n \to \infty} \left[ \frac{L_{avg, n}}{n} \right] = H
\]
Fidelity Criteria:

- Objective (Quantitative)
  - A number (vector) is calculated

- Subjective (Qualitative)
  - Judgment of different subject is used in limited level.
Digital Image Processing

Image Compression

• Objective Criteria:

− $f(x, y)$: input image

− $\hat{f}(x, y)$: approximated image

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

$$E_{abs} = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} |\hat{f}(x, y) - f(x, y)|$$

$$E_{rms} = \left( \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} |\hat{f}(x, y) - f(x, y)|^2 \right)^{1/2}$$

$$SNR_{ms} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} |\hat{f}(x, y)|^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} |f(x, y) - \hat{f}(x, y)|^2}$$
• Objective Criteria:
  – PSNR: Peak SNR

\[
PSNR = 10 \log_{10} \left( \frac{L^2}{\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left| \hat{f}(x, y) - f(x, y) \right|^2} \right)
\]

\( L \): Maximum level of images (255 for 8bits)
## Subjective Test

<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>
Example:

- Original and three approximations.

\[ E_{rms} = 5.17 \]

\[ E_{rms} = 15.67 \]

\[ E_{rms} = 14.17 \]
Image Compression Model:

- **Encoder**: Create a set of symbols from image data.
  - **Source Encoder**: Reduce input redundancy.
  - **Channel Encoder**: Increase noise immunity (noise robustness)
- **Decoder**: Construct output image from transmitted symbols
  - **Source Decoder**: Reverse of Source encoder
  - **Channel Decoder**: Error detection/correction.
Image Compression

- **Source Encoder-Decoder:**
  - **Mapper:** Map input data to suitable format for interpixel redundancies reduction (i.e., Run-Length coding)
  - **Quantizer:** Reduce accuracy of mapper output according to fidelity criteria (i.e., Psychovisual Redundancy)
  - **Symbol Encoder:** Create Fix or variable length symbols (code redundancy)
• Channel Encoder-Decoder*: 
  – Add controlled redundancy to source encoded data
  – Increase noise robustness.
  – Hamming (7-4) as an example:

4 bit binary input data: \( b_3 b_2 b_1 b_0 \)

7 bit binary output data: \( h_7 h_6 h_5 h_3 h_2 h_1 \Leftrightarrow b_0 b_1 b_2 h_4 b_3 h_2 h_1 \)

\[
\begin{align*}
  h_1 &= b_3 \oplus b_2 \oplus b_1 \\
  h_2 &= b_3 \oplus b_1 \oplus b_0 \\
  h_4 &= b_2 \oplus b_1 \oplus b_0 \\
\end{align*}
\]

Parity bits, Error check:

\[
\begin{align*}
  c_1 &= h_1 \oplus h_3 \oplus h_5 \oplus h_7 \\
  c_2 &= h_2 \oplus h_3 \oplus h_6 \oplus h_7 \\
  c_4 &= h_4 \oplus h_5 \oplus h_6 \oplus h_7 \\
\end{align*}
\]
Image Compression Standards

- See Tables 8.3 and 8.4 for more details
Image Compression

• Error-Free Compression:
  – ZIP, RAR, TIFF(LZW/ZIP)
    • Compression Ratio: 2-10
  – Steps:
    • Alternative representation to reduce interpixel redundancy
    • Coding the representation to reduce coding redundancy
Digital Image Processing

Image Compression

• Variable-Length Coding:
  – Variable-Length Coding
    • Assign variable bit length to symbols based on its probability:
      • Low probability (high information) more bits
      • High Probability (low information) less bits
Huffman Coding:

- Uses frequencies (Probability) of symbols in a string to build a variable rate prefix code.
- Each symbol is mapped to a binary string.
- More frequent symbols have shorter codes.
- No code is a prefix of another. (Uniquely decodable)
- It is optimum!
- CCITT, JBIG2, JPEG, MPEG (1, 2, 4), H261-4
Image Compression

- Huffman Coding:
  - Example:
    - We have three symbols a, b, c, and d.

```
a  0
b  100
c  101
d  11
```
Image Compression

- Huffman Coding:
  - A source string: `aabddcaa`
  - Fixed Length Coding: 16 bits (ordinary coding)
    - `00 00 01 11 11 10 00 00`
  - Variable length coding: 14bits (Huffman coding)
    - `0 0 100 11 11 101 0 0`
  - Uniquely Decodable:

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

```
a  a  b  d  d  c  a  a
```
Huffman Coding:

- Consider an information source: \( S = (s_1, s_2, \ldots, s_m) \)
- Assume symbols are ranked decreasing with respect to occurrence probabilities:
  \[
p(s_1) \geq p(s_2) \geq \cdots \geq p(s_{m-1}) \geq p(s_m)
  \]
- We seek for optimum code, the lengths of codewords assigned to the source symbols should be:
  \[
l_1 \leq l_2 \leq \cdots \leq l_{m-1} \leq l_m
  \]
Image Compression

• Huffman Coding:
  – Huffman (1951) Derived the following rules:
    • Two least probable source symbols have equal-length codewords.
      \[ l_1 \leq l_2 \leq \cdots \leq l_{m-1} = l_m \]
    • These two codewords are identical except for the last bits, with binary 0 and 1, respectively.
    • These source can be combined to form a new symbol.
      – Its occurrence probability is \( P(s_m) + P(s_{m-1}) \)
      – Its codeword is common prefix of order \( l_{m-1} \) of the two codewords assigned to \( S_m \) and \( S_{m-1} \), respectively.
Huffman Coding:

- The new set of source symbols thus generated is referred to as the first auxiliary source alphabet, which is one source symbol less than the original source alphabet.
- In the first auxiliary source alphabet, we can rearrange the source symbols according to a nonincreasing order of their occurrence probabilities.
- The same procedure can be applied to this newly created source alphabet.
- The second auxiliary source alphabet will again have one source symbol less than the first auxiliary source alphabet.
- This is repeated until we form a single source symbol with a probability of 1.
- Start from the source symbol in the last auxiliary source alphabet and trace back to each source symbol in original source alphabet to find the codewords.
• Huffman Coding:
  – Source Reduction sequence.

<table>
<thead>
<tr>
<th>Original source</th>
<th>Source reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td>Probability</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.4</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.3</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.1</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.1</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.06</td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.04</td>
</tr>
</tbody>
</table>

**FIGURE 8.11**
Huffman source reductions.
• More Illustration:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.05</td>
<td>0000</td>
</tr>
<tr>
<td>b</td>
<td>0.05</td>
<td>0001</td>
</tr>
<tr>
<td>c</td>
<td>0.1</td>
<td>001</td>
</tr>
<tr>
<td>d</td>
<td>0.2</td>
<td>01</td>
</tr>
<tr>
<td>e</td>
<td>0.3</td>
<td>10</td>
</tr>
<tr>
<td>f</td>
<td>0.2</td>
<td>110</td>
</tr>
<tr>
<td>g</td>
<td>0.1</td>
<td>111</td>
</tr>
</tbody>
</table>
### Image Compression

#### Example:

$L_{\text{avg}} = 2.2 \ \text{bits/symbol}$

<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</td>
<td>1</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.3</td>
<td>00</td>
<td>0.3</td>
<td>00</td>
<td>0.3</td>
<td>00</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.1</td>
<td>011</td>
<td>0.1</td>
<td>011</td>
<td>0.2</td>
<td>010</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.1</td>
<td>0100</td>
<td>0.1</td>
<td>0100</td>
<td>0.1</td>
<td>011</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.06</td>
<td>01010</td>
<td>0.1</td>
<td>0101</td>
<td>0.1</td>
<td>011</td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.04</td>
<td>01011</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 8.12** Huffman code assignment procedure.
Image Compression

• Example:
  – 8bits monochrome
    • Estimated Entropy: 7.3838 bits/pixel
  – Huffman’s Coding:
    • MATLAB: 7.428 bits/pixel, CR = 1.077, RD = 0.0715
### Image Compression

- Near Optimal

<table>
<thead>
<tr>
<th>Source symbol</th>
<th>Probability</th>
<th>Binary Code</th>
<th>Huffman</th>
<th>Truncated Huffman</th>
<th>B2-Code</th>
<th>Binary Shift</th>
<th>Huffman Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.2</td>
<td>00000</td>
<td>10</td>
<td>11</td>
<td>C00</td>
<td>000</td>
<td>10</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.1</td>
<td>00001</td>
<td>110</td>
<td>011</td>
<td>C01</td>
<td>001</td>
<td>11</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.1</td>
<td>00010</td>
<td>111</td>
<td>0000</td>
<td>C10</td>
<td>010</td>
<td>110</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.06</td>
<td>00011</td>
<td>0101</td>
<td>0101</td>
<td>C11</td>
<td>011</td>
<td>100</td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.05</td>
<td>00100</td>
<td>00000</td>
<td>00010</td>
<td>C00C00</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.05</td>
<td>00101</td>
<td>00001</td>
<td>00011</td>
<td>C00C01</td>
<td>101</td>
<td>1110</td>
</tr>
<tr>
<td>$a_7$</td>
<td>0.05</td>
<td>00110</td>
<td>00010</td>
<td>00100</td>
<td>C00C10</td>
<td>110</td>
<td>1111</td>
</tr>
<tr>
<td>Block 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_8$</td>
<td>0.04</td>
<td>00111</td>
<td>00011</td>
<td>00101</td>
<td>C00C11</td>
<td>111000</td>
<td>0010</td>
</tr>
<tr>
<td>$a_9$</td>
<td>0.04</td>
<td>01000</td>
<td>00110</td>
<td>00110</td>
<td>C01C00</td>
<td>111001</td>
<td>0011</td>
</tr>
<tr>
<td>$a_{10}$</td>
<td>0.04</td>
<td>01001</td>
<td>00111</td>
<td>00111</td>
<td>C01C01</td>
<td>111010</td>
<td>00110</td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>0.04</td>
<td>01010</td>
<td>00100</td>
<td>01000</td>
<td>C01C10</td>
<td>111011</td>
<td>00100</td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>0.03</td>
<td>01011</td>
<td>01001</td>
<td>01001</td>
<td>C01C11</td>
<td>111000</td>
<td>00101</td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>0.03</td>
<td>01100</td>
<td>01110</td>
<td>100000</td>
<td>C10C00</td>
<td>111101</td>
<td>001110</td>
</tr>
<tr>
<td>$a_{14}$</td>
<td>0.03</td>
<td>01101</td>
<td>01111</td>
<td>100001</td>
<td>C10C01</td>
<td>111110</td>
<td>001111</td>
</tr>
<tr>
<td>Block 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{15}$</td>
<td>0.03</td>
<td>01110</td>
<td>01100</td>
<td>100010</td>
<td>C10C10</td>
<td>1111111000</td>
<td>0010010</td>
</tr>
<tr>
<td>$a_{16}$</td>
<td>0.02</td>
<td>01111</td>
<td>010000</td>
<td>100011</td>
<td>C10C11</td>
<td>1111111001</td>
<td>0000111</td>
</tr>
<tr>
<td>$a_{17}$</td>
<td>0.02</td>
<td>10000</td>
<td>010001</td>
<td>100100</td>
<td>C11C00</td>
<td>1111111010</td>
<td>00001110</td>
</tr>
<tr>
<td>$a_{18}$</td>
<td>0.02</td>
<td>10001</td>
<td>01010</td>
<td>100101</td>
<td>C11C01</td>
<td>1111111101</td>
<td>00001100</td>
</tr>
<tr>
<td>$a_{19}$</td>
<td>0.02</td>
<td>10010</td>
<td>001011</td>
<td>100110</td>
<td>C11C10</td>
<td>1111111100</td>
<td>0000101</td>
</tr>
<tr>
<td>$a_{20}$</td>
<td>0.02</td>
<td>10011</td>
<td>011010</td>
<td>100111</td>
<td>C11C11</td>
<td>1111111101</td>
<td>00001110</td>
</tr>
<tr>
<td>$a_{21}$</td>
<td>0.01</td>
<td>10100</td>
<td>011011</td>
<td>101000</td>
<td>C00C00C00</td>
<td>111111110</td>
<td>00001111</td>
</tr>
</tbody>
</table>

**Entropy**: 4.0

**Average length**: 5.0, 4.05, 4.24, 4.65, 4.59, 4.13
Huffman drawbacks:

- For large number symbols, construction of optimal Huffman code is nontrivial.
- Huffman is block coding:
  - That is, some codeword having an integral number of bits is assigned to a source symbol.
  - A message may be encoded by cascading the relevant codewords. It is the block-based approach that is responsible for the limitations of Huffman codes.
  - Computationally not efficient for a message with all possible symbols.
Golomb Coding (1):

- Non-negative integer inputs
- Exponentially decaying probability distribution
- Inputs may be near optimal codes (simple than Huffman)
- JPEG-LS and AVS

Notation:

\[ \left\lfloor x \right\rfloor : \text{Largest integer less than or equal to } x, \text{Floor} \]

\[ \left\lceil x \right\rceil : \text{Smallest integer greater than or equal to } x, \text{Ceil} \]
Golomb Coding (2):

- Goal:
  - Golomb Coding of \( n \) (Non-Negative) with respect to \( m > 0 \): \( G_m(n) \)
  - Combination of unary coding quotient \( \lfloor n/m \rfloor \) and binary representation of remainder “\( n \mod m \)”
  - Unary coding of integer \( q \) is: \( q \) “1s” followed by a “0”
• Golomb Coding (3):
  – Steps:
    • Form unary coding of \(\lceil n/m \rceil\)
    • With \(k = \lceil \log_2 m \rceil\), \(c = 2^k - m\), \(r = n \mod m\) compute truncated remainder:
      \[
      r' = \begin{cases} 
      r \text{ truncated to } k - 1 \text{ bits} & 0 \leq r < c \\
      r + c \text{ truncated to } k \text{ bits} & \text{o.w.} 
      \end{cases}
      \]
    • Concatenate the results of two steps
Golomb Coding Example (1):

- Goal: $G_4(9)$
- Step #1: $\lfloor 9/4 \rfloor = 2 \xrightarrow{\text{Unary Coding}} 110$
- Step #2:
  \[
  k = \lfloor \log_2 4 \rfloor = 2, \quad c = 2^2 - 4 = 0
  \]
  \[
  r = 9 \mod 4 = 1 (0001) \xrightarrow{\text{truncated to } k=2} r' = 01
  \]
- Step #3: Concatenate 110 and 01: 11001
**Digital Image Processing**

**Image Compression**

- **Golomb Coding Example (2):**

<table>
<thead>
<tr>
<th>( n )</th>
<th>( G_1(n) )</th>
<th>( G_2(n) )</th>
<th>( G_4(n) )</th>
<th>( G_{exp}(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>01</td>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
<td>100</td>
<td>010</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>1110</td>
<td>101</td>
<td>011</td>
<td>1100</td>
</tr>
<tr>
<td>4</td>
<td>11110</td>
<td>1100</td>
<td>1000</td>
<td>11001</td>
</tr>
<tr>
<td>5</td>
<td>111110</td>
<td>1101</td>
<td>1001</td>
<td>11010</td>
</tr>
<tr>
<td>6</td>
<td>1111110</td>
<td>11100</td>
<td>1010</td>
<td>11011</td>
</tr>
<tr>
<td>7</td>
<td>11111110</td>
<td>11101</td>
<td>1011</td>
<td>111000</td>
</tr>
<tr>
<td>8</td>
<td>111111110</td>
<td>111100</td>
<td>11000</td>
<td>1110001</td>
</tr>
<tr>
<td>9</td>
<td>1111111110</td>
<td>111101</td>
<td>11001</td>
<td>1110010</td>
</tr>
</tbody>
</table>
Digital Image Processing

Image Compression

• Golomb Coding (4):

  – Easy (binary shift operation) for: \( m = 2^k, \ c = 0, \ r' = r = n \mod m \)
  
  • Golomb-Rice or Rice Codes

  – Optimality condition:

    • Non-negative geometrically distribution:

      \[
      P(n) = (1 - \rho) \rho^n, \ 0 < \rho < 1
      \]

    • With:

      \[
      m = \left\lfloor \frac{\log_2 (1 + \rho)}{\log_2 (1/\rho)} \right\rfloor
      \]
• Usability (1):
  – Intensity image distribution are NOT geometrical.
  – Difference images are.

• How handle negative value:
  – Map to Odd and even non-negative integer:
    \[
    M(n) = \begin{cases} 
    2n & n \geq 0 \\
    2|n| - 1 & n < 0 
    \end{cases}
    \]
Digital Image Processing

Image Compression

- Usability (2):
  - Geometrically distribution(s)
  - Difference image histogram
  - Mapped histogram

FIGURE 8.10
(a) Three one-sided geometric distributions from Eq. (8.2-2); (b) a two-sided exponentially decaying distribution; and (c) a reordered version of (b) using Eq. (8.2-4).
Image Compression

- Use Golomb Coding in image:
  - Consider $P(n-\mu)$ instead of $P(n)$, $\mu$: mean intensity
  - Map the negative image.
  - Code using $G_m(n)$
Digital Image Processing

Image Compression

• Golomb Image Coding Example (1):

Mapping

Number of pixels

0 50 100 150 200 250

0 1000 2000 3000 4000 5000 6000 7000

Probability

0.75 0.7 0.65 0.6 0.55 0.5 0.45 0.4 0.35 0.3

0 1 2 3 4

n - \mu

0 1 2 3 4 5 6 7 8

M(n - \mu)
Golomb Image Coding Results (1):

- $G_1(n)$
  - $C=4.5$
  - 88% of theoretical compression ratio (4.5/5.1)
  - 96% of Huffman coding compression (Both in MATLAB)
    - Less computational efforts
Golomb Image Coding Example (2):

- $G_1(n)$
  - $C=0.0922!!!$
  - Data Expansion

- Golomb is model based
Digital Image Processing

Image Compression

• Golomb Exponential Coding of order $k$:
  – Step #1: Find non-negative integer $i$ such that:
    $$\sum_{j=0}^{i-1} 2^{j+k} \leq n < \sum_{j=0}^{i} 2^{j+k}$$
    • And form unary coding of $i$.
  – Step #2: Truncate the following binary representation to $k+I$ (LSB)
    $$n - \sum_{j=0}^{i-1} 2^{j+k}$$
  – Step #3: Concatenate results of step #1 and #2
  – $k=0$ is known as *Elias-gamma* code
• Example:

– Let’s compute: $G^0_{\exp}(8)$

*Step #1:*

$k = 0 \Rightarrow i = \left\lfloor \log_2 9 \right\rfloor = 3$ \text{(Unary Coding)} $\rightarrow 1110$

*Step #2:*

$$8 - \sum_{j=0}^{3-1} 2^{j+k} = 8 - 1 - 2 - 4 = 1$$ \text{(Unary Coding)} $\rightarrow 0001$ \text{(Trunc. to 3+0)} $\rightarrow 001$

*Step #3: 1110001*
Digital Image Processing

Image Compression

- Arithmetic Coding:
  - Stream-based (A string of source symbols is encoded as a string of code symbols):
  - Free of the integer number of bits/symbol restriction and more efficient.
  - Arithmetic coding may reach the theoretical bound of coding efficiency specified in the noiseless source coding theorem for any information source.
  - JBIG1, JBIG2, JPEG-2000, H264, MPEG-4 AVC
**Image Compression**

- **Arithmetic Coding, Basic idea:**
  - With $n$ bits, a code interval of $[0,1]$ divided to $2^n$ parts with length of $2^{-n}$.
  - Vise versa $A=2^{-n}$ can be coded using $-\log_2 A$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0.125</th>
<th>0.250</th>
<th>0.375</th>
<th>0.500</th>
<th>0.625</th>
<th>0.750</th>
<th>0.875</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
• Arithmetic Coding-Basic Idea:

  – Represent the entire input stream as a small interval between the range $[0,1]$. 
  – 1. Divide interval $([0,1])$ to subintervals according to probability distribution. 
  – The first symbol interval is taken and again divide to sub interval with length of relative to probabilities. 
  – Now take second symbol interval and continue ...
### Arithmetic Encoding By Example:

- **Stream is** *baca*

<table>
<thead>
<tr>
<th>$S_i$</th>
<th>$P_i$</th>
<th>Range</th>
<th>Range</th>
<th>Range</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.5</td>
<td>[0.0-0.5)</td>
<td>[0.5-0.625)</td>
<td>[0.5-0.5625)</td>
<td>[0.59375-0.609375)</td>
</tr>
<tr>
<td>b</td>
<td>0.25</td>
<td>[0.5-0.75)</td>
<td>[0.625-0.6875)</td>
<td>[0.5625-0.59375)</td>
<td>[0.609375-0.6171875)</td>
</tr>
<tr>
<td>c</td>
<td>0.25</td>
<td>[0.75-1.0)</td>
<td>[0.6875-0.75)</td>
<td>[0.59375-0.625)</td>
<td>[0.6171875-0.625)</td>
</tr>
</tbody>
</table>
Digital Image Processing

Image Compression

• Arithmetic Decoding By Example:
  – Code is 0.59375
    • 0.5<0.59375<0.75 → b (b section of main interval)
    • 0.5<0.59375<0.625 → a (a section of subinterval b)
    • 0.59375<0.59375<0.625 → c (c section of subinterval a)
    • 0.59375<0.59375<0.609375 → a (a section of subinterval c)
  – 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
    • Stop for Number = 0
Image Compression

• Implementation:
  – Change \([0,1]\) to \([0000\text{H},FFFF\text{H}]\)
  – Change Probabilities value
Image Compression

- Arithmetic Coding

**FIGURE 8.13**
Arithmetic coding procedure.
Image Compression

- Lempel-Ziv-Welch (LZW) Coding
  - Reduce Spatial-Coding redundancy
  - Assign fixed-length codewords to variable-length sequences of source symbols.
  - No priori knowledge about intensity probability
  - GIF, TIFF, PNG, PDF
Digital Image Processing

Image Compression

• Introductory Example:

<table>
<thead>
<tr>
<th>Gray level</th>
<th>Count</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>12</td>
<td>0.375</td>
</tr>
<tr>
<td>95</td>
<td>4</td>
<td>0.125</td>
</tr>
<tr>
<td>169</td>
<td>4</td>
<td>0.125</td>
</tr>
<tr>
<td>243</td>
<td>12</td>
<td>0.375</td>
</tr>
</tbody>
</table>

1.81 bits/pixel
Digital Image Processing

Image Compression

- Introductory Example:

  - Second Order Entropy:

<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>0.250</td>
</tr>
<tr>
<td>(21, 95)</td>
<td>4</td>
<td>0.125</td>
</tr>
<tr>
<td>(95, 169)</td>
<td>4</td>
<td>0.125</td>
</tr>
<tr>
<td>(169, 243)</td>
<td>4</td>
<td>0.125</td>
</tr>
<tr>
<td>(243, 243)</td>
<td>8</td>
<td>0.250</td>
</tr>
<tr>
<td>(243, 21)</td>
<td>4</td>
<td>0.125</td>
</tr>
</tbody>
</table>

1.25 bits/pixel
• **Meta Algorithm**

  - Initialize the dictionary to contain all strings of length **one**.
  - Find the **longest** string W in the dictionary that matches the current input.
  - Emit the dictionary index for W to output and remove W from the input.
  - Add W followed by the next symbol in the input to the dictionary.
  - Go to Step 2.
Image Compression

• Example \textsuperscript{Wikipedia}:
  - String to be code: \texttt{TOBEORNOTTOBEORTOBEORNOT#}
  - 1) Initial Dictionary:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
<th>Symbol</th>
<th>Code</th>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>I</td>
<td>9</td>
<td>R</td>
<td>18</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>J</td>
<td>10</td>
<td>S</td>
<td>19</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>K</td>
<td>11</td>
<td>U</td>
<td>20</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>L</td>
<td>12</td>
<td>V</td>
<td>21</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>M</td>
<td>13</td>
<td>W</td>
<td>22</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
<td>N</td>
<td>14</td>
<td>X</td>
<td>23</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>O</td>
<td>15</td>
<td>Y</td>
<td>24</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>P</td>
<td>16</td>
<td>Z</td>
<td>25</td>
</tr>
<tr>
<td>H</td>
<td>8</td>
<td>Q</td>
<td>17</td>
<td>#</td>
<td>26</td>
</tr>
</tbody>
</table>
### Example: TOBEORNOTTOBEORTOBEORNOT#

<table>
<thead>
<tr>
<th>Input Code</th>
<th>Output Sequence</th>
<th>New Dictionary Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Full</td>
</tr>
<tr>
<td>20</td>
<td>T</td>
<td>27: TO</td>
</tr>
<tr>
<td>15</td>
<td>O</td>
<td>28: OB</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>29: BE</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>30: EO</td>
</tr>
<tr>
<td>15</td>
<td>O</td>
<td>31: OR</td>
</tr>
<tr>
<td>18</td>
<td>R</td>
<td>32: RN</td>
</tr>
<tr>
<td>14</td>
<td>N</td>
<td>33: NO</td>
</tr>
<tr>
<td>15</td>
<td>O</td>
<td>34: OT</td>
</tr>
<tr>
<td>20</td>
<td>T</td>
<td>35: TT</td>
</tr>
<tr>
<td>27</td>
<td>TO</td>
<td>36: TOB</td>
</tr>
<tr>
<td>29</td>
<td>BE</td>
<td>37: BEO</td>
</tr>
<tr>
<td>31</td>
<td>OR</td>
<td>38: OR?</td>
</tr>
<tr>
<td>36</td>
<td>TOB</td>
<td>39: TOBT</td>
</tr>
<tr>
<td>30</td>
<td>EO</td>
<td>40: EOB</td>
</tr>
<tr>
<td>32</td>
<td>RN</td>
<td>41: EOR</td>
</tr>
<tr>
<td>34</td>
<td>OT</td>
<td>42: RNO</td>
</tr>
</tbody>
</table>
Image Compression

0. Initialize a dictionary by all possible gray values (0-255)

1. Input current pixel

2. If the current pixel combined with previous pixels form one of existing dictionary entries

   Then

   2.1 Move to the next pixel and repeat Step 1

Else

   2.2 Output the dictionary location of the currently recognized sequence (which is not include the current pixel)

   2.3 Create a new dictionary entry by appending the currently recognized sequence in 2.2 with the current pixel

   2.4 Move to the next pixel and repeat Step 1
Image Compression

• **LZW for Image coding:**
  
  – Initialize a dictionary by all possible gray values (0-255)
  
  – **(1)** Input current pixel
  
  – **(2)** If the *(current pixel concatenate previous pixels)* exists in dictionary, then
    
    • **(2-1)** Move to the next pixel and repeat Step 1
  
  – Else
    
    • **(2-2)** Output the dictionary location of the currently recognized sequence *(which is not include the current pixel)*
    
    • **(2-3)** Create a new dictionary entry by appending the currently recognized sequence in (2-2) with the current pixel
    
    • **(2-4)** Move to the next pixel and repeat Step 1
Image Compression

Dictionary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>257</td>
<td>39-126</td>
</tr>
<tr>
<td>258</td>
<td>126-126</td>
</tr>
<tr>
<td>259</td>
<td>126-39</td>
</tr>
<tr>
<td>260</td>
<td>39-39-126</td>
</tr>
<tr>
<td>261</td>
<td>126-126-39</td>
</tr>
<tr>
<td>262</td>
<td>39-39-126-126</td>
</tr>
</tbody>
</table>

Input

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
<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>
<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>
<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>

Sequences

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
<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>
<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>
<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>

Encoded Output (9 bits)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
<tr>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>258</td>
<td>258</td>
</tr>
<tr>
<td>260</td>
<td>260</td>
</tr>
</tbody>
</table>

E. Fatemizadeh, Sharif University of Technology, 2011
Binary image compression:

- A very simple approach: 8 pixels to one byte (code redundancy)

- Constant Area Coding (CAC):
  - Image is divided to blocks \((p \times q)\):
    - White, Black, and Mixed
  - Coding:
    - Most Probable (W/B/M) will code by one bit (0)
    - Two others will code by two bits (10/11)
    - Mixed code is prefix of its data
• Binary image compression:
  – White Block Skipping (WBS):
    • 0 for whole white blocks
    • 1 is used as prefix for mixed or whole black (less probable)
  – Symbol-based (Token-based) Coding:
    • Image modeled as a collection of frequently occurred sub-images (symbols or tokens)
    • \((x_i,y_i,t_i):(x_i,y_i)\) are position of symbols and \(t_i\) position of symbol in dictionary

– JBIG2 (Text Halftone, Generic region), Page: 561-2
• Symbol-based (Token-based) Coding:
  – Image modeled as a collection of frequently occurred sub-images (symbols or tokens)
  – \((x_i, y_i, t_i)\):\((x_i, y_i)\) are position of symbols and \(t_i\)
    position of symbol in dictionary

<table>
<thead>
<tr>
<th>Token</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><img src="symbol_0.png" alt="Symbol 0" /></td>
</tr>
<tr>
<td>1</td>
<td><img src="symbol_1.png" alt="Symbol 1" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="symbol_2.png" alt="Symbol 2" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Triplet</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 2, 0)</td>
</tr>
<tr>
<td>(3, 10, 1)</td>
</tr>
<tr>
<td>(3, 18, 2)</td>
</tr>
<tr>
<td>(3, 26, 1)</td>
</tr>
<tr>
<td>(3, 34, 2)</td>
</tr>
<tr>
<td>(3, 42, 1)</td>
</tr>
</tbody>
</table>
• One dimensional Run Length Coding (RLE):

  – Basic idea:
    • Code each cluster of white (1) and Black (0) pixel by its length (Or cluster of constant gray level).
    • Two Approach:
      – Specify value of first run of each row.
      – Assume each row begin with white (its length may be zero)
    – We may use variable length coding for each length of black or white.
    – Facsimile (FAX) and BMP
Image Compression

- Run-Length Coding Implementation:
  - One-dimensional CCITT
  - Two-Dimensional CCITT
    - Coding Black-to-white and white-to-black transition
  - Read pages 555-559
Image Compression

• Bit-Plane Coding:
  – Ordinary Coding:
    \[ A = a_{m-1} 2^{m-1} + a_{m-2} 2^{m-2} + \cdots + a_1 2^1 + a_0 2^0 \]
  – We could have 8 Binary images, but its drawback:
    • Small changes in gray level can have significant effect on bit-plane images: 127=01111111 and 128=10000000, all eight images are different!
  – Gray Code:
    \[ g_i = a_i \oplus a_{i+1}, i = 0,1,\ldots,m-2 \]
    \[ g_{m-1} = a_{m-1} \]
  – 127=11000000 and 128=01000000
Image Compression

- Bit-Plane Coding Example
Digital Image Processing

Image Compression

Gray Images are less complex

• Example:
  – Bit-Planes (7-4)

Binary

Gray

Bit 7

Bit 6

Bit 5

Bit 4
Image Compression

Gray Images are less complex

- Example:
  - Bit-Planes (3-0)
Image Compression

- **JBIG2 Results:**

<table>
<thead>
<tr>
<th>Coefficient $m$</th>
<th>Binary Code (PDF bits)</th>
<th>Gray Code (PDF bits)</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6,999</td>
<td>6,999</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>12,791</td>
<td>11,024</td>
<td>1.16</td>
</tr>
<tr>
<td>5</td>
<td>40,104</td>
<td>36,914</td>
<td>1.09</td>
</tr>
<tr>
<td>4</td>
<td>55,911</td>
<td>47,415</td>
<td>1.18</td>
</tr>
<tr>
<td>3</td>
<td>78,915</td>
<td>67,787</td>
<td>1.16</td>
</tr>
<tr>
<td>2</td>
<td>101,535</td>
<td>92,630</td>
<td>1.10</td>
</tr>
<tr>
<td>1</td>
<td>107,909</td>
<td>105,286</td>
<td>1.03</td>
</tr>
<tr>
<td>0</td>
<td>99,753</td>
<td>107,909</td>
<td>0.92</td>
</tr>
</tbody>
</table>
• Block Transform Coding:
  – A Reversible-Linear transform maps the image to a set of coefficients.
  – These coefficients then quantized and coded. (Local/Global)


**Image Compression**

- **Transform Selection:**
  - Compression is done during quantization **NOT** transforming.
  - Type of transformation relate to Compression ratio.
    \[
    T(u, v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x, y) r(x, y; u, v)
    \]
    \[
    g(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) s(x, y; u, v)
    \]
  - Separable Transform:
    \[
    r(x, y; u, v) = r_1(x, u) r_2(y, v)
    \]
  - Symmetric Transform:
    \[
    r(x, y; u, v) = r_1(x, u) r_1(y, v)
    \]
Image Compression

Transform Coding:

- DFT (M=N):

\[ r(x, y; u, v) = e^{-j2\pi(ux+vy)/n} \]

\[ s(x, y; u, v) = \frac{1}{n^2} e^{j2\pi(ux+vy)/n} \]

- Fast
- Complex
- Gibbs Phenomenon
Digital Image Processing

Image Compression

- DFT Basis Function:
Image Compression

- Walsh-Hadamard Transform (WHT):

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

\[
n = 2^m, b_k(z) : k\text{th bit (right to left) of binary representation of } z
\]

\[
p_0(u) = b_{m-1}(u)
\]

\[
p_1(u) = b_{m-1}(u) + b_{m-2}(u)
\]

\[
p_2(u) = b_{m-2}(u) + b_{m-3}(u)
\]

\[
\vdots
\]

\[
p_{m-1}(u) = b_1(u) + b_0(u)
\]
Image Compression

- Walsh-Hadamard Basis Function:
Image Compression

- Discrete Cosine Transform (DCT):

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

\[
\alpha(u) = \begin{cases} 
\sqrt{\frac{1}{n}} & u = 0 \\
\sqrt{\frac{2}{n}} & u = 1, 2, \ldots, n-1 
\end{cases}
\]
Image Compression

- DCT Basis Function:
Digital Image Processing

Image Compression

• Example:
  - $n=8$,
  - Truncating 50% of coefficients

<table>
<thead>
<tr>
<th>Method</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFT</td>
<td>2.32</td>
</tr>
<tr>
<td>WHT</td>
<td>1.78</td>
</tr>
<tr>
<td>DCT</td>
<td>1.13</td>
</tr>
</tbody>
</table>

Example images showing compression results for DFT, WHT, and DCT methods.
Image Compression

- Transform Coding Details:
  - Consider a sub-image (block-wise processing):

  \[ g(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) s(x, y; u, v) \]

  **Matrix Formulation:**

  \[ G = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) S_{uv} \]

  \[ S_{uv} = s \left[ h(i, j; u, v) \right]^{(i,j)=(n-1,n-1)}_{(i,j)=(0,0)} \]
• Transform Coding Details:
  – Transform Masking Function:

\[
\chi(u, v) = \begin{cases} 
0 & T(u, v) \text{ in a predefined region} \\
1 & \text{o.w.} 
\end{cases}
\]

\[
\hat{G} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \chi(u, v) T(u, v) S_{uv} \iff \text{Compressed Image}
\]
Transform Coding Details:

- Error Analysis:

\[ e_{ms}^2 = E \left\{ \left\| G - \hat{G} \right\|^2 \right\} \]

\[ = E \left\{ \left\| \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v)S_{uv} - \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \chi(u,v)T(u,v)S_{uv} \right\|^2 \right\} \]

\[ = E \left\{ \left\| \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v)\left[1 - \chi(u,v)\right]S_{uv} \right\|^2 \right\} \]

\[ = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \sigma_{T(u,v)}^2 \left[1 - \chi(u,v)\right] \quad \text{Orthonormality of Transform Coeffs} \]

\[ G \sim N(0, \sigma^2) \]
Transform Compression:

- How to select Transform:
  - Who collect more information in less coefficients.

- Best Transform:
  - KL (Karhune-Loeve)
    - Data Dependent!

- Most used transform:
  - DCT:
    - Single Chip Implementation
    - Packing the most information into the fewest coefficients.
    - Minimum Blocking Artifact.
Image Compression

• Periodicity Implicit DFT and DCT
  – Gibbs phenomenon causes erroneous boundary
  • Blocking artifacts

**Figure 8.32** The periodicity implicit in the 1-D (a) DFT and (b) DCT.
• **Block size effect:**
  
  – **Small Blocks:**
    - Faster
    - More blocking effect (correlation between neighboring pixels)
    - Low compression ratio
  
  – **Large Block:**
    - Slower
    - Better compression in “flat” regions.
    - Lower Blocking Effect
  
  – Power of 2, for fast implementation.
Image Compression

- Block-Size and Transform Effects:

**FIGURE 8.26**
Reconstruction error versus subimage size.
Block Size effect for DCT:

- Reconstruct using 25% of the DCT coefficients

**FIGURE 8.27** Approximations of Fig. 8.27(a) using 25% of the DCT coefficients and (b) 2 × 2 subimages, (c) 4 × 4 subimages, and (d) 8 × 8 subimages. The original image in (a) is a zoomed section of Fig. 8.9(a).
Bit Allocation:

- Bio Allocation: Truncation-Quantization-Coding
- How to select retained coefficients.
  - Maximum Magnitude (Threshold Coding)
  - Maximum Variance (Zonal Coding)

- Threshold Coding:
  - Keeping $P$-largest coefficients.

- Zonal Coding:
  - DCT Coeffs. are considered as RV’s, all statistics computed using ensemble of transformed subimages.
  - Keeping $P$-largest variances (same $-P (u,v)$’s for all subimages.)
• Threshold vs. Zonal Coding:
  – Keep coefficients 8 out of 64

**RMS** = 4.5

**RMS** = 6.5

**FIGURE 8.28**
Approximations of Fig. 8.9(a) using 12.5% of the 8 × 8 DCT coefficients:
(a) — (b) threshold coding results; (c) — (d) zonal coding results. The difference images are scaled by 4.
• Zonal Coding Implementation:
  – Ensemble average over MN/n^2 subimages transforms and select non zero χ(u,v), Zonal Masking. (High variances)
    • 1 or 0 for each position in transform domain, Zonal Mask
    • Number of bits used to code each coefficients, Zonal bit allocation
  – Quantization of the retained coefficients:
    • Fixed Length (Uniformly Quantized)
    • Optimal Quantizer:
      – A model for DCT coefficients:
        » Rayleigh for DC component. (A positive RV’s)
        » Laplacian or Gaussian for others (CLT theorem)
Zonal Coding Example:

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Z-Coding ↔ # of bits

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Threshold Coding Implementation:

- An Adaptive mask selection of non-zero $\chi(u,v)$
- Location of maximum magnitude vary from one sub-image to another.

- Re-order 2D DCT to a 1D (Run Length Coded Sequence) in a zigzag arrangement.

![Diagram showing threshold coding implementation](image-url)
Why zigzag arrangement:

- This is done so that the coefficients are in order of increasing frequency.
- This improves the compression of run-length encoding.
### Threshold Coding Example:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

T-Coding ←→ Zigzag tracing
Thresholding:

- A single global threshold for all subimages.
  - Different CR from image to image
- A different threshold for each subimages.
  - \textit{N-Largest coding} (same number of images are discarded)
  - Same CR for all images.
- A variable threshold as a function of location.
  - Different CR from image to image.
  - Combined Quantization and Thresholding.
Image Compression

Location Variant Thresholding:

**Compression:**

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

\(Z(u, v)\): Elements of Transform Normalization

**Decompression:**

\[
\tilde{T}(u, v) = Z(u, v) \hat{T}(u, v) \xrightarrow{\text{inv}} \tilde{f}(x, y)
\]
• Thresholding-Quantization:
  – Left for specific $Z(u,v)=c$
  – Map of $Z(u,v)$

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

$$\hat{T}(u,v) = k, \quad kc - \frac{c}{2} \leq T(u,v) \leq kc + \frac{c}{2}$$

$$\hat{T}(u,v) = 0, \quad T(u,v) \leq \frac{Z(u,v)}{2}$$
• Quantization Effect:

\[ Z(u,v) \quad 2Z(u,v) \quad 4Z(u,v) \]

\[ 8Z(u,v) \quad 16Z(u,v) \quad 32Z(u,v) \]
Image Compression

• JPEG – Based on the Text Book
  – Three different coding systems:
    • Lossy *baseline coding system* (most application)
    • Greater Compression/Progressive reconstruction (Web)
    • Lossless independent coding (reversible compression)
Image Compression

• Baseline system (Sequential baseline system):
  – Input and Output precision: 8 bits
  – Quantized DCT coefficients precision: 11 bits
  – Performs in three steps: DCT, Quantization, VL coding

• Preprocessing:
  – Level shifting \(2^{k-1}\), k bit depth

• Coding:
  – Different VLS coding for nonzero AC and DC components.
### Image Compression

- Sample subimages (8*8)

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>55</td>
<td>61</td>
<td>66</td>
<td>70</td>
<td>61</td>
<td>64</td>
<td>73</td>
</tr>
<tr>
<td>63</td>
<td>59</td>
<td>66</td>
<td>90</td>
<td>109</td>
<td>85</td>
<td>69</td>
<td>72</td>
</tr>
<tr>
<td>62</td>
<td>59</td>
<td>68</td>
<td>113</td>
<td>144</td>
<td>104</td>
<td>66</td>
<td>73</td>
</tr>
<tr>
<td>63</td>
<td>58</td>
<td>71</td>
<td>122</td>
<td>154</td>
<td>106</td>
<td>70</td>
<td>69</td>
</tr>
<tr>
<td>67</td>
<td>61</td>
<td>68</td>
<td>104</td>
<td>126</td>
<td>88</td>
<td>68</td>
<td>70</td>
</tr>
<tr>
<td>79</td>
<td>65</td>
<td>60</td>
<td>70</td>
<td>77</td>
<td>63</td>
<td>58</td>
<td>75</td>
</tr>
<tr>
<td>85</td>
<td>71</td>
<td>64</td>
<td>59</td>
<td>55</td>
<td>61</td>
<td>65</td>
<td>83</td>
</tr>
<tr>
<td>87</td>
<td>79</td>
<td>69</td>
<td>68</td>
<td>65</td>
<td>76</td>
<td>78</td>
<td>94</td>
</tr>
</tbody>
</table>
### Image Compression

- **Level shifting (-128)**

-65  -70  -57  -6  26  -22  -58  -59  
-61  -67  -60  -24  -2  -40  -60  -58  
-43  -57  -64  -69  -73  -67  -63  -45  
-41  -49  -59  -60  -63  -52  -50  -34
Image Compression

• Forward DCT Transform:

\[
\begin{array}{cccccccc}
-415 & -29 & -62 & 25 & 55 & -20 & -1 & 3 \\
7 & -21 & -62 & 9 & 11 & -7 & -6 & 6 \\
-46 & 8 & 77 & -25 & -30 & 10 & 7 & -5 \\
-50 & 13 & 35 & -15 & -9 & 6 & 0 & 3 \\
11 & -8 & -13 & -2 & -1 & 1 & -4 & 1 \\
-10 & 1 & 3 & -3 & -1 & 0 & 2 & -1 \\
-4 & -1 & 2 & -1 & 2 & -3 & 1 & -2 \\
-1 & -1 & -1 & -2 & -1 & -1 & 0 & -1 \\
\end{array}
\]
Image Compression

**Normalized:**

\[
\begin{bmatrix}
-26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 & 0 \\
1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 & 0 \\
-4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

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

\[
\hat{T}(0,0) = -26
\]
Zig-Zag Scanning:

$[-26 -31 -3 -2 -62 -41 -41 15 02 00 -12 00 00 0 -1 -1 EOB]$

Two different Coding for DC and AC
- See Tables A-3, A-4, A-5 for codebook

1010110 0100 001 0100 0101 100001 0110 100011 001 100011 001
001 100101 11100110 110110 0110 11110100 000 1010
• Quantized Coefficients:

\[
\begin{array}{cccccccc}
-26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\
1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]


**Digital Image Processing**

**Image Compression**

- **Denormalization:**

\[
\begin{array}{cccccccc}
-416 & -33 & -60 & 32 & 48 & 0 & 0 & 0 \\
12 & -24 & -56 & 0 & 0 & 0 & 0 & 0 \\
-42 & 13 & 80 & -24 & -40 & 0 & 0 & 0 \\
-56 & 17 & 44 & -29 & 0 & 0 & 0 & 0 \\
18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\hat{T}^{\ast}(u,v) = Z(u,v)\hat{T}(u,v)
\]

\[
\hat{T}^{\ast}(0,0) = -416
\]

\[
T(0,0) = -415
\]
**Inverse DCT Transform:**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-70</td>
<td>-64</td>
<td>-61</td>
<td>-64</td>
<td>-69</td>
<td>-66</td>
<td>-58</td>
<td>-50</td>
</tr>
<tr>
<td>-72</td>
<td>-73</td>
<td>-61</td>
<td>-39</td>
<td>-30</td>
<td>-40</td>
<td>-54</td>
<td>-59</td>
</tr>
<tr>
<td>-68</td>
<td>-78</td>
<td>-58</td>
<td>-9</td>
<td>13</td>
<td>-12</td>
<td>-48</td>
<td>-64</td>
</tr>
<tr>
<td>-59</td>
<td>-77</td>
<td>-57</td>
<td>0</td>
<td>22</td>
<td>-13</td>
<td>-51</td>
<td>-60</td>
</tr>
<tr>
<td>-54</td>
<td>-75</td>
<td>-64</td>
<td>-23</td>
<td>-13</td>
<td>-44</td>
<td>-63</td>
<td>-56</td>
</tr>
<tr>
<td>-52</td>
<td>-71</td>
<td>-72</td>
<td>-54</td>
<td>-54</td>
<td>-71</td>
<td>-71</td>
<td>-54</td>
</tr>
<tr>
<td>-35</td>
<td>-47</td>
<td>-61</td>
<td>-66</td>
<td>-60</td>
<td>-48</td>
<td>-44</td>
<td>-44</td>
</tr>
</tbody>
</table>
Image Compression

• Level Shifting (+128):

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>64</td>
<td>67</td>
<td>64</td>
<td>59</td>
<td>62</td>
<td>70</td>
<td>78</td>
</tr>
<tr>
<td>56</td>
<td>55</td>
<td>67</td>
<td>89</td>
<td>98</td>
<td>88</td>
<td>74</td>
<td>69</td>
</tr>
<tr>
<td>60</td>
<td>50</td>
<td>70</td>
<td>119</td>
<td>141</td>
<td>116</td>
<td>80</td>
<td>64</td>
</tr>
<tr>
<td>69</td>
<td>51</td>
<td>71</td>
<td>128</td>
<td>149</td>
<td>115</td>
<td>77</td>
<td>68</td>
</tr>
<tr>
<td>74</td>
<td>53</td>
<td>64</td>
<td>105</td>
<td>115</td>
<td>84</td>
<td>65</td>
<td>72</td>
</tr>
<tr>
<td>76</td>
<td>57</td>
<td>56</td>
<td>74</td>
<td>75</td>
<td>57</td>
<td>57</td>
<td>74</td>
</tr>
<tr>
<td>83</td>
<td>69</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>61</td>
<td>67</td>
<td>78</td>
</tr>
<tr>
<td>93</td>
<td>81</td>
<td>67</td>
<td>62</td>
<td>69</td>
<td>80</td>
<td>84</td>
<td>84</td>
</tr>
</tbody>
</table>
### Difference Image:

\[
\begin{array}{cccccccc}
-6 & -9 & -6 & 2 & 11 & -1 & -6 & -5 \\
7 & 4 & -1 & 1 & 11 & -3 & -5 & 3 \\
2 & 9 & -2 & -6 & -3 & -12 & -14 & 9 \\
-6 & 7 & 0 & -4 & -5 & -9 & -7 & 1 \\
-7 & 8 & 4 & -1 & 6 & 4 & 3 & -2 \\
3 & 8 & 4 & -4 & 2 & 6 & 1 & 1 \\
2 & 2 & 5 & -1 & -6 & 0 & -2 & 5 \\
-6 & -2 & 2 & 6 & -4 & -4 & -6 & 10 \\
\end{array}
\]
Digital Image Processing

Image Compression

• Example:

1:25

1:52
Image Compression

- JPEG (Joint Photographic Expert Group)
  - Color Models
  - Image Subsampling
  - Coding (DC and AC)
Image Compression

- YUV Color model:

\[
U = B' - Y' \quad V = R' - Y' \\
\begin{bmatrix}
Y' \\
U \\
V
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
-0.299 & -0.587 & 0.886 \\
0.701 & -0.587 & -0.114
\end{bmatrix}
\begin{bmatrix}
R' \\
G' \\
B'
\end{bmatrix}
\]

Original color image

\( Y' \)

\( U \)

\( V \)
Digital Image Processing

Image Compression

- YIQ Color Model:

\[
I = 0.492111(R' - Y') \cos 33^\circ - 0.877283(B' - Y') \sin 33^\circ \\
Q = 0.492111(R' - Y') \sin 33^\circ + 0.877283(B' - Y') \cos 33^\circ \\
\begin{bmatrix} Y' \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}
\]

Original color image

\(Y'\)

\(I\)

\(Q\)
Digital Image Processing

Image Compression

- Chromatic components subsampling:
Image Compression

- Chrominance Decimation of eye:

- 4:4:4
- 4:2:2
- 4:1:1
- 4:2:0

- Pixel with only Y value
- Pixel with only U and V values
- Pixel with Y, U, and V values
Digital Image Processing

Image Compression

- Compression Block Diagram

![Compression Block Diagram](image-url)
Image Compression

- Decompression Block Diagram
Digital Image Processing

Image Compression

- **DCT on Image block**
  - Each image is divided into 8×8 blocks. The 2D DCT is applied to each block image $f(i, j)$, with output being the DCT coefficients $F(u,v)$ for each block.
Image Compression

• Quantization:
  
  – $F(u,v)$ represents a DCT coefficient, $Q(u,v)$ is a “quantization matrix” entry, and $F(u,v)$ represents the quantized DCT coefficients which JPEG will use in the succeeding entropy coding.

  • The quantization step is the main source for loss in JPEG compression.

  • The entries of $Q(u,v)$ tend to have larger values towards the lower right corner. This aims to introduce more loss at the higher spatial frequencies
Image Compression

- Quantization Table:
  - Default $Q(u,v)$ values obtained from psychophysical studies with the goal of maximizing the compression ratio while minimizing perceptual losses in JPEG images.

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

- Luminance Quantization Table

\[
\hat{F}(u,v) = \text{round} \left( \frac{F(u,v)}{Q(u,v)} \right)
\]

- Chrominance Quantization Table
Digital Image Processing

Image Compression

- DCT + Quantization:

Encoder

Decoder
• DPCM: Differential Pulse Code Modulation
  – Used for DC components compression:
    • DC coefficient is unlikely to change drastically within a short distance so we expect DPCM codes to have small magnitude and variance.

\[
\begin{align*}
150 & \\
155 & \\
149 & \\
152 & \\
144 & \Rightarrow 150 \\
5 & \\
-6 & \\
3 & \\
-8 & 
\end{align*}
\]

– Entropy coding of DC components:
  • Huffman
Image Compression

• RLC: Run Length Coding
  – Used for AC components compression:
    • RLC is effective if the information source has the property that symbols tend to form continuous groups. In this case, the quantized DCT coefficients tend to form continuous groups of 0s.
  – Used in Conjunction with zig-zag scan
  – Entropy coding of DC components:
    • Huffman
Image Compression

• RLC: Run Length Coding

\[ \hat{F}(u, v) \]

\[
\begin{array}{cccccccc}
32 & 6 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Lossless Predictive Coding:

- Code only *new information* in each pixel.
- *New Information*: Difference between two successive pixel.

\[ f_n : \text{New incoming pixel value.} \]
\[ \hat{f}_n : \text{Prediction of } f_n \text{ based on a limited range history} \]

- \( e_n \) is coded with a variable-length coding (symbol Encoder)

\[ e_n = f_n - \hat{f}_n \]
• Prediction:
  – An optimal (local/global) and adaptive predictor:
    \[
    \hat{f}_n = \text{round} \left( \sum_{i=1}^{m} \alpha_i f(n - i) \right)
    \rightarrow f(x, y) = \text{round} \left( \sum_{i=1}^{m} \alpha_i f(x, y - i) \right)
    \]
  – A nearest integer operator is needed
Image Compression

• Coder and Decoder:

Input sequence → $f(n)$ → $e(n)$ → Symbol encoder → Compressed sequence

Predictor → Nearest integer → $f(n)$

Compressed sequence → $e(n)$ → $f(n)$ → Decompressed sequence

Symbol decoder → $f(n)$ → $e(n)$ → Predictor
Image Compression

• Lossless Predictive Coding:
  – For 2D images:
  \[ f^\hat{}(x, y) = \text{round}\left(\sum_{i=-m}^{m} \sum_{j=-m}^{m} \alpha_i f(x - i, y - j)\right) \]
  – Example:
  \[ \hat{f}_n = \text{round}(\alpha f(x, y - 1)) \]
  • Zero difference is code with 128
  • Negative/Positive error are coded with dark light region.
Digital Image Processing

Image Compression

- Example:
  - Prediction residual
  - $\alpha = 1$
  - $e(x, y) = f(x, y) - \hat{f}(x, y)$
  - CR = $8/3.99 = 2$
  - Residual PDF

$$P_e(e) = \frac{1}{\sqrt{2\pi}\sigma_e} e^{-\frac{|e|}{\sigma_e}}$$
Digital Image Processing

Image Compression

• Example (Temporal Redundancy):

\[
\hat{f}(x, y, t) = \text{round}\left[\alpha f(x, y, t - 1)\right], \quad \alpha = 1
\]

\[
e(x, y, t) = f(x, y, t) - \hat{f}(x, y, t - 1)
\]

E. Fatemizadeh, Sharif University of Technology, 2011
Image Compression

- Motion Compensated Prediction Residuals
  - Self study (589-596)
• Lossy Compression:
  – JPEG, JPEG2000, GIF
  – 1:10 to 1:50
Image Compression

- **Lossy Predictive Coding:**
  - Map prediction error into a limited range of outputs (control compression ratio)

\[
\hat{f}_n = \hat{e}_n + \hat{f}_n
\]
**Delta Modulation:**

\[ \hat{f}_n = \alpha \hat{f}_{n-1} \quad \alpha < 1 \]

\[ \hat{e}_n = \begin{cases} 
+\xi & e_n > 0 \\
-\xi & \text{O.W.}
\end{cases} \]

Singel bit code (1bit/pixel)

---

<table>
<thead>
<tr>
<th>Input</th>
<th>Encoder</th>
<th>Decoder</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( f )</td>
<td>( \hat{f} )</td>
<td>( e )</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>14.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>20.5</td>
<td>-6.5</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>14.0</td>
<td>1.0</td>
</tr>
<tr>
<td>14</td>
<td>29</td>
<td>20.5</td>
<td>8.5</td>
</tr>
<tr>
<td>15</td>
<td>37</td>
<td>27.0</td>
<td>10.0</td>
</tr>
<tr>
<td>16</td>
<td>47</td>
<td>33.5</td>
<td>13.5</td>
</tr>
<tr>
<td>17</td>
<td>62</td>
<td>40.0</td>
<td>22.0</td>
</tr>
<tr>
<td>18</td>
<td>75</td>
<td>46.5</td>
<td>28.5</td>
</tr>
<tr>
<td>19</td>
<td>77</td>
<td>53.0</td>
<td>24.0</td>
</tr>
</tbody>
</table>
**Optimal Prediction:**

\[ E\{e_n^2\} = E\left\{\left| f_n - \hat{f}_n \right|^2 \right\}, \quad \hat{f}_n = \hat{e}_n + \hat{f}_n \approx e_n + \hat{f}_n = f_n \]

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

\[ E\{e_n^2\} = E\left\{f_n - \sum_{i=1}^{m} \alpha_i f_{n-i} \right\}^2 \Rightarrow \alpha = R^{-1}r \]

**R**: Image Autocorrelation Matrix, \( R(i, j) = E\{f_{n-i}f_{n-j}\}_{i,j=1} \)

\[ r(k) = E\{f_n f_{n-k}\}_{k=1}^m \Rightarrow \sigma_e^2 = \sigma^2 - \sum_{i=1}^{m} \alpha_i E\{f_n f_{n-i}\} \]
Image Compression

- Optimal Prediction:
  - Computation for image by image is hard.
  - Global coefficient is computed by assuming a simple image model.

\[
E\{f(x, y)f(x-i, y-i)\} = \sigma^2 \rho_v^i \rho_h^j
\]

\[
\hat{f}(x, y) = \alpha_1 f(x, y-1) + \alpha_2 f(x-1, y-1) + \alpha_3 f(x-1, y) + \alpha_4 f(x-1, y+1)
\]

\[
\alpha_1 = \rho_h, \quad \alpha_2 = -\rho_h \rho_v, \quad \alpha_3 = \rho_v, \quad \alpha_4 = 0
\]

\[
\rho_h \text{ and } \rho_v : \text{Horizontal and Vertical correlation coefficient}
\]

Dynamic Range Const.: \[
\sum_{i=1}^{m} \alpha_i \leq 1
\]
Results of Prediction Error Coding:

\[ \hat{f}(x, y) = 0.97 f(x, y - 1) \]

\[ \hat{f}(x, y) = 0.5 f(x, y - 1) + 0.5 f(x - 1, y) \]

\[ \hat{f}(x, y) = 0.75 f(x, y - 1) + 0.75 f(x - 1, y) - 0.5 f(x - 1, y - 1) \]

\[ \hat{f}(x, y) = \begin{cases} 0.97 f(x, y - 1) & \Delta h \leq \Delta v \\ 0.97 f(x - 1, y) & \text{O.W.} \end{cases} \]

\[ \Delta h = |f(x - 1, y) - f(x - 1, y - 1)| \]

\[ \Delta v = |f(x, y - 1) - f(x - 1, y - 1)| \]
Image Compression

• Example:
  – Prediction Errors

\[
\text{rms} = 11.1 \quad \text{rms} = 9.8
\]

\[
\text{rms} = 9.1 \quad \text{rms} = 9.7
\]
• Optimal Quantizer:
  – \( t = q(s) \)
  – \( q(s) = -q(-s) \)
  – Map \( s \) in \((s_i, s_{i+1}]\) to \( t_i \)
  – What is best \( s_i \) and \( t_i \)
**Lloyd-Max Quantizer**

- For even $p(s)$, $E\{(s_i-t_i)^2\}$ will minimize with these conditions:

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

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

$$
s_{-i} = -s_i, \quad t_{-i} = -t_i
$$
• **Lloyd-Max Quantizer**
  
  – $L=2, 4, 8$
  
  – Laplacian pdf ($\sigma=1$)
  
  – $t_i-t_{i-1} = s_i-s_{i-1} = \theta$

<table>
<thead>
<tr>
<th>Levels</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$s_i$</td>
<td>$t_i$</td>
<td>$s_i$</td>
</tr>
<tr>
<td>1</td>
<td>$\infty$</td>
<td>0.707</td>
<td>1.102</td>
</tr>
<tr>
<td>2</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>1.810</td>
</tr>
<tr>
<td>3</td>
<td>$\infty$</td>
<td>2.285</td>
<td>1.576</td>
</tr>
<tr>
<td>4</td>
<td>$\infty$</td>
<td>2.994</td>
<td></td>
</tr>
<tr>
<td>$\theta$</td>
<td>1.414</td>
<td>1.087</td>
<td>0.731</td>
</tr>
</tbody>
</table>
Image Compression

- Wavelet Coding

![Diagram of wavelet coding process](image.png)
Image Compression

- Wavelet Selection:

<table>
<thead>
<tr>
<th>Wavelet</th>
<th>Haar</th>
<th>Daubechies</th>
<th>Symlet</th>
<th>Cohen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image" alt="Haar" /></td>
<td><img src="image" alt="Daubechies" /></td>
<td><img src="image" alt="Symlet" /></td>
<td><img src="image" alt="Cohen" /></td>
</tr>
</tbody>
</table>
Wavelet Selection:

- Truncating the transforms below 1.5:

<table>
<thead>
<tr>
<th>Wavelet</th>
<th>Filter Taps (Scaling + Wavelet)</th>
<th>Zeroed Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haar (see Ex. 7.10)</td>
<td>2 + 2</td>
<td>33.8%</td>
</tr>
<tr>
<td>Daubechies (see Fig. 7.8)</td>
<td>8 + 8</td>
<td>40.9%</td>
</tr>
<tr>
<td>Symlet (see Fig. 7.26)</td>
<td>8 + 8</td>
<td>41.2%</td>
</tr>
<tr>
<td>Biorthogonal (see Fig. 7.39)</td>
<td>17 + 11</td>
<td>42.1%</td>
</tr>
</tbody>
</table>
Image Compression

• Decomposition Level Selection:
  – More scales \( \rightarrow \) More computational efforts

– An Experiments:
  • Biorthogonal wavelet;
  • Fixed global threshold (25);
  • Truncate only details

<table>
<thead>
<tr>
<th>Decomposition Level (Scales or Filter Bank Iterations)</th>
<th>Approximation Coefficient Image</th>
<th>Truncated Coefficients (%)</th>
<th>Reconstruction Error (rms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(256 \times 256)</td>
<td>74.7%</td>
<td>3.27</td>
</tr>
<tr>
<td>2</td>
<td>(128 \times 128)</td>
<td>91.7%</td>
<td>4.23</td>
</tr>
<tr>
<td>3</td>
<td>(64 \times 64)</td>
<td>95.1%</td>
<td>4.54</td>
</tr>
<tr>
<td>4</td>
<td>(32 \times 32)</td>
<td>95.6%</td>
<td>4.61</td>
</tr>
<tr>
<td>5</td>
<td>(16 \times 16)</td>
<td>95.5%</td>
<td>4.63</td>
</tr>
</tbody>
</table>
Image Compression

- Jpeg 2000 (607-614)
  - DC Level shift
  - Convert using color video transform (RGB $\rightarrow$ YC$_b$C$_r$)
  - Convert the whole image to tile components
  - Using different filter for lossy or loss-free compression
  - Quantized the coefficients
  - Arithmetic Coding
Digital Image Processing

Image Compression

• Example:
  – C=25, 52, 75, 105.
Digital Image Processing

Image Compression

- Image Watermarking
  - Skipped
Digital Image Processing

Image Compression

• Matlab Command
  – Huffmandeco, huffmandict, huffmanenco
  – Arithdeco, arithenco
  – Dpcmdeco, dpcmenco
  – Lloyds, quantiz
  – wcompress (wavelet image compression)