THE BASELINE JPEG ALGORITHM (1/2)
-- SUMMARY --

1. It operates on 8×8 blocks of the input image
2. Mean-normalization (subtract 128 from each pixel)
3. Transform: DCT-transform each block
4. Quantization
   - An 8×8 quantization matrix Q is user-provided
   - Each block is divided by Q (point by point)
   - The terms are then rounded to their nearest integers
   - Remark: Up to 4 quantization matrices per image are allowed (for example, one for luminance, and one for each of the three color components)

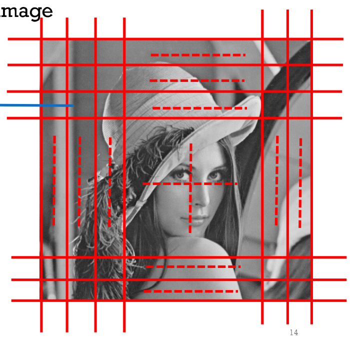
THE BASELINE JPEG ALGORITHM (2/2)
-- SUMMARY --

5. Entropy-coding of the DC coefficients (the top left coefficient of each quantized block) using DPCM+Huffman
   - Huffman-encode the DC residuals derived from the difference between each DC and the DC of the preceding block
6. Entropy-coding of the AC (i.e., non-DC) coefficients
   - Zigzag-order the quantized coefficients of each block
   - Record for each nonzero coefficient both its distance (called run) to the preceding nonzero coefficient in the zigzag sequence, and its value (called level)
   - Huffman code the [run,level] terms using one single Huffman table for all the AC's of the image

THE BASELINE JPEG ALGORITHM IN DETAIL (1/15)
-- BLOCK-ORIENTED --

1. It operates on 8×8 blocks of the input image

$$\begin{bmatrix} 143 & 147 & 149 & 152 & 156 & 147 & 146 & 149 \\ 151 & 146 & 143 & 154 & 148 & 144 & 153 & 132 \\ 147 & 143 & 145 & 149 & 144 & 145 & 128 & 133 \\ 152 & 145 & 145 & 144 & 146 & 134 & 130 & 137 \\ 146 & 143 & 142 & 147 & 124 & 127 & 139 & 138 \\ 139 & 145 & 139 & 127 & 126 & 135 & 139 & 141 \\ 145 & 137 & 124 & 130 & 138 & 136 & 140 & 144 \\ 144 & 124 & 136 & 134 & 137 & 139 & 142 & 145 \end{bmatrix}$$



Mean-normalize

DCT

Quantize

Entropy-code

THE BASELINE JPEG ALGORITHM IN DETAIL (2/15)
-- MEAN-NORMALIZATION OF BLOCKS --

2. Mean-normalization (subtract 128 from each pixel)

$$\begin{bmatrix} 143 & 147 & 149 & 152 & 156 & 147 & 146 & 149 \\ 151 & 146 & 143 & 154 & 148 & 144 & 153 & 132 \\ 147 & 143 & 145 & 149 & 144 & 145 & 128 & 133 \\ 152 & 145 & 145 & 144 & 146 & 134 & 130 & 137 \\ 146 & 143 & 142 & 147 & 124 & 127 & 139 & 138 \\ 139 & 145 & 139 & 127 & 126 & 135 & 139 & 141 \\ 145 & 137 & 124 & 130 & 138 & 136 & 140 & 144 \\ 144 & 124 & 136 & 134 & 137 & 139 & 142 & 145 \end{bmatrix} \longrightarrow \begin{bmatrix} 15 & 19 & 21 & 24 & 28 & 19 & 18 & 21 \\ 23 & 18 & 15 & 26 & 20 & 16 & 25 & 4 \\ 19 & 15 & 17 & 21 & 16 & 17 & 0 & 5 \\ 24 & 17 & 17 & 16 & 18 & 6 & 2 & 9 \\ 18 & 15 & 14 & 19 & -4 & -1 & 11 & 10 \\ 11 & 17 & 11 & -1 & -2 & 7 & 11 & 13 \\ 17 & 9 & -4 & 2 & 10 & 8 & 12 & 16 \\ 16 & -4 & 8 & 6 & 9 & 11 & 14 & 17 \end{bmatrix}$$

THE BASELINE JPEG ALGORITHM IN DETAIL (3/15)
-- DCT APPLIED ON EACH MEAN-NORMALIZED BLOCK --

3. Transform: DCT-transform each block

$$\begin{bmatrix} 15 & 19 & 21 & 24 & 28 & 19 & 18 & 21 \\ 23 & 18 & 15 & 26 & 20 & 16 & 25 & 4 \\ 19 & 15 & 17 & 21 & 16 & 17 & 0 & 5 \\ 24 & 17 & 17 & 16 & 18 & 6 & 2 & 9 \\ 18 & 15 & 14 & 19 & -4 & -1 & 11 & 10 \\ 11 & 17 & 11 & -1 & -2 & 7 & 11 & 13 \\ 17 & 9 & -4 & 2 & 10 & 8 & 12 & 16 \\ 16 & -4 & 8 & 6 & 9 & 11 & 14 & 17 \end{bmatrix} \xrightarrow{\textbf{DCT}} \underbrace{\begin{bmatrix} 103.4 & 12.4 & 6.0 & 2.1 & 8.1 & 5.7 & -0.7 & -0.4 \\ 31.7 & 11.6 & -22.8 & -0.7 & -0.1 & -0.5 & -4.7 & -1.8 \\ 11.0 & -21.2 & -3.7 & 8.8 & 2.3 & 0.1 & -0.2 & 5.6 \\ 0.2 & -4.9 & 10.3 & -8.6 & -8.9 & -2.5 & -7.6 & -1.4 \\ 4.9 & -0.4 & -1.9 & -8.9 & 6.6 & 2.6 & 3.6 & 3.6 \\ 0.7 & -0.9 & -0.9 & 4.1 & 7.2 & -15.5 & 2.8 & 1.8 \\ -3.1 & -1.0 & -2.5 & -5.5 & -5.2 & -6.7 & 10.7 & -1.1 \\ -2.9 & -0.2 & -1.2 & -2.7 & 3.6 & 2.6 & 0.4 & -6.4 \end{bmatrix}}_{\textbf{DCT output D}}$$

THE BASELINE JPEG ALGORITHM IN DETAIL (4/15)
-- QUANTIZATION --

4. Quantization
   - An 8×8 quantization matrix Q is user-provided
   - Each block D is divided by Q (point by point)
   - The terms are then rounded to their nearest integers (using NINT)

- Example:

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

THE BASELINE JPEG ALGORITHM IN DETAIL (5/15)

-- QUANTIZATION: ILLUSTRATION & EXPLANATION --

103.4	12.4	6.0	2.1	8.1	5.7	-0.7	-0.4
31.7	11.6	-22.8	-0.7	-0.1	-0.5	-4.7	-1.8
11.0	-21.2	-3.7	8.8	2.3	0.1	-0.2	5.6
0.2	-4.9	10.3	-8.6	-8.9	-2.5	-7.6	-1.4
4.9	-0.4	-1.9	-8.9	6.6	2.6	3.6	3.6
0.7	-0.9	-0.9	4.1	7.2	-15.5	2.8	1.8
-3.1	-1.0	-2.5	-5.5	-5.2	-6.7	10.7	-1.1
-2.9	-0.2	-1.2	-2.7	3.6	2.6	0.4	-6.4

./

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

DCT output D

Quantization matrix Q

NINT(D./Q)

$$\begin{bmatrix} 6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & -2 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

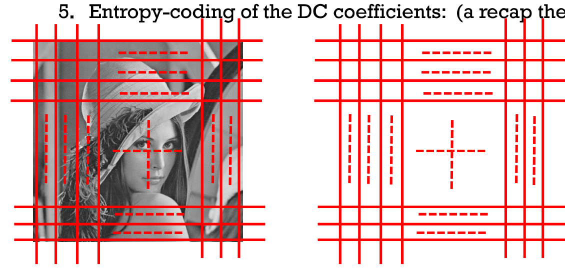
This is called the **DC** term of the block

- Q specifies 64 different quantizers, one per frequency coefficient
- As you move to the bottom right of Q, the numbers tend to get larger. Why?
- A quantizer Q_a where $Q_a(x) = \text{NINT}\left(\frac{x}{a}\right)$ is a uniform quantizer of interval length $= a$.???

THE BASELINE JPEG ALGORITHM IN DETAIL (6/15)

-- ENTROPY CODING OF THE DC TERMS --

5. Entropy-coding of the DC coefficients: (a recap then general description)



Mean-normalize+DCT+Quantize all the blocks

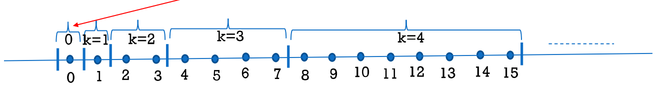
a. Collect all the DC terms, one DC term from each DCTed+quantized block, forming a 1D sequence
b. Apply DPCM on the sequence
c. Apply Huffman coding as explained next

Note: this is the only place in JPEG where any "processing-interactions" between blocks occur

THE BASELINE JPEG ALGORITHM IN DETAIL (7/15)

-- DETAILS OF ENTROPY CODING OF THE DC TERMS (1/2) --

5. Entropy-coding of the DC coefficients:
   a. The DC residuals are in the range [-2047,2047]
   b. Thus, the magnitude of each residual is between 0 and $2047 = 2^{11} - 1$, inclusive.
   c. Divide this range into 12 subranges, or categories, where category k ranges from 2^{k-1} to $2^k - 1$ inclusive.
      - Note that $0 \le k \le 11$, and for $k = 0$, category 0 has only the integer 0



0

THE BASELINE JPEG ALGORITHM IN DETAIL (8/15)

-- DETAILS OF ENTROPY CODING OF THE DC TERMS (2/2) --

5. Entropy-coding of the DC coefficients: Let r be a DC residual.
   d. $r \in [-2047{,}2407] \Rightarrow 0 \le |r| \le 2047 = 2^{10} + 1023 \Rightarrow |r| = 2^{k-1} + t$, for some k and t,
      - $0 \le t \le 2^{k-1} - 1,\ 0 \le k \le 11$
      - t can be represented in binary using $k - 1$ bits.

   If $|r| = 0$, take $k = 0$ and $t = 0$;
   If $|r| = 1$, then $k = 1$ and $t = 0$

   e. Therefore, r can be uniquely represented by $s, k,$ and t, where s is the sign of r, and k and t are as above. Represent $r \equiv [k, s, t]$ temporarily
   f. Develop a Huffman code for the 12 categories ($0 \le k \le 11$), where every codeword is at most 16 bits long

   Exercise: How would you find the probabilities of the 12 categories, for a given input image?

   g. Encode each DC residual r as a binary string hsm where
      - h is the Huffman codeword of the residual's category k
      - s= sign of the residual; s=0 if the residual is negative, 1 if positive
      - m= the $(k-1)$-bit binary representation of t

   Exercise: How would you modify the Huffman algorithm to produce a Huffman tree of height at most 16?

THE BASELINE JPEG ALGORITHM IN DETAIL (9/15)

-- ENTROPY CODING OF THE AC TERMS (1/7) --

6. Entropy-code of the AC (i.e., non-DC) coefficients of each block separately
   a. Zigzag ordering of NINT(D./Q)
   b. List the elements in that order : 6 1 4 1 1 0 0 -2 -2 allzeros

$$\begin{bmatrix} 6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & -2 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

THE BASELINE JPEG ALGORITHM IN DETAIL (10/15)

-- ENTROPY CODING OF THE AC TERMS (2/7) --

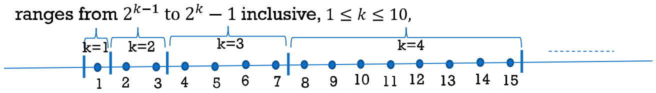
c. Record for each nonzero AC coefficient both its distance (called run) to the preceding nonzero coefficient in the zigzag sequence, and its value (called level): represent every non-zero AC as [run,level]
   - Example: 6 **1 4 1 1 0 0 -2 -2 allzeros** The AC terms
   - **[0,1] [0,4] [0,1] [0,1] [2,-2] [0,-2] EOB // EOB= end of block**

b. Huffman code the [run,level] terms using one single Huffman table for all the AC's of the image (explained next)

THE BASELINE JPEG ALGORITHM IN DETAIL (11/15)

-- ENTROPY CODING OF THE AC TERMS (3/7) --

1. All non-zero AC terms are of magnitude $\leq\ 2^{10}-1$
2. Let x be a non-zero AC term, $x \equiv [run, level]$, and let $d = run$, i.e., the length of the zero run between x and the previous nonzero AC term.
   - $level$ =value of x. For more convenience, we'll use x instead of *level*
3. $1 \leq |x| \leq 2^{10}-1$
4. Divide the range from 1 to $2^{10}-1$, into 10 categories, where category k ranges from 2^{k-1} to $2^k - 1$ inclusive, $1 \leq k \leq 10$,



THE BASELINE JPEG ALGORITHM IN DETAIL (12/15)

-- ENTROPY CODING OF THE AC TERMS (4/7) --

5. Represent the value of x by its sign s and its magnitude $|x|$
   - $s\ =\ 0$ if $x < 0$, $\qquad s = 1$ if $x > 0$
6. For whatever the value of $|x|$, there is a unique k where $2^{k-1} \leq |x| \leq 2^k - 1$
   - k is the category of x
7. $|x| = 2^{k-1} + t$, where $0 \leq t \leq 2^{k-1} - 1$. So, t can be represented with $k-1$ bits
8. Therefore, the value of x can be uniquely represented by $s, k,$ and t
9. Combined with its run d, the AC term x can be presented as $[d, k, t, s]$
   - This is intermediate epresentation, especially for d and k

THE BASELINE JPEG ALGORITHM IN DETAIL (13/15)

-- ENTROPY CODING OF THE AC TERMS (5/7) --

We are at $x \equiv [d, k, t, s]$

10. The run-length d is between 0 and 62
11. Express $d\ =\ 15p\ +\ r$, where $r = 0,1,2, \dots, 14$ and $p = 0, 1, 2, 3, 4$
12. r can be represented with 4 bits $r_3r_2r_1r_0$, different from 1111 (why?)
13. p can be represented with $11110000_1\ 11110000_2\ \dots\ 11110000_p$
14. This implies that d is $11110000_1\ 11110000_2\ \dots\ 11110000_p\ \ r_3r_2r_1r_0$
15. The category k, being between 1 and 10, can be represented with 4 bits $k_3k_2k_1k_0$
16. Therefore, the $[d, k]$ in the $[d, k, t, s]$ representation of AC term x, is represented as $11110000_1\ 11110000_2\ \dots\ 11110000_p\ \ r_3r_2r_1r_0k_3k_2k_1k_0$
17. This representation of $[d, k]$ can be viewed as a sequence of $p + 1$ bytes

THE BASELINE JPEG ALGORITHM IN DETAIL (14/15)

-- ENTROPY CODING OF THE AC TERMS (6/7) --

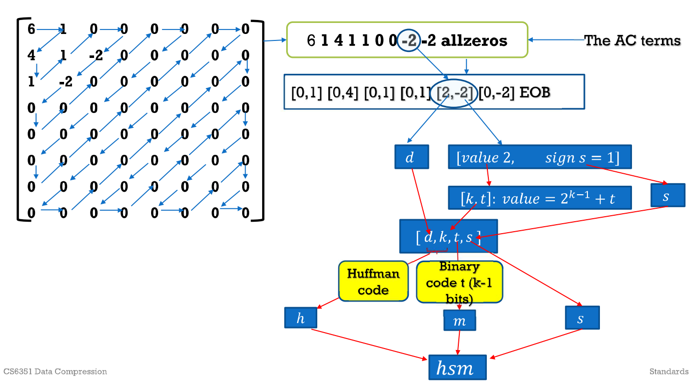
16. Therefore, the $[d, k]$ in the $[d, k, t, s]$ representation of AC term x, is represented as $11110000_1\ 11110000_2\ \dots\ 1111000_p\ \ r_3r_2r_1r_0k_3k_2k_1k_0$
17. This representation of $[d, k]$ can be viewed as a sequence of $p + 1$ bytes
18. The last byte $r_3r_2r_1r_0k_3k_2k_1k_0$ represents $15 \times 10 =\ 150$ values (why?)
19. Add to those the byte 11110000 and the end-of-block (EOB) symbol to signal the end of the nonzero AC terms in a block
20. This results in 152 different symbols needed for the $[d, k]$ representations
21. Build a Huffman table for those 152 symbols, where every codeword is at most 16 bits long
    - The probabilities of those symbols can be derived from the image being coded

THE BASELINE JPEG ALGORITHM IN DETAIL (15/15)

-- ENTROPY CODING OF THE AC TERMS (7/7) --

22. JPEG encodes each quantized non-zero AC term, whose intermediary representation is $[d, k, t, s]$, as ***hsm*** where
    - ***h*** is the Huffman codeword of $[d, k]$ (just explained on the previous slide)
    - ***s*** is the 1-bit sign of the AC term; s=0 if AC term negative, 1 if positive
    - ***m*** is the $(k-1)$-bit binary representation of t
23. End of AC coding and the whole JPEG encoding

DIAGRAM SUMMARY OF AC ENCODING



CONTINUING WITH THE EXAMPLE OF THE BLOCK (1/5)

-- EXAMPLE HUFFMAN TABLE FOR LENA --

d=Length of Zero Run	Category k	Code-length	Codeword
0	1	2	00
0	2	2	01
0	3	3	100
0	4	4	1011
0	5	5	11010
0	6	6	111000
0	7	7	1111000
…	…	…	…
1	1	4	1100
1	2	6	111001
1	3	7	1111001
1	4	9	111110110
…	…	…	…

d=Length of Zero Run	Category k	Code-length	Codeword
2	1	5	11011
2	2	8	11111000
…	…	…	…
3	1	6	111010
3	2	9	111110111
…	…	…	…
4	1	6	111011
5	1	7	1111010
6	1	7	1111011
7	1	8	11111001
8	1	8	11111010
9	1	9	111111000
10	1	9	111111001
…	…	…	…
EOB		4	1010

CONTINUING WITH THE EXAMPLE OF THE BLOCK (2/5)

The DC value 6 is coded separately along with other DC terms

- The block example in zigzag order:
  - 6 **1 4 1 1 0 0 -2 -2 allzeros**
  - **[0,1] [0,4] [0,1] [0,1] [2,-2] [0,-2] EOB**
- The AC terms are coded as follows:
  - Represent the sequence **[0,1] [0,4] [0,1] [0,1] [2,-2] [0,-2] EOB** as a sequence of $[d,k,t,s]$'s where t for now is in decimal $[d,x] \rightarrow [d,k,t,s]$ where $|x| = 2^{k-1}+t,\quad s = \text{sign}(x)$
    - [0,1,0,1] [0,3,0,1] [0,1,0,1] [0,1,0,1] [2,2,0,0] [0,2,0,0] EOB
  - Illustrations of $[d,x] \rightarrow [d,k,t,s]$:
    - [0,4] -> [0,3,0,1] because $4 = 2^{3-1}+0$, so $k=3, t=0$, and $s=1$ since $4>0$
    - [2,-2]->[2,2,0,0] because $|-2| = 2 = 2^{2-1}+0$, so $k=2, t=0$, and $s=0$ since -2<0
  - Now as sequence of $[d,k,t,s]$'s where t is now in $(k-1)$-bit binary
    [0,1,-,1] [0,3,00,1] [0,1,-,1] [0,1,-,1] [2,2,0,0] [0,2,0,0] EOB

CONTINUING WITH THE EXAMPLE OF THE BLOCK (3/5)

- The block example in zigzag order:
  - 6 **1 4 1 1 0 0 -2 -2 allzeros**
  - **[0,1] [0,4] [0,1] [0,1] [2,-2] [0,-2] EOB**
- The AC terms are coded as follows:
  - Sequence: [0,1,-,1] [0,3,00,1] [0,1,-,1] [0,1,-,1] [2,2,0,0] [0,2,0,0] EOB
  - Code([0,1,-,1])is hsm where h=Huffman([0,1])=00, s=1, m=empty, so: 00 1
  - Code([0,3,00,1]) is hsm where h=Huffman([0,3])=100, s=1, m=00, so: 100 1 00
  - Code([2,2,0,0]) is hsm where h=Huffman([2,2])= 11111000, s=0, m=0, so: 11111000 0 0
  - Code([0,2,0,0]) is hsm where h=Huffman([0,2])= 01, s=0, m=0, so: 01 0 0
  - Code(EOB) is 1010
- The final code of the ACs in block: 00 1 100 1 00 00 1 00 1 11111000 0 0 01 0 0 1010

CONTINUING WITH THE EXAMPLE OF THE BLOCK (4/5)

-- EXAMPLE HUFFMAN TABLE FOR LENA --

d=Length of Zero Run	Category k	Code-length	Codeword
0	1	2	00
0	2	2	01
0	3	3	100
0	4	4	1011
0	5	5	11010
0	6	6	111000
0	7	7	1111000
…	…	…	…
1	1	4	1100
1	2	6	111001
1	3	7	1111001
1	4	9	111110110
…	…	…	…

d=Length of Zero Run	Category k	Code-length	Codeword
2	1	5	11011
2	2	8	11111000
…	…	…	…
3	1	6	111010
3	2	9	111110111
…	…	…	…
4	1	6	111011
5	1	7	1111010
6	1	7	1111011
7	1	8	11111001
8	1	8	11111010
9	1	9	111111000
10	1	9	111111001
…	…	…	…
EOB		4	1010

CONTINUING WITH THE EXAMPLE OF THE BLOCK (5/5)

- The final code of the ACs in block: 00 1 100 1 00 00 1 00 1 11111000 0 0 01 0 0 1010
- Number of bits to code the AC terms: 33 bits
- Bitrate per symbol (out of the 63 AC symbols): $\frac{33}{63} = 0.52$ bits/symbol
  - The denominator 63 is the gross number of all the AC terms in the block
- Compression ratio based on those ACs alone: $\frac{63\times 8}{33} = 15.27$
  - The numerator is the number of bits in the uncoded 8×8 block at 8 bits per pixel
  - 8×8=64 but we removed one pixel to correspond to the non-counting of the DC term in these simple calculations

JPEG DECODING

1. Entropy-decode the bitstream back to the quantized blocks
2. Dequantize:
   - Multiply each block coefficient by the corresponding coefficient of the quantization matrix Q
3. Apply the inverse DCT transform on each block
4. Denormalize: add 128 to each coefficient

JPEG DECODING EXAMPLE (1/5)

- Take the example block before, and assume we performed the entropy decoding on it, returning it back to quantized form, then multiply by Q

$$\underbrace{\begin{bmatrix} 6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & -2 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\textbf{Quantized block}} .\times \underbrace{\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}}_{\textbf{Quantization matrix Q}}$$

JPEG DECODING EXAMPLE (2/5)

- Dequantized Block

$$\underbrace{\begin{bmatrix} 96 & 11 & 0 & 0 & 0 & 0 & 0 & 0 \\ 48 & 12 & -28 & 0 & 0 & 0 & 0 & 0 \\ 14 & -26 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\textbf{Dequantized block}}$$

JPEG DECODING EXAMPLE (3/5)

- Apply inverse DCT on it, and denormalize (by adding 128), we get:

$$\underbrace{\begin{bmatrix} 143 & 147 & 153 & 157 & 157 & 154 & 149 & 145 \\ 145 & 147 & 151 & 154 & 153 & 149 & 144 & 141 \\ 146 & 147 & 149 & 149 & 146 & 142 & 137 & 134 \\ 146 & 146 & 145 & 142 & 139 & 135 & 132 & 130 \\ 145 & 143 & 140 & 136 & 133 & 131 & 130 & 130 \\ 141 & 138 & 134 & 131 & 130 & 131 & 134 & 135 \\ 136 & 134 & 130 & 128 & 129 & 133 & 139 & 142 \\ 133 & 131 & 127 & 126 & 129 & 135 & 142 & 147 \end{bmatrix}}_{\textbf{Reconstructed block}}$$

JPEG DECODING EXAMPLE (4/5)

- Compute the error (original block – reconstructed block):

$$\underbrace{\begin{bmatrix} 143 & 147 & 149 & 152 & 156 & 147 & 146 & 149 \\ 151 & 146 & 143 & 154 & 148 & 144 & 153 & 132 \\ 147 & 143 & 145 & 149 & 144 & 145 & 128 & 133 \\ 152 & 145 & 145 & 144 & 146 & 134 & 130 & 137 \\ 146 & 143 & 142 & 147 & 124 & 127 & 139 & 138 \\ 139 & 145 & 139 & 127 & 126 & 135 & 139 & 141 \\ 145 & 137 & 124 & 130 & 138 & 136 & 140 & 144 \\ 144 & 124 & 136 & 134 & 137 & 139 & 142 & 145 \end{bmatrix}}_{\textbf{Original block}} - \underbrace{\begin{bmatrix} 143 & 147 & 153 & 157 & 157 & 154 & 149 & 145 \\ 145 & 147 & 151 & 154 & 153 & 149 & 144 & 141 \\ 146 & 147 & 149 & 149 & 146 & 142 & 137 & 134 \\ 146 & 146 & 145 & 142 & 139 & 135 & 132 & 130 \\ 145 & 143 & 140 & 136 & 133 & 131 & 130 & 130 \\ 141 & 138 & 134 & 131 & 130 & 131 & 134 & 135 \\ 136 & 134 & 130 & 128 & 129 & 133 & 139 & 142 \\ 133 & 131 & 127 & 126 & 129 & 135 & 142 & 147 \end{bmatrix}}_{\textbf{Reconstructed block}}$$

JPEG DECODING EXAMPLE (5/5)

- The resulting error block is:

$$\underbrace{\begin{bmatrix} 0 & 0 & -4 & -5 & -1 & -7 & -3 & 4 \\ 6 & -1 & -8 & 0 & -5 & -5 & 9 & -9 \\ 1 & -4 & -4 & 0 & -2 & 3 & -9 & -1 \\ 6 & -1 & 0 & 2 & 7 & -1 & -2 & 7 \\ 1 & 0 & 2 & 11 & -9 & -4 & 9 & 8 \\ -2 & 7 & 5 & -4 & -4 & 4 & 5 & 6 \\ 9 & 3 & -6 & 2 & 9 & 3 & 1 & 2 \\ 11 & -7 & 9 & 8 & 8 & 4 & 0 & -2 \end{bmatrix}}_{\textbf{Error block}}$$

- MSE=5.2, SNR=28.31