7. Introduction to image compression

- Image data is fundamentally different from text.
- Scale of measurement is different:
  - Pixels: *interval / ratio scale*:
    Neighboring pixels tend to have numerically close colour values
  - Characters: *nominal scale*:
    ASCII values of neighbouring characters do not correlate numerically
- Dedicated compression methods are required.

Image data types

- *Bi-level* images
  - 1 bit per pixel, 2 colors (black and white)
- *Grey-scale* images
  - 8 bits per pixel, 256 colors (shades of grey)
- *Color* palette images
  - 8 bits per pixel, 256 colors (representative subset)
- *True color* images
  - 3x8 bits per pixel, ≈ 16.8 million colors

Main types of image compression

- **Lossless:**
  - The original image can be returned precisely
  - Seldom needed for photographs
  - Exception: x-rays

- **Lossy:**
  - Only an approximation of the original image can be returned.
  - Takes advantage of the limitations of the human visual system.
  - Enables much higher compression ratios than the lossless approach (close to 10 x).

7.1. Lossless compression of bi-level images

Applications:

- Telefax
- Engineering drawings
- Document imaging (scanning and digital archiving)

Basis of compression:

- Different proportions of black and white pixels.
- *Clustering* of same-coloured pixels (black/white areas)
- Pixel rows are considered *bit vectors*; compressions methods are chosen accordingly.

Run-length coding

- ‘Run’ = sequence of instances of the same bit
- Runs of 0’s and 1’s alternate
- 0’s (white) usually dominate (-> ink on white paper)
- Two simple alternatives
  - Number of zeroes ended by 1
    0001000011100000010011 00… -> 3, 4, 0, 0, 6, 2, 0, …
  - Alternating lengths of 0- and 1-runs; the first bit must be stored
    0001000011100000010011 00… -> <0>, 3, 1, 4, 3, 6, 1, 2, 2, …
- The run-lengths are encoded by whatever method:
  - Huffman
  - Universal coding of numbers (gamma, delta, Fibonacci, …)

Interpolative coding: Effective non-statistical coding of number sequences

- Developers: Moffat & Stuiver 1996 / 2000
- Suitable for any sequences of integers;
  especially good if *small values are clustered*.
- Difference from universal coding of numbers:
  the code values depend on neighbors
- Clearly better than encoding the numbers independently

Ideas:

- Transform the sequence into a *cumulative* sequence
- Encode the *last number first*, then the middle one,
  then the middle of first/second half, recursively
  (cf. binary search)
- At each step (except first), the lower and upper bounds are known,
  determining the number of required bits, for *semi-fixed-length* code
- The bounds get closer when recursion proceeds.

Interpolative coding: Example

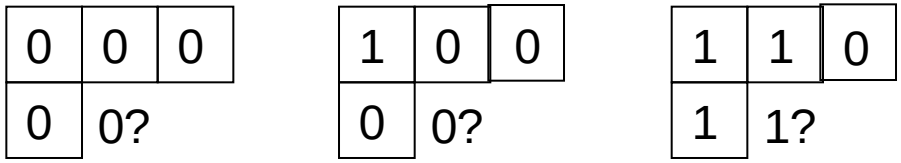
- Assume nonnegative integers
- Original sequence: 4, 2, 0, 3, 5, 1, 2, 3
- Cumulative sequence: 4, 6, 6, 9, 14, 15, 17, 20
- Steps:
  1. Encode 20 with universal code, e.g. gamma code: 9 bits
  2. Encode the middle element 9; it is between 0..20, so use either $\lfloor \log_2 21 \rfloor = 4$ or $\lceil \log_2 21 \rceil = 5$ bits for it.
  3. Encode the middle element 6 of the front part; it is between 0 and 9, so use either $\lfloor \log_2 10 \rfloor = 3$ or $\lceil \log_2 10 \rceil = 4$ bits for it.
  4. Encode the middle element 15 of the rear part; it is between 9 and 20, so use either $\lfloor \log_2 12 \rfloor = 3$ or $\lceil \log_2 12 \rceil = 4$ bits for it.
  5. Etc.

  Result: 29 bits

  Comparison: All numbers gamma-coded: 39 bits

Predictive run-length coding

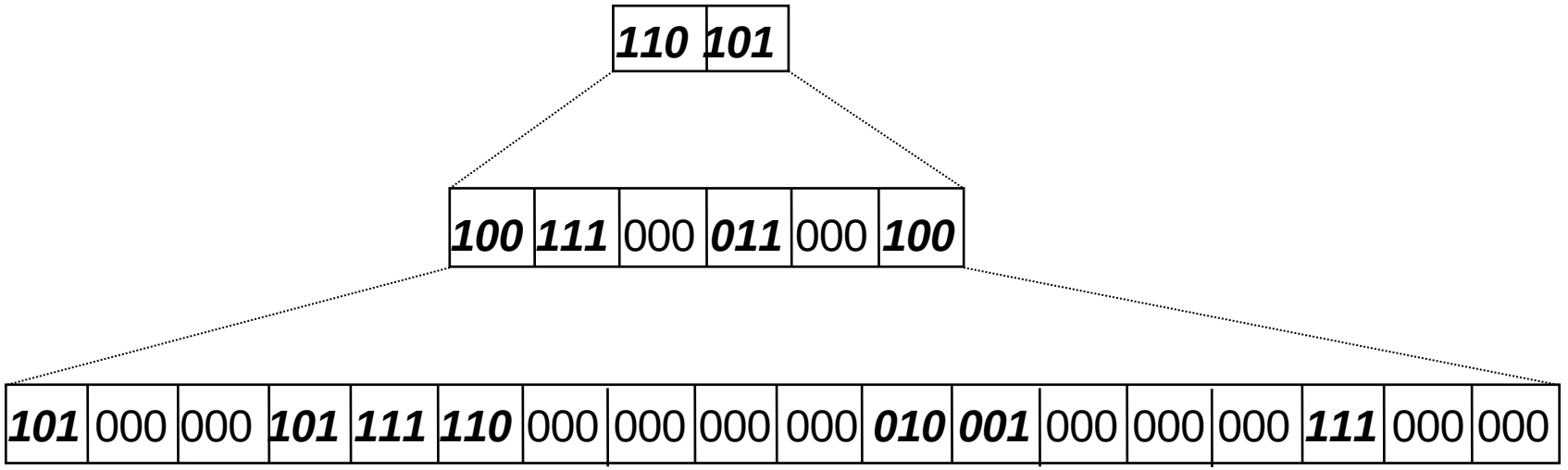
- Predict each pixel on the basis of its processed neighbours; result = success / failure.
- Transform each pixel: 0 = success,1 = failure



- Similar pixels are often *clustered*;
  the proportion of 0's grows and compression improves.
- Decoder repeats the prediction, and is able to do the reverse transform

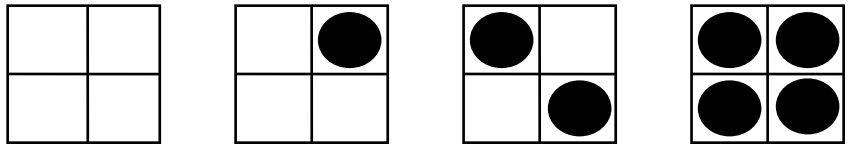
Another simple bit-vector coding: *Block coding*

- Partition the bit vector into fixed-length blocks..
- Encode an all-zero block by 0, and others by <1, block>
- The flag bits can be block-coded recursively → *Hierarchical block coding*



2-dimensional block coding

- Image is partitioned into rectangular boxes.
- Encode: white box = 0, non-white box = <1, box pixels>

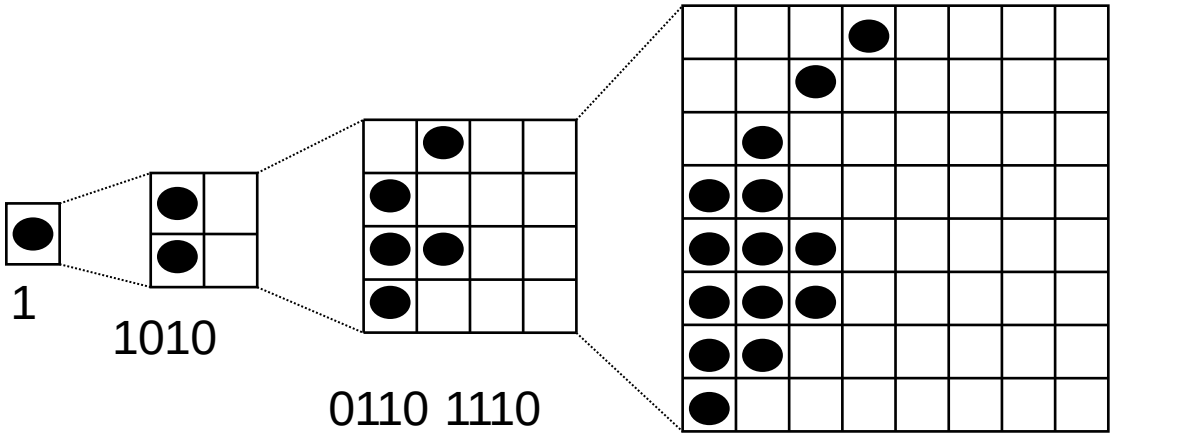


0 1 0100 1 1001 1 1111

- Recursion on flag bits creates a hierarchy
- Prediction transformation applicable also here:
  - The whole picture must be transformed before block encoding.
  - Reverse transform when block decoding is completed.

Hierarchical block coding

Example:



Telefax compression

- *Group 1*: Analog scheme, speed $\leq$ 6 min / A4 sheet.
- *Group 2*: Analog scheme, speed $\leq$ 3 min / A4 sheet
- *Group 3*: Digital scheme, 1- or 2-dim. compression, speed $\leq$ 1 min / A4 sheet.
- *Group 4*: Digital scheme, 2-dim. compression, speed $\leq$ 1 min / A4 sheet.

Telefax compression (cont.)

1-dimensional telefax scheme:

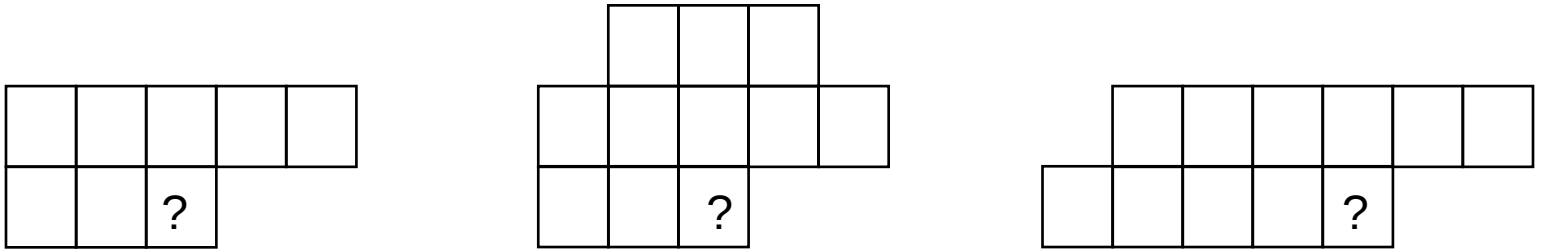
- Run-length coding
- Runs represented as $length = 64 \times m + t$
- Huffman-code m and t (separately for black and white)

2-dimensional telefax scheme:

- Pixel row encoded on the basis of the previous row.
- Max K–1 rows by 2-dim., then 1-dim. compression
- K = 2 (normal), = 4 (high resolution), = ∞ (group 4)

JBIG

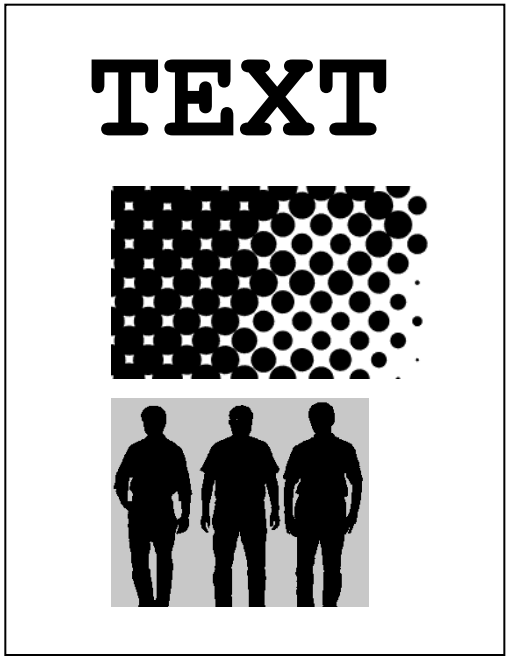
- Joint Bi-Level Image Compression Group, 1993
- Standardized by ISO, CCITT and IEC.
- Effective method for bi-level compression.
- *Context model*: 7 or 10 neighbouring (already processed) pixel values define the context for the current pixel.



- 128 or 1024 different *QM-coders* for final coding.
- *Sequential* and *progressive* modes;
  inter-level prediction in progressive mode

JBIG2

- Newer (2000), more effective version of JBIG
- Application areas: PDF documents, document images, telefax, wireless transmission, printer spooling
- Divides the image into three types of *regions*:
  - *Symbol regions* → text as image; *dictionary* coding
  - *Halftone regions* → halftone (raster) image as a bi-level image; *dictionary* coding
  - *Generic regions* → Others; *predictive* coding



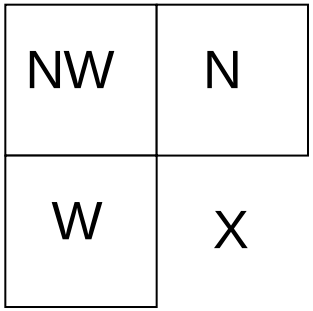
7.2. Lossless compression of grey-scale images

- *Lossy* is more common
- In critical applications (e.g. medical X-rays) *lossless*.
- Bi-level techniques can be applied to *bit planes*:
  8 bits per pixel, 256 different grey levels,
  - 1. bit plane = most significant bits from all pixels
  - 2. bit plane = second most significant bits from all pixels
  - Etc.
- Bit-plane compression can be improved by first *Gray-coding* the pixel values
  - Adjacent values differ by only one bit.
  - Example for 3 bits: 000, 001, 011, 010, 110, 111, 101, 100
  - The previous (encoded) bit plane can predict the next one.

Lossless JPEG

- JPEG offers both lossless and lossy options.
- Lossless JPEG applies a *linear prediction model*.
- Predicted grey level of a pixel is a linear function of 1, 2 or 3 neighbouring pixels (north, west, northwest)
- 7 alternative prediction formulas (+ no-prediction option)

1. $X = N$
2. $X = W$
3. $X = NW$
4. $X = N + W - NW$
5. $X = W + (N - NW) / 2$
6. $X = N + (W - NW) / 2$
7. $X = (N + W) / 2$



Lossless JPEG (cont.)

- *Prediction errors* are encoded with
  - Huffman coding
  - Arithmetic coding
  - Some other (fast) non-optimal coding of numbers (e.g. start-step-stop code).
- The decoder makes the *same* prediction and adds the decoded error.
- The first pixel transmitted as such.
- The first pixel row can be predicted only from ‘west’.
- Compression ratio typically $\approx$ 50%.

Other methods for lossless image compression

CALIC (Context Adaptive Lossless Image Compression):

- Uses a larger (7 pixels) context for prediction
- Analyses the context to choose the best prediction function
- Makes an initial prediction + refinement.

		NN	NNE
	NW	N	NE
WW	W	X	

JPEG-LS:

- Initial prediction as *median* of N, NW and W.
- Refined prediction on the basis of statistics about the errors that occurred *earlier* in the context of the *same* <NE, N, NW, W> context.
- Better than old lossless JPEG, slightly worse than CALIC

7.3. Example of lossy image compression: JPEG

- Joint Photographic Experts Group 1986-92
- ISO standard 1994
- Both lossless (see above) and lossy modes
- Enables usually compression ratios of more than 10:1
- Lossy JPEG is the most used compression for photographic color/grey-scale images.
- JPEG2000 gives better compression, but has not (yet) surpassed the old JPEG in popularity
- Modelling phase is totally different from earlier techniques.

Basis of JPEG: Discrete Cosine Transform (DCT)

- Transforms the image into *frequency domain*.
- The image is represented as a linear combination of basis functions, which are variants of the cosine functions, with various wavelengths (frequencies).
- Resembles Fourier transform, but the result is in the domain of real numbers (not complex).
- The result of the transform consists of *coefficients* of the transform.
- *Discrete* ($\neq$ continuous) means that the transform can be done as sums of a finite number of terms.
- *Fast cosine transform* takes time $O(n \log_2 n)$ for n pixels.

The magic formulas

Cosine transform:

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\left\{ f(x,y)\cdot\cos\frac{(2x+1)u\pi}{2N}\cdot\cos\frac{(2y+1)v\pi}{2N}\right\}$$

Reverse transform:

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\left\{\alpha(u)\alpha(v)C(u,v)\cdot\cos\frac{(2x+1)u\pi}{2N}\cdot\cos\frac{(2y+1)v\pi}{2N}\right\}$$

where

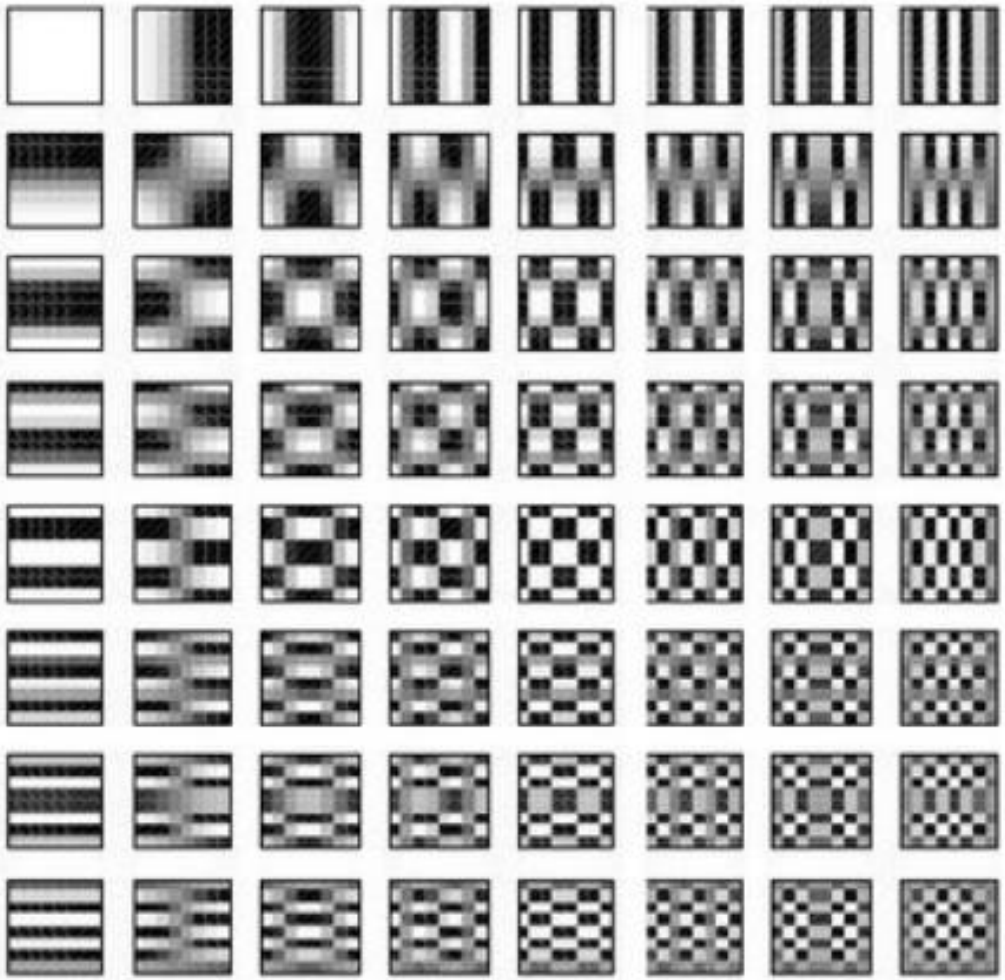
$\alpha(u) = \sqrt{1/N}$ for u=0, and $\sqrt{2/N}$ for u>0

Blocking

- JPEG processes the image in blocks of 8 x 8 pixels
- Reasons:
  - Faster transform
  - Regularities do not usually span far
- Blocks are coded almost independently
- Drawbacks:
  - There are similarities across block boundaries which are not taken advantage of.
  - For high compression rates, so called *blocking artifacts* may appear.
- Example of block transform:

  See: *http://en.wikipedia.org/wiki/JPEG*

DCT basis functions in graphical form

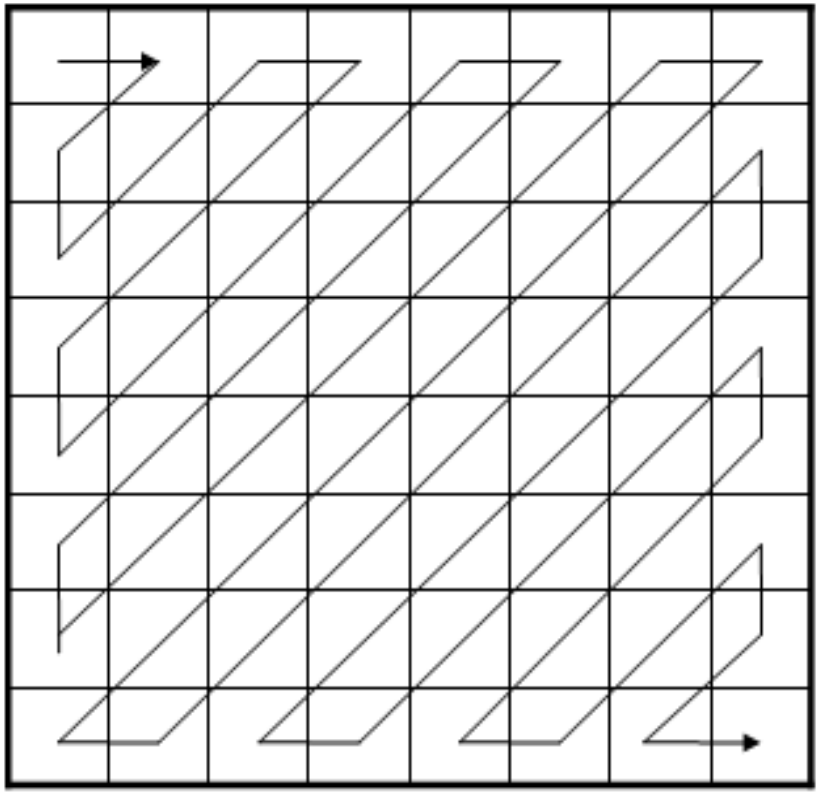


The next step: Quantification

- The lossy step
- Division of coefficients by suitable integers & rounding
- Reduces precision
- Smaller loss for low-frequency components:
  - More important visually
  - Smaller divisors for top-left area, higher for the bottom-right area
- Quantization tables are standardized for different qualities
- Note: division of 8 x 8 matrices element-by-element
- Example of quantization:
  See: *http://en.wikipedia.org/wiki/JPEG*

The next step: Linearization of the matrix

- So-called zig-zag order:



Encoding of the vector

- Observation: Most of the values zero
- The non-zero values are clustered to the left
- The first value: '*DC-coefficien*t'
  - Compute difference from the first value of the neighbour block.
  - Huffman-code the differences
- Other 63 ('*AC-coefficients*')
  - Tailored *run-length coding*

Encoding of 63 AC-coefficients

(1) Run-lengths and element sizes as *pairs*:

(a) 160 normal pairs:

- Number of zeroes before the next non-zero element (0..15)
- Number of *significant bits* in the non-zero element (1..10)

(b) Two special pairs:

- <0, 0> that represents EOB = end-of-block
- <15, 0> that represents a sequence of plain zeroes

Total size of the alphabet: 162

Apply Huffman code using default code tables.

(2) Non-zero element values:

Encode as base-2 numbers using the given bit count.

JPEG decoding

Reverse process:

1. Decode Huffman
2. Recover the quantized matrix:
   - Set the DC-coefficient to top-left
   - Recover the zig-zag order from the linear order (63 coefficients)
3. Multiply by quantization matrix (elementwise);
   this produces *roughly* the original DCT-coefficients.
4. Perform the inverse DCT transform.
5. Collect the blocks into a recovered image.

Example of blocking artifacts

Compression ratio 7:1

Compression ratio 30:1

Compression of color images

- Three components, e.g. RGB (Red-Green-Blue)
- High correlation between component images: Separate compression of each does not pay
- Solution: transformation to a different color model, where the components are less correlated
- YUV (YIQ):
  - Luminance channel: The most important to the human visual system
  - Two chrominance channels: Less important, and therefore usually *subsampled* 2:1 in both dimensions (→ pixel count reduces to ¼).
- Separate encoding of component images with different coding parameters

Color model conversions

- RGB → YUV:
  - Y = 0.299R + 0.587G + 0.114B
  - U = 0.492(B – Y)
  - V = 0.877(R – Y)

- YUV → RGB
  - R = Y + 1.140V
  - G = Y – 0.395U – 0.581V
  - B = Y + 2.033U

Converting a color image (Lena) to YUV

Summary of image compression

- Photographic images need different modeling methods, compared to text, graphics, or bi-level images.
- Modeling should *decorrelate* the image by well-chosen transformations
- The loss of information should be hidden from the user.
- When using Huffman coding for numeric data, the alphabet should be carefully selected to avoid dependencies
- When using Huffman, predefined (static) tables are preferable in image compression; there can be several alternative tables for different image types and compression qualities.