# 12	ECE 253a	Digital Image Processing	Pamela Cosman	11/4/11

Introductory material for image compression

Motivation:

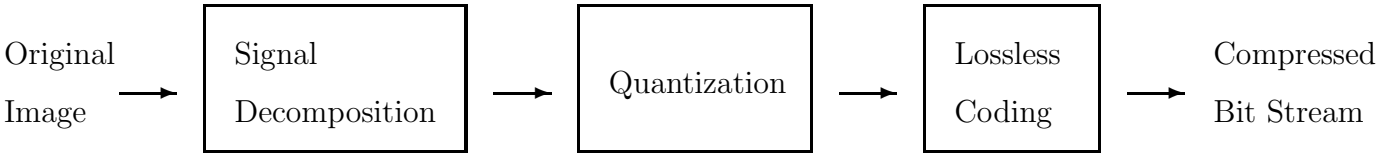
- Low-resolution color image: 512×512 pixels/color, 24 bits/pixel $\approx$ 3/4 MB
- 3000×2000 pixels, 24 bits/pixel $\approx$ 18 MB
- Video SDTV: 640×480 pixels $\times$ 24 bits/pixel color $\times$ 30 images/sec = 221 Mbps
- HDTV: 1280×720 pixels $\times$ 24 bits/pixel color $\times$ 60 images/sec = 1.3 Gbps
- 3D Stereo, Multiview, High-dynamic range ...?

Why can we compress images? 3 types of redundancy:

1. Coding redundancy: Not everything is equally probable
2. Interpixel redundancy: there are correlations between neighboring pixels, between color planes, between successive frames temporally
3. Psychovisual redundancy: the human visual system doesn't see everything anyway

General system for image compression:

A compression system typically consists of one or more of the following operations, which may be combined with each other or with additional signal processing:



Signal decomposition: the image is transformed into another space, or decomposed into a collection of images. Typically this is done by linear transformation by a Fourier or discrete cosine transform or by wavelet or subband filtering.

Quantization: analog or high rate digital pixels are converted into a relatively small number of bits. This operation is "lossy" as it is nonlinear and noninvertible, so information is lost. The conversion can operate on individual pixels (scalar quantization) or groups of pixels (vector quantization).

Lossless coding: Binary words are chosen to represent whatever symbols come out of the previous step. Huffman coding is an example of lossless coding. This step is invertible. It is also called entropy coding.

We'll look at each of these operations in turn, starting with ...

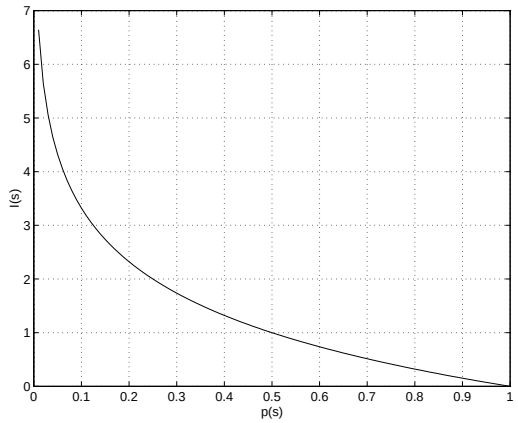
Lossless coding

Let S be a source that produces symbols from a finite alphabet $s_1, s_2, \ldots, s_j$ with probabilities of occurrence $p(s_1), p(s_2), \ldots, p(s_j)$.

The information associated with symbol s_i is, by definition:

$$I(s_i) = \log \frac{1}{p(s_i)}$$

Generally we take the log to the base 2, and the units of information are called bits.



Entropy (first-order entropy) is defined to be the average information per source output:

$$H(S) = \sum_{i=1}^{j} p(s_i) I(s_i) = -\sum_{i=1}^{j} p(s_i) \log_2 p(s_i) \text{ bits/symbol}$$

We define 0log0 to be 0 in this formula. Shannon showed that the entropy is a lower bound on the average number of bits needed to report the output of the source.

Source coding:

Let X be a random variable taking on values $s_1, s_2, \ldots, s_J$ from a finite alphabet $\mathcal{A}$.

Let $\mathcal{D}^*$ be the set of finite length strings of symbols from a D-ary alphabet.

A source code C for the random variable X is a mapping from $\mathcal{A}$ to $\mathcal{D}^*$.

Let $C(s_i)$ denote the codeword corresponding to s_i.

Let $l(s_i)$ denote the length of $C(s_i)$.

The expected length of the code is

$$El(C) = \sum_{x \in \mathcal{A}} p(x) l(x)$$

Without loss of generality, we can assume that the D-ary alphabet is $\mathcal{D} = \{0, 1, \ldots, D-1\}$.

Examples of codes

ASCII is a *fixed-length code.*

$$a \rightarrow 1000011 \text{ and } A \rightarrow 1000001$$

The same number of bits (7) is used to represent each symbol. If we want to reduce the number of bits required to represent different messages, we should use different numbers of bits to represent different symbols. Use fewer bits for things that occur more often. This would be a *variable length code.*

Suppose we have a 4-letter alphabet, with probabilities $p(a_0) = 1/2$, $p(a_1) = 1/4$, $p(a_2) = p(a_3) = 1/8$. The entropy for this source is 1.75 bits/symbol. Consider the following 4 codes:

Input Letter	Probability	Code 1	Code 2	Code 3	Code 4
a_0	1/2	0	0	1	0
a_1	1/4	0	1	01	01
a_2	1/8	1	00	001	011
a_3	1/8	10	11	000	0111
Average Length		1.125	1.25	1.75	1.875

Problem with code 1: Two symbols have the same codeword.

Problem is fixed with code 2, but still the output may not be uniquely decodable. Suppose the receiver gets the sequence 00. What was the input? a_0a_0 or a_2?

Codes 3 and code 4 look OK.

Definition: A code is **non-singular** if every element of $\mathcal{A}$ maps into a different string in $\mathcal{D}^*$.

$$x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j)$$

Non-singularity means that we will have an unambiguous description of a single value of X. But we generally want to send a sequence of values. So we would need to separate them by commas. This is inefficient.

Definition: The extension C^* of a code C is the mapping from finite length strings of A to finite length strings of D defined by the concatenation:

$$C^*(x_1x_2 \ldots x_n) = C(x_1)C(x_2) \ldots C(x_n)$$

Definition: A code is called **uniquely decodable** if its extension is non-singular.

This means that any encoded string has only one possible source string which produced it.

Definition: A code is called a **prefix code** or an **instantaneous code** if no codeword is a prefix of any other codeword.

This means that the end of a codeword is immediately recognizable. The code is self-punctuating.

More examples of codes:

Input Letter	Singular	Non-singular but not U.D.	U.D. but not instantaneous	instantaneous
a_0	0	0	10	0
a_1	0	010	00	10
a_2	0	01	11	110
a_3	0	10	110	111

We will now restrict ourselves to considering an alphabet size of 2 (binary codes) and only prefix codes. It can be shown that although uniquely decodable codes are a larger class of codes than prefix codes, they offer no advantages in terms of lower average codeword length. For any U.D. code, one can find a prefix code with the same codeword lengths.

Optimal binary prefix code: we use the word optimal to mean that it provides the minimum average length. An optimum binary prefix code has the following properties:

- If symbol a is more probable than symbol b ($p(a) > p(b)$) then $l(a) \leq l(b)$
- The two least probable input symbols have codewords which are equal in length and differ only in the final symbol

Proof:

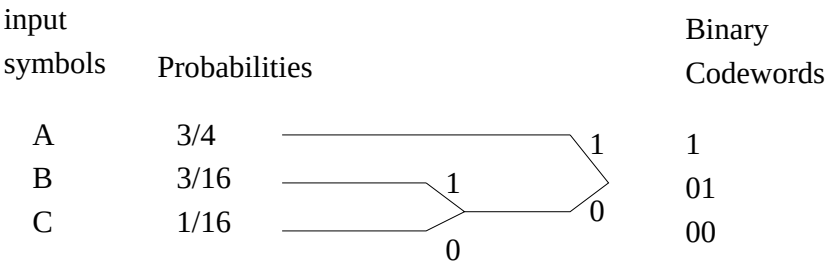
- Exchanging the two codewords will cause a strict decrease in the average length. Hence the original code could not have been optimal.
- Suppose they have different lengths. Since a prefix of the longer codeword cannot be itself a codeword, we can delete the final symbol of the longer word without it being confused for any other codeword. This strictly decreases the average length and hence the original code could not have been optimal. Thus the two least probable codewords must have equal lengths.

  Suppose they differ in some position other than the final one. We could then remove the final symbol and shorten the code without confusion.

These properties provide an iterative design technique for optimal codes, called Huffman coding.

HUFFMAN CODING

The input alphabet symbols can be considered to correspond to the terminal nodes of a code tree which we are to design. We try to find an optimal code for the alphabet, then for the successively reduced alphabet.



The entropy of the source is H = 1.014 bits per symbol. The average length of this code is L = 1.25 bits per symbol.

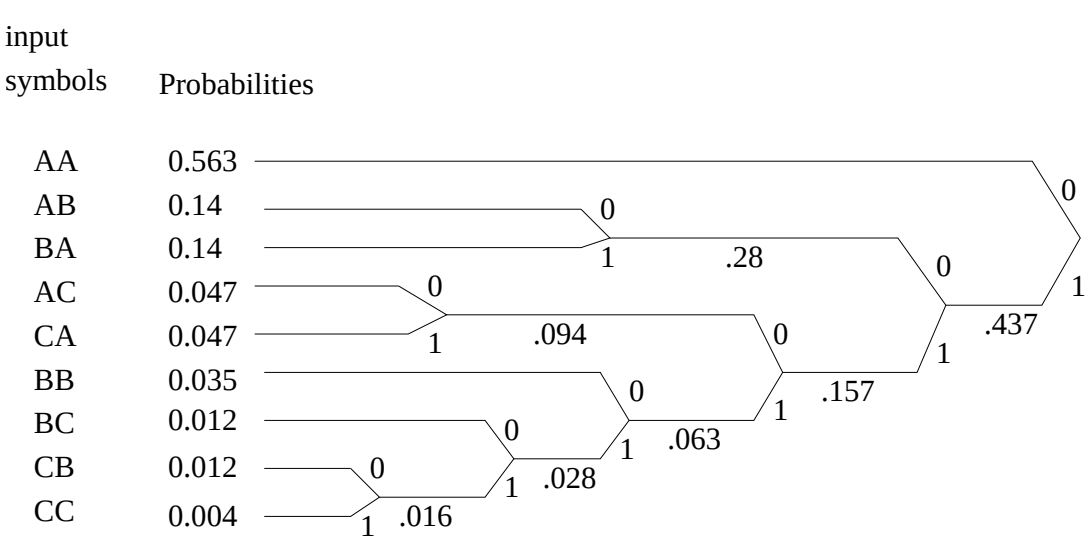
The efficiency of the code is

$$\text{efficiency } = \frac{H}{L} = \frac{1.014}{1.25} \approx 0.81$$

The redundancy of the code is

$$\text{redundancy } = |H - L| = |1.014 - 1.25| = .236$$

If we take pairs of inputs together, we can form a Huffman code for this paired source:



The expected length is now 2.075 bits per 2 symbols, so 1.0375 bits per symbol. The efficiency is up to 1.014 / 1.0375 ≈ 0.98.

Data Compression: A Non-assignment

Each equation contains the initials of words that will make it correct. Find the missing words. The questions can be considered to be good examples of data compression, or of the redundancy of English. As an example, the first equation has been done for you.

A) 26 = L. of the A. 26 = Letters of the Alphabet

B) 7 = W. of the A. W. _____

C) 1001 = A. N. _____

D) 12 = S. of the Z. _____

E) 54 = C. in the D. (with J.) _____

F) 9 = P. in the S. S. _____

G) 88 = P. K. _____

H) 13 = S. on the A. F. _____

I) 32 = D. F. at which W. F. _____

J) 18 = H. on the G. C. _____

K) 90 = D. in a R. A. _____

L) 200 = D. for P. G. in M. _____

M) 8 = S. on a S. S. _____

N) 3 = B. M. (S. H. T. R.) _____

O) 4 = Q. in a G. _____

P) 24 = H. in a D. _____

Q) 1 = W. on a U. _____

R) 5 = D in a Z. C. _____

S) 57 = H. V. _____

T) 11 = P. on a F. T. _____

U) 1000 = W. that a P. is W. _____

V) 29 = D. in F. in a L. Y. _____

W) 64 = S. on a C. _____

X) 40 = D. and N. of the G. F. _____

Y) 87 = F. S. and S. _____