Image Compression

Relative Data Redundancy

$R = \frac{b' - b}{C}$

where $C = compression\ ratio$

and $b\ and\ b' = bits$

$C = b/b'$

Why do we need compression ?

- Data storage
- Data transmission

How can we implement Compression?

Coding Redundancy

Most 2D intensity arrays contain more bits than required to represent intensities.

Spatial and temporal Redundancy

Pixels of most 2D intensity arrays are correlated spatially and video sequences are temporally correlated.

Irrelevant Information

Most 2D intensity arrays consists information that is ignored by the human visual system.

$r_k = intensity$, then

Probability, $p_r(r_k) = \frac{n_k}{MN}$

$l(r_k) = bits\ used\ to\ represent\ r_k$

then, average bits required to represent each pixel:

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

Measuring Image Information

A random event E with probability $\mathcal{P}(E)$ is said to contain

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

units of information.

Possible Events = $\{a_1, a_2, a_3, \ldots, a_{\mathcal{J}}\}$
Their associated probabilities = $\{P(a_1), P(a_2), \ldots, P(a_{\mathcal{J}})\}$

The average information per source output, called the entropy of the source

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

a_j is called source symbols. Because they are statistically independent, the source is called **zero-memory** source.

Huffman Coding

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

FIGURE 8.8 Huffman code assignment procedure.

The average length of this code is

$$L_{avg} = 0.4*1 + 0.3*2 + 0.1*3 + 0.1*4 + 0.06*5 + 0.04*5$$
$$= 2.2 \text{ bits/pixel}$$

Golomb Coding

Step 1: Unary Code

Represent integer n by n $1s$ followed by a 0.

Form the unary code of $\lfloor n/m \rfloor$

Step 2

Let $K = \lceil \log_2 m \rceil$

$C = 2^k - m$

$r = n \mod m$

$$r' = \begin{cases} r \; truncated \; to \; k-1 \; bits & 0 \le r < c \\ r + c \; truncated \; to \; k \; bits & otherwise \end{cases}$$

Step 3

Concatenate the results of step 1 and step 2.

$G_4(9):$

$\lfloor 9/4 \rfloor = 2,$

the unary code is 110

$k = \lceil \log_2 4 \rceil = 2, c = 2^2 - 4 = 0,$

$r = 9 \mod 4 = 1.$

$r' = 01$

$G_4(9) = 11001$

Arithmetic Coding

$$A = (u^{n-1} - l^{n-1})$$

$$l^n = l^{n-1} + A \times F_x(x_n - 1)$$

$$u^n = l^{n-1} + A \times F_x(x_n)$$

where $x_n =$ current symbol integer value ($a_2 = 2$)

then find $ans = \frac{l^m + u^m}{2}$

Deciphering

Find F values by taking prefix sums

$$F(0) = 0,\ F(1) = P(1),\ F(2) = P(1) + P(2),\ F(3) = P(1) + P(2) + P(3),\ and\ so\ on$$

LZW (Dictionary Coding)

Lempel-Ziv-Welch coding, assigns fixed length code words to variable length sequences of source symbols.

Used in

- TIFF
- GIF
- PDF

The Algorithm

- A codebook or dictionary containing the source symbol is constructed.
- The first 256 words of the dictionary are assigned to gray levels $0 - 255$
- Remaining part of the dictionary is filled with sequences of gray levels.

Important Features

1. The Dictionary is created while the data are being encoded. So encoding can be done *on the fly*.
2. The dictionary is not required to be transmitted. The dictionary will be built while decoding.
3. If the dictionary overflows we have to reinitialize it and add a bit to each one of the code words.
4. Choosing a large dictionary size avoids overflow, but spoils compressions.

If a new word is encountered, the word is sent as output in the uncompressed form and is entered into dictionary as a new entry.

Add doublets to the dictionary if not present and send the code for a single character
If the doublet is already present add the triplet and send the code for the doublet
Check for the **longest prefix** at every step in the dictionary and add a letter to it and add it to the dictionary and send the code for that longest prefix.

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

Run Length Encoding

- Used to reduce size of a repeating string of characters.
- This repeating string is called a **run**, typically encodes a run of symbol into 2 bytes, a **count** and a **symbol**.
- Can compress any size of data
- Cannot achieve high compression ratios compared to other methods.
- Easy to implement and is quick to execute.
- Supoorted by most bitmap formats.

Encoding

$string = WWWWBWWBBWWWW$

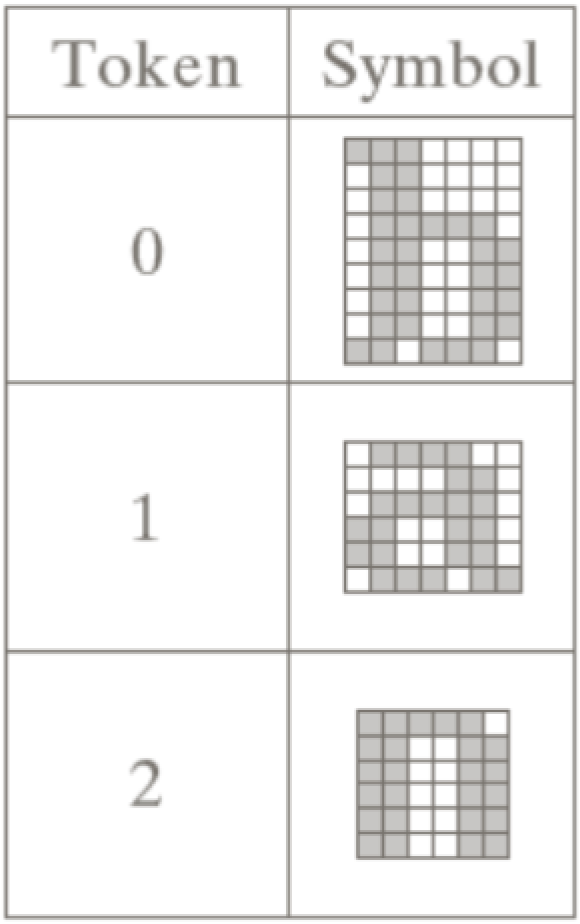
$encoding = 4W1B2W2B4W$

$format = < count >< symbol > \dots$

Symbol Based Coding

Images are represented in forms of triplets $(x_1, y_1, t_1),\ (x_2, y_2, t_2), \dots$

where (x, y) is the location and t is the symbol index in dictionary.



Bit Plane Coding

- Represent the image intensity values into its binary representation
- Lets say the binary representation of the largest number used m bits, so we need to form m bit planes where the 1^{st} plane will have 1^{st}(MSB) digit of the binary representation of every values, we will do the same step for m planes
- Then encode each plane with run length coding (first converting the 2D array to 1D linear array).

A problem $127 = 01111111$ and $128 = 10000000$, the run length has decreased in all the bit planes!

SOLUTION: Gray Code

In gray code, the representation of adjacent gray levels will differ only in 1 bit.

Let $g_{m-1} \ldots \ldots g_1 g_0$ represent the gray cod representation of a binary number.

Then:

$$g_i = a_i \oplus a_{i+1} \qquad 0 \le i \le m-2$$

$$g_{m-1} = a_{m-1}$$

In gray code:

127 = 01000000

128 = 11000000

$a_{m-1} = MSB$

Take xor of 2 adjacent bits except for MSB.

Gray Decoding

The MSB is retained as such, i.e.,

$$a_i = g_i \oplus a_{i+1} \qquad 0 \le i \le m-2$$

$$a_{m-1} = g_{m-1}$$

Differential Pulse Code Modulation (DPCM)

Take $A=0,\ B=1,\ C=2,\ D=3,...$ and encode the string

Works better than RLE for many digital images.

Image transform

2 main types

- Orthogonal transform
  - example: Walsh-Hadamand transform, DCT
- Subband tranform
  - Wavelet transform

Orthogonal matrix $\mathbf{W}$

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix}$$

Walsh Hadamand Transform

- Decompose a function into a linear combination of rectangular basis functions, called **Walsh functions**, of value $+1$ and -1.
- For Hadamand Ordering, the transformation matrix is obtained by substituting transformation kernel

$$s(x,u) = \frac{1}{\sqrt{N}}(-1)^{\sum_{i=0}^{N-1} b_i(x) b_i(u)}$$

Letting $\mathbf{H}_N$ denote the Hadamard matrix of order N, a simple recursive relationship for generating Hadamard-ordered transfomation matrices is

$$\mathbf{A}_W = \frac{1}{\sqrt{N}}\mathbf{H}_N$$

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$$

$$H_1 = [1] \qquad H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Bit Allocation

The overall process of truncating, quantizing, coding the coefficients of the transformed subimage is commonly called **bit allocation**.

Zonal Coding

The retained coefficients are selected on the basis of **max variance**.

Thresold Coding

The retained coefficients are selected on the basis of **max magnitude**.

JPEG Compression

- Stands for Joint Photographic Experts Group.
- Used on 24-bit color files (RGB)
- Works well on photographic images.
- Lossy compression technique, yet yields an excellent quality image with high compression.

Different Coding Systems

Defines 3 of them:

- A lossy baseline coding system
- an extended coding system for greater compression, higher precision
- A lossless independent coding system

Steps

- Convert RGB to YUV (Optional).

- Divide the file into 8×8 blocks.
- Transform from spatial domain to frequency domain using Discrete cosine transform.
- Quantize by dividing each coefficient by an integer value and round off.
- Look at the resulting coefficients in a zig-zag manner. Do a RLE followed by Huffman coding.

YUV

- Stores color in terms of its **luminance** (brightness) and **hue**.
- Not required for JPEG compression, but gives better compression rates.
- Human eye is less sensitive to hue than luminance.

Discrete Cosine Transform

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

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