Image Compression

Image Data

- Inherently large
  - $MN\times$bytes_per_pixel

- Consider this
  - 30 frames per second video
    - 640x480 (R-G-B-A) frame
      - 4 bytes per pixel equals ~1MB per frame
    - 30MB for 1 second
    - Over 1GB for 1 minute
Image Compression

• We need a way to “compress” image data

• Conserve bandwidth
  - For backing store (disk drive, CD-ROM)
  - Network transmission

Data Compression

• Two categories
  - Information Preserving
    • Error free compression
    • Original data can be recovered completely
  - Lossy
    • Original data is approximated
    • Less than perfect
    • Generally allows much higher compression
Basics

• Data Compression
  - Process of reducing the amount of data required to represent a given quantity of information

• Data vs. Information
  - Data and Information are not the same thing

Basics

• Data vs. Information
  - Data
    • the means by which information is conveyed
    • various amounts of data can convey the same information
  - Information
    • “A signal that contains no uncertainty”
Example of data vs. information

Dataset 1

NxN\times1\text{byte} \\
(pixel data)

Dataset 2

NxN image \\
circle(0,0,r)

24\times1\text{byte} \\
(24\text{ ascii characters} \\
to\text{ store the above text})

Redundancy

• Redundancy
  - “data” that provides no relevant information
  - “data” that restates what is already known
• For example
  - say N1 and N2 denote the number of “data units” in two sets that represent the same information
    - Relative redundancy of the first set of data N1 is
      \[ R_D = 1 - \frac{1}{C_r} \]
      where \( C_r \) is the “Compression Ratio”
      \[ C_r = \frac{n1}{n2} \]
Redundancy

- Relative Redundancy of N1 to N2
  - $R_D = 1 - 1/C_r$
- Compression Ratio
  - $C_r = n_1 / n_2$

- if $(n_2 = n_1)$
  - $C_r = 1$, $R_D = 0$
    - Compression Ratio is 1, implying there is no redundancy
- if $(n_2 << n_1)$
  - Highly redundant data
  - N2 is much more “compact” than N1
  - i.e., high compression ratio

Example

- $N_1 = 10$ & $N_2 = 1$

- Compression Ratio
  - $C_R = N_1/N_2 = 10$ (or 10:1)
- $R_D = 1 - (1/10)$ or 0.9
- Implying 90% of the data in N1 is redundant
Compression

- Given a set of $N$ data units
- Try to remove redundant data that is not needed to express the underlying information
- Hopefully your new set, $N' \ll N$

Image Compression

- Three basic types of redundancies
  - Coding Redundancy
  - Interpixel Redundancy
  - Psycho-visual Redundancy
- Image compression
  - Reduce one or more of these redundancies
Coding Redundancy

• Recall the Histogram
  \( p_r(r_k) = \frac{n_k}{n} \quad k = 0, 1, 2, \ldots, L-1 \)
  \( L \) is the number of gray levels
  \( n_k \) is the number of pixels with gray level \( r_k \)
  \( n \) is the total number of pixels
  \( p_r(r_k) \) is the probability that \( r_k \) will occur in this image

• Let \( L(r_k) \) = the number of bits needed to represent the gray level \( r_k \)

Coding Redundancy

• Let \( L(r_k) \) = the number of bits needed to represent the gray level \( r_k \)

• The average number of bits needed to represented each pixel is
  \( L_{avg} = \sum L(r_k) p_r(r_k) \)

• Thus, the total number of bits needed to code an image
  \( N*M*L_{avg} \)
Coding Redundancy

- **Example**
  - Consider an image with 8 gray levels
  - \( L_1(r_k) \) is a natural binary coding set
    - \( m \)-bit counting sequence i.e. \( 2^m \)
    - thus, to encode 8 distinct codes needs \( m = 3 \)
  - Each pixel uses 3 bits
    - \( \text{Lavg} = 3 \text{bits} \)

- **Coding Redundancy**
  - However, consider using the codes in this table

<table>
<thead>
<tr>
<th>( r_k )</th>
<th>( p(r_k) )</th>
<th>Code 1</th>
<th>( l(r_k) )</th>
<th>Code 2</th>
<th>( l(r_k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.19</td>
<td>000</td>
<td>3</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>1/7</td>
<td>0.25</td>
<td>001</td>
<td>3</td>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>2/7</td>
<td>0.11</td>
<td>010</td>
<td>3</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>3/7</td>
<td>0.16</td>
<td>011</td>
<td>3</td>
<td>0101</td>
<td>3</td>
</tr>
<tr>
<td>4/7</td>
<td>0.08</td>
<td>100</td>
<td>3</td>
<td>0001</td>
<td>3</td>
</tr>
<tr>
<td>5/7</td>
<td>0.06</td>
<td>101</td>
<td>3</td>
<td>00001</td>
<td>5</td>
</tr>
<tr>
<td>6/7</td>
<td>0.03</td>
<td>110</td>
<td>3</td>
<td>000001</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0.02</td>
<td>111</td>
<td>3</td>
<td>000000</td>
<td>6</td>
</tr>
</tbody>
</table>

- \( \text{Lavg} = \sum l_2(r_k)p(r_k) = 2(0.19) + 2(0.25) + 2(0.21) + 3(0.16) + 4(0.08) + 5(0.06) + 6(0.03) + 6(0.02) \)

\( \text{Lavg} = 2.7 \text{ bits} \)
Coding Redundancy

- $L_{1\text{avg}} = 3$ bits
- $L_{2\text{avg}} = 2.7$ bits

- $Cr = \frac{L_{1\text{avg}}}{L_{2\text{avg}}} = \frac{3}{2.7}$ or 1.1
- $Rd = 1 - \frac{1}{1.11} = 0.099$
- Thus approximately 10 percent of the data in $L_1$ is redundant

"Variable Length Coding"

- Assigning fewer bits to more probable symbols achieves compression
- This process is commonly referred to as "variable-length-coding"

- What is the pitfall?
  - Equally probable symbols
    - Generally no gain . .
    - Then don't encode . .
  - Need to know how to choose the "symbol"
Variable Length Coding and Compression

- If gray-levels of an image are coded using more symbols than absolutely necessary
- It is said to contain coding redundancy
- Variable length coding is very powerful and is frequently applied
  - Not just for images
  - Characters strings too
    - Gzip, Zip, etc use a form of variable length coding

Interpixel Redundancy

- Think of an average binary image
- The value of a pixel can be reasonably predicted from its neighbors
- Sometimes called
  - Spatial Redundancy
  - Geometric Redundancy
  - Interframe Redundancy
  - Well call it "interpixel" redundancy
Interpixel Redundancy

- In order to reduce redundancy in 2D pixel arrays
- We can convert them into more efficient, “non-visual” format
- Transforms of this type are called “mappings”
  - they are reversible if the original image elements can be reconstructed from the transformed data sets

Run-length Encoding

- Each scan line, \( y \), of an binary image \( f(:,y) \) is encoded as a sequence of 0 and 1s
  \[ f(0,y) f(1,y) f(2,y) \ldots f(n-1, y) \]

- An alternative
  - encode as a pair of sequences
  - \((g_1,r_1)(g_2,r_2)\ldots(g_i,ri)\ldots(g_n,rn)\)
  - where \( g_1 \) encodes the gray-level (0 or 1)
  - and \( ri \) encodes the "run" length in pixels
Example

• The entire image in previous example
  - 1024x343 pixels (1 bit)
  - required only 12,166 runs
    • each run requires 11 bits to encode
  - N1 = 1024x343 (the binary image)
  - N2 = 12,166 * 11 (run length encoded image)

• Cr = (1024)(343)/(12,166*11) = 2.63

• Rd = 1 - 1/2.63 = 0.62
  - N1 (the binary image) is roughly 62% redundant
Psychovisual Redundancy

• “Brightness” as perceived by the eye is dependent on many things

• The fact is, the eye does not respond with equal sensitivity to all visual information

• Some information has less relative importance than other information

• This information is said to be psychovisually redundant
  - It can be eliminated without significantly impairing visual “quality”

Psychovisual Redundancy

• Unlike Coding and Interpixel Redundancy
  - Psychovisual is associated with real or quantifiable visual information

  - It is possible only because the elimination of this data does not effect the visual process
    • But we lose quantifiable information

  - Thus, this type of reduction is called “quantification”
    • And it is lossy
Lots of examples

• Improved Gray-Scale Quantization (IGS)
  • Maps 8 bits to 4 bits
  • Adds some random noise to pixels before quantization
  • This noise helps removes our sensitivity to edges created by quantization

Original  Straight  IGS
Quantization  Quantization

Lots of examples

• Color Quantization is very common
• MPEG and (digital video) DV Standards
  - Terms like
  - 4:2:2
    • For every 4 intensity pixels
      - only 2 color pixels are stored (in X and Y direction)
  - 4:1:1
    • For every 4 intensity pixels
      - only 1 color pixels are stored (in X and Y direction)
**Temporal Interlacing**

- TV signal is interlaced
- 1 frame is composed of "2 fields"
  - one with odd scanlines
  - one with even scanlines
- Frame rate is 29.9 per second (NTSC)
  - 24 per second (PAL)
  - But ~60 fields per second are displayed
- Exploits temporal and spatial coherency
- progressive scan
  - Full frame

---

**Interlacing**

```
<table>
<thead>
<tr>
<th>Complete frame</th>
<th>Odd Field</th>
<th>Even Field</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Odd (De-interlacing)   Even
Fidelity Criteria

- Psychovisual data reduction results in a loss of quantitative visual information
- Because information of interest may be lost
  - a repeatable or reproducible means of quantifying the nature and extent of information loss is desirable

Fidelity Criteria

- Two classes of criteria are used
  - Objective fidelity criteria
  - Subjective fidelity criteria

- Objective
  - loss can be expressed as a function of
    - the input image and
    - the resulting compressed (then decompressed) image
Objective Fidelity Criteria

• Let \( f(x,y) \) be the input image
• Let \( f'(x,y) \) be the "approximated" \( f(x,y) \) from compression/decompression

• For any value of \( x \) and \( y \), the error \( e(x,y) \) between \( f(x,y) \) and \( f'(x,y) \) is
  \[- e(x,y) = f'(x,y) - f(x,y) \]

Objective Fidelity Criteria

• The total error between the two images
  \[ e = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f'(x,y) - f(x,y)] \]

• The Root-Mean-Square Error
  \[ e_{\text{rms}} = \frac{1}{MN} \left[ \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f'(x,y) - f(x,y)]^2 \right]^{1/2} \]
Objective Fidelity Criteria

- Commonly used fidelity criteria is
  - mean-square signal-to-noise ratio defined as:

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

  (Mean squared error)

Subjective Fidelity Criteria

- Human-based criteria
- Usually side-by-side comparison

<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>
Image Compression Models

f(x,y) → Source encoder → Channel encoder → Channel → Channel decoder → Source decoder → f(x,y)

f(x,y) → Mapper → Quantizer → Symbol Encoder → Inverse Mapper → f(x,y)

Error-Free Image Compression (Lossless Compression)
Error-Free Compression

- To reduce only coding redundancy
- Most popular technique is
  - Huffman Encoding
    - This is a variable-length coding technique

  - Huffman encoding yields the smallest possible number of code "symbols" per source symbol
    - Optimal for a fixed number of source symbols
    - subject that source symbols are coded one at a time

Huffman Encoding

- Huffman encoding
- Perform a series of source reductions
  - Ordering the probabilities of the symbols occurrence

  - combine the lowest probable symbols into a single symbol that replaces them in the next reduction

  - assigns codes based on these reduction
    - This is “variable-length” coding
    - So, we will have to build variable-length codes
Huffman Encoding (Reduction)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Original Source</th>
<th>Source Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>a2</td>
<td>0.4</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>a6</td>
<td>0.3</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>a1</td>
<td>0.1</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>a4</td>
<td>0.1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>a3</td>
<td>0.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a5</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Huffman Encoding

(Reduction)

<table>
<thead>
<tr>
<th>Original Source</th>
<th>Source Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td>Probability</td>
</tr>
<tr>
<td>a2</td>
<td>0.4</td>
</tr>
<tr>
<td>a6</td>
<td>0.3</td>
</tr>
<tr>
<td>a1</td>
<td>0.1</td>
</tr>
<tr>
<td>a4</td>
<td>0.1</td>
</tr>
<tr>
<td>a3</td>
<td>0.06</td>
</tr>
<tr>
<td>a5</td>
<td>0.04</td>
</tr>
</tbody>
</table>
## Huffman Encoding (Reduction)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Source Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>a2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>a6</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>a1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>a4</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>a3</td>
<td>0.06</td>
<td>0.1</td>
</tr>
<tr>
<td>a5</td>
<td>0.04</td>
<td></td>
</tr>
</tbody>
</table>

## Code assignment

<table>
<thead>
<tr>
<th>Sym</th>
<th>Prob</th>
<th>Code</th>
<th>Source Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>a2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>a6</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>a1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>a4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>a3</td>
<td>0.06</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>a5</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Code assignment

<table>
<thead>
<tr>
<th>Original Source</th>
<th>Source Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sym</strong></td>
<td><strong>Prob</strong></td>
</tr>
<tr>
<td>a2 0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>a6 0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>a1 0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>a4 0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>a3 0.06</td>
<td>0.1</td>
</tr>
<tr>
<td>a5 0.04</td>
<td></td>
</tr>
</tbody>
</table>

### Code assignment

<table>
<thead>
<tr>
<th>Original Source</th>
<th>Source Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sym</strong></td>
<td><strong>Prob</strong></td>
</tr>
<tr>
<td>a2 0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>a6 0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>a1 0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>a4 0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>a3 0.06</td>
<td>0.1</td>
</tr>
<tr>
<td>a5 0.04</td>
<td></td>
</tr>
</tbody>
</table>
## Code assignment

<table>
<thead>
<tr>
<th>Sym</th>
<th>Prob</th>
<th>Code</th>
<th>Source Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>a2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4 1 0.4 1 0.4 1 0.6 0</td>
</tr>
<tr>
<td>a6</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3 00 0.3 00 0.3 00 0.4 1</td>
</tr>
<tr>
<td>a1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1 011 0.2 010 0.3 01</td>
</tr>
<tr>
<td>a4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1 0100 0.1 011</td>
</tr>
<tr>
<td>a3</td>
<td>0.06</td>
<td>0.1</td>
<td>0.1 0101</td>
</tr>
<tr>
<td>a5</td>
<td>0.04</td>
<td>0.1</td>
<td>0.1 0101</td>
</tr>
</tbody>
</table>
Huffman Encoding

• Uniquely Decodable

• consider 010100111100

\[ a_3 \quad a_1 \quad a_2 \quad a_2 \quad a_6 \]

Symbol Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_2</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>a_6</td>
<td>0.3</td>
<td>00</td>
</tr>
<tr>
<td>a_1</td>
<td>0.1</td>
<td>011</td>
</tr>
<tr>
<td>a_4</td>
<td>0.1</td>
<td>0100</td>
</tr>
<tr>
<td>a_3</td>
<td>0.06</td>
<td>01010</td>
</tr>
<tr>
<td>a_5</td>
<td>0.04</td>
<td>01011</td>
</tr>
</tbody>
</table>

Huffman Encoding

• Very powerful encoding scheme
• Produces the optimal codes
  - Given an input symbol set
  - And its known distribution

• Need to store symbol table along with compressed data!