Image Compression: JPEG

Multimedia Systems (Module 4 Lesson 1)

Summary:

- JPEG Compression
  - DCT
  - Quantization
  - Zig-Zag Scan
  - RLE and DPCM
  - Entropy Coding
- JPEG Modes
  - Sequential
  - Lossless
  - Progressive
  - Hierarchical

Sources:

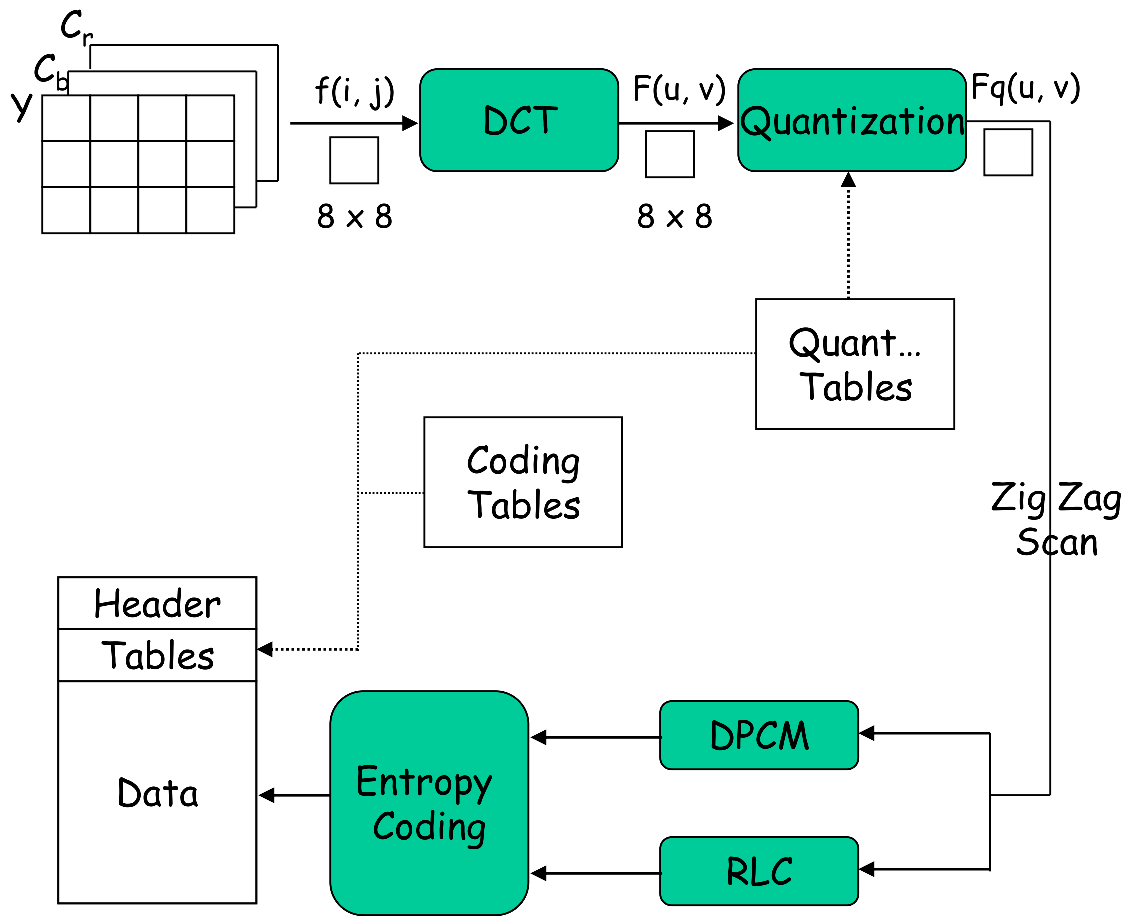
- The JPEG website:

  http://www.jpeg.org
- My research notes

Why JPEG

- The *compression ratio* of lossless methods (e.g., Huffman, Arithmetic, LZW) is not high enough for image and video compression.
- JPEG uses *transform coding*, it is largely based on the following observations:
  - Observation 1: A large majority of useful image contents change relatively slowly across images, i.e., it is unusual for intensity values to alter up and down several times in a small area, for example, within an 8 x 8 image block. A translation of this fact into the spatial frequency domain, implies, generally, lower spatial frequency components contain **more information** than the high frequency components which often correspond to less useful details and noises.
  - Observation 2: Experiments suggest that humans are more immune to loss of higher spatial frequency components than loss of lower frequency components.

JPEG Coding



Steps Involved:

1. Discrete Cosine Transform of each 8x8 pixel array $f(x,y) \rightarrow_T F(u,v)$
2. Quantization using a table or using a constant
3. Zig-Zag scan to exploit redundancy
4. Differential Pulse Code Modulation(DPCM) on the DC component and Run length Coding of the AC components
5. Entropy coding (Huffman) of the final output

DCT : Discrete Cosine Transform

DCT converts the information contained in a block(8x8) of pixels from ***spatial*** domain to the ***frequency*** domain.

- A simple analogy: Consider a unsorted list of 12 numbers between 0 and 3 -> (*2, 3, 1, 2, 2, 0, 1, 1, 0, 1, 0, 0*). Consider a transformation of the list involving two steps (1.) sort the list (2.) Count the frequency of occurrence of each of the numbers ->(*4,4,3,1*). : Through this transformation we lost the spatial information but captured the frequency information.
- There are other transformations which retain the spatial information. E.g., Fourier transform, DCT etc. Therefore allowing us to move back and forth between spatial and frequency domains.

1-D DCT:

$$F(\omega) = \frac{a(u)}{2} \sum_{n=0}^{N-1} f(n) \cos \frac{(2n+1)\omega\pi}{16}$$

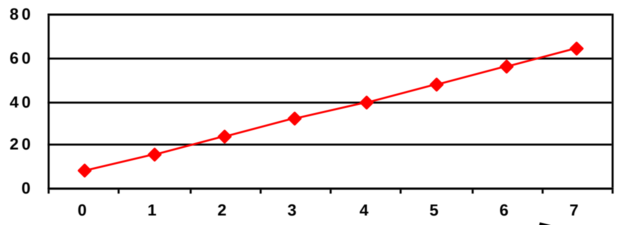
$$a(0) = \frac{1}{\sqrt{2}}$$

$$a(p) = 1 \quad [\, p \neq 0 \,]$$

1-D Inverese DCT:

$$f'(n) = \frac{a(u)}{2} \sum_{\omega=0}^{N-1} F(\omega) \cos \frac{(2n+1)\omega\pi}{16}$$

Example and Comparison



DCT

100	-52	0	-5	0	-2	0	0.4

100	-52	0	-5	~~0~~	~~-2~~	~~0~~	~~0.4~~

Inverse DCT

8	15	24	32	40	48	57	63

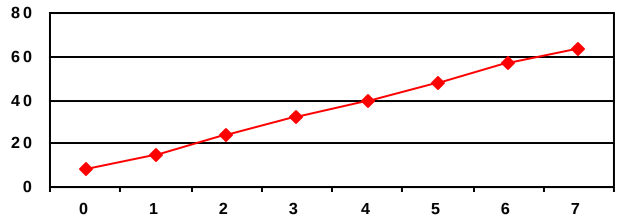
FFT

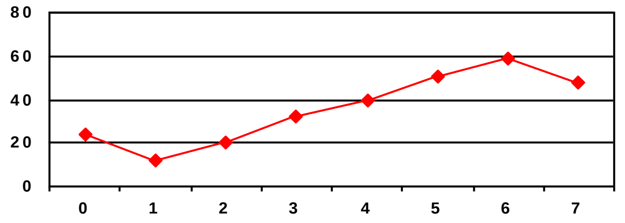
36	10	10	6	6	4	4	4

36	10	10	6	~~6~~	~~4~~	~~4~~	~~4~~

Inverse FFT

24	12	20	32	40	51	59	48



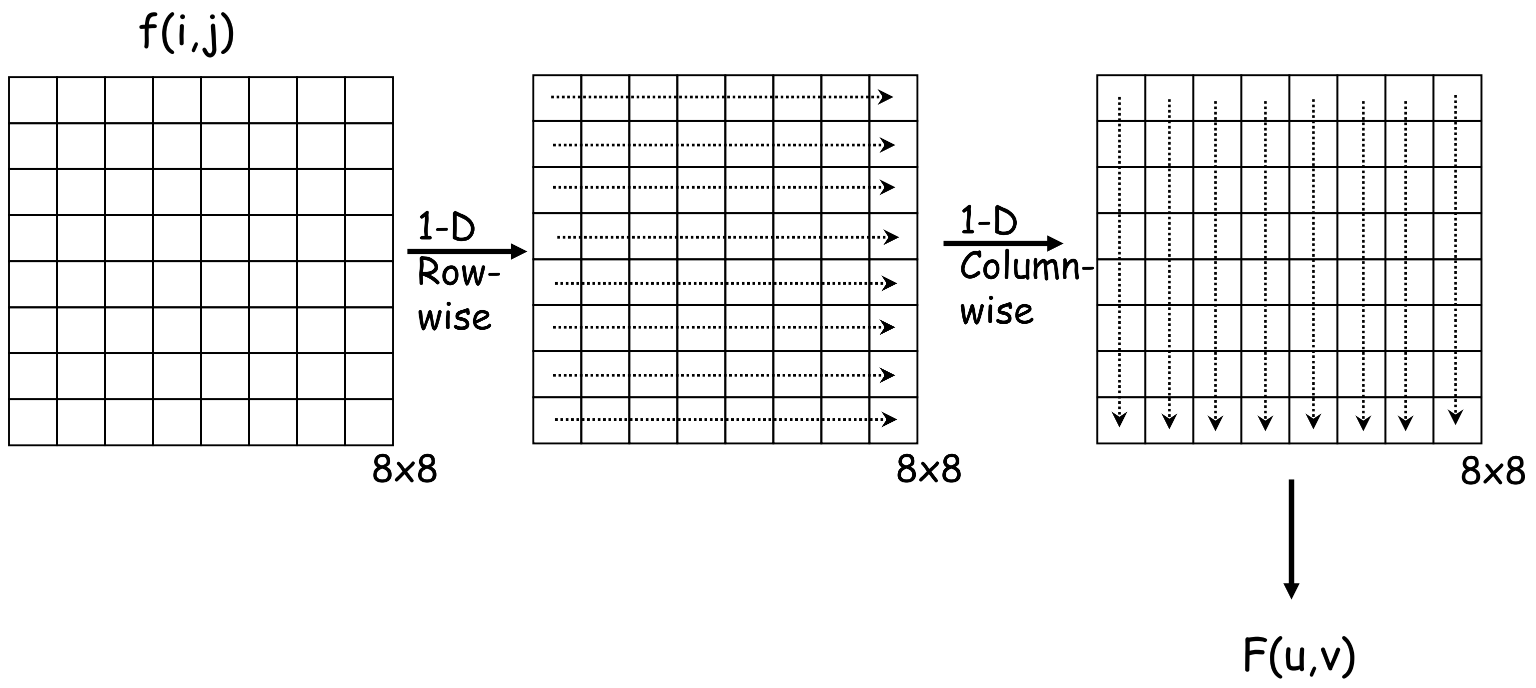


Example Description:

- f(n) is given from n = 0 to 7; (N=8)
- Using DCT(FFT) we compute F(ω) for ω = 0 to 7
- We truncate and use Inverse Transform to compute f'(n)

2-D DCT

- Images are two-dimensional; How do you perform 2-D DCT?
  - Two series of 1-D transforms result in a 2-D transform as demonstrated in the figure below



- F(0,0) is called the DC component and the rest of F(i,j) are called AC components

2-D Transform Example

- The following example will demonstrate the idea behind a 2-D transform by using our own cooked up transform: The transform computes a running cumulative sum.

f(i,j)

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

8x8

<u>My Transform:</u> $F_{my}(\omega) = \sum_{n=\omega}^{8} f(n)$

$\longrightarrow$ 1-D Row-wise

8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1
8	7	6	5	4	3	2	1

8x8

$\longrightarrow$ 1-D Column-wise

64	56	48	40	32	24	16	8
56	49	42	35	28	21	14	7
48	42	36	30	24	18	12	6
40	35	30	25	20	15	10	5
32	28	24	20	16	12	8	4
24	21	18	15	12	9	6	3
16	14	12	10	8	6	4	2
8	7	6	5	4	3	2	1

$F_{my}(u,v)$

8x8

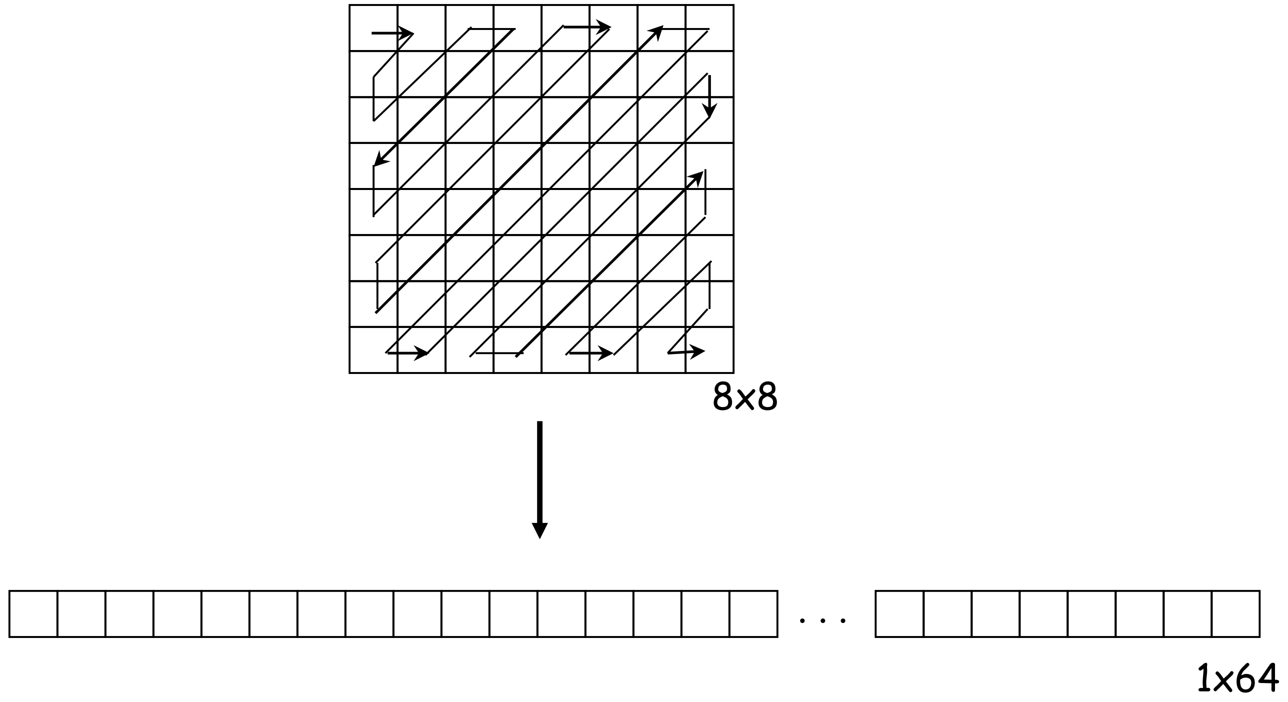
Note that this is only a hypothetical transform. Do not confuse this with DCT

Quantization

- Why? -- To reduce number of bits per sample
  F'(u,v) = round(F(u,v)/q(u,v))
- Example: 101101 = 45 (6 bits).
  Truncate to 4 bits: 1011 = 11. (Compare 11 x 4 =44 against 45)
  Truncate to 3 bits: 101 = 5. (Compare 8 x 5 =40 against 45)
  Note, that the more bits we truncate the more precision we lose
- Quantization error is the main source of the Lossy Compression.
- **Uniform Quantization:**
  - q(u,v) is a constant.
- **Non-uniform Quantization -- Quantization Tables**
  - Eye is most sensitive to low frequencies (upper left corner in frequency matrix), less sensitive to high frequencies (lower right corner)
  - Custom quantization tables can be put in image/scan header.
  - JPEG Standard defines two default quantization tables, one each for luminance and chrominance.

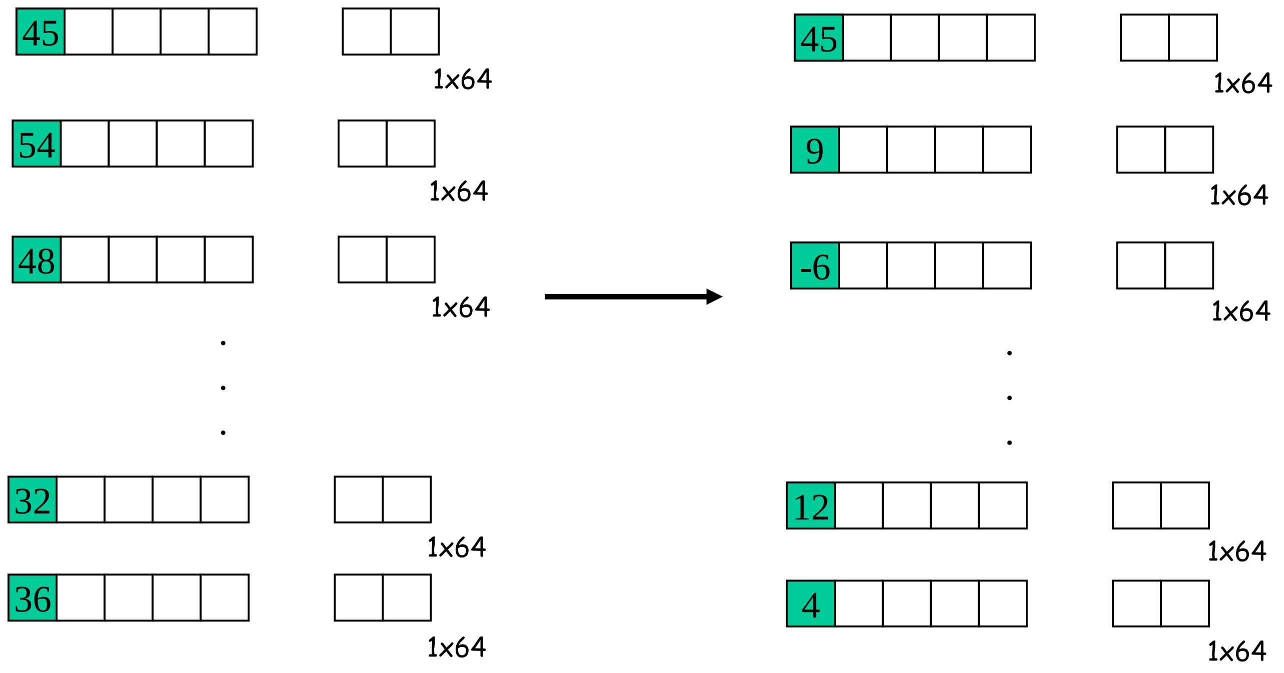
Zig-Zag Scan

- Why? -- to group low frequency coefficients in top of vector and high frequency coefficients at the bottom
- Maps 8 x 8 matrix to a 1 x 64 vector



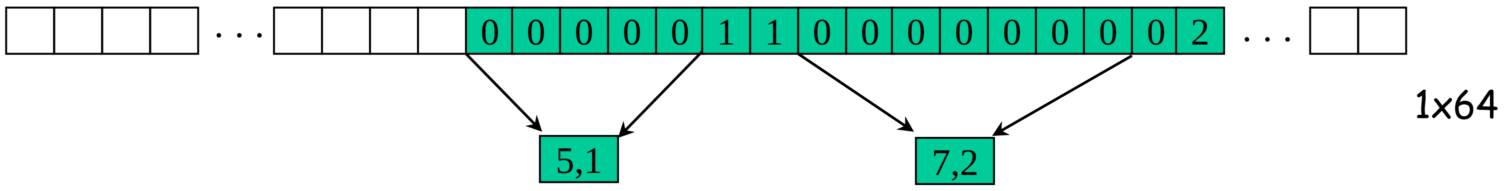
DPCM on DC Components

- The DC component value in each 8x8 block is large and varies across blocks, but is often close to that in the previous block.
- Differential Pulse Code Modulation (DPCM): Encode the difference between the current and previous 8x8 block. Remember, smaller number -> fewer bits



RLE on AC Components

- The 1x64 vectors have a lot of zeros in them, more so towards the end of the vector.
  - Higher up entries in the vector capture higher frequency (DCT) components which tend to be capture less of the content.
  - Could have been as a result of using a quantization table
- Encode a series of 0s as a (*skip*,*value*) pair, where *skip* is the number of zeros and *value* is the next non-zero component.
  - Send (0,0) as end-of-block sentinel value.



Entropy Coding: DC Components

- DC components are differentially coded as (**SIZE**,**Value**)
  - The code for a **Value** is derived from the following table

SIZE	**Value**	**Code**
0	0	---
1	-1,1	0,1
2	-3, -2, 2,3	00,01,10,11
3	-7,…, -4, 4,…, 7	000,…, 011, 100,…111
4	-15,…, -8, 8,…, 15	0000,…, 0111, 1000,…, 1111
.		.
.		.
11	-2047,…, -1024, 1024,… 2047	…

Size_and_Value Table

Entropy Coding: DC Components (Contd..)

- DC components are differentially coded as (**SIZE**,**Value**)
  - The code for a **SIZE** is derived from the following table

SIZE	**Code Length**	**Code**
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Huffman Table for DC component SIZE field

Example: If a DC component is 40 and the previous DC component is 48. The difference is -8. Therefore it is coded as:

1010111

0111: The value for representing –8 (see Size_and_Value table)

101: The size from the same table reads 4. The corresponding code from the table at left is 101.

Entropy Coding: AC Components

- AC components (range -1023..1023) are coded as (S1,S2 pairs):
  - **S1: (RunLength/SIZE)**
    - **RunLength:** The length of the consecutive zero values [0..15]
    - **SIZE:** The number of bits needed to code the *next* nonzero AC component's value. [0-A]
    - (0,0) is the End_Of_Block for the 8x8 block.
    - **S1** is Huffman coded (see AC code table below)
  - **S2: (Value)**
    - **Value:** Is the value of the AC component.(refer to size_and_value table)

Partial Huffman Table for AC Run/Size Pairs

Run/ SIZE	Code Length	Code
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

Run/ SIZE	Code Length	Code
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
… 15/A	More	Such rows

Entropy Coding: Example

40	12	0	0	0	0	0	0
10	-7	-4	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Example: Consider encoding the AC components by arranging them in a zig-zag order -> 12,10, 1, -7 2 0s, -4, 56 zeros

12: read as zero 0s,12: (0/4)12 → 10111100

1011: The code for (0/4 from AC code table)

1100: The code for 12 from the Size_and_Value table.

10: (0/4)10 → 10111010

1: (0/1)1 → 001

-7: (0/3)-7 → 100000

2 0s, -4: (2/3)-4 → 1111110111011

1111110111: The 10-bit code for 2/3

011: representation of –4 from Size_and_Value table.

56 0s: (0,0) → 1010 (Rest of the components are zeros therefore we simply put the EOB to signify this fact)

JPEG Modes

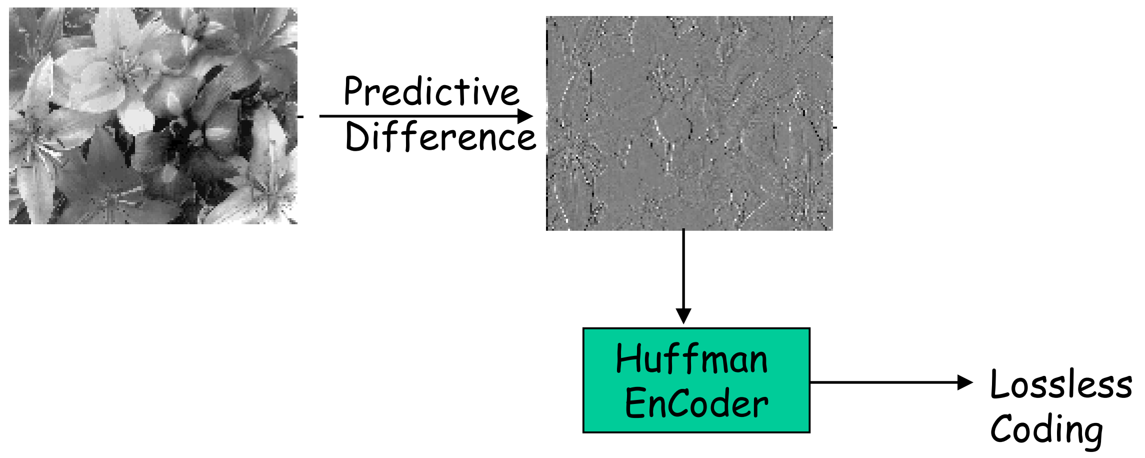
Sequential Mode:

- Each image is encoded in a single left-to-right, top-to-bottom scan.
  - The technique we have been discussing so far is an example of such a mode, also referred to as the **Baseline Sequential Mode.**
  - It supports only 8-bit images as opposed to 12-bit images as described before.

JPEG Modes

Lossless Mode:

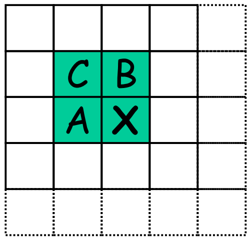
- Truly lossless
- It is a predictive coding mechanism as opposed to the baseline mechanism which is based on DCT and quantization(the source of the loss).
- Here is the simple block diagram of the technique:



Lossless Mode (Contd..)

Predictive Difference:

- For each pixel a predictor (one of 7 possible) is used that *best* predicts the value contained in the pixel as a combination of up to 3 neighboring pixels.
- The difference between the predicted value and the actual value (**X**)contained in the pixel is used as the *predictive difference* to represent the pixel.
- The predictor along with the predictive difference are encoded as the pixel's content.
- The series of pixel values are encoded using huffman coding



Predictor	Prediction
P1	A
P2	B
P3	C
P4	A+B-C
P5	A + (B-C)/2
P6	B + (A-C)/2
P7	(A+B)/2

Notes:

- The very first pixel in location (0, 0) will always use itself.
- Pixels at the first row always use P1,
- Pixels at the first column always use P2.
- The best (of the 7) predictions is always chosen for any pixel.

JPEG Modes

Progressive Mode: It allows a coarse version of an image to be transmitted at a low rate, which is then progressively improved over subsequent transmissions.

- *Spectral Selection* : Send DC component and first few AC coefficients first, then gradually some more ACs.

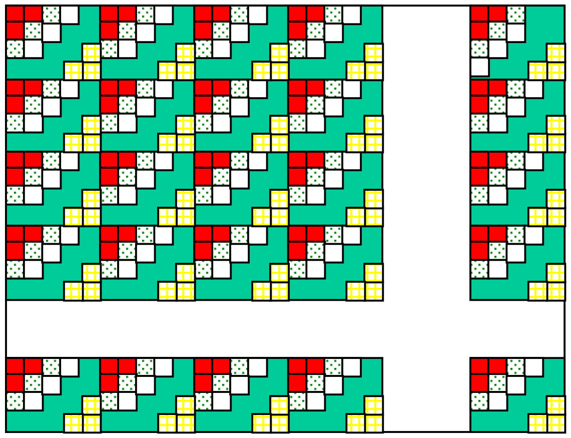
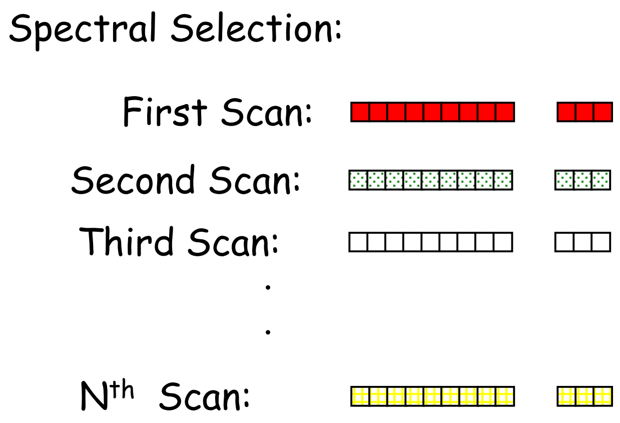
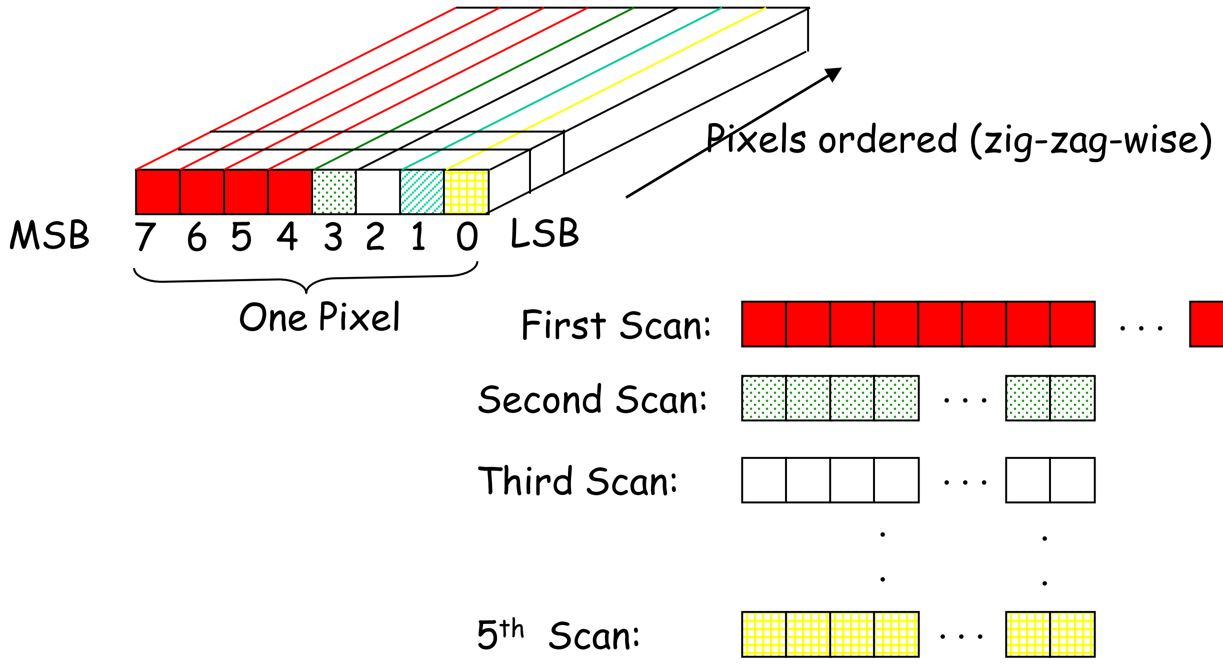


Image Pixels



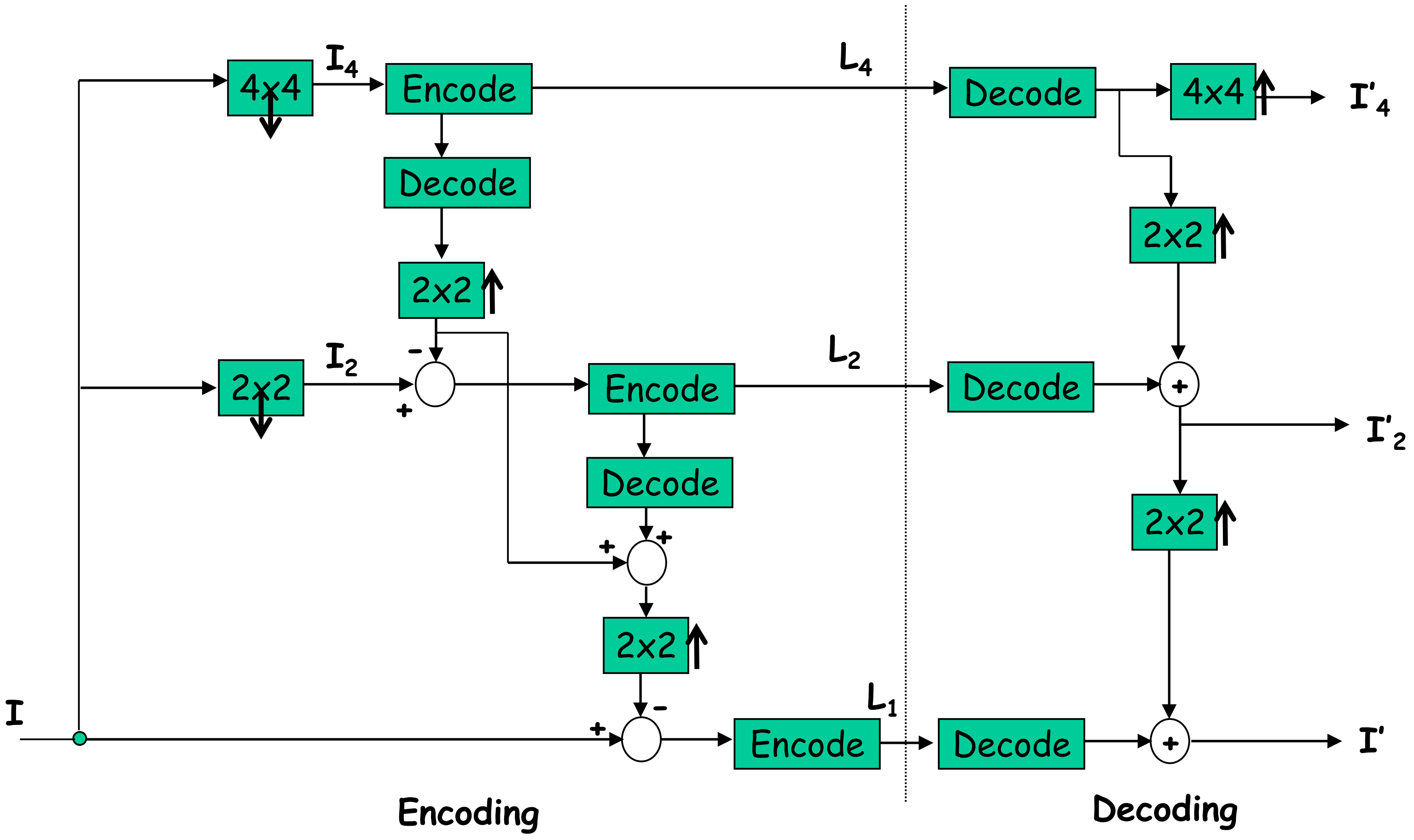
Progressive Mode

- *Successive Approximation* : All the DCT components are sent few bits at a time: For example, send n1 (say,4) bits (starting with MSB) of all pixels in the first scan, the next n2(say 1) bits of all pixels in the second and so on.



Hierarchical Mode

- Used primarily to support multiple resolutions of the same image which can be chosen from depending on the target's capabilities.
- The figure here shows a description of how a 3-level hierarchical encoder/decoder works:



I
4x4 ↓
I4
Encode
L4
Decode
2x2 ↑
2x2 ↓
I2
-
+
Encode
L2
Decode
+
+
2x2 ↑
-
+
Encode
L1
Decode
4x4 ↑
I'4
2x2 ↑
+
I'2
2x2 ↑
+
I'
Encoding
Decoding