Image Compression

Background

- Reduce the amount of data to represent a digital image
  - Storage and transmission
    - Consider the live streaming of a movie at standard definition video
    - A color frame is $720 \times 480$ pixel, with 24-bits per pixel
    - Number of bits transmitted per frame: 8,294,400
    - At 30 frames per second, this data needs to make it in $1/30s$, giving the transmission speed of 248,832,000 bits per second, or 237Mb per second
    - A two-hour movie has 895,795,200,000 bits or 8.543Gb (close to 1 TB)
  - For HD view, we have to deal with $1980 \times 1080$ pixels, each of 24 bits (1080p)
    - The 4K UHD standard requires $3840 \times 2160$ or 8,294,400 pixels per frame (24bpp)
    - The 8K UHD standard requires $7680 \times 4320$ or 33,177,600 pixels per frame (24bpp)
    - Both 4K and 8K are transmitted at the rate of a minimum of 60fps
  - Residential data speed – 300Mbps (Charter) to 1Gbps (AT&T)
  - Remove redundant data (color map)
  - Possibly organize data (run length encoding)

- Bandwidth compression
  - Use analog methods to reduce video transmission bandwidth
  - Digital methods based on Shannon’s information theory, modeling signal as a probabilistic distribution of data

- Information preserving vs lossy compression

Fundamentals

- Data compression
  - Reduce the amount of data required to represent a given quantity of information
  - Data vs information
    - Data as a means to convey information

- Data redundancy
  - Items that may not be pertinent to information being conveyed
    - Irrelevant or repeated information
  - Central issue in digital image compression
  - Mathematically quantifiable
    - Two data sets to convey the same information in $b$ and $b'$ bits (or information units), respectively
    - Relative data redundancy of first set, denoted by $R$, is defined as
      \[ R = 1 - \frac{1}{C} \]
      
      $C$ is the compression ratio given by $\frac{b}{b'}$
      
      - $C = 10$ (or 10:1) implies that $b$ contains 10 bits of data for every bit of data in $b'$
• Corresponding relative data redundancy of larger representation is 0.9 ($R = 0.9$), indicating that 90% of data is redundant
  * $b = b' \Rightarrow C = 1$ and $R = 0$
  * $b' \ll b \Rightarrow C \to \infty$ and $R = 1$, indicating significant compression and highly redundant data
  * $b' \gg b \Rightarrow C \to 0$ and $R \to -\infty$, indicating second data set contains much more data than the first, leading to data expansion
  * Range of $C$ and $R$ is given by $(0, \infty)$ and $(-\infty, 1)$, respectively
  * Compression ratio specified as $n : 1$, with redundancy given by $\frac{n-1}{n}$

– Redundancy to exploit in image compression (Figure 8.1)

1. Coding redundancy
   * Code: A system of symbols (letters/numbers/bits) used to represent a body of information or a set of events
   * Each piece of information or event is assigned a sequence of code symbols, called a code word
   * Number of symbols in code word is its length
   * 8-bit codes used to represent intensities in images contain more bits than necessary to represent the intensities

2. Spatial and temporal redundancy
   * Spatially correlated pixels – each pixel is generally similar or dependent on neighboring pixel
   * In a video sequence, pixels may be temporally correlated

3. Psychovisual redundancy
   * Information that is ignored by human visual system
   * Almost all images contain such information

• Coding redundancy
  – Use histogram analysis to develop codes to reduce the amount of data
  – Assume gray scale image (applicable to color as well)
  – Discrete random variable $r_k$ in the interval $[0, L - 1]$ used to represent the intensities of an $M \times N$ image
    \[
    p_r(r_k) = \frac{n_k}{MN}, \quad k = 0, 1, 2, \ldots, L - 1
    \]
    * $p_r(r_k)$ is the probability of occurrence for gray level $r_k$
    * $L$ is the number of discrete gray levels
    * $n_k$ is the number of occurrences of gray level $k$
  – Let number of bits required to represent each value of $r_k$ be $l(r_k)$
    * Average number of bits $L_{avg}$ required to represent each pixel is
    \[
    L_{avg} = \sum_{k=0}^{L-1} l(r_k)p_r(r_k)
    \]
    * If the intensities are represented using a natural $m$-bit fixed-length code, $L_{avg}$ reduces to $m$ bits
    * Total number of bits required to code an $M \times N$ image is $MN L_{avg}$
  – Natural/straight binary code: Each event or piece of information (gray-scale) is assigned one of $2^m$ $m$-bit binary codes from an $m$-bit binary counting sequence
  – Representing the gray levels of an image with a natural $m$-bit binary code reduces $L_{avg}$ to $m$ bits
    \[
    L_{avg} = \sum_{k=0}^{L-1} mp_r(r_k)
    \]
    \[
    = m \sum_{k=0}^{L-1} p_r(r_k)
    \]
    \[
    = m
    \]
– Example 8.1
  * Table 8.1: Contains intensity distribution of Figure 8.1a
  * 8-bit binary code used to represent four possible intensities
  * If we use a scheme such as code 2 in Table 8.1
    \[ L_{\text{avg}} = 0.25(2) + 0.47(1) + 0.25(3) + 0.03(3) \]
    \[ = 1.81 \text{bits} \]
  * With code 2, number of bits needed to represent the entire image
    \[ MNL_{\text{avg}} = 256 \times 256 \times 1.81 = 118,621 \text{bits} \]
  * Compression ratio
    \[ C = \frac{256 \times 256 \times 8}{256 \times 256 \times 1.81} \approx 4.42 \]
  * Redundancy
    \[ R = 1 - \frac{1}{4.42} = 0.774 \]
  * Compression comes from assigning fewer bits to the code for more probable intensity values
    * \( l_2(r_{128}) = 1 \)
    * \( l_2(r_{255}) = 3 \)
    * In the best fixed-length code, we have the natural 2-bit counting sequence \{00, 01, 10, 11\} giving the compression of 4:1 instead of 4.42:1 with variable length code

– Example of variable length coding
  * 8 discrete gray levels in an image, with distribution given by

<table>
<thead>
<tr>
<th>( r_k )</th>
<th>( p_r(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>0</td>
<td>0.19</td>
<td>000</td>
<td>3</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0.25</td>
<td>001</td>
<td>3</td>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0.21</td>
<td>010</td>
<td>3</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0.16</td>
<td>011</td>
<td>3</td>
<td>001</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0.08</td>
<td>100</td>
<td>3</td>
<td>0001</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0.06</td>
<td>101</td>
<td>3</td>
<td>00001</td>
<td>5</td>
</tr>
<tr>
<td>6</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>

  * For code 1, \( L_{\text{avg}} \) is 3 bits
  * For code 2, \( L_{\text{avg}} \) is given by
    \[ L_{\text{avg}} = \sum_{k=0}^{7} l_2(r_k)p_r(k) \]
    \[ = 2(0.19) + 2(0.25) + 2(0.21) + 3(0.16) + 4(0.08) + 5(0.06) + 6(0.03) + 6(0.02) \]
    \[ = 2.7 \text{bits} \]
  * Compression ratio \( C_R = \frac{3}{2.7} = 1.11 \)
  * Redundancy \( R_D = 1 - \frac{1}{1.11} = 0.099 \)
  * Assign shortest code words to gray levels that occur most frequently in the image
  * Referred to as variable-length encoding

  – Coding redundancy is the phenomenon when the code does not minimize the average number of bits per pixel
  * Always present when the gray levels are represented by a straight binary code

  • Spatial/temporal redundancy
– Computer-generated collection of constant intensity lines in Figure 8.1b
  1. All 256 intensities are equally probable; uniform histogram
     * Image cannot be compressed by variable length encoding
     * In this case, fixed length code actually minimizes the number of bits needed for the image
  2. Since line intensities are selected randomly, pixels are independent of one another in the vertical direction
  3. Since pixels along each line are identical, they are maximally correlated (completely dependent on one another) in the horizontal direction
     * Significant spatial redundancy that can be minimized by run-length pairs or mappings
     * Run-length pair tells the start of a new intensity and the number of consecutive pixels with that intensity
     * Compresses 2D representation in terms of 8-bit intensity value times 256 columns times 256 rows, or
       \[(256 \times 256 \times 8) / ((256 + 256) \times 8)\] or 128:1
     – For temporal redundancy, we can compute the difference from one frame to next and use that as the code
     – Mapping said to be reversible if the pixels of original array can be reconstructed without error from transformed data; it is irreversible otherwise
• Irrelevant information
  – Remove superfluous data from the set
  – Information ignored by human visual system or extraneous to the intended use of image
  – Figure 8.1c: Homogeneous field of gray, represented by its average intensity alone, or a single byte
     * Original image of 256 \times 256 pixels represented by 1 byte
     * Compression of 65,536:1
  – Figure 8.3a: Histogram of Figure 8.1c
     * Several intensity values close to each other, integrated by human visual system as they are close to each other
  – Histogram equalized version shown in Figure 8.3b
     * Makes intensity changes visible and reveals two constant intensity regions
     * Real information lost by average representation with one byte
     * Removal of such information, that may be ignored by human visual processing system, is called quantization
     * Quantization refers to mapping of a broad range of values to a narrow range
     * Quantization is an irreversible operation because information is lost
• Measuring image information
  – Minimum number of bits needed to represent an image based on information theory
  – Information modeled as a probabilistic process
  – A random event \( E \) with probability \( P(E) \) contains
    \[ I(E) = \log \frac{1}{P(E)} = -\log P(E) \]
    units of information
     * \( P(E) = 1 \Rightarrow I(E) = 0 \)
     * Event always occurs implies that there is no uncertainty associated with the event; no information will be transferred by communicating that the event has occurred
  – The unit to measure information is defined by the logarithmic base in the definition of \( I(E) \) above
     * For base \( m \) logarithm, unit of information is \( m \)-ary
     * For base 2, unit of information is \textit{bit}
       \[ P(E) = \frac{1}{2} \Rightarrow I(E) = -\log_2 \frac{1}{2} = 1 \text{ bit} \]
       * Amount of information when one of two equally likely events occur is 1 bit (one of two outcomes)
Entropy

* Let there be a discrete set of statistically independent random events given by source symbols \( \{a_1, a_2, \ldots, a_J\} \)
* Let the probability associated with each of those events be \( \{P(a_1), P(a_2), \ldots, P(a_J)\} \)
* The average information for this set of events is the entropy, given by

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

* Because they are statistically independent, the source is called a zero-memory source

Consider an image to be the output of an imaginary zero-memory “intensity source”

* The symbol probabilities can be estimated by histogram
* The intensity source’s entropy is given by

\[
\tilde{H} = - \sum_{k=0}^{L-1} p_r(r_k) \log_2(p_r(r_k))
\]

- Use of base 2 logarithm indicates that the average information per intensity output of the image is in bits
- You need at least \( \tilde{H} \) bits/pixel to encode the intensity values of this image

* The entropy of the image in Figure 8.1 is estimated by using the probabilities from Table 8.1

\[
\tilde{H} = -[0.25 \log_2 0.25 + 0.47 \log_2 0.47 + 0.25 \log_2 0.25 + 0.03 \log_2 0.03

= -[0.25(-2) + 0.47(-1.09) + 0.25(-2) + 0.03(-5.06)]

= 1.6614 \text{bpp}
\]

* Entropy of images in Figure 8.1b and 8.1c is computed as 8 bpp and 1.566 bpp

• Shannon’s first theorem

- We computed the variable length code to represent the intensities in image 8.1a using 1.81 bpp
- Entropy estimation for the same gives us 1.6614 bpp
- Shannon’s first theorem, also called noiseless coding theorem assures that the entropy estimation value can be indeed used to represent the image
- Represent groups of \( n \) consecutive source symbols with a single code word
- It shows that

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

where \( L_{\text{avg},n} \) is the average number of code symbols required to represent all \( n \)-symbol groups
- Done by using a hypothetical source that produces \( n \)-symbol blocks using the symbols of original source
- Image is a “sample” of intensity source; a block of \( n \) source symbols corresponds to a group of \( n \) adjacent pixels
- Compute a variable length code for \( n \) pixel blocks using relative frequencies of the blocks
- \( n \)th extension of a hypothetical intensity source with 256 intensity values has 256\(^n\) possible \( n \)-pixel blocks
  * With \( n = 2 \), we have a 65,536 element histogram, and up to 65,536 variable length code words must be generated
- Computational complexity limits the usefulness of the extension coding approach in practice
- The lower limit breaks down when the pixels in the image are correlated
  * Blocks of correlated pixels can be coded with fewer pixels than predicted by this equation
  * When the output of a source depends on a finite number of preceding outputs, the source is called Markov or finite memory source
● Fidelity criteria
  
  – Means to quantify the loss of information, in an objective and subjective manner
  
  1. Objective fidelity criterion
  
  * Information loss can be quantified as a mathematical expression
  * Root-mean-square (RMS) error between two images
  * Let the input image be given by \( f(x, y) \) and the approximation after compression and decompression be given by \( \hat{f}(x, y) \)
  * The error between two images is the difference between them and is quantified by
  \[
  e(x, y) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]
  \]
  * The RMS error between two images is given by
  \[
  e_{\text{rms}} = \sqrt{\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left(\hat{f}(x, y) - f(x, y)\right)^2}
  \]
  * \( \hat{f}(x, y) \) can be considered to be the sum of the original image \( f(x, y) \) and an error \( e(x, y) \)
  * The mean-squared signal-to-noise ratio of the output image is defined by
  \[
  \text{SNR}_{\text{ms}} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x, y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[\hat{f}(x, y) - f(x, y)\right]^2}
  \]
  * The RMS value of SNR, denoted by \( \text{SNR}_{\text{rms}} \) is obtained by taking the square root of \( \text{SNR}_{\text{ms}} \)
  
  2. Subjective fidelity criterion
  
  * Objective fidelity criterion can be used to evaluate information loss
  * Subjective fidelity criterion presents the uncompressed images to a set of humans and averages their evaluation
  * Evaluations can be based on a rating scale, such as a Likert scale
  * Another way to compare is by showing the two images side-by-side and get user perception on a scale (much worse, worse, same, better, much better)
  
  – Figure 8.4
  
  * Three different approximations of the same image
  * Computed RMS errors are: 5.17, 15.67, and 14.17, respectively
  * Table 8.2 – Rating scale of Television Allocations Study Organization
  * Figure 8.4c has broken lines implying loss of crucial information
    · It has far less information compared to Figure 8.2b, yet it is ranked ahead of that
    · Subjective criterion may be more appropriate
  
  – Image compression models
  
  – Figure 8.5
  
  – Two distinct components: encoder and decoder
  – Codec: A device or program that is capable of both encoding and decoding
  – \( \hat{f} \) may or may not be an exact replica of \( f \)
  * Error-free, Lossless, or information preserving compression
  – Encoding or compression process
Image Compression

* Designed to remove the redundancies through a series of three independent processes

1. Mapper
   - Transforms $f$ into a nonvisual format designed to remove spatial and temporal redundancy
   - Reversible operation
  ![](https://image.pollinations.ai/prompt/1. Mapper
   - Transforms $f$ into a nonvisual format designed to remove spatial and temporal redundancy
   - Reversible operation
   - May not directly reduce the amount of data required to represent the image
   - Exemplified by run-length encoding
   - Image may also be mapped into a set of less correlated transform coefficients that must be further processed to achieve compression
   - In video applications, mapper may use previous frames to reduce temporal redundancy

2. Quantizer
   - Reduces the accuracy of the mapper’s output in accordance with pre-established fidelity criterion
   - Goal is to remove irrelevant information
   - Irreversible operation
   - Omitted for error-free compression
   - In video, the bit-rate of the encoded output is measured and used to adjust quantizer operation to maintain predetermined average output rate; visual quality may change from one frame to next

3. Symbol coder
   - Fixed or variable length code to represent quantizer output
   - Shortest code words are assigned to most frequently observed quantizer values to minimize coding redundancy
   - Reversible operation

   - Decoding or decompression process
   - Two components: symbol decoder and inverse mapper
   - No inverse quantizer because quantization loses information

- Image formats, containers, and compression standards
  - Image file format
    - Standard way to organize and store image data
    - Defines data arrangement and type of compression
  - Image container
    - Similar to file format but handles multiple types of image data
  - Image compression standard
    - Procedure to compress and decompress images

- Figure 8.6
- Tables 8.3 and 8.4

Basic compression methods

- Huffman coding
  - Yields the smallest number of code symbols per source
  - In terms of Shannon’s first theorem, resulting code is optimal for a fixed value of $n$, subject to the constraint that the source symbols be coded one at a time
  - In practice, source symbols may be intensities of an image, or output of an intensity mapping operation (pixel difference, run length)
  - Start with ordering the probabilities of symbols under consideration
– Combine the lowest probability symbols into a single symbol that replaces them in the next source reduction
– Figure 8.7
  * Hypothetical set of symbols ordered by decreasing probability
  * Combine symbols to form *compound symbols* iteratively until only two symbols are left, maintaining the probability order at every step
  * Code each reduced source starting with the smallest source and working back to original source
  * Minimum length binary code for a two symbol source are the symbols 0 and 1
  * Figure 8.8
  * Apply symbols recursively starting from two symbols till all the symbols are assigned
  * Average length of the code

\[
L_{\text{avg}} = 0.4 \times 1 + 0.3 \times 2 + 0.1 \times 3 + 0.1 \times 4 + 0.06 \times 5 + 0.04 \times 5
\]

\[
= 2.2 \text{bpp}
\]

– Huffman code is called a *block code* because each source symbol is mapped into a fixed sequence of code symbols
– The code is *instantaneous* because each code word in a string of code symbols can be decoded without referencing subsequent symbols
– It is *uniquely decodable* because any string of code symbols can be decoded in only one way
– Decoding 010100111100
– Example
  * Figure 8.9
  * Code with 7.428bpp
  * Estimated entropy for the image is 7.3838bpp
  * Compressed representation exceeds estimated entropy by

\[
512^2 \times (7.428 - 7.3838) = 11,587 \text{bits}
\]

* Compression ratio \( C = 8/7.428 = 1.077 \)
* Relative redundancy \( R = 1 - (1/1.077) = 0.0715 \)
– For a large number of symbols to be coded, construction of Huffman code is a nontrivial task

• Golomb coding
– Lossless data compression method using a family of data compression codes
– Coding of nonnegative integer inputs with exponentially decaying probability distributions
– Uses a tunable parameter \( m \) to divide an input value \( n \) into two parts:
  * \( r \) is the result of dividing \( n \) by \( m \), kept as a unary code
  * \( r' \) is the remainder \((n \%m)\), kept as a truncated binary code
– Given a nonnegative integer \( n \) and a positive integer divisor \( m > 0 \), the Golomb code of \( n \) with respect to \( m \), denoted \( G_m(n) \), is a combination of the unary code of quotient \([n/m]\) and the binary representation of \( n \% m \)
– \( G_m(n) \) is constructed as follows
  1. Form the unary code of \([n/m]\)
     * Unary code of an integer \( q \) is defined as \( q \) 1s followed by a zero
  2. Let \( k = \lceil \log_2 m \rceil \), \( c = 2^k - m \), \( r = n \% m \), and compute truncated remainder \( r' \) such that

\[
 r' = \begin{cases} 
 r \text{ truncated to } k - 1 \text{ bits} & 0 \leq r < c \\
 r + c \text{ truncated to } k \text{ bits} & \text{otherwise}
\end{cases}
\]
  3. Concatenate the results of steps 1 and 2
– Example: Compute $G_4(9)$

1. Determine the unary code of $\lfloor 9/4 \rfloor = 2$ which is 110
2. Let $k = \lceil \log_2 4 \rceil = 2$, $c = 2^2 - 4 = 0$, and $r = 9\%4 = 1001_2 \text{rest} = 0001_2$
   $r \not< c$ implying that $r'$ is $r + c$ truncated to $k$ bits, or $r' = 0001$ truncated to 2 bits, or $r' = 01$
3. Concatenating the results from steps 1 and 2, we have
   $$G_4(9) = 11001$$

– For the special case of divisor $m = 2^k$, $c = 0$ and $r' = r = n\%m$ truncated to $k$ bits for all $n$

* Divisions required to generate the Golomb code become binary shift operations
* Computationally simpler code, also called Golomb-Rice or Rice code

– Table 8.5

* $G_1$, $G_2$ and $G_4$ codes for the first 10 nonnegative integers
* $G_1$ is the unary code of the nonnegative integers because $\lfloor n/1 \rfloor = n$ and $n\%1 = 0$ for all $n$

– There are many Golomb codes to choose from based on the selection of divisor $m$

– Optimal Golomb codes

* Shortest average code length of all uniquely decipherable codes
* Integers to be represented are geometrically distributed with probability mass function
  $$P(n) = (1 - p)\rho^n$$
  for some $0 < \rho < 1$
* Also, $m$ is given by
  $$m = \left\lceil \frac{\log(1 + \rho)}{\log(1/\rho)} \right\rceil$$

* Figure 8.10: Plots $P(n)$ for three values of $\rho$ (small integers are much more probable than larger ones)
  1. Figure 8.10a: Three one-sided geometric distributions
  2. Figure 8.10b: Two-sided exponentially decaying distribution
  3. Figure 8.10c: A reordered version of Figure 8.10b
* The negative differences can be handled with a mapping such as
  $$M(n) = \left\{ \begin{array}{cc}
  2n & n \geq 0 \\
  2|n| - 1 & n < 0 
  \end{array} \right.$$  

Mapped integers can be efficiently coded using an appropriate Golomb-Rice code

– Example: Figure 8.1(c) and its histogram in Figure 8.3(a)

* Let $n$ be a nonnegative integer intensity in the image, $0 \leq n \leq 255$
* Let $\mu$ be the mean intensity
* Let $P(n - \mu)$ be the two-sided distribution in Figure 8.11a
  * Plot generated by normalizing the histogram of Figure 8.3a
* Normalizing the histogram by total number of pixels and shifting the normalized values to the left by 128 (subtracting the mean intensity from image) we get one-sided distribution $P(M(n - \mu))$ of Figure 8.11b
* Encode the normalized values using Golomb code
* Encoded representation is 4.5 times smaller than the original image ($C = 4.5$)
* $G_1$ code realizes 4.5/5.1 or 88% of the compression possible with variable length coding

– Image of Figure 8.9a

* If the intensities are coded using the same $G_1$ code as above, $C = 0.0922$
* This leads to data expansion (instead of compression)
− This is because the probabilities of intensities in Figure 8.9a are much different than the geometric distribution required for Golomb code

- Table 8.5 (again)
- Column 5 of the table contains the first ten codes of zeroth order exponential Golomb code, denoted $G_{\text{exp}}^0(n)$
- Exponential-Golomb codes are useful for encoding of run lengths because both short and long runs are coded efficiently
- An order $k$ exponential-Golomb code is computed as follows
  1. Find an integer $i \geq 0$ such that
     $$\sum_{j=0}^{i-1} 2^j + k \leq n < \sum_{j=0}^{i} 2^j + k$$
     and form the unary code of $i$. If $k = 0$, $i = \lfloor \log_2(n + 1) \rfloor$ and the code is also known as Elias gamma code
  2. Truncate the binary representation of
     $$n - \sum_{j=0}^{i-1} 2^j + k$$
     to $k + 1$ least significant bits
  3. Concatenate the results of the above two steps
- Compute $G_{\text{exp}}^0(8)$
  - Let $i = \lfloor \log_2(9) \rfloor = 3$ in step 1 as $k = 0$
  - We have
    $$\sum_{j=0}^{3-1} 2^j + 0 \leq 8 < \sum_{j=0}^{3} 2^j + 0$$
    $$\sum_{j=0}^{3} 2^j \leq 8 < \sum_{j=0}^{3} 2^j$$
    $$7 \leq 8 < 15$$
  - Unary code of 3 is 1110
  - Truncating the binary representation using step 2 above
    $$8 - \sum_{j=0}^{3-1} 2^j + 0 = 8 - \sum_{j=0}^{2} 2^j$$
    $$= 8 - 7$$
    $$= 1$$
    $$= 0001$$
    yielding 001 by truncating to $3 + 0$ least significant bits
  - Concatenation yields the code as 1110001

- Arithmetic coding
  - Form of entropy coding used in lossless data compression
  - Generates nonblock codes
    * No one-to-one correspondence between source symbols and code words
    * An entire sequence of source symbols is assigned to a single arithmetic code word
    * Encodes an entire message into a single number of arbitrary precision, a fraction $n$, where $0 \leq n < 1$
  - As the number of symbols in the sequence increases, the interval used to represent it becomes smaller, and the number of bits required to represent the interval become larger
  - Each symbol of the sequence reduces the size of interval by its probability of occurrence
  - Achieves the bound on compression established by Shannon’s first theorem
  - Properties of arithmetic coding
When applied to independent and identically distributed (IID) source symbols, the compression of each symbol is provably optimal.

Effective in a wide range of situations and compression ratios.

Simplifies automatic modeling of complex sources; yields near optimal or significantly improved compression for non-IID source symbols.

Based on simple arithmetic, supported efficiently on all CPUs.

Suitable for use as a compression black box by non experts.

- Practical issues
  - Excessive complexity for coding applications
  - Most efficient implementations covered by patents, some now expired or obsolete.
  - Efficient implementations difficult to understand.

- Notation
  - Data source represented by $\Omega$.
  - Source symbols $s_k$ coded as integer numbers in the set \{0, 1, ..., $M - 1$\}.
  - Let $S = \{s_1, s_2, \ldots, s_N\}$ be a sequence of $N$ random symbols put out by $\Omega$.
  - Assume that the source symbols are IID, with probability $p(m) = \text{Prob}\{s_k = m\}$, $m = 0, 1, 2, \ldots, M - 1$; $k = 1, 2, \ldots, N$.

- Figure 8.12
  - Coding a five symbol message $a_1, a_2, a_3, a_3, a_4$ from a four symbol source.
  - In the beginning, message occupies the entire interval [0, 1), divided into four regions based on probabilities of each symbol.

<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>

- In first iteration, associate first symbol with subinterval [0, 0.2).

- Each subsequent iteration corresponds to refining the code for next symbol.
  - Distribute all the symbols on the current subinterval as per their probability.
  - Narrow the subinterval based on the new symbol.

- The code can be represented by any number in the final interval.

- The coding can be described as:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Input symbol</th>
<th>Interval base $b_k$</th>
<th>Interval end $l_k$</th>
<th>Interval length $l_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>$a_1$</td>
<td>0.0</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>$a_2$</td>
<td>0.04</td>
<td>0.08</td>
<td>0.04</td>
</tr>
<tr>
<td>3</td>
<td>$a_3$</td>
<td>0.056</td>
<td>0.072</td>
<td>0.016</td>
</tr>
<tr>
<td>4</td>
<td>$a_3$</td>
<td>0.0624</td>
<td>0.0688</td>
<td>0.0064</td>
</tr>
<tr>
<td>5</td>
<td>$a_4$</td>
<td>0.06752</td>
<td>0.0688</td>
<td>0.00128</td>
</tr>
</tbody>
</table>

- Code can be described as any number in the range [0.06752, 0.06880), for example 0.068.
  - Just three digits to represent a five symbol message.
  - 0.6 decimal digits per source symbol.
  - Entropy of the source is 0.58 decimal digits per symbol.

- As the length of the sequence to be coded increases, resulting arithmetic code approaches the bound established by Shannon’s first theorem.
  - Coding performance falls short of bound due to...
1. Addition of end-of-message indicator to separate one message from another
2. Use of finite precision arithmetic
   - Practical implementations address the second problem by introducing
     Scaling strategy: Renormalizes each subinterval to \([0, 1]\) range before subdividing it in accordance with symbol probabilities
     Rounding strategy guarantees that the truncations associated with finite precision arithmetic do not prevent coding subintervals from being represented accurately

- Decoding the code \(s\)
  - In iteration \(i\), find \(\hat{s}_i\) as
    \[
    \hat{s}_i = \{ s : c(s) \leq \hat{v}_i \}
    \]
  - Compute the next interval as
    \[
    \hat{v}_{i+1} = \frac{\hat{v}_i - c(\hat{s}_i)}{p(\hat{s}_i)}
    \]

- Adaptive context dependent probability estimates
  - Help improve the accuracy of probabilities
  - Update symbol probabilities as symbols are coded or become known
  - Probabilities adapt to the local statistics of symbols being coded
  - Context
    - Predefined neighborhood of pixels around the symbols being coded
  - Causal context
    - Limited to symbols that have already been coded
  - Figure 8.13
    - Context determination
    - Probability estimation
      - Number of contexts and their probabilities
        \(P(0|a = 0), P(1|a = 0), P(0|a = 1), P(1|a = 1)\)
    - Appropriate probabilities are passed to arithmetic coder as a function of the current context

- LZW (Lempel-Ziv-Welch) coding
  - Error-free compression approach that also addresses spatial redundancies in an image
  - Assigns fixed-length codewords to variable length sequences of source symbols
  - Does not require a priori knowledge of the probability of occurrence of the symbols to be encoded
  - Example
    - Figure 8.9a (512 \(\times\) 512 8-bit image)
    - Uncompressed TIFF version requires 286,740 bytes (262,144 bytes for pixels plus 24,596 bytes of overhead)
    - With TIFF’s LZW compression, the file is 224,420 bytes
    - Compression ratio \(C\) = 1.28
    - For Huffman coding, \(C\) = 1.077
    - Additional compression in LZW due to removal of image’s spatial redundancy
  - Coding process
    - Build a codebook or dictionary of source symbols
    - For 8-bit monochrome images, first 256 words of dictionary are assigned to intensity 0, 1, \ldots, 255
    - Examine intensity sequences; sequences not in dictionary are placed in algorithmically determined (possibly next unused) locations
      - If the first two pixels of image are white, the sequence “255–255” may be assigned to location 256
      - Next time two consecutive white pixels are encountered, they can be represented by codeword 256, or the address containing the sequence “255–255”
If a 9-bit 512-word dictionary is used in the coding process, the original (8+8) bits to represent the two pixel sequence are replaced by a 9-bit codeword.

- Size of dictionary
  - Too small? Detection of matching sequences is less likely
  - Too large? Size of code words will adversely affect compression performance

- Example
  - Consider the following $4 \times 4$ 8-bit image of a vertical edge
    
    \[
    \begin{array}{cccc}
    39 & 39 & 126 & 126 \\
    39 & 39 & 126 & 126 \\
    39 & 39 & 126 & 126 \\
    39 & 39 & 126 & 126 \\
    \end{array}
    \]

  - Initial dictionary
    
    | Dictionary Location | Entry |
    |---------------------|-------|
    | 0                   | 0     |
    | 1                   | 1     |
    | ...                 | ...   |
    | 255                 | 255   |
    | 256                 | -     |
    | ...                 | ...   |
    | 511                 | -     |

  - Image encoded by processing pixels in the left-to-right, top-to-bottom manner
    - Concatenate each successive pixel value to a currently recognized sequence (Column 1 in Table 8.7)
    - Search the dictionary for each concatenated sequence and if found, replace by the newly concatenated and recognized sequence
    - If there is no concatenated sequence, address of the currently recognized sequence is output as the next encoded value

  - Original 128-bit image is reduced to 90 bits (10 9-bit codes)
    - Compression ratio $C = 1.42$

- Run-length coding
  - Run-length pairs
    - Start of a new intensity and number of consecutive pixels of that intensity
  - Standard approach in FAX coding
  - For few runs of identical pixels, may result in data expansion
  - Example: RLE in BMP file format
    - Image data represented in two modes: encoded and absolute
      - Either mode can occur anywhere in the image
    - Encoded mode
      - Uses two byte RLE representation
      - First byte has number of consecutive pixels
      - Second byte contains color index
      - 8-bit color index selects intensity (color or gray) from a table of 256 possible intensities
    - Absolute mode
      - First byte is 0
      - Second byte signals one of four possible conditions
<table>
<thead>
<tr>
<th>Second byte value</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>End of line</td>
</tr>
<tr>
<td>1</td>
<td>End of image</td>
</tr>
<tr>
<td>2</td>
<td>Move to a new position</td>
</tr>
<tr>
<td>3-255</td>
<td>Specify pixels individually</td>
</tr>
</tbody>
</table>

- When the second byte is 2, the next two bytes contain unsigned horizontal and vertical offsets to a new position in the image
- If the second byte is between 3 and 255, it contains the number of uncompressed pixels that follow
- Total number of bytes must be aligned to 16-bit word boundary
- Uncompressed BMP file of the $512 \times 512 \times 8$ bit image of Figure 8.9a requires 263,244 bytes of memory
  - Compression with BMP’s RLE option gives us 267,706 bytes
  - Compression ratio $C = 0.98$ (expansion)
- Figure 8.1c gives $C = 1.35$ with BMP RLE option
  - Particularly effective to compress binary images
    - Adjacent pixels are more likely to be identical (only two intensities)
    - Each image row can be replaced by just a sequence of lengths using a convention
      1. Specify the value of first run of each row
      2. Assume that each row begins with a white run, possibly of length zero
  - Additional compression may be achieved by variable length coding of run lengths
    - Encode black and white run lengths separately using variable length codes, specifically tailored to their own statistics
    - Let $a_j$ represent a black run of length $j$
      - Estimate the probability of $a_j$ in entire image
      - Estimate of entropy of this black run-length is
        $$H_0 = - \sum_{j=1}^{J} P(a_j) \log P(a_j)$$
      - Estimate of white run entropy $H_1$ is computed similarly
      - Run length entropy is given by
        $$H_{RL} = \frac{H_0 + H_1}{L_0 + L_1}$$
        where $L_0$ and $L_1$ are the average values of black and white run lengths, respectively
  - 1D CCITT compression
    - Each line of an image is encoded as a series of variable length Huffman code words
      - Each code word represents the length of alternate white and black runs
    - Two types of code words
      1. Terminating codes
      2. Makeup codes
  - Reading assignment
    - Symbol-based coding
      - Image represented as a collection of frequently occurring sub-images or symbols
      - Useful for document-type images containing bitmaps of characters
      - Each symbol stored in a symbol dictionary
      - Image coded as a set of triplets $\{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots\}$
        - Each $(x_i, y_i)$ specifies the location of a symbol
* Each $t_i$ is the address of the symbol in the dictionary
  - Figure 8.17
  * Input image size – $9 \times 51 \times 1 = 459$ bits
  * Coded representation
    - 3 bytes per triplet
    - 6 triplets
  * Coded size – $(3 \times 6) + ((9 \times 7) + (6 \times 7 + (6 \times 6))) = 285$ bits
  * Compression ratio – $C = 1.61$
  - JBIG2 compression

- Bit-plane coding
  - Process bit-planes individually and use run-length and symbol-based codes
  - Simplest approach: Create $m$ 1-bit bit planes
    * Problem: Pixel of intensity 127 ($01111111_2$) next to a pixel of intensity 128 ($10000000_2$) will have a 0-1 (or 1-0) transition in every bit-plane
  - Problem solved by representing the image by an $m$-bit gray code computed as
    \[
    g_i = a_i \oplus a_{i+1} \quad 0 \leq i \leq m - 1
    \]
    \[
    g_{m-1} = a_{m-1}
    \]
    * $\oplus$ indicates exclusive-or operation
    * Resulting successive code words differ in only one bit position
    * $127_{10} = 01111111_{2} \Rightarrow 01000000_{2}$
    * $128_{10} = 10000000_{2} \Rightarrow 11000000_{2}$
  - Figure 8.19 and 8.20

- Block transform coding
  - Divide the image into small non-overlapping blocks of equal size, such as $8 \times 8$ pixels
  - Process the blocks independently using a 2D transform
  - Use a reversible linear transform to map each block into a set of transform coefficients, which are then quantized and coded
  - For most images, a significant number of coefficients have small magnitudes and can be coarsely quantized or discarded without image distortion
  - Figure 8.21
  - Forward transform
    * Decorrelates the pixels of each block
    * Packs as much information as possible into the smallest number of transform coefficients
  - Quantization
    * Selectively eliminates or more coarsely quantizes the coefficients that carry the least amount of information in a predefined sense
    * The selected coefficients have the smallest impact on reconstructed subimage quality
  - Coding
    * Adaptive transform coding if transform coding is adapted to local image content; nonadaptive transform coding if fixed for all subimages

- Transform selection
  - Depends on the amount of reconstruction error tolerated and available computational resources
- Compression achieved by quantization of transformed coefficients
- Given subimage $g(x, y)$ of size $n \times n$
- Forward transform
  \[ T(u, v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x, y)r(x, y, u, v) \]
- Inverse transform
  \[ g(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v)s(x, y, u, v) \]
- $r(x, y, u, v)$ and $s(x, y, u, v)$ are forward and inverse transformation kernels, also called basis functions or basis images