Lecture 14  Image Compression

1. What and why image compression
2. Basic concepts
3. Encoding/decoding, entropy
What is Data and Image Compression?

- Data compression is the art and science of representing information in a compact form.
- Data is a sequence of symbols taken from a discrete alphabet.
Why do we need Image Compression?

Still Image
• One page of A4 format at 600 dpi is > 100 MB.
• One color image in digital camera generates 10-30 MB.
• Scanned 3”×7” photograph at 300 dpi is 30 MB.

Digital Cinema
• $4K\times2K\times3 \times 12 \text{ bits/pel} = 48 \text{ MB/frame or 1 GB/sec}$
  or 70 GB/min.
Why do we need Image Compression?

1) Storage
2) Transmission
3) Data access
   1990-2000
   Disc capacities : 100MB -> 20 GB (200 times!)
   but seek time    : 15 milliseconds \(\rightarrow\) 10 milliseconds
   and transfer rate: 1MB/sec \(\rightarrow\) 2 MB/sec.

Compression improves overall response time in some applications.
Source of images

• Image scanner
• Digital camera
• Video camera,
• Ultra-sound (US), Computer Tomography (CT), Magnetic resonance image (MRI), digital X-ray (XR), Infrared.
• etc.
Image types

IMAGE COMPRESSION

- Video images
- Gray-scale images
- Binary images
- True colour images
- Colour palette images

UNIVERSAL COMPRESSION

- Textual data
Why we can compress image?

- Statistical redundancy:
  1) Spatial correlation
     a) Local - Pixels at neighboring locations have similar intensities.
     b) Global - Reoccurring patterns.
  2) Spectral correlation – between color planes.
  3) Temporal correlation – between consecutive frames.

- Tolerance to fidelity:
  1) Perceptual redundancy.
  2) Limitation of rendering hardware.
Lossy vs. Lossless compression

Lossless compression: reversible, information preserving
text compression algorithms,
binary images, palette images

Lossy compression: irreversible
grayscale, color, video

Near-lossless compression:
medical imaging, remote sensing.
Rate measures

Bitrate: \( \frac{\text{size of the compressed file}}{\text{pixels in the image}} = \frac{C}{N} \) bits/pel

Compression ratio: \( \frac{\text{size of the original file}}{\text{size of the compressed file}} = \frac{N \cdot k}{C} \)
Distortion measures

Mean average error (MAE): 
\[ MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - x_i| \]

Mean square error (MSE): 
\[ MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i)^2 \]

Signal-to-noise ratio (SNR): 
\[ SNR = 10 \cdot \log_{10} \left[ \frac{\sigma^2}{MSE} \right] \] (decibels)

Pulse-signal-to-noise ratio (PSNR): 
\[ PSNR = 10 \cdot \log_{10} \left[ \frac{A^2}{MSE} \right] \] (decibels)

A is amplitude of the signal: \( A = 2^8 - 1 = 255 \) for 8-bits signal.
Other issues

- Coder and decoder computation complexity
- Memory requirements
- Fixed rate or variable rate
- Error resilience
- Symmetric or asymmetric
- Decompress at multiple resolutions
- Decompress at various bit rates
- Standard or proprietary
Entropy

Set of symbols (alphabet) $S=\{s_1, s_2, \ldots, s_N\}$, $N$ is number of symbols in the alphabet.

Probability distribution of the symbols: $P=\{p_1, p_2, \ldots, p_N\}$

According to Shannon, the entropy $H$ of an information source $S$ is defined as follows:

$$H = -\sum_{i=1}^{N} p_i \cdot \log_2(p_i)$$
Entropy

The amount of information in symbol $s_i$, in other words, the number of bits to code or code length for the symbol $s_i$:

$$H(s_i) = -\log_2(p_i)$$

The average number of bits for the source $S$:

$$H = -\sum_{i=1}^{N} p_i \cdot \log_2(p_i)$$
Entropy for binary source: N=2

\( S = \{0, 1\} \)

\( p_0 = p \)

\( p_1 = 1 - p \)

\[
H = -(p \cdot \log_2 p + (1 - p) \cdot \log_2 (1 - p))
\]

H=1 bit for \( p_0 = p_1 = 0.5 \)
Entropy for uniform distribution: $p_i=1/N$

Uniform distribution of probabilities: $p_i=1/N$:

$$H = -\sum_{i=1}^{N} \left(\frac{1}{N}\right) \cdot \log_2 \left(\frac{1}{N}\right) = \log_2(N)$$

Examples:

- $N=2$: $p_i=0.5$; $H=\log_2(2) = 1$ bit
- $N=256$: $p_i=1/256$; $H=\log_2(256) = 8$ bits
How to get the probability distribution?

1) Static modeling:
   a) The same code table is applied to all input data.
   b) One-pass method (encoding)
   c) No side information

2) Semi-adaptive modeling:
   a) Two-pass method: (1) analysis and (2) encoding.
   b) Side information needed (model, code table)

3) Adaptive (dynamic) modeling:
   a) One-pass method: analysis and encoding
   b) Updating the model during encoding/decoding
   c) No side information
Static vs. Dynamic: Example

\[ S = \{a, b, c\}; \ \text{Data: } a, a, b, a, a, c, a, a, b, a. \]

1) Static model: \( p_i = 1/10 \)

\[ H = -\log_2(1/10) = 1.58 \text{ bits} \]

2) Semi-adaptive method: \( p_1 = 7/10; \ p_2 = 2/10; \ p_3 = 1/10; \)

\[ H = -(0.7 \log_2 0.7 + 0.2 \log_2 0.2 + 0.1 \log_2 0.1) = 1.16 \text{ bits} \]
3) Adaptive method: Example

\[ S = \{a,b,c\}; \text{ Data: } a,a,b,a,a,c,a,a,b,a. \]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( p_i )</td>
<td>0.33</td>
<td>0.5</td>
<td>0.2</td>
<td>0.5</td>
<td>0.57</td>
<td>0.13</td>
<td>0.56</td>
<td>0.60</td>
<td>0.18</td>
<td>0.58</td>
</tr>
<tr>
<td>( H )</td>
<td>1.58</td>
<td>1.0</td>
<td>2.32</td>
<td>1.0</td>
<td>0.81</td>
<td>3.0</td>
<td>0.85</td>
<td>0.74</td>
<td>2.46</td>
<td>0.78</td>
</tr>
</tbody>
</table>

\[ H = \frac{1}{10}(1.58 + 1.0 + 2.32 + 1.0 + 0.81 + 3.0 + 0.85 + 0.74 + 2.46 + 0.78) \]

= 1.45 bits/char

1.16 < 1.45 < 1.58

S.-Ad.    Ad.    Static
Coding methods

- Shannon-Fano Coding
- Huffman Coding
- Predictive coding
- Block coding

- Arithmetic code
- Golomb-Rice codes
Shannon-Fano Code: A top-down approach

1) Sort symbols according to their probabilities:
   \[ p_1 \leq p_2 \leq \ldots \leq p_N \]

2) Recursively divide into parts, each with approx. the same number of counts (probability)
Shannon-Fano Code: Example (1 step)

<table>
<thead>
<tr>
<th>sᵢ</th>
<th>pᵢ</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15/39</td>
</tr>
<tr>
<td>B</td>
<td>7/39</td>
</tr>
<tr>
<td>C</td>
<td>6/39</td>
</tr>
<tr>
<td>D</td>
<td>6/39</td>
</tr>
<tr>
<td>E</td>
<td>5/39</td>
</tr>
</tbody>
</table>

A, B, C, D, E

A, B
15 + 7 = 22

C, D, E
6 + 6 + 5 = 17

0

1
Shannon-Fano Code: Example (2 step)

<table>
<thead>
<tr>
<th>s_i</th>
<th>p_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15/39</td>
</tr>
<tr>
<td>B</td>
<td>7/39</td>
</tr>
<tr>
<td>C</td>
<td>6/39</td>
</tr>
<tr>
<td>D</td>
<td>6/39</td>
</tr>
<tr>
<td>E</td>
<td>5/39</td>
</tr>
</tbody>
</table>

A, B, C, D, E

00 11

A, B

15 + 7 = 22

A

15

B

7

C, D, E

6 + 6 + 5 = 17

C

6

D, E

6 + 5 = 11
Shannon-Fano Code: Example (3 step)

<table>
<thead>
<tr>
<th>$s_i$</th>
<th>$p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15/39</td>
</tr>
<tr>
<td>B</td>
<td>7/39</td>
</tr>
<tr>
<td>C</td>
<td>6/39</td>
</tr>
<tr>
<td>D</td>
<td>6/39</td>
</tr>
<tr>
<td>E</td>
<td>5/39</td>
</tr>
</tbody>
</table>

A, B, C, D, E
15, 7, 6, 6, 5

A, B
15 + 7 = 22

C, D, E
6 + 6 + 5 = 17

D, E
6 + 5 = 11
### Shannon-Fano Code: Example (Result)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>$p_i$</th>
<th>$-\log_2(p_i)$</th>
<th>Code</th>
<th>Subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15/39</td>
<td>1.38</td>
<td>00</td>
<td>2*15</td>
</tr>
<tr>
<td>B</td>
<td>7/39</td>
<td>2.48</td>
<td>01</td>
<td>2*7</td>
</tr>
<tr>
<td>C</td>
<td>6/39</td>
<td>2.70</td>
<td>10</td>
<td>2*6</td>
</tr>
<tr>
<td>D</td>
<td>6/39</td>
<td>2.70</td>
<td>110</td>
<td>3*6</td>
</tr>
<tr>
<td>E</td>
<td>5/39</td>
<td>2.96</td>
<td>111</td>
<td>3*5</td>
</tr>
</tbody>
</table>

Total: 89 bits

![Binary tree](image)

$$H = \frac{89}{39} = 2.28 \text{ bits}$$
## Shannon-Fano Code: Encoding

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
</tr>
<tr>
<td>B</td>
<td>01</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>110</td>
</tr>
<tr>
<td>E</td>
<td>111</td>
</tr>
</tbody>
</table>

**Message:** B A B A C A C A D E

**Codes:** 01 00 01 00 10 00 10 00 110 111

**Bitstream:** 0100010010001000110111

**Binary tree:**

```
     1
    / \
   0   0
  /     \   \
A      1   1
       /   \   \   
      0     0    1   1
     /     /       /   \
    B     C       D    E
```

**Binary tree:**

- **Root:** 1
- **Left child:** 0
  - **Left grandchild:** A
  - **Right grandchild:** B
- **Right child:** 0
  - **Left grandchild:** C
  - **Right grandchild:** (null)
- **Right child:** 1
  - **Right grandchild:** D
  - **Right grandchild:** E
Shannon-Fano Code: Decoding

A - 00
B - 01
C - 10
D - 110
E - 111

Bitstream: 0100010010001000110111 (23 bits)

Codes: 01 00 01 00 10 00 10 00 110 111

Message: B A B A C A C A D E

Binary tree

A

0
B
1
C

0
D
1
E
1

0

1
Huffman Code: A bottom-up approach

INIT:
Put all nodes in an OPEN list, keep it sorted all times according their probabilities;

REPEAT
a) From OPEN pick two nodes having the lowest probabilities, create a parent node of them.
   b) Assign the sum of the children’s probabilities to the parent node and inset it into OPEN
   c) Assign code 0 and 1 to the two branches of the tree, and delete the children from OPEN.
Huffman Code: Example

<table>
<thead>
<tr>
<th>Symbol</th>
<th>$p_i$</th>
<th>$-\log_2(p_i)$</th>
<th>Code</th>
<th>Subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15/39</td>
<td>1.38</td>
<td>0</td>
<td>2*15</td>
</tr>
<tr>
<td>B</td>
<td>7/39</td>
<td>2.48</td>
<td>100</td>
<td>3*7</td>
</tr>
<tr>
<td>C</td>
<td>6/39</td>
<td>2.70</td>
<td>101</td>
<td>3*6</td>
</tr>
<tr>
<td>D</td>
<td>6/39</td>
<td>2.70</td>
<td>110</td>
<td>3*6</td>
</tr>
<tr>
<td>E</td>
<td>5/39</td>
<td>2.96</td>
<td>111</td>
<td>3*5</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>87 bits</strong></td>
</tr>
</tbody>
</table>

H = 87/39 = 2.23 bits
Huffman Code: Decoding

A - 0
B - 100
C - 101
D - 110
E - 111

Bitstream: 1000100010101010110111 (22 bits)
Codes: 100 0 100 0 101 0 101 0 110 111
Message: B A B A C A C A D E

Binary tree
Properties of Huffman code

- Optimum code for a given data set requires two passes.
- Code construction complexity $O(N \log N)$.
- Fast lookup table based implementation.
- Requires at least one bit per symbol.
- Average codeword length is within one bit of zero-order entropy (Tighter bounds are known): $H \leq R < H+1$ bit
- Susceptible to bit errors.
Unique prefix property

No code is a prefix to any other code, all symbols are the leaf nodes

Shannon-Fano and Huffman codes are prefix codes

Legend: Shannon (1948) and Fano (1949); Huffman (1952) was student of Fano at MIT. Fano: "Construct minimum-redundancy code → final exam is passed!"
Predictive coding

1) Calculate prediction value: \( y_i = f(\text{neighbourhood of } x_i) \).
2) Calculating the prediction error: \( e_i = y_i - x_i \).
3) Encode the prediction error \( e_i \).
Predictive model for grayscale images

\[ y = x_i - x_{i-1} \]

Histogram of the original image and Residual image

Entropy: \( H_o = 7.8 \text{ bits/pel} \)  
\[ H_r = 5.1 \text{ bits/pel} \]
Coding without prediction

\[ f_0 = 8; \quad p_0 = p = \frac{8}{64} = 0.125; \]
\[ f_1 = 56; \quad p_1 = (1-p) = \frac{56}{64} = 0.875 \]

Entropy:

\[ H = -\left(\frac{8}{64} \log_2 \left(\frac{8}{64}\right) + \frac{56}{64} \log_2 \left(\frac{56}{64}\right)\right) = 0.544 \text{ bits/pel} \]
Prediction for binary images by pixel above

Entropy:

\[ H = -\left(\frac{16}{64}\log_2\frac{16}{64} + \frac{48}{64}\log_2\frac{48}{64}\right) = 0.811 \text{ bits/pel} \]

Wrong predictor!
Prediction for binary images pixel to the left

Entropy:

\[ H = -\left(\frac{1}{64}\log_2\left(\frac{1}{64}\right) + \frac{63}{64}\log_2\left(\frac{63}{64}\right)\right) = 0.116 \text{ bits/pel} \]

Good predictor!
Comparison of predictors:

- Without prediction: $H = 0.544$ bits/pel
- Prediction by pixel above: $H = 0.811$ bits/pel (bad!)
- Prediction by pixel to the left: $H = 0.116$ bits/pel (good!)
Shortcoming of Huffman codes

Alphabet: a, b.
$p_a = p = 0.99$,  $p_b = q = 0.01$

1) Entropy

\[ H_1 = - (p \log_2(p) + q \log_2(q)) = 0.081 \text{ bits/pel} \]

2) Huffman code: $p_a = '0'$,  $p_b = '1'$

Bitrate  $R_1 = 1*p + 1*q = p + q = 1 \text{ bit/pel!}$

Make a new alphabet blocking symbols!
Block coding: \( n=2 \)

New alphabet: 'A'='aa', 'B'='ab', 'C'='ba', 'D'='bb'

\[
p_A=p^2=0.9801, \ p_B=pq=0.0099, \ p_C=pq=0.0099, \ p_D=q^2=0.0001
\]

1) Entropy: \[
H_2 = -(0.9801 \log_2(0.9801) + 0.0099 \log_2(0.0099) + 0.0099 \log_2(0.0099) + 0.0001 \log_2(0.0001)) = \\
= (0.0284 + 0.0659 + 0.0659 + 0.0013)/2 = \textbf{0.081} \text{ bits/pel}
\]

Why \( H_2 = H_1 \)?

2) Huffman code: \( c_A='0', \ c_B='10', \ c_C='110', \ c_D='111' \)

\[
L_A=1, \ L_B=2, \ L_C=2, \ L_D=3
\]

Bitrate \( R_2 = (1*p_A+2*p_B+3*p_C+3*p_D)/2 = \textbf{0.515} \text{ bits/pel} \)
Block coding: n=3

'A'='aaa' -> \( p_A = p^3 \)
'B'='aab', 'C'='aba', 'D'='baa' -> \( p_B = p_C = p_D = p^2q \)
'E'='abb', 'F'='bab', 'G'='bba' -> \( p_E = p_F = p_G = pq^2 \)
'H'='bbb' -> \( p_H = q^3 \)

Huffman code:
\[ c_A = '0', \quad c_B = '10', \quad c_C = '110', \quad c_D = '1110' \]
\[ c_E = '111100', \quad c_F = '111101', \quad c_G = '111110', \quad c_H = '111111' \]

Entropy \( H_3 \)?

Bitrate:
\[ R_3 = \frac{(1*p_A + 2*p_B + 3*p_C + 4*p_D + 6*(p_E + p_F + p_G + p_H))}{3} = \]
\[ = 0.353 \text{ bits/pel} \]
Block coding: $n \rightarrow \infty$

$p_a = p = 0.99, \quad p_b = q = 0.01$

Entropy $H_n = 0.081$ bits/pel

Bitrate for Huffman coder:

- $n = 1$: $R_1 = 1.0$ bit
- $n = 2$: $R_2 = 0.515$ bits
- $n = 3$: $R_3 = 0.353$ bits

If block size $n \rightarrow \infty$?

$H_n \leq R_n < H_n + \frac{1}{n}$

Problem - alphabet size and Huffman table size grows exponentially with number of symbols $n$ blocked.
Block coding: Example 2, n=1

1) Entropy

\[ H = - \left( \frac{56}{64} \log_2 \left( \frac{56}{64} \right) + \frac{8}{64} \log_2 \left( \frac{8}{64} \right) \right) = 0.544 \text{ bits/pel} \]

2) Huffman code: a = '0'; b = '1'

Bitrate: R = 1 bit/pel
Block coding: Example 2, n=4

1) Entropy

\[ H = -\left( \frac{12}{16} \log_2 \left( \frac{12}{16} \right) + \frac{4}{16} \log_2 \left( \frac{4}{16} \right) \right) / 4 = 0.203 \text{ bits/pel} \]

2) Huffman code: A='0', B='1'

Bitrate \( R = (1 \cdot p_A + 1 \cdot p_B) / 4 = 0.250 \text{ bits/pel} \)
Binary image compression

- Run-length coding
- Predictive coding
- READ code
- Block coding
- G3 and G4
- JBIG: Prepared by Joint Bi-Level Image Expert Group in 1992
Compressed file size

Model size

n=1: Model size: \(p_a, p_b\) → \(2^1 \times 8\) bits

n=2: Model size: \(p_A, p_B, p_C, p_D\) → \(2^2 \times 8\) bits

n=k: Model size: \(\{p_A, p_B, \ldots, p_D\}\) → \(2^k \times 8\) bits

Compressed data size for S symbols in input file:

\[R \times S\] bits, where R is bitrate (bits/pel)

Total size: Model size + \(R \times S\) bits

Difference between entropy H and bitrate R!
Run-length coding idea

- Pre-processing method, good when one symbol occurs with high probability or when symbols are dependent
- Count how many repeated symbol occur
- Source 'symbol' = length of run

Example: …, 4b, 9w, 2b, 2w, 6b, 6w, 2b, ...
Run-length encoding: CCITT standard

Resolution:
Image: 1728*1,188 or 2 Mbytes
Transmission time: T=7 min
Run-length encoding: Example

<table>
<thead>
<tr>
<th>$n$</th>
<th>white runs</th>
<th>black runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00110101</td>
<td>0000110111</td>
</tr>
<tr>
<td>1</td>
<td>000111</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>0111</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>1011</td>
<td>011</td>
</tr>
<tr>
<td>5</td>
<td>1100</td>
<td>0011</td>
</tr>
<tr>
<td>6</td>
<td>1110</td>
<td>0010</td>
</tr>
<tr>
<td>7</td>
<td>1111</td>
<td>00011</td>
</tr>
<tr>
<td>8</td>
<td>10011</td>
<td>000101</td>
</tr>
<tr>
<td>9</td>
<td>10100</td>
<td>000100</td>
</tr>
<tr>
<td>10</td>
<td>00111</td>
<td>0000100</td>
</tr>
<tr>
<td>11</td>
<td>01000</td>
<td>0000101</td>
</tr>
<tr>
<td>12</td>
<td>001000</td>
<td>0000111</td>
</tr>
</tbody>
</table>

RL Code
- 4 b '011'
- 9 w '10100'
- 2 b '11'
- 2 w '0111'
- 6 b '0010'
- 6 w '1110'
- 2 b '11'
Run-length Huffman encoding: $0 \leq n \leq 63$

<table>
<thead>
<tr>
<th>$n$</th>
<th>white runs</th>
<th>black runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00111011</td>
<td>0000111101</td>
</tr>
<tr>
<td>1</td>
<td>000111</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>0111</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>1011</td>
<td>011</td>
</tr>
<tr>
<td>5</td>
<td>1100</td>
<td>0011</td>
</tr>
<tr>
<td>6</td>
<td>1110</td>
<td>0010</td>
</tr>
<tr>
<td>7</td>
<td>1111</td>
<td>00011</td>
</tr>
<tr>
<td>8</td>
<td>10011</td>
<td>000101</td>
</tr>
<tr>
<td>9</td>
<td>10100</td>
<td>000100</td>
</tr>
<tr>
<td>10</td>
<td>00111</td>
<td>0000100</td>
</tr>
<tr>
<td>11</td>
<td>01000</td>
<td>0000101</td>
</tr>
<tr>
<td>12</td>
<td>001000</td>
<td>0000111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>0000011</td>
<td>00000110111</td>
</tr>
<tr>
<td>23</td>
<td>0000100</td>
<td>00000101000</td>
</tr>
<tr>
<td>24</td>
<td>0101000</td>
<td>00000010111</td>
</tr>
<tr>
<td>25</td>
<td>0101011</td>
<td>00000011000</td>
</tr>
<tr>
<td>26</td>
<td>0100111</td>
<td>00000110010</td>
</tr>
<tr>
<td>27</td>
<td>0100100</td>
<td>00001100111</td>
</tr>
<tr>
<td>28</td>
<td>0011000</td>
<td>00011001100</td>
</tr>
<tr>
<td>29</td>
<td>00000010</td>
<td>00011001101</td>
</tr>
<tr>
<td>30</td>
<td>00000011</td>
<td>00000110100</td>
</tr>
<tr>
<td>31</td>
<td>00011010</td>
<td>00000011000</td>
</tr>
</tbody>
</table>
Run-length Huffman encoding: $n > 63$

Examples:
- $n=30w$: code='00000011'
- $n=94w=64w+30w$: code='1101100000000011'
- $n=64w=64w+0w$: code='1101100110101'
Predictive coding: Idea

• Predict the pixel value on the basis of past pixel(s)
• Send ‘0’ if prediction is correct, ‘1’ if prediction is not correct.
  
  Predictor for $x_i$: $y_i = x_{i-1}$
  
  Prediction error: $e_i = x_i - x_{i-1}$

Example: alphabet $S = \{0, 1\}$

Data: (0) 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 → $H=1.0$ bit

Errors: 0 0 0 0 1 0 0 0 0 0 0 0 0 -1 0 0 0
  (If $e < 0$ then $e = e + 2$) Why 2?

Errors: 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 → $H=0.5$ bit
### Four-pixel prediction function

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Prediction</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Configuration 1" /></td>
<td><img src="image2" alt="Prediction 1" /></td>
<td>99.76 %</td>
</tr>
<tr>
<td><img src="image3" alt="Configuration 2" /></td>
<td><img src="image4" alt="Prediction 2" /></td>
<td>62.99 %</td>
</tr>
<tr>
<td><img src="image5" alt="Configuration 3" /></td>
<td><img src="image6" alt="Prediction 3" /></td>
<td>83.97 %</td>
</tr>
<tr>
<td><img src="image7" alt="Configuration 4" /></td>
<td><img src="image8" alt="Prediction 4" /></td>
<td>87.98 %</td>
</tr>
<tr>
<td><img src="image9" alt="Configuration 5" /></td>
<td><img src="image10" alt="Prediction 5" /></td>
<td>71.05 %</td>
</tr>
<tr>
<td><img src="image11" alt="Configuration 6" /></td>
<td><img src="image12" alt="Prediction 6" /></td>
<td>86.59 %</td>
</tr>
<tr>
<td><img src="image13" alt="Configuration 7" /></td>
<td><img src="image14" alt="Prediction 7" /></td>
<td>70.10 %</td>
</tr>
<tr>
<td><img src="image15" alt="Configuration 8" /></td>
<td><img src="image16" alt="Prediction 8" /></td>
<td>95.19 %</td>
</tr>
<tr>
<td><img src="image17" alt="Configuration 9" /></td>
<td><img src="image18" alt="Prediction 9" /></td>
<td>96.64 %</td>
</tr>
<tr>
<td><img src="image19" alt="Configuration 10" /></td>
<td><img src="image20" alt="Prediction 10" /></td>
<td>77.14 %</td>
</tr>
<tr>
<td><img src="image21" alt="Configuration 11" /></td>
<td><img src="image22" alt="Prediction 11" /></td>
<td>94.99 %</td>
</tr>
<tr>
<td><img src="image23" alt="Configuration 12" /></td>
<td><img src="image24" alt="Prediction 12" /></td>
<td>61.41 %</td>
</tr>
<tr>
<td><img src="image25" alt="Configuration 13" /></td>
<td><img src="image26" alt="Prediction 13" /></td>
<td>61.41 %</td>
</tr>
<tr>
<td><img src="image27" alt="Configuration 14" /></td>
<td><img src="image28" alt="Prediction 14" /></td>
<td>78.74 %</td>
</tr>
<tr>
<td><img src="image29" alt="Configuration 15" /></td>
<td><img src="image30" alt="Prediction 15" /></td>
<td>78.60 %</td>
</tr>
<tr>
<td><img src="image31" alt="Configuration 16" /></td>
<td><img src="image32" alt="Prediction 16" /></td>
<td>91.82 %</td>
</tr>
</tbody>
</table>
READ Code (1)

- Code the location of run boundary relative to the previous row.
  READ = "Relative Element Address Designate"
- The READ code includes three coding modes:
  o Pass mode
  o Vertical mode
  o Horizontal mode

1. $a_0$ is the first pel of a new codeword and can be black or white.
2. $a_1$ is the first pel to the right of $a_0$ with a different color.
3. $a_2$ is the first pel to the right of $a_1$ with a different color.
4. $b_1$ is the first pel on the reference line to the right of $a_0$ with a different color.
5. $b_2$ is the first pel on the reference line to the right of $b_1$ with a different color.
• **Vertical mode:**
  The position of each color change is coded with respect to a nearby change position of the same color on the reference line, if one exists. "Nearby" is taken to mean within three pixels.

• **Horizontal mode:**
  There is no nearby change position on the reference line, one-dimensional run-length coding - called

• **Pass code:**
  The reference line contains a run that has no counterpart in the current line; next complete run of the opposite color in the reference line should be skipped.
### READ: Codes for modes

<table>
<thead>
<tr>
<th>Mode:</th>
<th>Codeword:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass</td>
<td>0001</td>
</tr>
<tr>
<td>Horizontal</td>
<td>$001 + H_w(wl) + H_b(bl)$</td>
</tr>
<tr>
<td>+3</td>
<td>0000011</td>
</tr>
<tr>
<td>+2</td>
<td>000011</td>
</tr>
<tr>
<td>+1</td>
<td>011</td>
</tr>
<tr>
<td>Vertical</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>010</td>
</tr>
<tr>
<td>-2</td>
<td>000010</td>
</tr>
<tr>
<td>-3</td>
<td>0000010</td>
</tr>
</tbody>
</table>

$wl$ = length of the white run  \hspace{1cm}  $bl$ = length of the black run

$H_w$ = Huffman code of white run  \hspace{1cm}  $H_b$ = Huffman code of black run

(For Huffman codes see previous slides)
• There is an all-white line above the page, which used as the reference line for the 1st scan line of the page.
• Each line is assumed start with a white pixel, which is ignored by receiver.
• Pointer $a_0$ is set to an imaginary white pel on the left of the coding line, and $a_1$ is set to point to the 1st black pel on the coding line. The first run length is $|a_0a_1|-1$.
• Pointers $b_1$ and $b_2$ are set to point to the start of the 1st and 2nd runs on the reference line, respectively.
• The encoder assumes an extra pel on the right of the line, with a color opposite that of the last pixel.
Pass (a) and Vertical mode (b1,b2)

Run length $b_1b_2$ coded. New $a_0$ becomes old $b_2$.

(b1)

Run length $a_1b_1$ coded. New $a_0$ becomes old $a_1$.
Horizontal mode (c1,c2)

Run length $a_1 b_1$ coded. New $a_0$ becomes old $a_1$.

(c1)

Reference line $\rightarrow$

Coding line $\rightarrow$

Run lengths $a_0 a_1$ (white) and $a_1 a_2$ (black) coded. New $a_0$ becomes old $a_2$. 
Flowchart

Reference line := all white

Read coding line. Set a0 before 1st pel.

Determine a1, b1, and b2

b2 to the left of a1?

Yes

Compute Pass mode code

Set a0 under b2

No

a1b1 ≤ 3

Yes

Compute Vertical mode code

No

Determine a2

Compute Horizontal mode code

Set a0 to a2

No

EOL

Set a0 to a1

Yes

EOP

END

Reference line equal to coding line
**READ Code: Example**

<table>
<thead>
<tr>
<th>n</th>
<th>white runs</th>
<th>black runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00110101</td>
<td>0000110111</td>
</tr>
<tr>
<td>1</td>
<td>000111</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>0111</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>1011</td>
<td>011</td>
</tr>
<tr>
<td>5</td>
<td>1100</td>
<td>0011</td>
</tr>
<tr>
<td>6</td>
<td>1110</td>
<td>0010</td>
</tr>
<tr>
<td>7</td>
<td>1111</td>
<td>000111</td>
</tr>
<tr>
<td>8</td>
<td>10011</td>
<td>000101</td>
</tr>
<tr>
<td>9</td>
<td>10100</td>
<td>000100</td>
</tr>
<tr>
<td>10</td>
<td>00111</td>
<td>0000100</td>
</tr>
<tr>
<td>11</td>
<td>01000</td>
<td>0000101</td>
</tr>
<tr>
<td>12</td>
<td>001000</td>
<td>0000111</td>
</tr>
</tbody>
</table>

**Mode:**

<table>
<thead>
<tr>
<th>Pass</th>
<th>Codeword:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal</td>
<td>$001 + H_W(wl) + H_b(bl)$</td>
</tr>
<tr>
<td>Vertical</td>
<td>$011$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode:</th>
<th>Codeword:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0001</td>
</tr>
<tr>
<td>+3</td>
<td>000011</td>
</tr>
<tr>
<td>+2</td>
<td>000011</td>
</tr>
<tr>
<td>+1</td>
<td>011</td>
</tr>
<tr>
<td>-1</td>
<td>010</td>
</tr>
<tr>
<td>-2</td>
<td>000010</td>
</tr>
<tr>
<td>-3</td>
<td>0000010</td>
</tr>
</tbody>
</table>
Block Coding: Idea

• Divide the image into blocks of pixels.
• A totally white block (all-white block) is coded by ’0’.
• All other blocks (non-white blocks) thus contain at least one black pixel. They are coded with a 1-bit as a prefix followed by the contents of the block (bit by bit in row-major order) or with Huffman code.
• The Block Coding can be applied to difference (error) image for predictive coding approach.

(see also Lecture 2)
Block Coding: Huffman codes for $k=0, 1$

<table>
<thead>
<tr>
<th>Original images</th>
<th>Error images</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>40.67 0</td>
<td>54.36 0</td>
</tr>
<tr>
<td><img src="image3" alt="Image" /></td>
<td><img src="image4" alt="Image" /></td>
</tr>
<tr>
<td>2.90 10111</td>
<td>8.22 100</td>
</tr>
<tr>
<td><img src="image5" alt="Image" /></td>
<td><img src="image6" alt="Image" /></td>
</tr>
<tr>
<td>2.87 10110</td>
<td>7.81 1111</td>
</tr>
<tr>
<td><img src="image7" alt="Image" /></td>
<td><img src="image8" alt="Image" /></td>
</tr>
<tr>
<td>2.58 111101</td>
<td>7.53 1110</td>
</tr>
<tr>
<td><img src="image9" alt="Image" /></td>
<td><img src="image10" alt="Image" /></td>
</tr>
<tr>
<td>2.50 111100</td>
<td>7.51 1101</td>
</tr>
</tbody>
</table>
Block Coding: Huffman codes for $k=2$

<table>
<thead>
<tr>
<th>Encoder Instruction</th>
<th>Entropy</th>
<th>Code</th>
<th>Encoder Instruction</th>
<th>Entropy</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.23</td>
<td>11111</td>
<td></td>
<td>2.51</td>
<td>10111</td>
</tr>
<tr>
<td></td>
<td>5.27</td>
<td>1000</td>
<td></td>
<td>2.04</td>
<td>10101</td>
</tr>
<tr>
<td></td>
<td>5.74</td>
<td>1010</td>
<td></td>
<td>1.79</td>
<td>110011</td>
</tr>
<tr>
<td></td>
<td>5.67</td>
<td>1001</td>
<td></td>
<td>1.43</td>
<td>110001</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
<td>11100101</td>
<td></td>
<td>2.05</td>
<td>10110</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>11100100</td>
<td></td>
<td>0.77</td>
<td>101000</td>
</tr>
</tbody>
</table>
Block Coding: Huffman codes for $k=3, 4$

<table>
<thead>
<tr>
<th></th>
<th>Input Symbols:</th>
<th>Output Symbols:</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Symbol 1]</td>
<td>1.81</td>
<td>111000</td>
</tr>
<tr>
<td>![Symbol 2]</td>
<td>2.21</td>
<td>111011</td>
</tr>
<tr>
<td>![Symbol 3]</td>
<td>1.75</td>
<td>1110011</td>
</tr>
<tr>
<td>![Symbol 4]</td>
<td>2.18</td>
<td>111010</td>
</tr>
<tr>
<td>![Symbol 5]</td>
<td>18.29</td>
<td>110</td>
</tr>
<tr>
<td><strong>Entropy</strong></td>
<td><strong>Average code length:</strong></td>
<td><strong>Entropy</strong></td>
</tr>
<tr>
<td>2.93</td>
<td>2.99</td>
<td>2.50</td>
</tr>
</tbody>
</table>
Hierarchical block encoding: Principle

• In the hierarchical variant of the block coding the bit map is first divided into b*b blocks (typically 16*16).

• These blocks are then divided into quadtree structure of blocks in the following manner:
  - If a particular b*b block is all-white, it is coded by ’0’.
  - Otherwise the block is coded by ’1’ and then divided into four equal sized subblocks which are recursively coded in the same manner.
Hierarchical block encoding: (1)

Code: '1'

Code: '0111'
Hierarchical block encoding ()

L=3
Codes: 0011 0111 1000

L=4
Codes: 0111 1111
      1111 1100
      0101 1010

Totally: 1+4+12+24 = 41 bits
Hierarchical block encoding: Example

Image to be compressed:

Code bits:

<table>
<thead>
<tr>
<th>1</th>
<th>0111</th>
<th>0011 0111 1000</th>
<th>0111 1111 1111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0111 1111 1111</td>
<td>0101 1010 1100</td>
</tr>
</tbody>
</table>

1+4+12+24=41
CCITT Group 3 (G3) and Group 4 (G4)

- The RLE and READ algorithms are included in image compression standards, known as CCITT G3 and G4. (used in FAX-machines).
CCITT Group 3 (G3)

• Every k-th line is coded by RLE-method and the READ-code is applied for the rest of the lines.
• The first (virtual) pixel is white
• EOL code after every line to synchronize code
• Six EOL codes after every page
• Binary documents only
CCITT Group 4 (G4)

• All lines are codes by READ
• The first reference line (above image) is white
• EOL code after every line to synchronize code
• Six EOL codes after every page
• Option for grayscale and color images
### G3 and G4: Results

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Low (200×100)</th>
<th>High (200×200)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme</td>
<td>G3</td>
<td>G4</td>
</tr>
<tr>
<td>Bits per pel</td>
<td>0.13</td>
<td>0.11</td>
</tr>
<tr>
<td>Seconds</td>
<td>57</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>0.09</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>74</td>
<td>61</td>
</tr>
</tbody>
</table>

7 min → 1 min
Comparison of algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPRESS</td>
<td>8.9</td>
</tr>
<tr>
<td>GZIP</td>
<td>10.8</td>
</tr>
<tr>
<td>BLOCK</td>
<td>11.2</td>
</tr>
<tr>
<td>RLE</td>
<td>7.9</td>
</tr>
<tr>
<td>2D-RLE</td>
<td>9.8</td>
</tr>
<tr>
<td>ORLE</td>
<td>18.0</td>
</tr>
<tr>
<td>G3</td>
<td>18.9</td>
</tr>
<tr>
<td>G4</td>
<td>17.9</td>
</tr>
<tr>
<td>JBIG</td>
<td>23.3</td>
</tr>
</tbody>
</table>

COMPRESS = Unix standard compression software
GZIP = Gnu compression software
PKZIP = Pkware compression software
BLOCK = Hierarchical block coding [KJ80]
RLE = Run-length coding [NM80]
2D-RLE = 2-dimensional RLE [WW92]
ORLE = Ordered RLE [NM80]
G3 = CCITT Group 3 [YA85]
G4 = CCITT Group 4 [YA85]
JBIG = ISO/IEC Standard draft [PM93]
Any analog quantity that is to be processed by a digital computer or digital system must be converted to an integer number proportional to its amplitude. The conversion process between analog samples and discrete-valued samples is called quantization.
Uniform quantizer: $M=8$ levels

Input-output characteristic of uniform quantizer
Nonuniform quantizer: M = 8 levels

Input-output characteristic of nonuniform quantizer

$$q(x)$$

<table>
<thead>
<tr>
<th>Code</th>
<th>$a_0=-\infty$</th>
<th>$a_8=+\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$[a_i, a_{i+1})$  $y_i$
Nonuniform quantizer: $M = 8$ levels

Input-output characteristic of nonuniform quantizer

<table>
<thead>
<tr>
<th>Code</th>
<th>$q(x)$</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td>$y_7$</td>
<td>$[a_i, a_{i+1})$</td>
<td>$y_i$</td>
</tr>
<tr>
<td>010</td>
<td>$y_6$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>$y_5$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>$y_4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>$y_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>$y_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>$y_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>$y_0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$a_0 = -\infty$
$a_8 = +\infty$
Quantization error

Quantized signal $q(x)$

Input signal $x$

Quantization error:

$$e(x) = x - q(x)$$
Distortion measure

Probability density function (pdf) of $x$ is $p(x)$

Quantization error: $e(x) = x - q(x)$

Mean (average value) $\mu$ of quantization error:

$$
\mu = E[x - q(x)] = \sum_{j=1}^{M} \int_{a_{j-1}}^{a_j} (x - y_i)p(x)dx
$$

Variance $\sigma^2$ of quantization error as distortion measure:

$$
\sigma^2 = E[(x - q(x))^2] = \sum_{j=1}^{M} \int_{a_{j-1}}^{a_j} (x - y_j)^2 p(x)dx
$$
Optimal quantization problem

Given a signal $x$, with probability density function (or histogram) $p(x)$, find a quantizer $q(x)$ of $x$, which minimizes the quantization error variance $\sigma^2$:

$$\sigma_{\text{opt}}^2 = \min_{\{a_j\}, \{y_j\}} \sum_{j=1}^{M} \int_{a_{j-1}}^{a_j} (x - y_j)^2 p(x) \, dx$$
Lossy image compression

- DPCM: Prediction error quantization
- Block Truncation Coding (BTC)
- Vector Quantization (VQ)
- Transform Coding (DCT, JPEG)
- Subband Coding
- Wavelet Coding (JPEG2000)

Data → Transformation → Quantization → Encoding → Bitstream

Model
Part 1: DPCM

\[ y = x_i - x_{i-1} \]

Histogram of the original image and Residual image

Entropy: \( H_o = 7.8 \) bits/pel

\( H_r = 5.1 \) bits/pel
Prediction error quantization with open loop

\[ e_i = x_i - x_{i-1} \rightarrow q(e_i) \]

DPCM is Differential Pulse Code Modulation
Quantization with open loop: Decoding

Problem: error accumulation!

W/o quantization
\[ x_n = x_{n-1} + e_i \]
\[ y_n = x_{n-1} + q(e_i) \]

With quantization
\[ y_n = y_{i-1} + q(e_n) \]
\[ y_n - x_n = [x_1 + q(e_2) + \ldots + q(e_n)] - [x_1 + e_2 + \ldots + e_n] = \]
\[ = (q(e_2) - e_2) + \ldots + (q(e_n) - e_n); \]

Variance:
\[ \sigma_y^2 = \sigma_x^2 + (n-1)\sigma_q^2 \]
Closed loop: Encoding

\[ e_i = x_i - z_{i-1} \]
\[ z_i = z_{i-1} + q(e_i) \]
Closed loop: Decoding

Error accumulation? No!

W/o quantization
\[ e_n = x_n - z_{n-1} \text{ or } x_n = z_{n-1} + e_n \Rightarrow z_n = z_{n-1} + q(e_n) \]

With quantization
\[ x_n - z_{n-1} = (z_{n-1} + e_n) - (z_{n-1} + q(e_n)) = e_n - q(e_n); \]
Example

• Open loop: quantization step is 8

- $x_j$: 81 → 109 → 129 → 165 → 209 → 221
- $e_j$: 28 → 20 → 36 → 44 → 12
- $[e_j/8]$: 4 → 3 → 5 → 6 → 2
- $q(e_j)$: 32 → 24 → 40 → 48 → 16
- $y_j$: 81 → 113 → 137 → 177 → 225 → 241

• Closed loop: quantization step is 8

- $x_j$: 81 → 109 → 129 → 165 → 209 → 221
- $e_j$: 28 → 16 → 36 → 40 → 12
- $q(e_j)$: 32 → 16 → 40 → 40 → 16
- $z_j$: 81 → 113 → 129 → 169 → 209 → 225
Entropy reduction:

\[
\Delta H = H_0 - H_1 = \log_2 \left( \frac{\sigma_0}{\sigma_1} \right)
\]

\[
\sigma_1^2 = 2\sigma_0^2 (1 - \rho(\Delta)),
\]

where \(\sigma_0^2\) is variance of the data \(x\),
\(\sigma_1^2\) is variance of the prediction error \(e\),
\(\rho(\Delta)\) is correlation coefficient of the pixels \(x_i\) and \(x_{i-1}\)

or

\[
\Delta H = 0.5\log_2 \left[ 2(1 - \rho(\Delta)) \right].
\]

Example: If \(\rho(\Delta) = 0.8\) → \(-\log_2[2*0.2]=1.3\) bits
If \(\rho(\Delta) = 0.9\) → \(-\log_2[2*0.1]=2.3\) bits
Optimum linear prediction

• 1-D Linear predictor: \( \hat{x}_i = \sum_{j=1}^{m} a_j x_{i-j} \)

Usually \( m=3 \)

• 2-D and 3-D linear predictors
Part 2. Block Truncation Coding

- Divide the image into $4 \times 4$ blocks;
- Quantize the block into two representative values $a$ and $b$;
- Encode (1) the representative values $a$ and $b$ and (2) the significance map in the block.

\[
\frac{\bar{x} = 7.94}{\sigma = 4.91}
\]

<table>
<thead>
<tr>
<th>Original</th>
<th>Bit-plane</th>
<th>Reconstructed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 9 12 15</td>
<td>0 1 1 1</td>
<td>2 12 12 12</td>
</tr>
<tr>
<td>2 11 11 9</td>
<td>0 1 1 1</td>
<td>2 12 12 12</td>
</tr>
<tr>
<td>2 3 12 15</td>
<td>0 0 1 1</td>
<td>2 2 12 12</td>
</tr>
<tr>
<td>3 3 4 14</td>
<td>0 0 0 1</td>
<td>2 2 2 12</td>
</tr>
</tbody>
</table>

$a = [2.3] = 2$  
$b = [12.3] = 12$
1. How to construct quantizer?

• The first two moments preserving quantization:

\[
<x> = \frac{1}{m} \sum_{i=1}^{m} x_i \quad <x^2> = \frac{1}{m} \sum_{i=1}^{m} x_i^2
\]

\[
\sigma^2 = <x^2> - <x>^2
\]

• Threshold for quantization: \( T = <x> \); \( n_a + n_b = m \)

\[
m < x > = n_a a + n_b b
\]

\[
m < x^2 > = n_a a^2 + n_b b^2
\]

\[
a = <x> - \sigma \cdot \sqrt{\frac{n_b}{n_a}}
\]

\[
b = \bar{x} + \sigma \cdot \sqrt{\frac{n_a}{n_b}}
\]
2. Optimal scalar quantizer ("AMBTC")

- Minimize quantization error:

\[
D = \min_{a,b,T} \left\{ \sum_{x_i < T} (x_i - a)^2 + \sum_{x_i \geq T} (x_i - b)^2 \right\}
\]

- Max-Lloyd solution:

\[
a = \frac{1}{n_a} \sum_{x_i < T} x_i
\]

\[
b = \frac{1}{n_b} \sum_{x_i \geq T} x_i
\]

\[
T = \frac{a + b}{2}
\]

- How to find the \(a, b, T\)? See Max-Lloyd algorithm.
Example of BTC

<table>
<thead>
<tr>
<th>Original</th>
<th>Bit-plane</th>
<th>Reconstructed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2  9 12 15</td>
<td>0  1 1 1 1</td>
<td>2  12 12 12</td>
</tr>
<tr>
<td>2  11 11  9</td>
<td>0  1 1 1 1</td>
<td>2  12 12 12</td>
</tr>
<tr>
<td>2  3 12 15</td>
<td>0  0 1 1 1</td>
<td>2  2 12 12</td>
</tr>
<tr>
<td>3  3  4 14</td>
<td>0  0 0 1 1</td>
<td>2  2  2 12</td>
</tr>
</tbody>
</table>

\[
x = 7.94
\]
\[
\sigma = 4.91
\]
\[
T = 9
\]
\[
n_a = 7
\]
\[
n_b = 9
\]
\[
a = [2.3] = \hat{2}
\]
\[
b = [12.3] = 12
\]
\[
D = \sigma_a^2 + \sigma_b^2 = 7 + 43 = 50
\]
Example of optimal quantizer ("AMBTC")

<table>
<thead>
<tr>
<th>Original</th>
<th>Bit-plane</th>
<th>Reconstructed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 9 12 15</td>
<td>0 1 1 1</td>
<td>3 12 12 12</td>
</tr>
<tr>
<td>2 11 11 9</td>
<td>0 1 1 1</td>
<td>3 12 12 12</td>
</tr>
<tr>
<td>2 3 12 15</td>
<td>0 0 1 1</td>
<td>3 3 12 12</td>
</tr>
<tr>
<td>3 3 4 14</td>
<td>0 0 0 1</td>
<td>3 3 3 12</td>
</tr>
</tbody>
</table>

\[ \bar{x} = 7.94 \quad \sigma = 4.91 \]

\[ T = 9 \quad n_a = 7 \quad n_b = 9 \]

\[ a = [2.7] = 3 \quad b = [12.0] = 12 \]

\[ D = \sigma^2_a + \sigma^2_b = 4 + 43 = 47 \]
Representative levels compression

• Main idea of BTC:
  Image → ”smooth part” + ”detailed part”
  (a and b) (bit-planes)

• We can treat set of a’s and b’s as an image:
  1. Predictive encoding of a and b
  2. Lossless image compression algorithm (FELICS, JPEG-LS, CALIC).
  3. Lossy compression: DCT (JPEG)
Significance bits compression

Binary image:
- Lossless binary image compression methods (JBIG, context modeling with arithmetic coding)
- Lossy image compression (vector quantization, with sub-sampling and interpolating missing pixels, filtering)
Bitrate and Block size

The number of pixels in block: $k^2$ pels

- **BTC:**
  1. Values ’a’ and ’b’: (8+8) bits
  2. Significance bits: $k^2$ bits

Bitrate: $R = \frac{16 + k^2}{k^2} = \left(1 + \frac{16}{k^2}\right)$ bit/pel

Example: $k=4$: $R = \left(1 + \frac{16}{4^2}\right) = 2$ bit/pel

- Bigger block $\rightarrow$ smaller bitrate $R$, bigger distortion $D$
- Smaller block $\rightarrow$ bigger bitrate $R$, smaller distortion $D$

Trade-off between Rate and Distortion
Quadtree segmentation

1. Divide the image into blocks of $m_1 \times m_1$ size.
2. FOR EACH BLOCK
   IF ($\sigma < \sigma_0$) THEN apply BTC
   ELSE divide into four subblocks: $m = m/2$
3. REPEAT step 2 UNTIL
   ($\sigma < \sigma_0$) OR $m = m_2$

here $m_2$ is minimal block size
The hierarchy of the blocks is represented by a quadtree structure.
Example of BTC

**AMBTC**

bpp = 2.00
mse = 40.51
Block size: 4x4

**HBTC-VQ**

bpp = 1.62
mse = 15.62
Block size: 2x2 .. 32x32

bpp=8
-block-
JPEG

- JPEG = Joint Photographic Experts Group

- Lossy coding of continuous tone still images (color and grayscale)

- Based on Discrete Cosine Transform (DCT):
  0) Image is divided into block $N \times N$
  1) The blocks are transformed with 2-D DCT
  2) DCT coefficients are quantized
  3) The quantized coefficients are encoded
JPEG: Encoding and Decoding

Source Image Data → FDCT → Quantizer → Entropy Encoder → Compressed Image Data

8x8 blocks

Table Specifications

Decompressor

Entropic Dequantizer

IDCT

Reconstructed Image Data

Table Specifications

Table Specifications
Divide image into $N \times N$ blocks

Input image

8x8 block
2-D DCT basis functions: \( N=8 \)
2-D Transform Coding

...
1-D DCT basis functions: $N=8$

\[ x_j = \sum_{k=0}^{N-1} \alpha(k) C(k) \cdot \cos \left[ \frac{(2j+1)k\pi}{2N} \right] \]

\[ \alpha(k) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } k = 0 \\ \sqrt{\frac{2}{N}} & \text{for } k = 1, 2, \ldots, N-1 \end{cases} \]
Zig-zag ordering of DCT coefficients

Converting a 2-D matrix into a 1-D array, so that the frequency (horizontal and vertical) increases in this order and the coefficients variance are decreasing in this order.
Example of DCT for image block

<table>
<thead>
<tr>
<th>Original block</th>
<th>Transformed block</th>
</tr>
</thead>
<tbody>
<tr>
<td>139 144 149 153 155 155 155 155</td>
<td>235.6 -1.0 -12.1 -5.2 2.1 -1.7 -2.7 1.3</td>
</tr>
<tr>
<td>144 151 153 156 159 156 156 156</td>
<td>-22.6 -17.5 -6.2 -3.2 -2.9 -0.1 0.4 -1.2</td>
</tr>
<tr>
<td>150 155 160 163 158 156 156 156</td>
<td>-10.9 -9.3 -1.6 1.5 0.2 -0.9 -0.6 -0.1</td>
</tr>
<tr>
<td>159 161 162 160 160 159 159 159</td>
<td>-7.1 -1.9 0.2 1.5 0.9 -0.1 0.0 0.3</td>
</tr>
<tr>
<td>159 160 161 162 162 155 155 155</td>
<td>-0.6 -0.8 1.5 1.6 -0.1 -0.7 0.6 1.3</td>
</tr>
<tr>
<td>161 161 161 161 160 157 157 157</td>
<td>1.8 -0.2 1.6 -0.3 -0.8 1.5 1.0 -1.0</td>
</tr>
<tr>
<td>162 162 161 163 162 157 157 157</td>
<td>-1.3 -0.4 -0.3 -1.5 -0.5 1.7 1.1 -0.8</td>
</tr>
<tr>
<td>162 162 161 161 163 158 158 158</td>
<td>-2.6 1.6 -3.8 -1.8 1.9 1.2 -0.6 -0.4</td>
</tr>
</tbody>
</table>

Matlab: `y=dct(x)`
Distribution of DCT coefficients

DC coefficient

DC: uniformly distributed

AC coefficient

AC: distribution resembles Laplacian pdf
Bit allocation for DCT coefficients

- Lossy operation to reduce bit-rate
- Vector or Scalar Quantizer?
- Set of optimal scalar quantizers?
- Set of scalar quantizers with fixed quantization tables
Bit allocation for DCT coefficients

Minimize the total distortion $D$

$$D = \min_{\{b_i\}} \left\{ \sum_{i=1}^{N} h_i \sigma_i^2 2^{-2b_i} \right\}$$

subject to $\sum_{i=1}^{N} b_i = B$

here $b_i$ is number of bits for coefficient $y_i$, $B$ is a given total number of bits,

$$h_i = \frac{1}{12} \left\{ \int \left[ p_i(x) \right]^{1/3} dx \right\}^3$$

See Lecture 10
Optimal bit allocation for DCT coefficients

Solution of the optimization task with Lagrange multiplier method:

Bitrate: \[ b_i = \frac{B}{N} + \frac{1}{2} \log_2 \left( \frac{\sigma_i^2}{\theta^2} \right) + \frac{1}{2} \log_2 \frac{h_i}{H} \]

Distortion: \[ D = NH \theta^2 2^{-B/N} \]

where \[ \theta^2 = \left( \prod_{k=0}^{N-1} \sigma_k^2 \right)^{1/N} \quad ; \quad H = \left( \prod_{k=0}^{N-1} h_k \right)^{1/N} \]
Distortion: \[ D = NH\theta^2 2^{-B/N} \]

where \[ \theta^2 = \left( \prod_{k=0}^{N-1} \sigma_k^2 \right)^{1/N} \]

Distortion \( D \) is minimal, if \( \theta^2 \) is minimal.
Product of diagonal elements is greater than or equal to the determinant of the (positive semidefinite) matrix.
Equality is attained iff the matrix is diagonal.
KLT provides minimum of \( \theta^2 \) (and minimum of distortion \( D \)) among other transforms!
Default quantization matrix $Q$

\[
y_q(k,l) = \text{round}[y(k,l)/Q(k,l)]
\]

Examples: 236/16 $\rightarrow$ 15  
-22/11 $\rightarrow$ -2

Matlab: $Qy=\text{quant}(y)$
Quantization of DCT coefficients: Example

<table>
<thead>
<tr>
<th>Quantization matrix</th>
<th>Quantized coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 11 10 16 24 40 51 61</td>
<td>15 0 -1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>12 12 14 19 26 58 60 55</td>
<td>-2 -1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>14 13 16 24 40 57 69 56</td>
<td>-1 -1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>14 17 22 29 51 87 80 62</td>
<td>-1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>18 22 37 56 68 109 103 77</td>
<td>0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>24 35 55 64 81 104 113 92</td>
<td>0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>49 64 78 87 103 121 120 101</td>
<td>0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>72 92 95 98 112 100 103 99</td>
<td>0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Ordered DCT coefficients: 15,0,-2,-1,-1,-1,0,0,-1,-1, 54{′0′}.
Dequantization

\[ z(k,l) = y_q(k,l) \cdot Q(k,l) \]

Examples: 236/16 → 15
-22/11 → -2

Matlab: \( z = \text{dequant} \ (Qy) \)

<table>
<thead>
<tr>
<th>Original DCT block</th>
</tr>
</thead>
<tbody>
<tr>
<td>240 0 -10 0 0 0 0 0</td>
</tr>
<tr>
<td>-24 -12 0 0 0 0 0 0</td>
</tr>
<tr>
<td>-14 -13 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantized coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 ( \begin{array} {cccccccc} 0 &amp; -1 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ -2 &amp; -1 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ -1 &amp; -1 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ -1 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \end{array} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dequantized coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>235.6 -1.0 -12.1 -5.2 2.1 -1.7 -2.7 1.3</td>
</tr>
<tr>
<td>-22.6 -17.5 -6.2 -3.2 -2.9 -0.1 0.4 -1.2</td>
</tr>
<tr>
<td>-10.9 -9.3 -1.6 1.5 0.2 -0.9 -0.6 -0.1</td>
</tr>
<tr>
<td>-7.1 -1.9 0.2 1.5 0.9 -0.1 0.0 0.3</td>
</tr>
<tr>
<td>-0.6 -0.8 1.5 1.6 -0.1 -0.7 0.6 1.3</td>
</tr>
<tr>
<td>1.8 -0.2 1.6 -0.3 -0.8 1.5 1.0 -1.0</td>
</tr>
<tr>
<td>-1.3 -0.4 -0.3 -1.5 -0.5 1.7 1.1 -0.8</td>
</tr>
<tr>
<td>-2.6 1.6 -3.8 -1.8 1.9 1.2 -0.6 -0.4</td>
</tr>
</tbody>
</table>
Inverse DCT

### Dequantized coefficients

<table>
<thead>
<tr>
<th></th>
<th>240</th>
<th>0</th>
<th>-10</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-24</td>
<td></td>
<td>-12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-14</td>
<td></td>
<td>-13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Decompressed block

<table>
<thead>
<tr>
<th></th>
<th>144</th>
<th>146</th>
<th>149</th>
<th>152</th>
<th>154</th>
<th>156</th>
<th>156</th>
<th>156</th>
</tr>
</thead>
<tbody>
<tr>
<td>148</td>
<td>150</td>
<td>152</td>
<td>154</td>
<td>156</td>
<td>156</td>
<td>156</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>156</td>
<td>157</td>
<td>158</td>
<td>158</td>
<td>157</td>
<td>156</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>161</td>
<td>161</td>
<td>162</td>
<td>161</td>
<td>159</td>
<td>157</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>163</td>
<td>164</td>
<td>163</td>
<td>162</td>
<td>160</td>
<td>158</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>164</td>
<td>164</td>
<td>164</td>
<td>162</td>
<td>160</td>
<td>158</td>
<td>157</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>161</td>
<td>162</td>
<td>162</td>
<td>162</td>
<td>161</td>
<td>159</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>159</td>
<td>161</td>
<td>161</td>
<td>162</td>
<td>161</td>
<td>159</td>
<td>158</td>
<td></td>
</tr>
</tbody>
</table>

### Original block

<table>
<thead>
<tr>
<th></th>
<th>139</th>
<th>144</th>
<th>149</th>
<th>153</th>
<th>155</th>
<th>155</th>
<th>155</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>144</td>
<td>151</td>
<td>153</td>
<td>156</td>
<td>159</td>
<td>156</td>
<td>156</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>155</td>
<td>160</td>
<td>163</td>
<td>158</td>
<td>156</td>
<td>156</td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>161</td>
<td>162</td>
<td>160</td>
<td>160</td>
<td>159</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>160</td>
<td>161</td>
<td>162</td>
<td>162</td>
<td>155</td>
<td>155</td>
<td></td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>161</td>
<td>161</td>
<td>161</td>
<td>160</td>
<td>157</td>
<td>157</td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>162</td>
<td>161</td>
<td>163</td>
<td>162</td>
<td>157</td>
<td>157</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See: \( x = \text{idct}(y) \)
Encoding of quantized DCT coefficients

- Ordered data: 15, 0, -2, -1, -1, -1, 0, 0, -1, -1, 54{'0'}.  
- Encoding:  
  - DC: ?  
  - AC: ?
Encoding of quantized DCT coefficients

• DC coefficient for the current block is predicted of that of the previous block, and error is coded using Huffman coding

• AC coefficients:

  (a) Huffman code, arithmetic code for non-zeroes
  (b) run-length encoding: (number of ’0’ s, non-’0’-symbol)
Performance of JPEG algorithm

8 bpp 0.6 bpp

0.37 bpp 0.22 bpp
Compression of color images

487x414 pixels,
Uncompressed, 600471 Bytes, 24 bpp
85502 Bytes, 3.39 bpp, CR=7

487x414 pixels
41174 Bytes, 1.63 bpp, CR=14.7
RGB vs YCbCr

- 24 bits RGB representation: apply DCT for each component separately
  - does not make use of the correlation between color components
  - does not make use of lowe sensitivity of the human eyes to chrominance component

- Convert RGB into a YCbCr representation: Y is luminance, and Yb, Yc are chrominance
  - Downsample the two chrominance components
RGB $\Leftrightarrow$ YCbCr conversion

\[
\begin{bmatrix}
Y \\
C_b \\
C_r
\end{bmatrix} = \begin{bmatrix}
0.299 & 0.587 & 0.114 \\
-0.169 & -0.331 & 0.500 \\
0.500 & -0.419 & -0.081
\end{bmatrix} \begin{bmatrix}
R \\
G \\
B
\end{bmatrix} + \begin{bmatrix}
0 \\
128 \\
128
\end{bmatrix}
\]

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} = \begin{bmatrix}
1.000 & -0.001 & 1.402 \\
1.000 & -0.344 & -0.714 \\
1.000 & 1.772 & 0.001
\end{bmatrix} \begin{bmatrix}
Y \\
C_b - 128 \\
C_r - 128
\end{bmatrix}
\]

Luminance Y and two chrominances Cb and Cr
Chrominance subsampling

- 4:4:4
- 4:2:2
- 4:1:1
- 4:2:0

Y pixel

Cb and Cr pixel
## Quantization of DCT coefficients

### For illuminance

<table>
<thead>
<tr>
<th>16</th>
<th>11</th>
<th>10</th>
<th>16</th>
<th>24</th>
<th>40</th>
<th>51</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>14</td>
<td>19</td>
<td>26</td>
<td>58</td>
<td>60</td>
<td>55</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>16</td>
<td>24</td>
<td>40</td>
<td>57</td>
<td>69</td>
<td>56</td>
</tr>
<tr>
<td>14</td>
<td>17</td>
<td>22</td>
<td>29</td>
<td>51</td>
<td>87</td>
<td>80</td>
<td>62</td>
</tr>
<tr>
<td>18</td>
<td>22</td>
<td>37</td>
<td>56</td>
<td>68</td>
<td>109</td>
<td>103</td>
<td>77</td>
</tr>
<tr>
<td>24</td>
<td>35</td>
<td>55</td>
<td>64</td>
<td>81</td>
<td>104</td>
<td>113</td>
<td>92</td>
</tr>
<tr>
<td>49</td>
<td>64</td>
<td>78</td>
<td>87</td>
<td>103</td>
<td>121</td>
<td>120</td>
<td>101</td>
</tr>
<tr>
<td>72</td>
<td>92</td>
<td>95</td>
<td>98</td>
<td>112</td>
<td>100</td>
<td>103</td>
<td>99</td>
</tr>
</tbody>
</table>

### For chrominance

<table>
<thead>
<tr>
<th>17</th>
<th>18</th>
<th>24</th>
<th>47</th>
<th>99</th>
<th>99</th>
<th>99</th>
<th>99</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>21</td>
<td>26</td>
<td>66</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>24</td>
<td>26</td>
<td>56</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>47</td>
<td>66</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
</tbody>
</table>
Performance of JPEG algorithm

- Grayscale 8 bits images:
  - 0.5 bpp: excellent quality

- Color 24 bits images:
  - 0.25-0.50 bpp: moderate to good
  - 0.50-0.75 bpp: good to very good
  - 0.75-1.00 bpp: excellent, sufficient for most applications
  - 1.00-2.00 bpp: indistinguishable from original
JPEG $\Rightarrow$ JPEG2000

JPEG: 0.25 bpp
JPEG2000: 0.25 bpp
JPEG 2000

- JPEG 2000 is a new still image compression standard
- "One-for-all" image codec:
  * Different image types: binary, grey-scale, color, multi-component
  * Different applications: natural images, scientific, medical remote sensing text, rendered graphics
  * Different imaging models: client/server, consumer electronics, image library archival, limited buffer and resources.
History

• Call for Contributions in 1996
• The 1st Committee Draft (CD) Dec. 1999
• Final Committee Draft (FCD) in March 2000
• Accepted as Draft International Standard in Aug. 2000
• Published as ISO Standard in Jan. 2002
Key components

• Transform
  – Wavelet
  – Wavelet packet
  – Wavelet in tiles

• Quantization
  – Scalar

• Entropy coding
  – (EBCOT) code once, truncate anywhere
  – Rate-distortion optimization
  – Context modeling
  – Optimized coding order
Key components

- Visual
  - Weighting
  - Masking
- Region of interest (ROI)
- Lossless color transform
- Error resilience
2-D wavelet transform

Original
128, 129, 125, 64, 65, ...

Transform Coeff.
4123, -12.4, -96.7, 4.5, ...
Quantization of wavelet coefficients

Transform Coeff.
4123, -12.4, -96.7, 4.5, ...

Quantized Coeff. (Q=64)
64, 0, -1, 0, ...
Quantizer with dead zone

\[ \nu[m, n] = \left\lfloor \frac{|s[m, n]|}{\delta} \right\rfloor, \]

Quantized Magnitude

\[ \chi[m, n] = \begin{cases} 
0 & + s \\
1 & - s 
\end{cases} \]

Sign
Entropy coding

Quantized Coeff. (Q=64)
64, 0, -1, 0, ...

Coded Bitstream

0 1 1 0 1 1 0 1 0 1 ...
Key features of EBCOT: Embedded Block Coding with Optimized Truncation

- Low memory requirement in coding and decoding
- Easy rate control
- High compression performance
- Region of interest (ROI) access
- Error resilience
- Modest complexity
Block structure in EBCOT

Encode each block separately & record the bitstream of each block. Block size is 64x64.
Progressive encoding
Quantizer with dead zone

\[ v[m, n] = \left\lfloor \frac{|s[m, n]|}{\delta} \right\rfloor, \]

Quantized Magnitude

\[ \chi[m, n] = \begin{cases} 
0 & +s \\
1 & -s
\end{cases} \]

Sign
ROI: Region of interest

Scale-down the coefficients outside the ROI so those are in lowerer bit-planes.
Decoded or refined ROI bits before the rest of the image.
ROI: Region of interest

- Sequence based code
  - ROI coefficients are coded as independent sequences
  - Allows random access to ROI without fully decoding
  - Can specify exact quality/bitrate for ROI and the BG
- Scaling based mode:
  - Scale ROI mask coefficients up (decoder scales down)
  - During encoding the ROI mask coefficients are found significant at early stages of the coding
  - ROI always coded with better quality than BG
  - Can't specify rate for BG and ROI
Tiling

- Image ⇒ Component ⇒ Tile ⇒ Subband ⇒ Code-Block ⇒ Bit-Planes
JPEG 2000 vs JPEG

DCT

WT
JPEG 2000 vs JPEG: Quantization

JPEG

JPEG 2000
JPEG 2000 vs JPEG: 0.3 bpp
JPEG 2000 vs JPEG: Bitrate=0.3 bpp

MSE=150
PSNR=26.2 db

MSE=73
PSNR=29.5 db
JPEG 2000 vs JPEG: Bitrate=0.2 bpp

MSE=320
PSNR=23.1 db

MSE=113
PSNR=27.6 db