

## Media Types

-data (text)  
-audio

-image  
-video

-graphics

Compression	{	Data	→	cannot afford to lose information
		Audio / image / video	→	can afford to lose information
		Graphics	→	maybe

## Some Compression Methods

- RLE, Huffman encoding etc. for LOSSLESS
- Sub-band coding, CELP for Audio
- JPEG, Wavelet, fractal for Images
- H.263, MPEG-1, MPEG-2 for video
- MPEG-4, Emerging standard considering all media types, including graphics.

## Overview of Compression

-Goal of compression is to reduce redundancies in information

-Types of redundancies

- coding redundancies
- spatio / temporal redundancy
- perceptual redundancy (Human)

Reducing coding redundancies is completely lossless

Reducing spatio / temporal redundancies can be lossy

Reducing perceptual redundancies is lossy

What kinds of events convey more information?

- Events:
- 1) There was a car accident
  - 2) It snowed in Edmonton on January 1
  - 3) It snowed in Las Vegas on July 10
  - 4) A Jumbo Jet crashed

- |         |   |
|---------|---|
| Event 3 | contains the most because it's the rarest event |
| Event 4 | eventful, but not as rare                       |
| Event 2 | not that important, because it happens so often |
| Event 1 | is almost not worth mentioning                  |

Let  $P(E)$  = Probability of an event  $E$  occurring

Info content in  $E$  proportional to  $\frac{1}{P(E)}$

Suppose  $b$  digits are used to code an Event or Symbol. Then, information content =  $\log_b \frac{1}{P(E)}$

Consider  $n$  events:  $E_1, E_2, E_3, \dots, E_n$

$H$  = Entropy = Average Information content of a set of events (or symbols)  
=  $\sum (\text{Probability of Event } E_i \text{ occurring}) \times (\text{Information content of } E_i)$

$P(E_1), P(E_2), \dots, P(E_n)$

where  $0 \leq P(E_i) \leq 1$  &  $\sum P(E_i) = 1$

$$H = - \sum P(E_i) \log_b P(E_i)$$

where  $b$  is base used for coding (e.g., 2 for binary)

When  $P(E_i) = \frac{1}{N}$ , for  $i = 1, 2, \dots, N$

the Entropy reaches its maximum value.

$$H_{\max} = \log_b N$$

where  $b$  is what base (i.e. 2 for binary)

For example, if base = 2 (i.e., Binary)

$$H_{\max} = \log_2 N$$

e.g. 4 symbols:

code		Prob	Entropy – $\log_2 p = 2$ if binary representable
00	A	$\frac{1}{4}$	
01	B	$\frac{1}{4}$	
10	C	$\frac{1}{4}$	
11	D	$\frac{1}{4}$	

Usually during coding, some symbols appear more likely than others. In this case, it is possible to have variable length codes representing these symbols in order to reduce the average code length and get close to the entropy.

## Huffman Coding

Idea is to create variable length codes depending on the probability of appearance of different symbols. Symbols that appear frequently got shorter codes.

The probability of a symbol (X) is nothing but the proportion of time X appears in the string of symbols to be coded. (P.344 in Image Processing text)

6 symbols: A B C D E F     $P(A) = 0.4$     $P(B) = 0.3$     $P(C) = 0.1$     $P(D) = 0.1$   
     $P(E) = 0.06$     $P(F) = 0.04$

Using fixed length codes, 000 = A, 001 = B, ...101 = F (**Length = 3**)

$$H = \text{entropy} = - \sum P(E_i) \times \lg P(E_i) = 2.14$$

for variable length codes, we use Huffman method:

### Step 1 Source Reduction

- list symbols in order from largest to smallest probability
- we then combine successive two small + probability symbols, until 2 left

A	0.4	-----	0.4	-----	0.4	-----	0.4	-----	0.4
B	0.3	-----	0.3	-----	0.3	-----	0.3	-----	0.6
C	0.1	-----	0.1	-----	0.1	-----	0.3	-----	
D	0.1	-----	0.1	-----	0.2	-----		-----	
E	0.06	-----	0.1	-----		-----		-----	
F	0.04	-----		-----		-----		-----	

### Step 2 Code Generation

In this step we work backwards, assigning variable length codes:

	(v)		(iv)		(iii)		(ii)		(i)
A	1	-----	1	-----	1	-----	1	-----	1
B	00	-----	00	-----	00	-----	00	-----	0
C	011	-----	011	-----	011	-----	01	-----	
D	0100	-----	0100	-----	010	-----		-----	
E	0100	-----	0101	-----		-----		-----	
F	01011	-----		-----		-----		-----	

Thus final Huffman codes will be:

A = 1      B = 00      C = 011      D = 0100      E = 01010      F = 01011

Average Huffman Code length = 2.2 bits/symbol

$$\begin{array}{cccccc} \text{A} & & \text{B} & & \text{C} & & \text{D} & & \text{E} & & \text{F} \\ \underbrace{\phantom{0.4 \times (1)}} & & \underbrace{\phantom{0.3 \times (2)}} & & \underbrace{\phantom{0.1 \times (3)}} & & \underbrace{\phantom{0.1 \times (4)}} & & \underbrace{\phantom{0.06 \times (5)}} & & \underbrace{\phantom{0.04 \times (5)}} \\ 0.4 \times (1) & + & 0.3 \times (2) & + & 0.1 \times (3) & + & 0.1 \times (4) & + & 0.06 \times (5) & + & 0.04 \times (5) \end{array}$$

Compared to entropy = 2.12 bits/symbol

$$\begin{array}{cccccc} \text{A} & & \text{B} & & \text{C} & & \text{D} & & \text{E} & & \text{F} \\ \underbrace{\phantom{0.4 \lg (0.4)}} & & \underbrace{\phantom{0.3 \lg (0.3)}} & & \underbrace{\phantom{0.1 \lg (.1)}} & & \underbrace{\phantom{0.1 \lg (.1)}} & & \underbrace{\phantom{0.06 \lg (.06)}} & & \underbrace{\phantom{0.04 \lg (.04)}} \\ - ( 0.4 \lg_2 (0.4) + 0.3 \lg (0.3) + 0.1 \lg (.1) + 0.1 \lg (.1) + 0.06 \lg (.06) + 0.04 \lg (.04) ) \end{array}$$

Note: codes are unique and identifiable without delimiters as individuals.

e.g.,            A = 1, and nothing else starts with 1  
                     B = 00, and nothing else starts with 00

### Run Length Encoding

from Huffman,...

A	0.4	1	1	20	20
B	0.3	00	2	15	30
C	0.1	011	3	5	15
D	0.1	0100	4	5	20
E	0.06	01010	5	3	15
F	0.04	01011	5	2	10
letters	prob	code	# of bits in code	total = 50	110 # of bits

Suppose we have 50 letters in a text with probabilities as given above

Using Huffman encoding:  $110/50 = 2.2$  bits / symbol

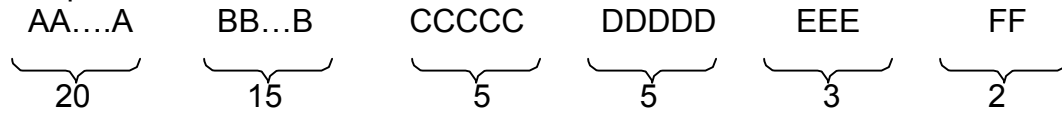
### Overhead for code:

-need to store (or transmit) the codes and corresponding letters.

e.g. (1,A), (00,B), ....

Huffman (or other similar codes) by itself does not take into account the arrangement of letters ABCDEF in the text.

For example:



5 bits can count up to 32

3 bits can represent 6 characters

Now, lets look at an alternate way of coding a string such as this.

20A

$$\begin{array}{rcl}
 5+3 & 8 & \\
 8 \times 6 & 48 \text{ bits} & \Rightarrow \quad 48/50 = 0.96 \text{ bits/symbol}
 \end{array}$$

## Combining Run-length with Huffman

-Usually done for Binary code stream

011000110011110101.....Binary stream

(1,2), (2,1), (3,0), (2,1), (2,0), (4,1).....  
1 2 3 2 2 4

3 Run Lengths

if we establish the first bit is 0, then it follows that it will alternate for each Run length thereafter.

0 1 2 3 2 2 4 } new stream

How can I use Huffman here??

We use Huffman encoding to code the run length

## Using Huffman with Run Lengths

1. First, use run length to encode the bit stream
2. Second, use Huffman to encode the run lengths

## Coding Text, e.g. Unix Compress

- Can use a dynamic dictionary (code book)

Option 1: Start with a default skeleton dictionary of frequently used words  
(context sensitive)

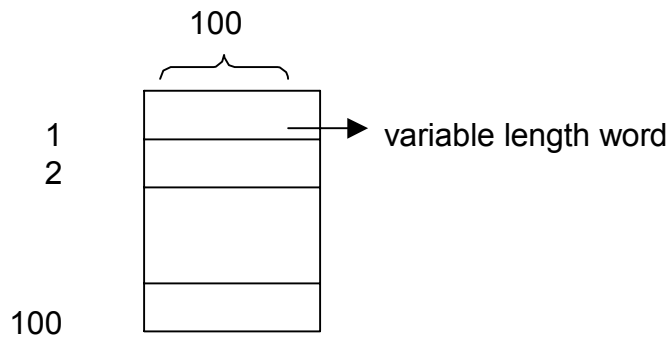
1	IS	}	Default dictionary
2	THE		
3	CAN		
	.		
	ANDRE	}	Dynamic dictionary

\* Universal Method (note: a priori knowledge of source statistics is required)

◆ Messages are encoded as a sequence of addresses to words in the dictionary

- Repeating patterns become words in the dictionary

- Superior to run length encoding in most cases



-original algorithm was developed by Ziv & Lempel

-Practical implementation was done by Welch, so its called Lempel-Ziv-Welch (LZW) coding

-Unix compress is a variation of this method

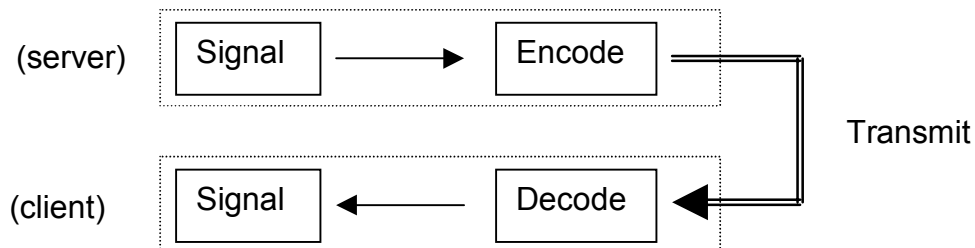
(no guarantee that LZW will do better than Huffman)



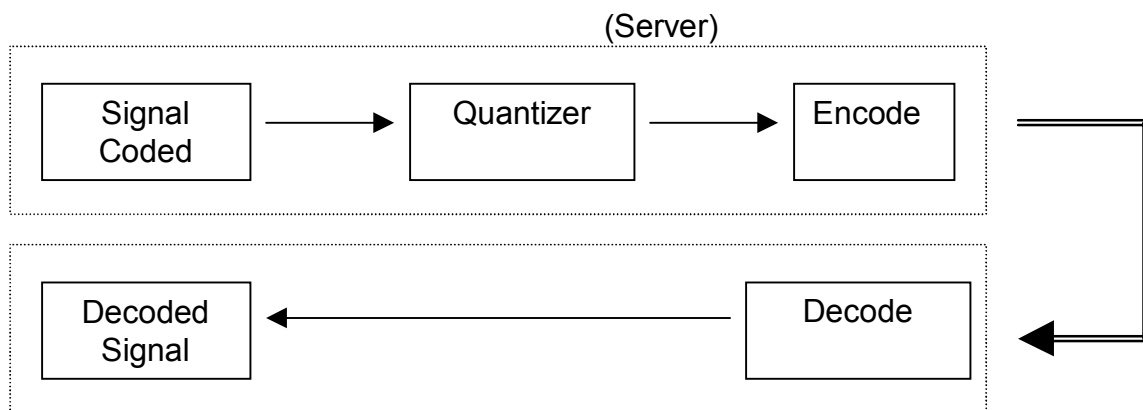
## General Models for Compression / Decompression

-they apply to symbols data, text, and to image but not video

### 1. Simplest model (Lossless encoding without prediction)



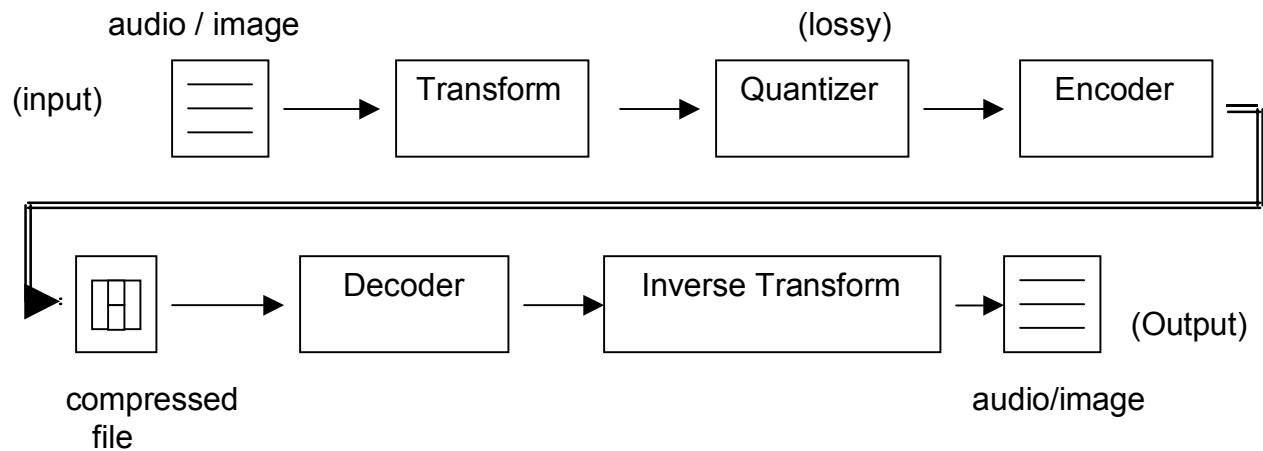
### 2. Lossy coding without prediction:



Quantization: → Initial Data      0, 1, 2, 3 ... 256  
Quantized Data 0, 1, 2, 3, 4, 8, 16, 32, 64, 128, 256

└→ Lossy

### 3. Transform Coding:



One of the most popular transforms is called the discrete cosine transform (DCT)

In the frequency domain, we can have:

- Fourier transform
- Sin transform
- Cosine transform

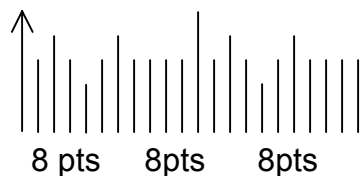
#### 1 – D DCT:

- forward transformation

$$F(u) = \frac{c(u)}{2} \sum_{x=0}^7 f(x) \cos \left[ \frac{(2x+1)\pi u}{16} \right], \quad u = 0, 1, 2, \dots, 7$$

- inverse transform

$$f(x) = \sum_{u=0}^7 \frac{c(u)}{2} F(u) \cos \left[ \frac{(2x+1)u}{16} \right], \quad x = 0, 1, 2, \dots, 7$$



$$c(u) = \begin{cases} \frac{1}{2} & \text{for } u = 0 \\ 1 & \text{for } u > 0 \end{cases}$$

## 2-D DCT [Works on 8 x 8 image blocks]

- forward:

$$F(u,v) = \frac{1}{N} C(u) c(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) * \cos \left[ \frac{u\pi(2x+1)}{2N} \right] \cos \left[ \frac{v\pi(2y+1)}{2N} \right]$$

- inverse

$$f(x,y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) c(u) C(v) * \cos \left[ \frac{u\pi(2x+1)}{2N} \right] \cos \left[ \frac{v\pi(2y+1)}{2N} \right]$$

$$F(0,0) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) = \text{average Grey level of } 8 \times 8 \text{ block}$$

- lower u,v values represent lower frequencies or slow transition (smooth variations) in a signal. Human perception is more sensitive to changes in smooth variations, so we use more precise quantization at lower u,v numbers and less precise with higher u,v levels; the higher u,v values represent sudden changes in a 1-D signal or sharp edges in a 2-D image

-compression is achieved by specifying 8 x 8 quantization table, which usually has larger values for higher frequencies (i.e. higher (u,v))

-default quantization tables are created taking human perception into account

-however, you can choose your own quantization tables.

## Some other classes of Transforms


- wavelets (to be discussed in labs. + programming assignment)
- Gabor filters

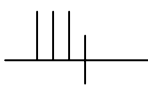
$f(x,y)$  usually 0 to 255 for 8 bit gray scale

#### 4. Predictive Coding

1-D Digital Audio

(Given Signal)  Prediction: last sample point

(Predicted signal) 

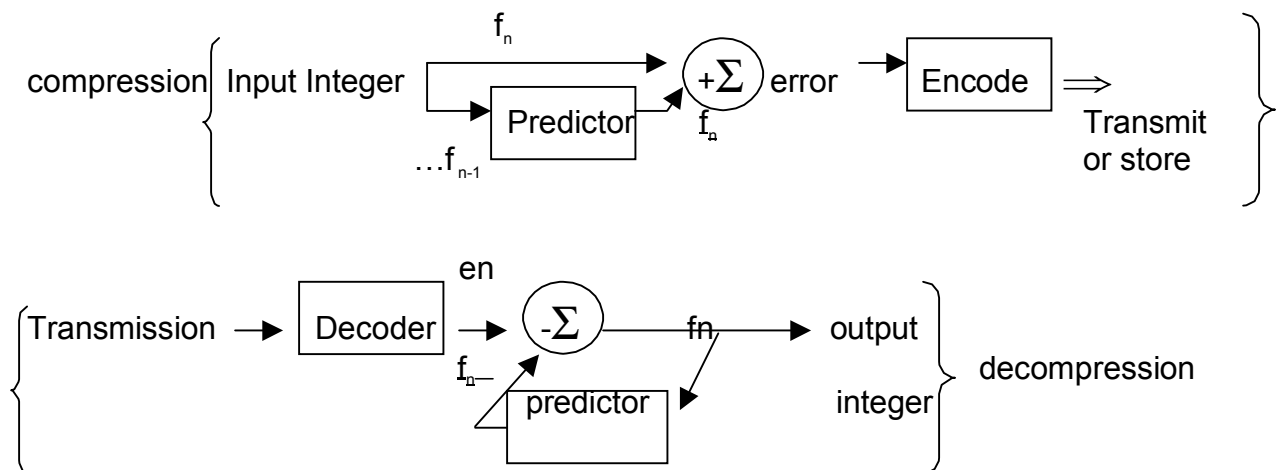
(error after pred) 

Predictive coding types 1) lossless 2) lossy

lossless  $\rightarrow$  does not quantize error in prediction

lossy  $\rightarrow$  quantize errors in prediction

LOSSLESS MODEL:



The most effective AUDIO codes use a form of linear prediction (LP). Some popular ones are:

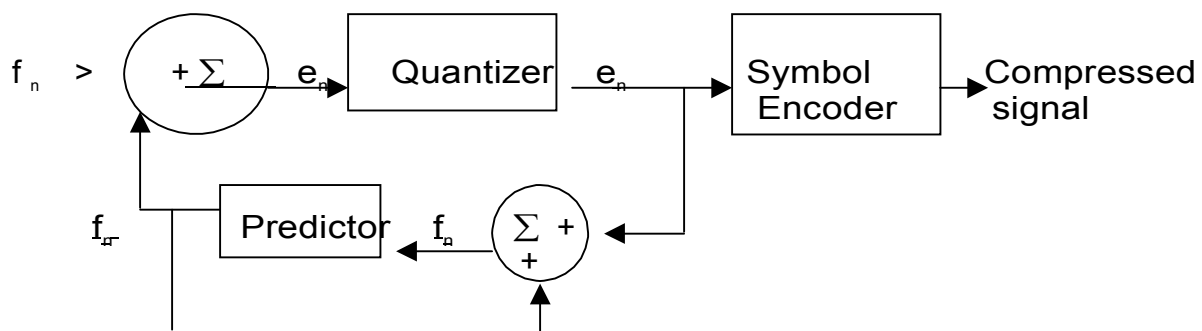
-CELP (Code Excited Linear Predictions)

### Lossy Predictive Coding:

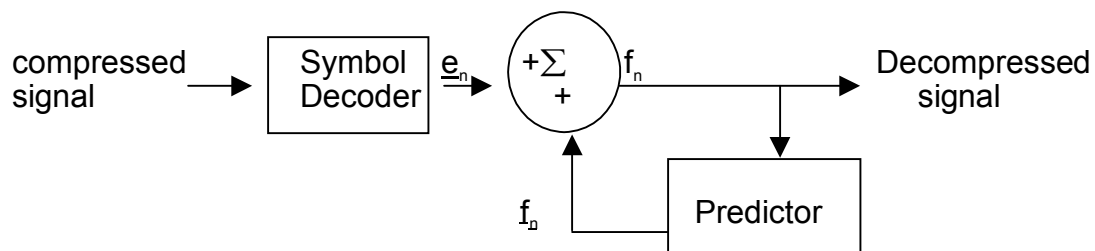
$\hat{f}_n$ : estimate of  $f_n$

$e_n = f_n - \hat{f}_n$  = error in estimate

### Coder



### Decoder

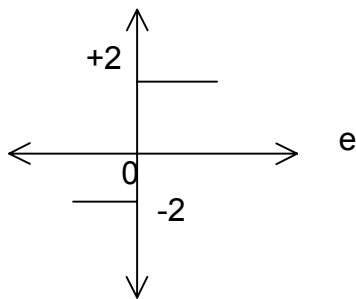


One of the simplest lossy predictive coding methods is known as “Delta Modulation”

1) Uses a simple predictor like  $\hat{f}_n = X(f_{n-1})$

2) Quantization is a simple Delta function with 2 levels depending on the error.

e quantized error



$$\underline{e}_n = \begin{cases} +2 & \text{for } e_n \geq 0 \\ -2 & \text{otherwise} \end{cases}$$

Example

$f_n : 10, 9, 15, 13, 6, 12, 15, 16, 18, 24, \dots$

lossless predictive coding  $f_n = f_{n-1}$ , i.e.,  $x = 1$   
 $e_n = f_n - f_n = f_n - f_{n-1}$

$e_n = 1, 6, -2, -7, 6, 1, 2, 6, \dots$

these  $e_n$ 's are then encoded using the symbol encoder

lossy predictive coding using diagram at the top (Delta mode)

$$f_n = d f_{n-1} = f_{n-1} \quad e_n = f_n - f_{n-1} = f_n - f_{n-1}$$

$\underline{e}_n : ???$

## JPEG Image Compression Standard

- standards are created by an international body of “experts” in a field
- CCITT was the initial body that developed the JPEG standard
- based on D.C.T.
- JTCL/ST../... JPEG2000 ! use wavelets
- Currently standards are being developed in the MPEG-4 and MPEG-7 areas

## Motion Encoding

H.261,H.263, MPEG-1, MPEG-2

## Multimedia (Encompassing Many Types of Media)

MPEG-4

## Motivation

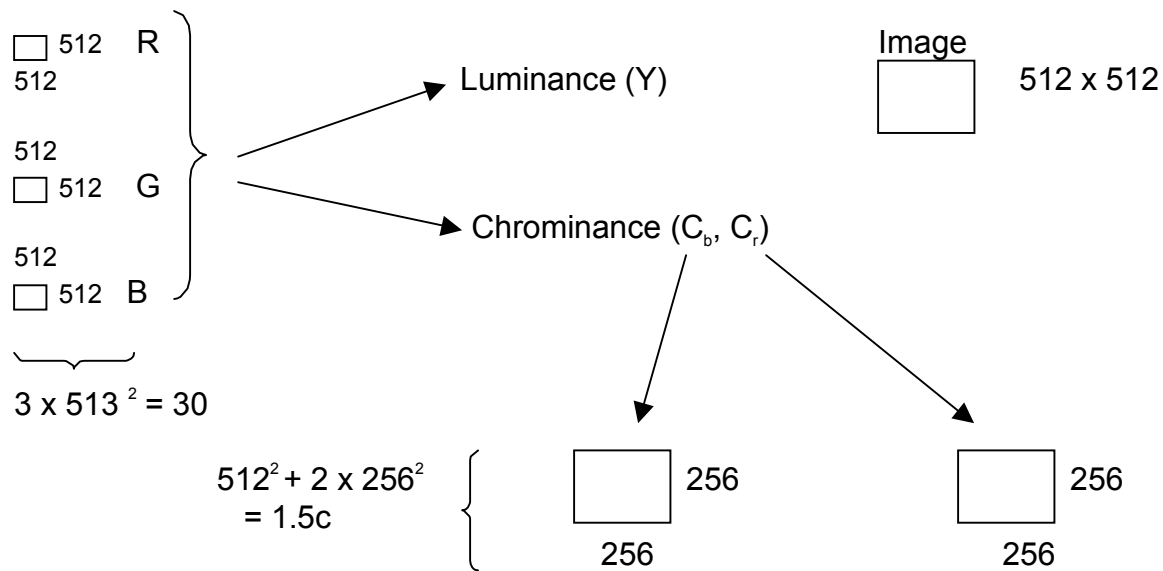
Why did the JPEG committee decide to choose certain options in their standard?

1) As size of image increases, transforms become more and more expensive. So, what sizes do we make the window and what transformation algorithm to use?

The JPEG committee chose to use an 8 x 8 DCT because MSE was not reduced significantly for larger windows (sub-image size) and DCT performed better than other well known transforms.

2) Human Eyes (Visual System) are more sensitive to luminance (brightness) changes rather than chrominance (color) changes.

Thus, JPEG has options for reducing the resolution of the chrominance part, compared to the luminance part.



As long as luminance is left unchanged, the chrominance change is less noticeable.

This saves 50% because

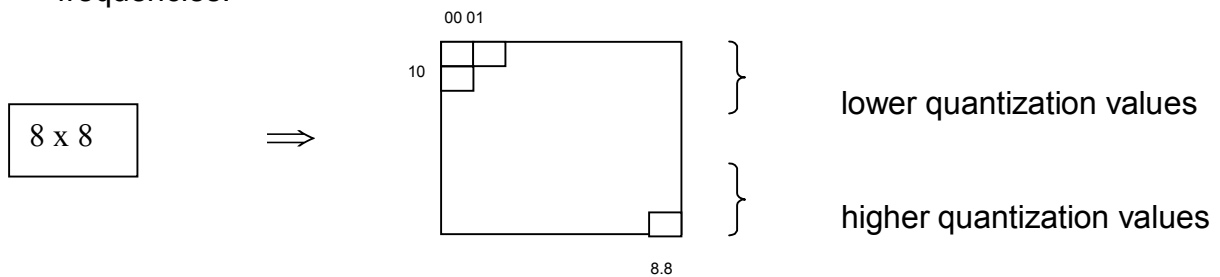
$$3 \times 512^2 = 3 S$$

initial

$$512^2 + (2 \times 256^2) = 1.5 S$$

final

- 3) Human eyes are more sensitive to loss at lower frequencies than loss at higher frequencies.





### JPEG Modes:

1) Lossless

2) Baseline Sequential

Every JPEG coder must support at least the Baseline standard

3) Progressive

4) Hierarchical

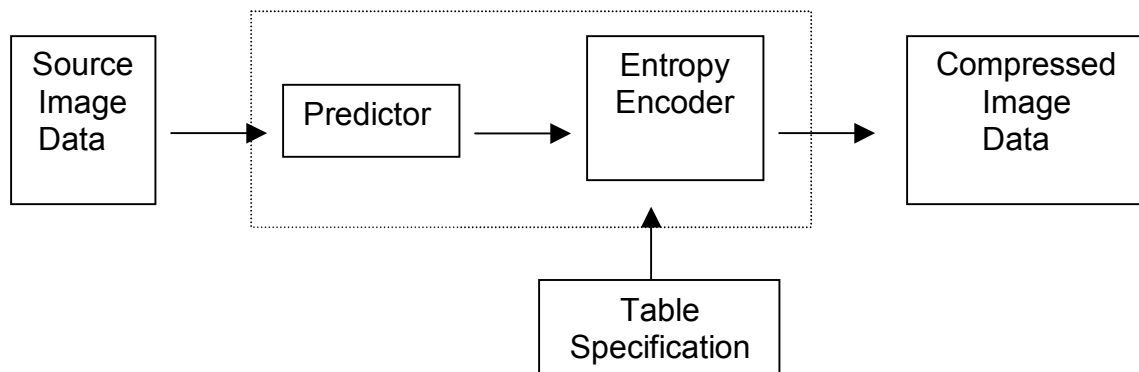
Methods 2, 3 and 4 are lossy transform coding based on DCT

Method 1 uses predictive coding → NO TRANSFORMATION

Progressive Mode: usually updates different frequency component (from LOW to HIGH) progressively.

Hierarchical Mode: create multi-resolution representation of images, and codes "Difference" between consecutive levels.

### Lossless Mode:

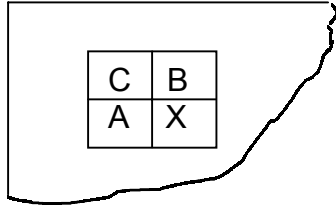


JPEG allows Huffman & Arithmetic encoding

Unlike audio, image compression must be 2 dimensional

### Prediction:

-Want to predict value of X given values of A,B,C in the 2 x 2 neighborhood.



### Predictors for lossless coding

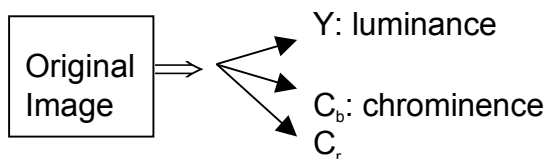
Predictor Code	Prediction
0	No Prediction
1	A
2	B
3	C
4	$A+B-C$
5	$A + [(B - C) / 2]$
6	$B + [(A - C) / 2]$
7	$(A+B) / 2$

example:

250	249
200	210

we can have positive as well as negative errors.

### Baseline Sequential Mode:



Similar compression methods are used for all components, with the following differences:

- (i) The Chrominance resolution is usually reduced
- (ii) The quantization tables are different for the chrominance and luminance parts.

OUTLINE of Steps in the JPEG Baseline Compression Method:

1. Transform the (R,G,B) image into 3 channels based on Luminance (Y) & Chrominance (Cb, Cr).
2. Each channel is divided into 8x8 blocks; images are padded if necessary to make rows and columns multiples of 8.
3. Each 8x8 image block is transformed into 8x8 frequency blocks by using the Discrete Cosine Transform (DCT).
4. The DCTs are usually computed to 11-bit precision if the R, G, B colors have 8-bit precision. This extra precision in DCT computation is kept to compensate for loss of accuracy when the frequency coefficients are divided by a Quantization matrix.
5. The DCT coefficients are ordered in a zig-zag format, starting with the top left corner corresponding to  $F(0,0)$ .
6. The DC coefficients ( $F(0,0)$ ) are coded separately from the other coefficients (AC coefficients).
7. The DC coefficients are coded from one block to the next using a predictive coding strategy. [The DC coefficient in one block can be predicted using the DC coefficient in the previous block.]
8. The AC coefficients are quantized in the zig-zag order, then treating these values as a sequence of bits a Run-length Coding method is used with the run-lengths being coded using Huffman or Arithmetic coding.
9. In PHOTOSHOP and other programs the amount of compression is controlled by a Quality parameter. A higher Quality implies that the values used for Quantization is smaller; this process can be achieved in two ways: (a) using a number of Quantization tables corresponding to different qualities, or (b) scaling a given quantization table depending on the quality required. A typical default JPEG quantization table is given below:

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