JPEG Lecture


The first and still principal image compression standard. circa 1990.

Conceptually fairly simple

There is a baseline version and extensions. I'll describe just the baseline version for black-white images.

Overview of JPEG Encoding


Overview of JPEG Decoding

4. Binary decode, 3. Inverse quantize, 2. Inverse transform, 1. Reassemble segments into image

Draw Block Diagram

Images

Black-white image $X = [X(i,j), i=1,...,n, j=1,...,n]$ as a matrix, e.g. 512x512.

- $i$ denotes row, $j$ denotes column

Ordinarily, each $X(i,j) \in \{0, 1, 2, ..., 255\}$; i.e. 8 bits per pixel, which is what JPEG ordinarily expects. 0 = black, 255 = white

We could reduce the rate by dropping bits. $R = 7$ looks very good (128 levels), $R = 6$ looks pretty good (64), $R = 5$ or below shows false contouring.

Details

1. Segment into 8x8 square blocks (each will be processed independently)

Let $B = [B(i,j), i=0,...,7, j=0,...,7]$ denote a one of the blocks. we'll describe how to quantize it.
2. Transformation:

encoder does DCT transform of each 8x8:

\[ C(u,v) = 4 c_u c_v \frac{1}{n^2} \sum_{i=0}^{7} \sum_{j=0}^{7} B(i,j) \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16}, \quad u,v = 0, \ldots, 7 \]

where \( c_u = \begin{cases} \frac{1}{\sqrt{2}}, & u = 0 \\ 1, & u = 1, \ldots, 7 \end{cases} \)

decoder does inverse DCT:

\[ B(i,j) = \sum_{u=0}^{7} \sum_{v=0}^{7} c_u c_v C(u,v) \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16}, \quad i,j = 0, \ldots, 7 \]

this shows the "meaning" of the coefficients, specifically we interpret \( C(u,v) \) as a frequency coefficient

\[ C(u,v) = \text{strength of sinusoidal component with horizontal frequency } u \text{ and vertical frequency } v \]

\( (u,v) \text{ th component} = C(u,v) \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16} \)

show picture of frequency basis matrices

Notes:

there is fast implementation based on FFT

\( C(u,v) \in \{-1024, \ldots, 1023\} \)

can also view as a matrix multiply \( C = TB \), where \( T \) is a 64x64 orthogonal matrix and \( C \) and \( B \) are 64-dimensional column vectors.
3. **Quantization**

encoder produces index that describes coefficient:

\[ I(u,v) = \text{round}\left(\frac{B(u,v)}{M(u,v)}\right) \]  
(round to nearest integer)

example: \( I(u,v) = 7 \) means that \( 6.5 \leq M(u,v) \leq B(u,v) \leq 7.5 \ M(u,v) \)

\( M \) is called "the quantization matrix". It's a table of integer quantizer accuracies.

"quantization matrix" is term used only by JPEG

Small values lead to fine quantizations.

decoder converts index to a quantized coefficient:

\[ \hat{C}(u,v) = I(u,v) \ M(u,v) \]

\( M \) can be specified by the encoder. There is no official standard, but JPEG did list an example \( M_0 \) that most implementations use. Actually, they typically, use

\[ M = M_0 / f, \]  
where \( f \) is quality factor. Large \( f \) leads to high quality.

Handout quantization matrix
Figure 10.3: $8 \times 8$ DCT basis functions.
4. **binary encode**

**DC coeff's**

\[ C(0,0) = \text{avg of block B} = \text{DC coefficient} \]

neighboring blocks are similar
binary encode \( I(0,0) - \text{previous} I(0,0) \) using variable length code

\[ C(0,0) - \text{previous} \hat{C}(0,0) \]

an example of predictive encoding (DPCM)

**AC coeff's** (the other coefficients)

zig-zag runlength encoding -- arrange coefficients in zig-zag order \( Z_1 \ Z_2 \ Z_3 \ Z_4 \ldots \)

many of these are zero, so

transform into value and runlength sequence \( V_1, R_1, V_2, R_2, \ldots \)

\( V_i = \text{ith nonzero value}, \ R_i = \text{number of zeros since last value.} \ R_i \text{ limited to 16.} \)

transform value \( V \) into sign, size/category \( S \) and value \( W \)

\[ V = \text{sign}(V) \ (2^S - 1 + W), \ S = 1, \ldots, 10, \ W = 0, \ldots, 2^{S-1}-1 \]

\( V \) ranges from \( 2^S - 1 \) to \( 2^{S-1} \) and the negative of these

there are \( 2^{S-1} \) values so it takes \( S-1 \) bits plus sign bit

encode \( (S_i, R_i) \) together using variable-length encoding table. (show table)
encode \( W_i \) using \( S_i-1 \) bits.

**Performance**

rate and distortion

**Complexity**

encoder: 2.25 multiplies, 9.25 additions, 1 comparision per pixel
(Rabbani and Jones, p. 122)

**Issues:**

**Why?**

transform
scalar quantize one at a time
uniform quantizer
why this binary coding techniques

we see the pattern -- processing, quantization, binary coding
Annex K (informative)

Examples and guidelines

This annex provides examples of various tables, procedures, and other guidelines.

K.1 Quantization tables for luminance and chrominance components

Two examples of quantization tables are given in tables K.1 and K.2. These quantization values have been used with good results on 8 bit per sample Y,Cb,Cr images of the format illustrated in Figure 13. Note that these quantization values are appropriate for the DCT normalization defined in A.3.3.

Table K.1 Luminance quantization table

<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>68</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>

Table K.2 Chrominance quantization table

<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>

If these quantization values are divided by 2, the resulting reconstructed image is usually nearly indistinguishable from the source image.

K.2 A procedure for generating the lists which specify a Huffman code table

A Huffman table is generated from a collection of statistics in two steps. The first step is the generation of the list of lengths and values which are in accord with the rules for generating the Huffman code tables. The second step is the generation of the Huffman code table from the list of lengths and values.

The first step, the topic of this section, is needed only for custom Huffman table generation and is done only in the encoder. In this step the statistics are used to create a table associating each value to be coded with the size (in bits) of the corresponding Huffman code. This table is sorted by code size.

A procedure for creating a Huffman table for a set of up to 256 symbols is shown in Figure K.1. Three vectors are defined for this procedure:

- FREQ(V) frequency of occurrence of symbol V.
- CODESIZE(V) code size of symbol V.
- OTHERS(V) index to next symbol in chain of all symbols in current branch of code tree.

where V goes from 0 to 256.
An Image Block

\[ f(j, k) = \begin{bmatrix}
  139 & 144 & 149 & 153 & 155 & 155 & 155 & 155 \\
  144 & 151 & 153 & 156 & 159 & 156 & 156 & 156 \\
  150 & 155 & 160 & 163 & 158 & 156 & 156 & 156 \\
  159 & 161 & 162 & 160 & 160 & 159 & 159 & 159 \\
  159 & 160 & 161 & 162 & 155 & 155 & 155 & 155 \\
  161 & 161 & 161 & 161 & 160 & 157 & 157 & 157 \\
  162 & 162 & 161 & 163 & 162 & 157 & 157 & 157 \\
  162 & 162 & 161 & 161 & 163 & 158 & 158 & 158 
\end{bmatrix} \]
The DCT transform of the image block

\[
F(u, v) = \begin{bmatrix}
1260 & -1 & -12 & -5 & 2 & -2 & -3 & 1 \\
-23 & -17 & -6 & -3 & -3 & 0 & 0 & -1 \\
-11 & -9 & -2 & 2 & 0 & -1 & -1 & 0 \\
-7 & -2 & 0 & 1 & 1 & 0 & 0 & 0 \\
-1 & -1 & 1 & 2 & 0 & -1 & 1 & 1 \\
2 & 0 & 2 & 0 & -1 & 1 & 1 & -1 \\
-1 & 0 & 0 & -1 & 0 & 2 & 1 & -1 \\
-3 & 2 & -4 & -2 & 2 & 1 & -1 & 0
\end{bmatrix}
\]
The usual quantization matrix

\[ Q(u, v) = \begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99
\end{bmatrix} \]
The quantized indices

\[ F^*(u, v) = \begin{bmatrix}
79 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
-2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix} \]
zīg-zāg ordering

\[
\begin{bmatrix}
0 & 1 & 5 & 6 & 14 & 15 & 27 & 28 \\
2 & 4 & 7 & 13 & 16 & 26 & 29 & 42 \\
3 & 8 & 12 & 17 & 25 & 30 & 41 & 43 \\
9 & 11 & 18 & 24 & 31 & 40 & 44 & 53 \\
10 & 19 & 23 & 32 & 39 & 45 & 52 & 54 \\
20 & 22 & 33 & 38 & 46 & 51 & 55 & 60 \\
21 & 34 & 37 & 47 & 50 & 56 & 59 & 61 \\
35 & 36 & 48 & 49 & 57 & 58 & 62 & 63 \\
\end{bmatrix}
\]
<table>
<thead>
<tr>
<th>Category</th>
<th>AC Coefficient Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-1, 1$</td>
</tr>
<tr>
<td>2</td>
<td>$-3, -2, 2, 3$</td>
</tr>
<tr>
<td>3</td>
<td>$-7, \cdots, -4, 4, \cdots, 7$</td>
</tr>
<tr>
<td>4</td>
<td>$-15, \cdots, -8, 8, \cdots, 15$</td>
</tr>
<tr>
<td>5</td>
<td>$-31, \cdots, -16, 16, \cdots, 31$</td>
</tr>
<tr>
<td>6</td>
<td>$-63, \cdots, -32, 32, \cdots, 63$</td>
</tr>
<tr>
<td>7</td>
<td>$-127, \cdots, -64, 64, \cdots, 127$</td>
</tr>
<tr>
<td>8</td>
<td>$-255, \cdots, -128, 128, \cdots, 255$</td>
</tr>
<tr>
<td>9</td>
<td>$-511, \cdots, -256, 256, \cdots, 511$</td>
</tr>
<tr>
<td>10</td>
<td>$-1023, \cdots, -512, 512, \cdots, 1023$</td>
</tr>
</tbody>
</table>

Table 10.1: AC coefficient grouping.
<table>
<thead>
<tr>
<th>Category</th>
<th>AC Coefficient Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-1, 1$</td>
</tr>
<tr>
<td>2</td>
<td>$-3, -2, 2, 3$</td>
</tr>
<tr>
<td>3</td>
<td>$-7, \ldots, -4, 4, \ldots, 7$</td>
</tr>
<tr>
<td>4</td>
<td>$-15, \ldots, -8, 8, \ldots, 15$</td>
</tr>
<tr>
<td>5</td>
<td>$-31, \ldots, -16, 16, \ldots, 31$</td>
</tr>
<tr>
<td>6</td>
<td>$-63, \ldots, -32, 32, \ldots, 63$</td>
</tr>
<tr>
<td>7</td>
<td>$-127, \ldots, -64, 64, \ldots, 127$</td>
</tr>
<tr>
<td>8</td>
<td>$-255, \ldots, -128, 128, \ldots, 255$</td>
</tr>
<tr>
<td>9</td>
<td>$-511, \ldots, -256, 256, \ldots, 511$</td>
</tr>
<tr>
<td>10</td>
<td>$-1023, \ldots, -512, 512, \ldots, 1023$</td>
</tr>
</tbody>
</table>

Table 10.1: AC coefficient grouping.
<table>
<thead>
<tr>
<th>Zero Run</th>
<th>Category</th>
<th>Code length</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
<td>1011</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td>11010</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>6</td>
<td>111000</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>7</td>
<td>1111000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1100</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
<td>111001</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>1111001</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>9</td>
<td>111110110</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>11011</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>8</td>
<td>11111000</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
<td>111010</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>9</td>
<td>111110111</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>6</td>
<td>111011</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7</td>
<td>1111010</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>7</td>
<td>1111011</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>8</td>
<td>11111001</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
<td>11111010</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>9</td>
<td>11111000</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>9</td>
<td>11111001</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>9</td>
<td>11111010</td>
</tr>
<tr>
<td>16 0's</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End of Block (EOB)</td>
<td>4</td>
<td>1010</td>
<td></td>
</tr>
</tbody>
</table>

Table 10.2: Example of JPEG AC Huffman code table.

(rewind to the end)
reconstructed image block

\[ \hat{f}(j, k) = \begin{bmatrix}
  144 & 146 & 149 & 152 & 154 & 156 & 156 & 156 \\
  148 & 150 & 152 & 154 & 156 & 156 & 156 & 156 \\
  155 & 156 & 157 & 158 & 158 & 157 & 156 & 155 \\
  160 & 161 & 161 & 162 & 161 & 159 & 157 & 155 \\
  163 & 163 & 164 & 163 & 162 & 160 & 158 & 156 \\
  163 & 163 & 164 & 164 & 162 & 160 & 158 & 157 \\
  160 & 161 & 162 & 162 & 162 & 161 & 159 & 158 \\
  158 & 159 & 161 & 161 & 162 & 161 & 159 & 158 
\end{bmatrix} \]
$e(j, k) = \begin{bmatrix}
-5 & -2 & 0 & 1 & 1 & -1 & -1 & -1 \\
-4 & 1 & 1 & 2 & 3 & 0 & 0 & 0 \\
-5 & -1 & 3 & 5 & 0 & -1 & 0 & 1 \\
-1 & 0 & 1 & -2 & -1 & 0 & 2 & 4 \\
-4 & -3 & -3 & -1 & 0 & -5 & -3 & -1 \\
-2 & -2 & -3 & -3 & -2 & -3 & -1 & 0 \\
2 & 1 & -1 & 1 & 0 & -4 & -2 & -1 \\
4 & 3 & 0 & 0 & 1 & -3 & -1 & 0 
\end{bmatrix}$

errors in reconstructed block