Image Compression

(Chapter 8)

Chapter Outline

This chapter deals with algorithms for compressing grayscale and colour images. *Before this chapter you should have studied the e-lecture on images and Appendix B.2.* After this chapter you should:

- have understood the nature of redundancy in grayscale and colour images.
- have understood the JPEG baseline lossy and JPEG lossless algorithms to the extent that:
  - you are able to execute by hand a (compression or decompression) algorithm on a given small sample input including the forward and inverse Discrete Cosine transform.
  - you are able to describe the (compression or decompression) algorithm in your own words, in a reasonably precise manner.
  - you are able to understand and explain, why these algorithms perform well. Your explanation and understanding should convey the intuition in a reasonably precise manner.

Outline

- Spatial redundancy.
- Lossless Compression.
  - Bi-level images.
  - Indexed colour images.
  - Grayscale images.
- Lossy compression.

Spatial Redundancy

- Images consist of *regions* of “similar” colour. Nature of similarity varies: 1-bit, 8-bit, 24-bit colour?
    - Bi-level images (e.g. Fax compression), 8-bit indexed images: adjacent colours will often be *identical*.
    - 24-bit colour images, 8-bit grayscale image: adjacent colours will be close in value, but *not identical*.
- This similarity is a little like *context* information in text compression (each pixel’s color can be predicted better by knowing the color of the pixels around it).

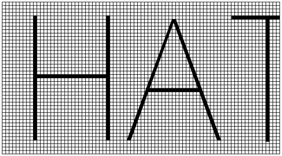
Modelling Spatial Redundancy: Bi-Level Images

"Images consist of *regions* of "similar" colour."

- Bi-level images – only black and white pixels.
  - "Similar" means same.
  - The image is divided into regions of *same* colour.
  - If we scan the image row by row, a pixel tends to be the *same* as the pixel before it.
- Run-length encoding (2-state Markov model).
- But Markov model does not model everything:
  - Successive rows are likely to be very similar.
  - If there is a letter 'I' in a fax then many successive rows will have a black run at the same place!

Bi-Level Image (ITU-T T.4 G3 2D)

Already seen: ITU-T T.4 Group 3 1D algorithm for compressing fax images.



Full spatial redundancy not exploited.

Line 1: 7W, 1B, 20W, 1B, 19W, 2B, 16W, 13B, ...

Line 2: 7W, 1B, 20W, 1B, 19W, 2B, 26W, 1B, ...

Successive runs have many similarities.

Modified READ encoding

Line 1: 7W, 1B, 20W, 1B, 19W, 2B, 16W, 13B, ...
Line 2: 7W, 1B, 20W, 1B, 19W, 2B, 26W, 1B, ...

- Simple idea: code Line 2 with respect to Line 1.

  0, 0, 0, 0, 0, 0, +10, -12
- The result has lots of 0s, so more compressible than just runs.
- Unfortunately, this "idea" is too simple: e.g. successive lines don't have same number of runs, real algorithm a bit more complicated.
- There are still better algorithms:
  - modified modified READ (ITU-T T.6 Group 4 2D algorithm)
  - JBIG/JBIG2. Also works for bit-plane lossless compression of gray-scale and colour images.

Compression Performance

Comparison of Bi-Level Image Compression Algorithms [Arps and Truong, 1994]

- Original file size: 460KB
- Compressed file size (KB):

Source description	1D	MR	MMR	JBIG
Letter	20.6	14.3	8.5	6.7
Sparse text	26.2	16.7	10.0	7.7
Dense text	135.7	105.7	92.1	70.7

Lossless: Indexed Colour

"Images consist of *regions* of "similar" colour."

- Indexed colour: only 256 different colours in the image.
    - Maybe only 4-5 shades of sky blue e.g. in the image.
    - "Similar" often means "same".
    - There are 256 different colours, so runs may not be too long.
        - Definitely not Capon model – it has only 2 states!
    - From a pixel's neighbours we can make a fair guess about its colour.
- Markov model + dictionary compression (LZW) used in GIF.

Modelling Spatial Redundancy: Grayscale

"Images consist of *regions* of 'similar' colour."

- Within an area, grayscale values are *similar*, not *same*.

▷ For most pixels: values similar to adjacent pixels.

  - Linear system model can be used within regions.

- At boundaries, sharp differences can show up.

▷ Most pixels *inside* regions.

  - $a \times b$ rectangle: area $= ab$, perimeter $= 2(a+b)$.
  - r radius circle: area $= \pi r^2$, perimeter $= 2\pi r$.

JPEG Lossless Algorithm

- JPEG = Joint Picture Experts Group, standardising organisation.
- Current standard finalized in 1994.
  - Successor standard JPEG 2000 was ratified in 2003 but is not in widespread use.
- The most common JPEG algorithm is the *lossy baseline* algorithm. This is covered later.

JPEG Predictive Coding

- The *context* of a pixel is the pixels around it.
- We try to *predict* the value of the pixel using the values of the pixels around it.
- If the prediction is $\hat{x}$ and the actual value is x, then the error in prediction is $x - \hat{x}$.
    - The error in prediction is called the *residual* value.
- If predictions are good, residual values should be strongly skewed to small values around 0, and are compressed using Huffman or arithmetic coding.

JPEG Predictive Coding

- Decompression is easy:
    - get all the residual values (decoding Huffman e.g.)
    - decode the pixels around a pixel
    - use the same prediction algorithm/formula at the decoder to predict the value.
    - Add in the residual value for that pixel.
- Need to avoid circular references: value of pixel x used to predict pixel y and vice versa.
    - Scan the image row-by-row.
    - Use pixels above and to the left of current pixel to predict current pixel.

JPEG Predictive Coding

- Input is a grayscale image I, with 256 shades of grey.
  - Each pixel is an integer from 0 (black) to 255 (white).

C	B
A	X

- JPEG lossless specifies eight predictors:

```
0. No prediction   4. X = A + B - C
1. X = A           5. X = A + (B - C)/2
2. X = B           6. X = B + (A - C)/2
3. X = C           7. X = (A + B)/2
```

- We use one of the predictors above to predict pixels except:
  - Top left value in the image: always left as is.
  - Top row: always use predictor X=A.
  - Left column: always use X=B.

Example

Input image:

124	122	123	120	121	122	123	124
123	122	121	124	120	121	119	121
128	126	124	122	119	118	117	116
126	124	62	64	62	63	58	56
127	125	63	61	60	58	60	62
122	124	63	61	60	58	60	62
123	120	63	61	61	59	61	62
127	125	62	60	61	59	61	62

Predictor $X = A$.

Example (cont'd)

124	122	123	120	121	122	123	124
123	122	121	124	120	121	119	121
128	126	124	122	119	118	117	116
126	124	62	64	62	63	58	56
127	125	63	61	60	58	60	62
122	124	63	61	60	58	60	62
123	120	63	61	61	59	61	62
127	125	62	60	61	59	61	62

Predictor $X = A$.

Example (cont'd)

124	-2 (122)	123	120	121	122	123	124
123	122	121	124	120	121	119	121
128	126	124	122	119	118	117	116
126	124	62	64	62	63	58	56
127	125	63	61	60	58	60	62
122	124	63	61	60	58	60	62
123	120	63	61	61	59	61	62
127	125	62	60	61	59	61	62

Predictor $X = A$.

Example (cont'd)

124	-2 (122)	123	120	121	122	123	124
123	122	121	124	120	121	119	121
128	126	124	122	119	118	117	116
126	124	62	64	62	63	58	56
127	125	63	61	60	58	60	62
122	124	63	61	60	58	60	62
123	120	63	61	61	59	61	62
127	125	62	60	61	59	61	62

Predictor $X = A$.

Example (cont'd)

124	−2	1 (123)	120	121	122	123	124
123	122	121	124	120	121	119	121
128	126	124	122	119	118	117	116
126	124	62	64	62	63	58	56
127	125	63	61	60	58	60	62
122	124	63	61	60	58	60	62
123	120	63	61	61	59	61	62
127	125	62	60	61	59	61	62

Predictor $X = A$.

Example (cont'd)

124	-2	1	-3	1	1	1	1
123	-1	-1	3	-4	1	-2	2
128	-2	-2	-2	-3	-1	-1	-1
126	-2	-62	2	-2	1	-5	-2
127	-2	-62	-2	-1	-2	2	2
122	2	-61	-2	-1	-2	2	2
123	-3	-57	-2	0	-2	2	1
127	-2	-63	-2	1	-2	2	1

Predictor $X = A$ (use $X = B$ for leftmost column).

Example (cont'd)

124	-2	1	-3	1	1	1	1
-1	-1	-1	3	-4	1	-2	2
5	-2	-2	-2	-3	-1	-1	-1
-2	-2	-62	2	-2	1	-5	-2
1	-2	-62	-2	-1	-2	2	2
-5	2	-61	-2	-1	-2	2	2
1	-3	-57	-2	0	-2	2	1
4	-2	-63	-2	1	-2	2	1

Predictor $X = A$ (use $X = B$ for leftmost column).

Distribution of Values in Residual Image

Original

56	1/64	124	6/64
58	3/64	125	2/64
59	2/64	126	2/64
60	5/64	127	2/64
61	7/64	128	1/64
62	7/64		
63	4/64		
64	1/64		
116	1/64		
117	1/64		
118	1/64		
119	2/64		
120	3/64		
121	4/64		
122	5/64		
123	4/64		

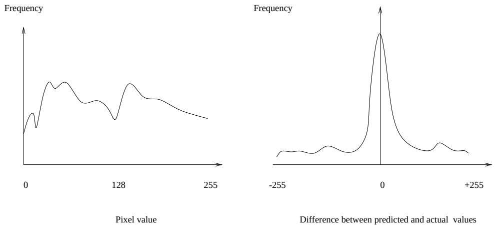
Residual

-63	1/64
-62	2/64
-61	1/64
-57	1/64
-5	2/64
-4	1/64
-3	3/64
-2	19/64
-1	8/64
0	1/64
1	12/64
2	9/64
3	1/64
4	1/64
5	1/64
124	1/64

- Raw data: 8 bits/pixel.
- We will Huffman/arithmetic coding to residual image.
- Entropy of original image (left) is 4.10 bits/pixel.
- Entropy of residual image (right) is 3.11 bits/pixel.

JPEG Lossless: Concluding Remarks

- Example image had only two regions, one mid-grey (around 128) and one dark grey (around 64). Typical images will have many more regions, e.g. regions around 220, regions around 180 etc.
  - hence, the entropy of the *original* image would be quite high, much closer to 8 bits/pixel.



- Can also be applied once each to Y, C_b and C_r components of a color image to get lossless color image compression.

Baseline JPEG: Outline

- First lossy compression algorithm we teach.
- Most commonly used JPEG algorithm.
- Overview:
  - Basics.
  - Colour-space transformation.
  - Downsampling.
  - Discrete Cosine Transform.
  - Quantisation.
  - Zigzag scan/RLE/Huffmann
  - Customisation

Baseline JPEG

- Want high compression ratio, "visually lossless" compression.
- "Visually lossless": figure out what the Human Visual System can't distinguish, and "delete" it.
- Exploit human visual system limitations (HVSL):
  - HVSL 1 Eye is far more sensitive to intensity (brightness) variations than colour variations.
  - HVSL 2 Eye cannot perceive rapid variation in small areas at normal resolution (high spatial frequencies).
- Also exploit Spatial Redundancy:
  - Adjacent pixels have similar values.

Baseline JPEG: Main steps

- Perform Colour-space Transformation.
- Downsample chrominance information (HVSL, SR).

Divide downsampled Y, C_b and C_r images into 8×8 blocks of pixels. For each block do:

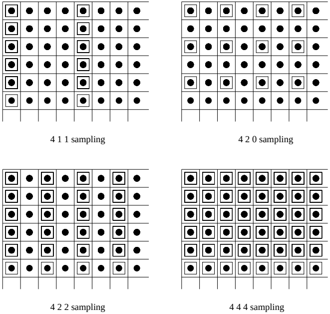
- Perform Discrete Cosine Transform (DCT).
- Quantise. (HVSL)
- Coding of quantised DCT coefficients:
  - Zig-zag scan and RLE. (HVSL)
  - Differential encoding of D/C coefficients. (SR)
  - Entropy coding.

Color-Space Transform

- Convert $RGB \rightarrow YC_bC_r$.
- Y component is "luminance" and represents the intensity (brightness) information. C_b and C_r are "chrominance" and represent colour information.
- This transformation can be reversed. It does not perform any compression.
- Why use it?
  - HVSL 1 Eye is far more sensitive to intensity variations than colour variations.
- We will compress the Y component in a less lossy way.

Downsampling

- "Downsample" C_b and C_r information.
  4:2:2 downsampling: 4/4 Y, 2/4 C_b, 2/4 C_r samples.
  4:1:1 downsampling: 4/4 Y, 1/4 C_b, 1/4 C_r samples.
  4:2:0 downsampling: 4/4 Y, 1/4 C_b, 1/4 C_r samples.



- 4:2:0 downsampling gives 50% reduction:
  - $4 + 4 + 4 \rightarrow 4 + 1 + 1$

Baseline JPEG: Main steps

$\surd$ Perform Colour-space Transformation.

$\surd$ Downsample chrominance information (HVSL, SR).

Divide downsampled Y, C_b and C_r images into 8×8 blocks of pixels. For each block do:

- Perform Discrete Cosine Transform (DCT).
- Quantise. (HVSL)
- Coding of quantised DCT coefficients:
  - Zig-zag scan and RLE. (HVSL)
  - Differential encoding of D/C coefficients. (SR)
  - Entropy coding.

Discrete Cosine Transform (DCT)

- Performs a couple of matrix multiplications.
- Determines the kind of variation there is across each block. Is the block relatively flat (all pixel values similar), or is there a lot of variation?
- This transformation can be reversed. It does not perform any compression.
- Why use it?
  - HVSL 2 Eye cannot perceive rapid variation in small areas at normal resolution (high spatial frequencies).
- We will compress the blocks with variations much more.

Discrete Cosine Transform (DCT)

The $n \times n$ DCT matrix is given by:

$$D_{i,j} = \begin{cases} \sqrt{\frac{1}{n}} \cos(i \cdot (2j+1) \cdot \pi/(2 \cdot n)) & \text{if } i = 0, \\ \sqrt{\frac{2}{n}} \cos(i \cdot (2j+1) \cdot \pi/(2 \cdot n)) & \text{otherwise.} \end{cases}$$

Degrees are measured in *radians*, so 2π radians is $360°$. For $n = 2$:

$$\begin{aligned} D_{0,0} = D_{0,1} &= \sqrt{1/2}\cos(0) = 1/\sqrt{2} \\ D_{1,0} &= \sqrt{1/2}\cos(\pi/4) = 1/\sqrt{2} \\ D_{1,1} &= \sqrt{1/2}\cos(3\pi/4) = -1/\sqrt{2} \end{aligned}$$

Note: $1/\sqrt{2} = 0.70710678\ldots = 0.71$ (2dp).

DCT matrix

2×2 DCT matrix (to 2dp):

$$\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} = \begin{pmatrix} 0.71 & 0.71 \\ 0.71 & -0.71 \end{pmatrix}$$

8×8 DCT matrix (used in JPEG), to 2dp:

```
0.35  0.35  0.35  0.35  0.35  0.35  0.35  0.35
0.49  0.42  0.28  0.10 -0.10 -0.28 -0.42 -0.49
0.46  0.19 -0.19 -0.46 -0.46 -0.19  0.19  0.46
0.42 -0.10 -0.49 -0.28  0.28  0.49  0.10 -0.42
0.35 -0.35 -0.35  0.35  0.35 -0.35 -0.35  0.35
0.28 -0.49  0.10  0.42 -0.42 -0.10  0.49 -0.28
0.19 -0.46  0.46 -0.19 -0.19  0.46 -0.46  0.19
0.10 -0.28  0.42 -0.49  0.49 -0.42  0.28 -0.10
```

8×8 DCT

Given 8×8 block A of pixels. Let D be the 8×8 DCT matrix.

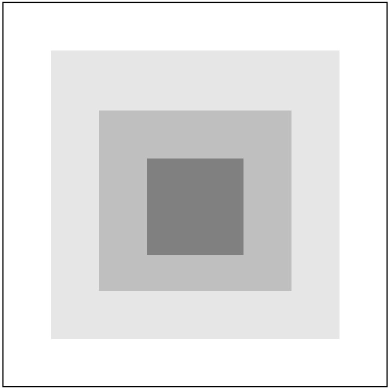
Forward DCT:

1. Subtract 128 from all entries of A, giving A'.
2. Compute the matrix $B = DA'D^T$ where D^T is the transpose of D.

Inverse DCT:

1. Recover $A' = D^T BD$.
2. Add 128 to all entries of A', giving A.

Example 8×8 block



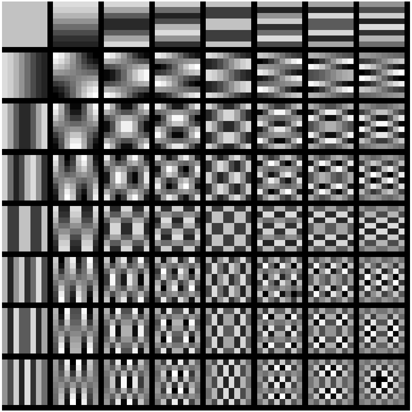
Example 8×8 block

255	255	255	255	255	255	255	255
255	223	223	223	223	223	223	255
255	223	191	191	191	191	223	255
255	223	191	127	127	191	223	255
255	223	191	127	127	191	223	255
255	223	191	191	191	191	223	255
255	223	223	223	223	223	223	255
255	255	255	255	255	255	255	255

Output of DCT

776	0	164	0	16	0	19	-0
-0	-0	0	0	0	0	-0	-0
164	0	-137	-0	21	0	-11	-0
0	0	-0	0	0	0	0	-0
16	-0	21	0	-48	0	9	0
0	0	0	0	-0	0	0	-0
19	-0	-11	0	9	0	-23	-0
-0	-0	-0	-0	-0	-0	-0	0

DCT "basic patterns"



- This is an 8×8 matrix of "basic patterns".
- The (i, j)th *coefficient* in the output of the DCT says "how much" of the (i, j)th basic pattern there is in the block.
- A large coefficient (positive or negative) means pattern is present.
    - Large negative coefficient means the inverse of the pattern is present.
- Coefficients close to 0 mean that the pattern is not present.

Meaning of DCT coefficients

- Top left entry is “D/C coefficient”.
  - Value is $\frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(a_{i,j}-128)$.
  - $a_{i,j}$ is the value in the i-th row and j-th column of the input block of pixels A.
  - $a_{i,j}-128$ is the value in the i-th row and j-th column of the the matrix A', which is the matrix A with 128 subtracted.
  - The D/C coefficient is N times the average value in A'.
- ▷ As we go to bottom right, the “basic patterns” are for “high-frequency” patterns.
- ▷ Bottom right entries are “A/C coefficients”.

Baseline JPEG: Main steps

- $\surd$ Perform Colour-space Transformation.
- $\surd$ Downsample chrominance information (HVSL, SR).

Divide downsampled Y, C_b and C_r images into 8×8 blocks of pixels. For each block do:

- $\surd$ Perform Discrete Cosine Transform (DCT).
- Quantise. (HVSL)
- Coding of quantised DCT coefficients:
  - Zig-zag scan and RLE. (HVSL)
  - Differential encoding of D/C coefficients. (SR)
  - Entropy coding.

Quantisation

- All DCT values are *quantised* (quantisation is the *main* lossy step in JPEG).
- Takes a *value* x and a *quantisation factor* q.
  - ▷ quantisation: round x/q to nearest integer z.
  - ▷ invert quantisation: output $z \cdot q$ (lossy recovery).

Example using quantisation factor $q = 16$

Value (x)	223	24	7	(8-bit)
Quantise	$\frac{223}{16} \to 14$	$\frac{24}{16} \to 2$	$\frac{7}{16} \to 0$	(4-bit)[1]
Recover	224	32	0	(8-bit)

- Quantisation by factor q loses about $\log_2 q$ bits.
- Large quantisation factor: more compression, less accurate.

[1] Not quite 4-bit but ignore this for now.

Quantisation in JPEG

- DCT coefficients are quantised using two 8×8 matrices of quantisation factors:
  - one called Q_Y used for quantising coefficients from the Y blocks (the *luminance* quantisation matrix) and
  - one called Q_C used for quantising coefficients from the C_b and C_r blocks (the *chrominance* quantisation matrix).
- The (i, j)-th DCT coefficient is quantised by a value $Q_{Y/C}[i, j]$, depending on whether it is from a Y block or a C_b or C_r block.
  - ▷ each DCT coefficient is quantised by a different amount.
- The quantisation matrices encode HVSLs.

Quantisation and HVSLs

- HVSL 2 Eye cannot perceive rapid variation in small areas at normal resolution (high spatial frequencies).
  - DCT coefficients in the bottom right hand corner indicate variation in blocks.
  - Quant. factors in bottom right hand corner are *much bigger*.
  - E.g. $Q_Y[0,0] = 16$ and $Q_Y[7,7] = 99$.
  - ▷ 776 is quantised to $776/16 = 48.5 \rightarrow 49$ and is recovered as $49 \times 16 = 784$. The value -23 in the lower right is quantised to 0 and recovered as 0.
- HVSL 1 Eye is far more sensitive to intensity variations than colour variations.
  - Quantisation factors in Q_C get large very quickly.
  - E.g. $Q_Y[3,3] = 29$ but $Q_C[3,3] = 99$.
  - Coefficients corresponding to colour variations are quantized more heavily.

Luminance (Y) Quantisation Table

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

Luminance (Y)

$$\begin{pmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}$$

Chrominance (C_b/C_r)

Quantisation of Example 8 × 8 Block

776/16	0	164/10	0	16/40	0	19/51	0
0	0	0	0	0	0	0	0
164/14	0	-137/16	0	21/57	0	-11/69	0
0	0	0	0	0	0	0	0
16/18	0	21/37	0	-48/68	0	9/103	0
0	0	0	0	0	0	0	0
19/49	0	-11/78	0	9/121	0	-23/120	0
0	0	0	0	0	0	0	0

49	0	16	0	1	0	0	0
0	0	0	0	0	0	0	0
12	0	-9	0	1	0	0	0
0	0	0	0	0	0	0	0
1	0	1	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

Baseline JPEG: Main steps

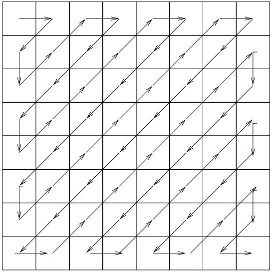
- $\surd$ Perform Colour-space Transformation.
- $\surd$ Downsample chrominance information (HVSL, SR).

Divide downsampled Y, C_b and C_r images into 8×8 blocks of pixels. For each block do:

- $\surd$ Perform Discrete Cosine Transform (DCT).
- $\surd$ Quantise. (HVSL)
- Coding of quantised DCT coefficients:
  - Zig-zag scan and RLE. (HVSL)
  - Differential encoding of D/C coefficients. (SR)
  - Entropy coding.

Zig-Zag scan and RLE

Read 8×8 matrix of quantised DCT coefficients in strange order:



49	0	16	0	1	0	0	0
0	0	0	0	0	0	0	0
12	0	-9	0	1	0	0	0
0	0	0	0	0	0	0	0
1	0	1	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

Why? Bottom right entries often zero, get runs of zeros.

```
49, 0, 0, 12, 0, 16, 0, 0, 0, 0, 1, 0, -9, 0, 1, 0, 0,
 0, 0, 0,  0, 0,  0, 1, 0, 1, 0, 0, 0,  0, 0, 0, 0, 0,
 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.
```

▷ Apply Run-Length Encoding.

Differential encoding of D/C coefficients

- D/C coefficient of a block is usually large.
- ▷ Related to average pixel value in block.
- Pixel averages in adjacent blocks usually similar.
- ▷ Adjacent blocks have similar D/C coefficients.
- ▷ Use predictive encoding: from the D/C coefficients of the adjacent blocks, predict the D/C coefficient of the current block and obtain a *residual value* (as in JPEG lossless).

Final step: Code the entire output so far using entropy (Huffman/arithmetic) coding.

Baseline JPEG: Main steps

- $\surd$ Perform Colour-space Transformation.
- $\surd$ Downsample chrominance information (HVSL, SR).

Divide downsampled Y, C_b and C_r images into 8×8 blocks of pixels. For each block do:

- $\surd$ Perform Discrete Cosine Transform (DCT).
- $\surd$ Quantise. (HVSL)
- $\surd$ Coding of quantised DCT coefficients:
  - $\surd$ Zig-zag scan and RLE. (HVSL)
  - $\surd$ Differential encoding of D/C coefficients. (SR)
  - $\surd$ Entropy coding.

Customisation

- The baseline algorithm can be customised:
  - Give your own quantisation tables.
  - Give your own Huffman codes.
  - After quantisation, further quantise (this is the *quality* parameter on most JPEG implementations.
- We have covered the lossless and baseline (lossy) JPEG algorithms. The JPEG standard also specifies *hierarchical* and *progressive* modes, which aim to give user an "early glimpse" of entire image. E.g. store the coded Y blocks at the start of the file, so the decoder can output an early grayscale version of the image.
- This concludes Chapter 8 (Images).