Multimedia compression

Why do we need compression?

- Crucial for a number of multimedia applications
  - Efficient transmission over communication channel
  - Efficient storage on storage devices

Motivation

- Image size
  - Color 12 megapixel image 4000×3000px = ~35Mb
  - Channel speed 1Mbit/s = transfer time: ~4,6 min
- Audio size
  - 3 minute song, 44100 16-bit samples per second ~= 15Mb
  - Channel speed 1Mbit/s = transfer time: ~2 min
- Video size
  - 1920 x 1080 color video, 25 FPS, 120 minutes = ~ 1Tb
  - Channel speed 1Mbit/s = transfer time: ~80 days

Compression of sensory data

- Redundancy of sensory data
  - Correlation across space and time
  - Human perceptual sensitivity
- Lossless compression
  - Minimize size without loosing information
- Lossy compression
  - Minimize size with controlled loss of information
  - Discard data that is not noticed by a human

Spatial correlation

Neighborhood pixels are frequently similar (image, video)

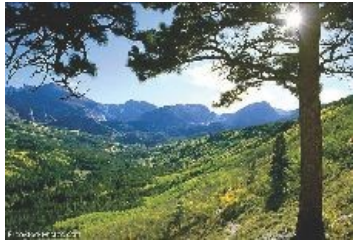
$I(x, y)$

$I(x, y) - I(x, y + 1)$

Spectrum correlation

Correlation across spectrum

(Similar intensity of colors in colors in all RGB channels)



Red

Green

Blue

Temporal correlation

Two frames in a video stream are very similar

$I_t(x, y)$

Frame t

$I_{t+1}(x, y)$

Frame t+1

$\Delta_t = I_t(x, y) - I_{t+1}(x, y)$

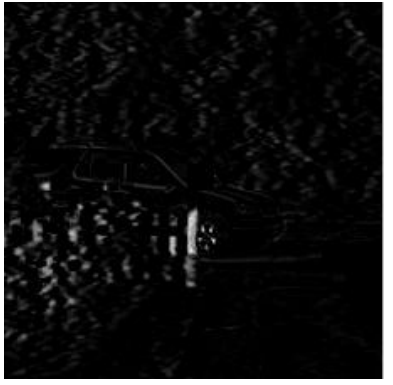
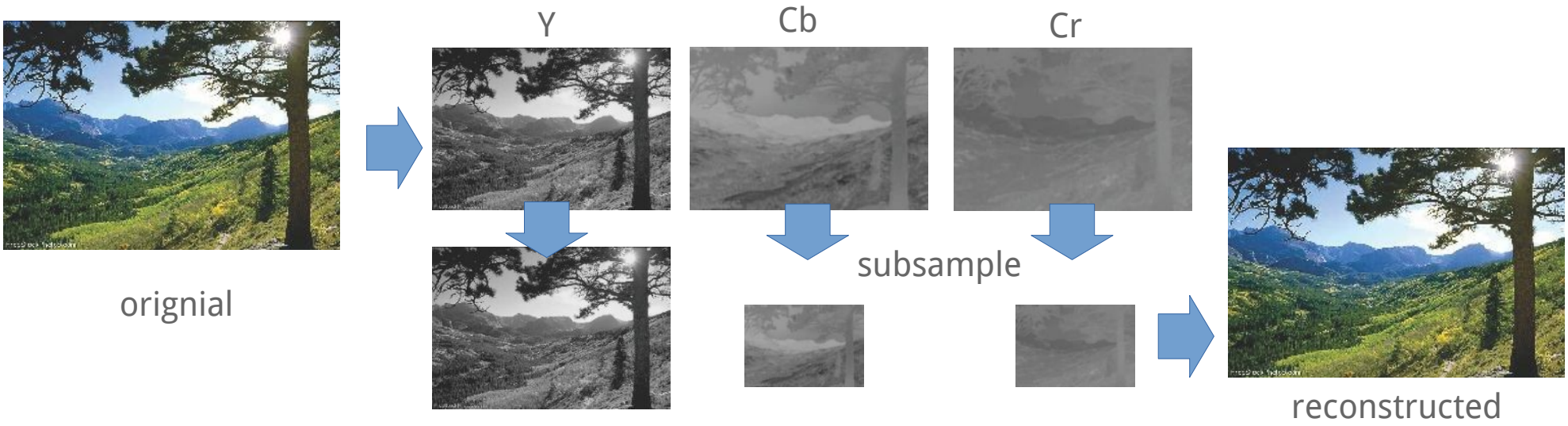


Image difference

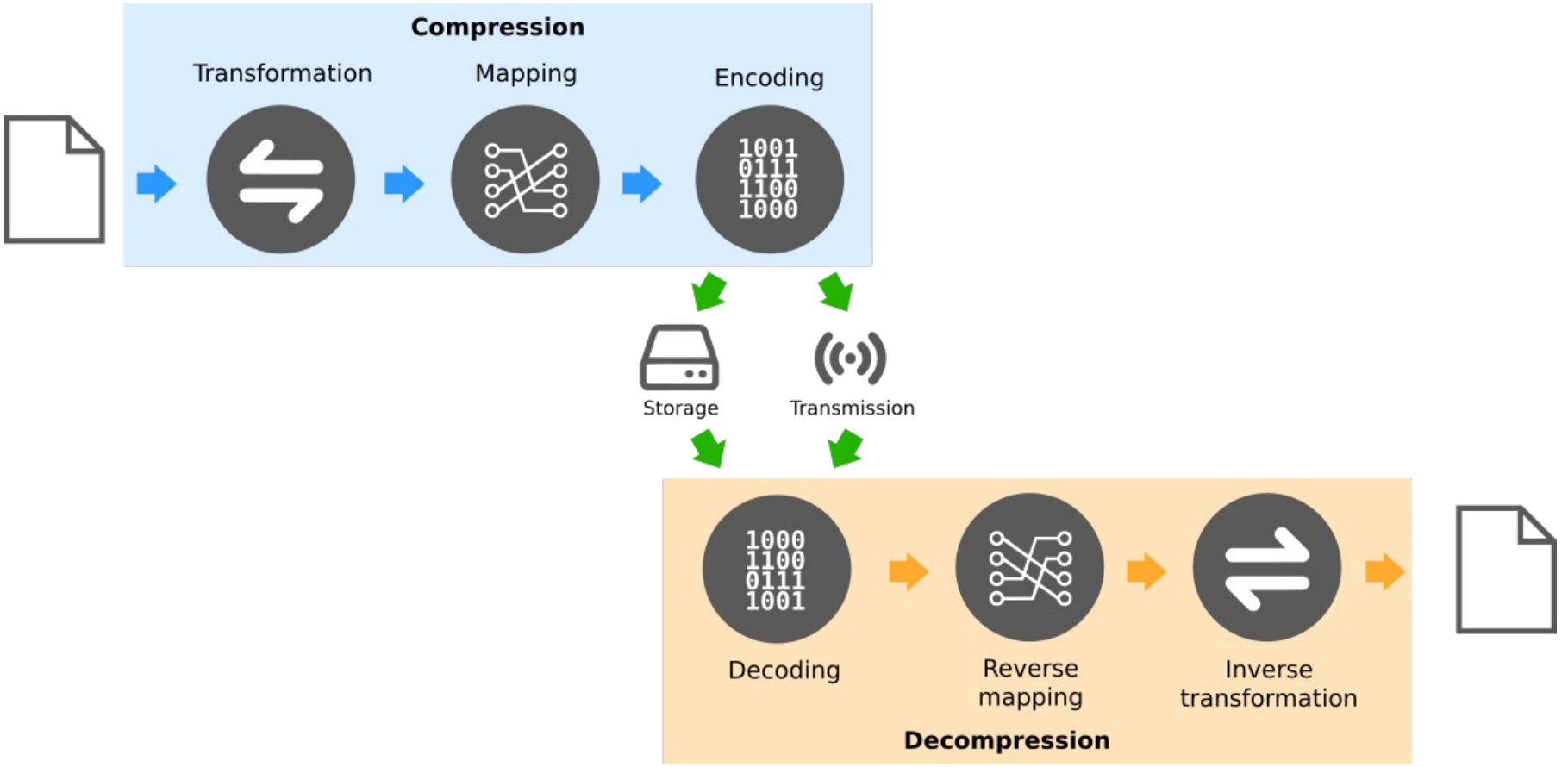
Perceptual sensitivity

Human perceptual system is differently sensitive to different types of information

(e.g. human vision is more sensitive to changes in intensity than in chroma)



Compression process



Lossless vs. lossy

- Lossless compression
  - Decoded information is equal to the original image (numerically/quantitatively and perceptually/qualitatively)
- Lossy compression
  - Decoded information should be perceptually equal to the original image (to a required level)
  - Achieves higher compression levels, removes redundant information
- Choice is application dependent
  - Biomedical signals have to be stored without loss
  - Entertainment industry can save space and bandwidth with lossy compression

Transforming signal/data

- Mapping signal/data to a form that is easier to encode
  - Reducing correlation in data
  - Reducing redundancy
  - Change statistical properties
- Typical transformations
  - Differential (predictive) coding
  - Bijective transformation to frequency space (e.g. DCT)
  - Convert colors from RGB to YCbCr (separate intensity and chroma)

From data to symbols

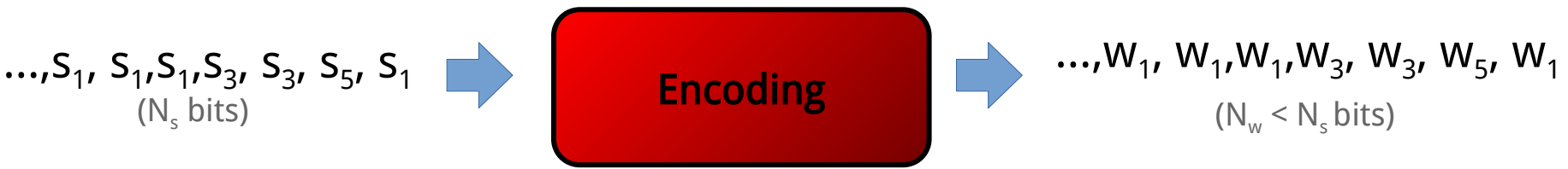
- Data to symbols for efficient encoding
- Split signal to smaller chunks (partitioning)
  - Split image into blocks – each block is a symbol
  - Encode each block separately
- Code sequences, repetitions (run-length coding)
  - Absolute correlation in subsequences of data
  - Blocks are sequences of repeated data

Run-length encoding - RLE

- Sequence of same numbers write as symbol (count, value)
  - Count – how many times the symbol is repeated
  - Value – the value of single symbol
- Sequence of data is transformed into sequence of symbols

Lossless encoding

- Transformation and mapping to symbols are preprocessing
- Most lossless data compression occurs in symbol encoding
  - Binary stream where each symbols is assigned a code word
  - Code dictionaries can be formed on-line or off-line
  - Entropy encoders minimize expected number of bytes to represent sequences of symbols



Lossless symbol coding

- Minimize size (number of bits) without losing information
- **Statistical methods** (Huffman, arithmetic)
  - More frequent symbols are assigned shorter code words (variable length coding - VLC)
  - Statistics of symbol occurrence in data stream can be fixed or computed on-the-fly
- **Dictionary methods** (LZ77, LZW, LZMA)
  - No statistics about symbol frequency
  - Dynamically generate code tables (dictionary) to encode sequences of various length
  - Store dictionary and references

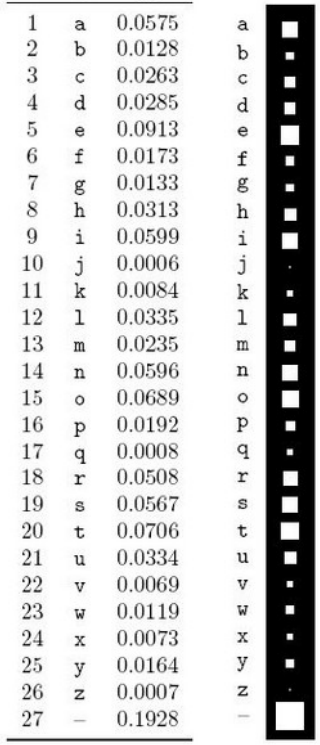
Information entropy

- How many bits required for each symbol
  - 4 bit – 16 symbols
  - 5 bit – 32 symbols
- Information entropy measures how many bits are needed to encode a single character based on probability of its appearance in a sequence

$$h(p_i) = -log_2(p_i)$$

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

i	a_i	p_i		$h(p_i)$
1	a	0.0575	a	4.1
2	b	0.0128	b	6.3
3	c	0.0263	c	5.2
4	d	0.0285	d	5.1
5	e	0.0913	e	3.5
6	f	0.0173	f	5.9
7	g	0.0133	g	6.2
8	h	0.0313	h	5.0
9	i	0.0599	i	4.1
10	j	0.0006	j	10.7
11	k	0.0084	k	6.9
12	l	0.0335	l	4.9
13	m	0.0235	m	5.4
14	n	0.0596	n	4.1
15	o	0.0689	o	3.9
16	p	0.0192	p	5.7
17	q	0.0008	q	10.3
18	r	0.0508	r	4.3
19	s	0.0567	s	4.1
20	t	0.0706	t	3.8
21	u	0.0334	u	4.9
22	v	0.0069	v	7.2
23	w	0.0119	w	6.4
24	x	0.0073	x	7.1
25	y	0.0164	y	5.9
26	z	0.0007	z	10.4
27	–	0.1928	–	2.4



$$H(X) = -log_2(1/27) = 4.75 bit/ch$$

$$H(X) = -\sum_{i=1}^{N} p_i log_2(p_i) = 4.1 bit/ch$$

Entropy coding

- Known set of symbols S of size N, each of them has a probability of occurring in stream
- Average length of word $\overline{w} = \sum_{i=1}^{N} p_i |w_i|$ $S = \{(s_1, p_1), (s_2, p_2), \cdots, (s_N, p_N)\}$
- Determine code words that minimize the average length of word and satisfy requirements:
  - Regular – different symbols are assigned different words
  - Unique – message can be understood in only one way
  - Instantaneous – each word can be decoded as soon as it is read

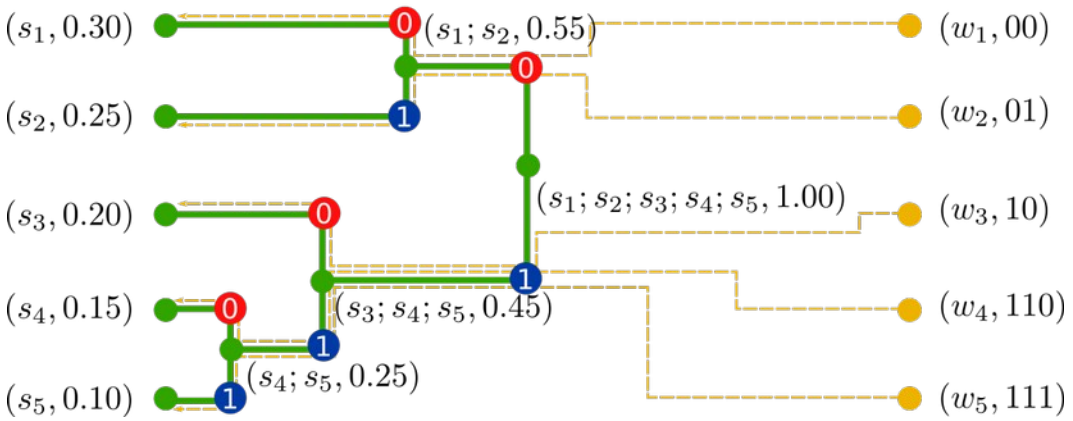
Huffman encoding/decoding

- Type of optimal prefix coding
- Encoding
  - Establish symbol statistics
  - Generate word codes
  - Map symbols to words
- Decoding
  - Binary search tree
  - Lookup table

Determining words by Huffman coding

- Sort symbols by decreasing probability of occurrence
- Merge two symbols with lowest probability and combine their probabilities
- Repeat until all symbols have words assigned

$S = \{(s_1, 0.30), (s_2, 0.25), (s_3, 0.20), (s_4, 0.15), (s_5, 0.10)\}$



Naive encoding (3bit codes) – 15 bit

$$s_1, s_3, s_1, s_2, s_2 \rightarrow 010001010011011$$

Huffman encoding – 10 bit

$$w_1, w_3, w_1, w_2, w_2 \rightarrow 0010000101$$

Huffman algorithm properties

- Each symbols separately requires a non-zero integer number of bits
  - Lower bound 1bit/symbol
  - Optimal when encoding each symbol separately
- Encoding based on statistical properties of the data-stream
  - Deviation from the model makes compression sub-optimal
  - On-line updates (adaptive) are computationally expensive
- Improved algorithms
  - Limit longest code words
  - Avoid one-symbol one-word mapping (arithmetic coding)

Dictionary coding

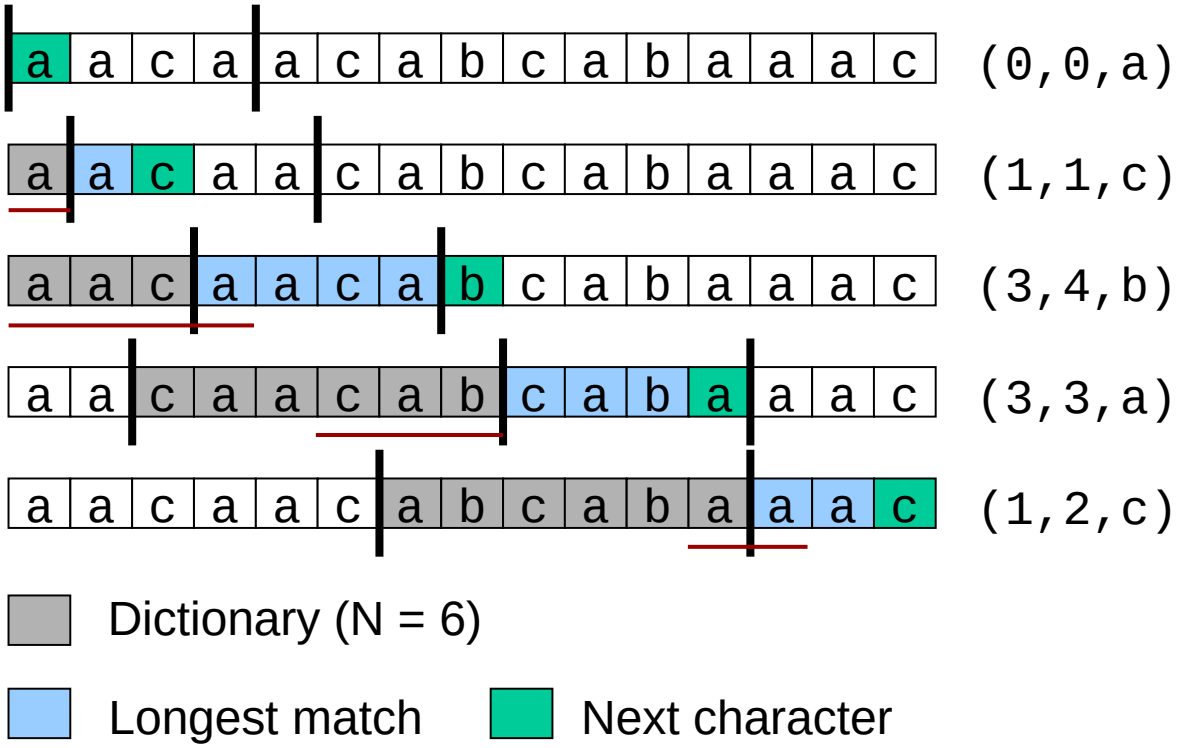
- Statistical coding requires to know symbol distribution in advance
- Dictionary coding schemes
  - Dynamic dictionary generation for variable length sequences of symbols
  - Code words with fixed length
  - Decoder reads message and reconstructs dictionary on-the-fly
  - Not suitable for short sequences (resulting bitstream can even increase in size)

LZ77 coding

- Lempel and Ziv (1977)
- Sliding-window compression
  - Maintain a window buffer of past N symbols
  - Can only decompress from the beginning
- Replace repeated occurrences of data with references to a single copy
  - Match encoded as a triplet command (offset, length, next)
  - Length can also include symbols that will be written by the command

LZ77 algorithm

```
while (stream not empty) do
begin
  find longest prefix p of
remaining stream starting in
window
  i = position of p in window
  j = length of p
  X = first char after p in
view
  output triplet (i,j,X)
  move j+1 chars
end
```



Improving LZ77

- LZ78 encoding
  - Explicit dictionary
  - Supports partial decompression
- LZW encoding (Lempel-Ziv-Welch)
  - Patented by Unisys (expired)
  - Dictionary pre-initialized with all possible symbols
  - When a match is not found, the current symbol is assumed to be the first symbol of an existing string in the dictionary

DEFLATE encoding

- Combination of LZ77 and Huffman encoding
  - LZ77 – duplicate string elimination
  - Huffman – replace symbols with prefix codes
- Configurable encoding
  - Set time that can be spend for searching sub-strings

Compression in multimedia

- Image compression
  - PNG compression
  - JPEG compression
- Video compression
  - MPEG video codecs
- Sound compression
  - FLAC compression
  - MPEG-1 Layer 3 compression

Encoding comparison

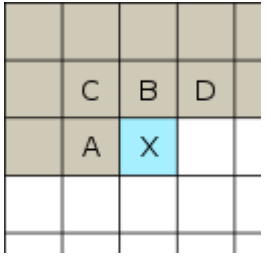
- Compression efficiency
  - What is the ratio between encoded and raw data size
- Delay
  - Time for compression/decompression
  - Amount of data processed at once (buffer)
- Implementation Complexity
- Robustness
  - Corruption of encoded data stream
- Scalability
  - Support for multiple profiles

PNG format

- Portable Network Graphics – 1996
- Lossless data compression
- Substitute for GIF format (patents, limitations)
  - Alpha channel (transparency)
  - RGB and indexed modes
- Two stage compression
  - Predictive filtering – encode difference to predicted value
  - DEFLATE compression

Predictive filtering

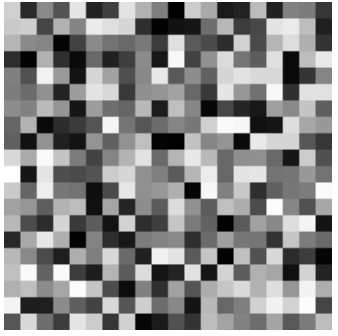
- Predicting pixel values based on value of previous pixel, encoding the difference
- Different filter types, chosen for each scan-line using heuristic algorithm
  - Unaltered value
  - Use value of A
  - Use value of B
  - Use mean value of A and B (round down)
  - Use A + B - C



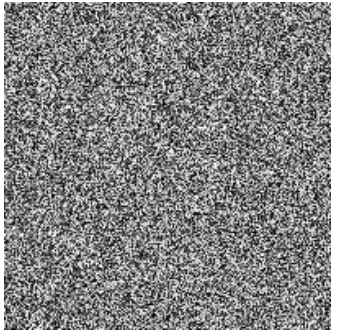
PNG examples

200 x 200 px RGB image – raw size: 120kB

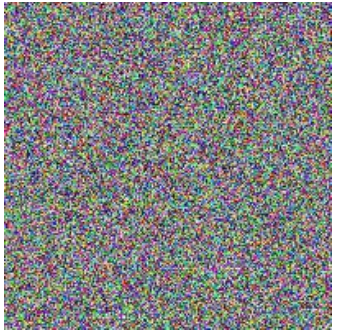
2 grays – 616 B



10 px chunks – 2,3 kB



Random gray – 105 kB



Random RGB – 120, 5 kB

Vector graphics – 22,1 kB

Photograph – 79,9 kB

Back to digital signal processing

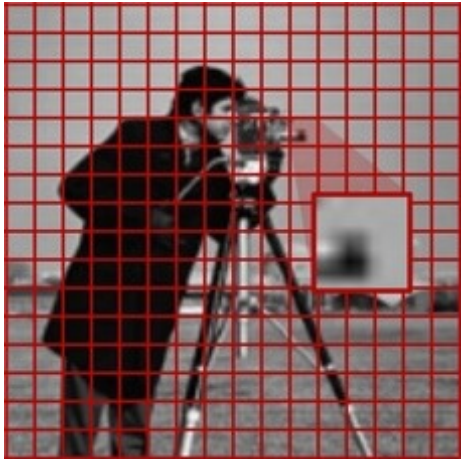
- Downsample
  - Fewer samples → fewer pixels
  - But: aliasing
- Quantize
  - Fewer bits/sample → fewer intensity levels
  - But: banding, blocking
- We have to do it cleverly to reduce side effects

Lossy compression in images

- Exploit spatial correlation
  - Local pixels are usually very similar
  - Divide image in small regions
  - Encode region in a way that some information is discarded

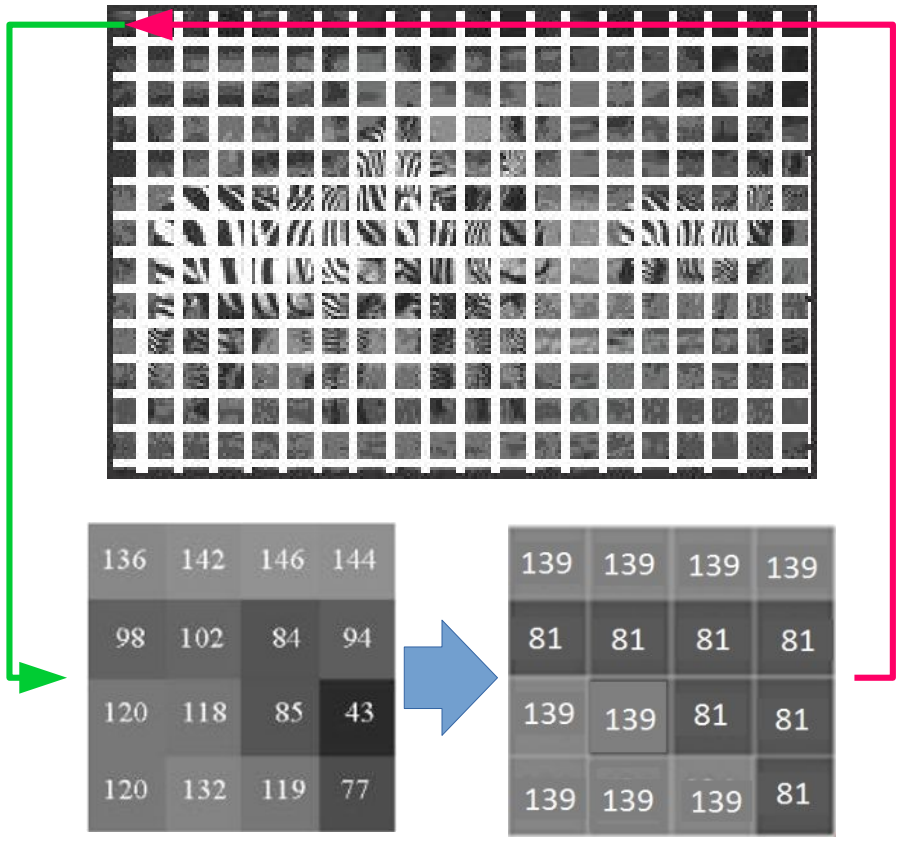
$I(x, y)$

$I(x, y) - I(x, y + 1)$



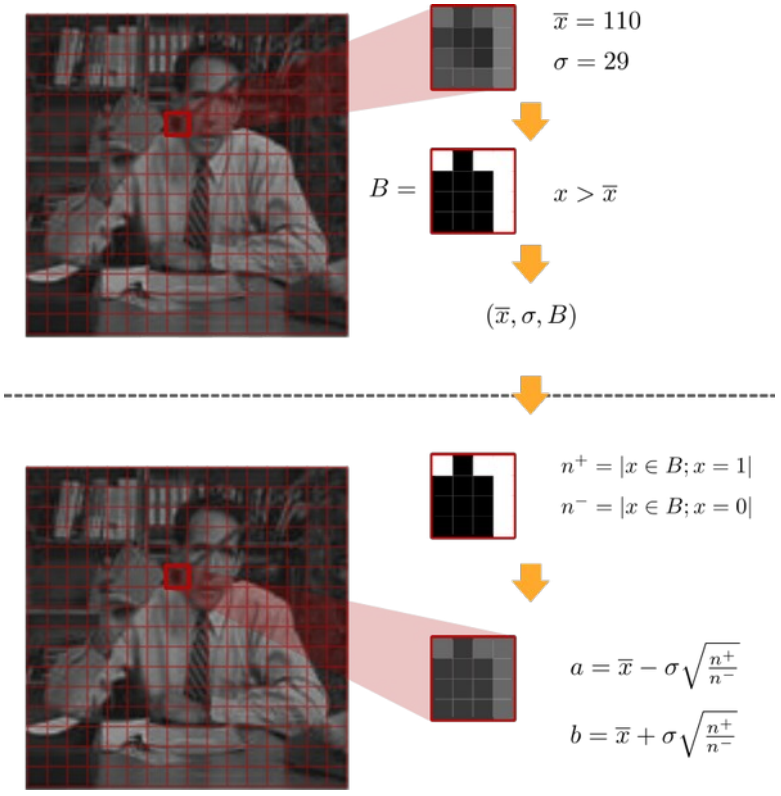
Block truncation coding (BTC)

- Divide image into square blocks
- Encode each block with only two intensities
  - Mean value and standard deviation of the block should not change
- Simple, but used in real life
  - Mars Pathfinder (1997)



BTC compression / decompression

- Compression
  - Divide image into n x n blocks
  - Threshold each block with mean value of values in the block
  - Store thresholded value (1bit per pixel), mean value (8bit) and standard deviation (8bit) – 4 bytes in total
- Decompression
  - Use mean and standard deviation to compute two values
  - Assign higher values to pixels above the threshold, lower to the ones below



Properties of BTC encoding

- Compression ratio (8bit values, 4x4 region) – 4:1
- Very simple method
  - Noticeable errors due to loss of information
  - Frequent edges between blocks
  - Artifacts around edges and in parts of image with low contrast where the values are slowly changing from one to another

The JPEG standard

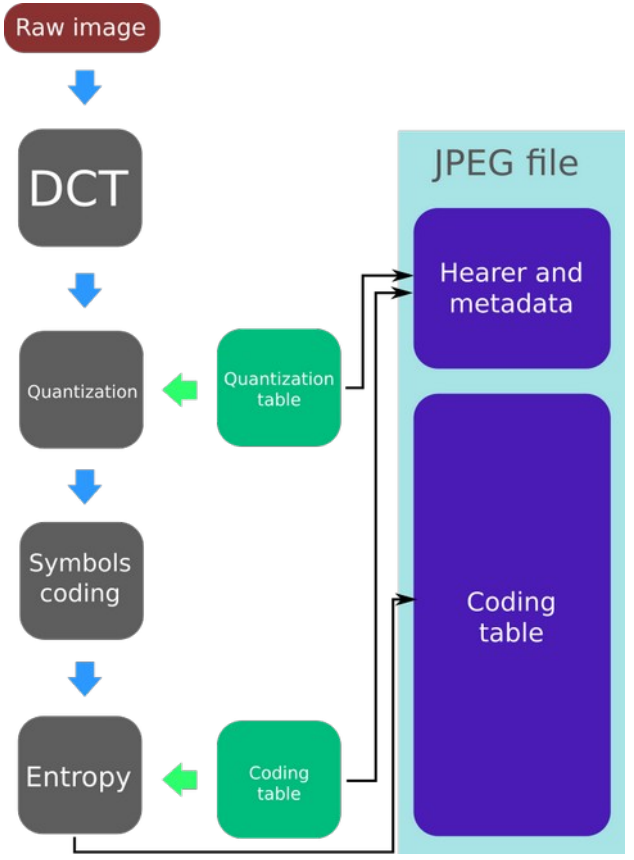
- Joint Photographic Experts Group (1992)
- Most popular standard for image compression
- Defines lossy and lossless compression
  - Lossy – DCT, quantization, RLE, Huffman
  - Lossless – predictive coding, Huffman
- Take into account human perception system
  - Discard information that is least relevant to human perception

The JPEG lossy compression

- Sequential and progressive coding
- Low computational requirements
- Suitable for all types of images, works better with photographs
  - Allows compromise between transfer speed and quality
  - Color depth: 8-12 bits
- File formats
  - Raw image data + metadata
  - JPEG File Interchange Format (JFIF) – multi-platform
  - Exchangeable Image file Format (EXIF) – digital cameras

JPEG lossy encoder

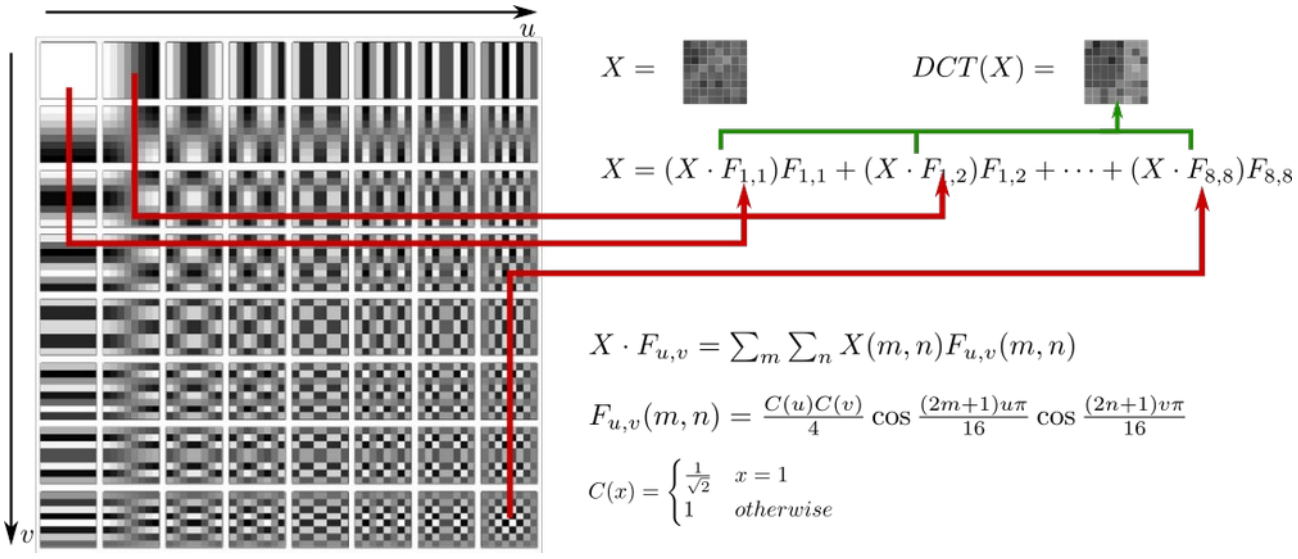
- Signal transformation (DCT)
  - Image divided in 8 x 8 blocks
  - Each block described using DCT coefficients
  - If the image size is not divisible by 8 add lines/columns
- Quantization of DCT coefficients
- Mapping to symbols
  - Encode difference between DC coefficients of sequential blocks
- Encode using Huffman or arithmetic encoding



Discrete Cosine Transformation - DCT

- Apply 2D DCT to each block
  - Projection (dot product) to 64 basis functions
  - Function representation in frequency domain

First component is direct current (DC) offset and represents the mean value in the block. Remaining components are alternating - AC



DCT coefficient quantization

- DCT coefficients are divided by quantization table and rounded
  - Defined using psychophysical tests (but not defined in standard)
  - Table quantizes higher frequencies more coarsely
- Controlled loss of information
  - Higher information is lost
  - Quality parameter scales the quantization matrix

$X =$

DCT2 coefficients

	1	2	3	4	5	6	7	8
1	80	-34	16	-9	-55	9	0	-24
2	450	-25	44	28	-30	31	41	-23
3	-217	-79	42	46	-28	14	19	-13
4	66	16	-19	-55	38	-55	-5	-8
5	75	-27	20	-51	30	42	-25	17
6	-8	-34	25	-8	7	19	-10	-7
7	-44	36	-27	54	-53	7	-3	-3
8	6	14	-14	-2	11	-24	10	20

$K =$

Quantization matrix

	1	2	3	4	5	6	7	8
1	10	10	50	50	100	100	100	100
2	10	10	50	50	100	100	100	100
3	50	50	50	50	100	100	100	100
4	50	50	50	50	100	100	100	100
5	100	100	100	100	100	100	100	100
6	100	100	100	100	100	100	100	100
7	100	100	100	100	100	100	100	100
8	100	100	100	100	100	100	100	100

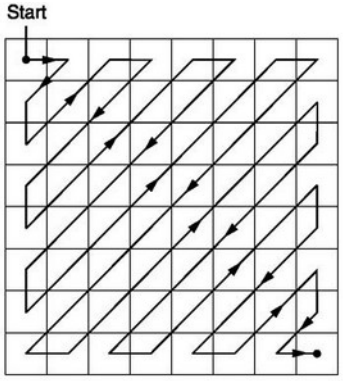
$\lfloor X/K \rfloor \cdot K =$

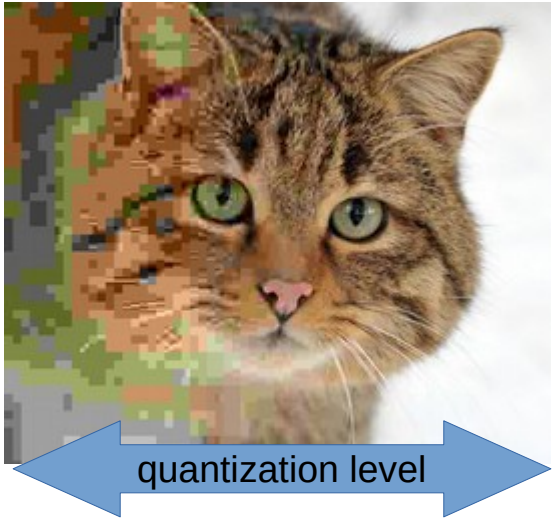
dct2 coefficients from the quantized data

	1	2	3	4	5	6	7	8
1	80	-30	0	0	-100	0	0	0
2	450	-30	50	50	0	0	0	0
3	-200	-100	50	50	0	0	0	0
4	50	0	0	-50	0	-100	0	0
5	100	0	0	-100	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	100	-100	0	0	0
8	0	0	0	0	0	0	0	0

DCT coefficients as symbols

- Separate encoding for DC (1 value) and AC components (63 values)
- Encoding DC component
  - Difference between sequential blocks
- Encoding AC coefficients
  - Ordered from lower to higher frequencies
  - Sequence encoded as a RLE sequence (contains sequences of zeros)
- Writing symbols
  - DC and AC symbols encoded using Huffman or arithmetic encoding
  - Huffman code maps are predefined or calculated on-the-fly





JPEG compression example

640x480 RGB image – raw size: 900kB

- Quality 100: 200kB, error: 0.55
- Quality 80: 47.7kB, error: 1.63
- Quality 60: 32.3kB, error: 2.14
- Quality 40: 25.0kB, error: 2.61
- Quality 20: 16.6kB, error: 3.34
- Quality 0: 5.6kB, error: 9.46

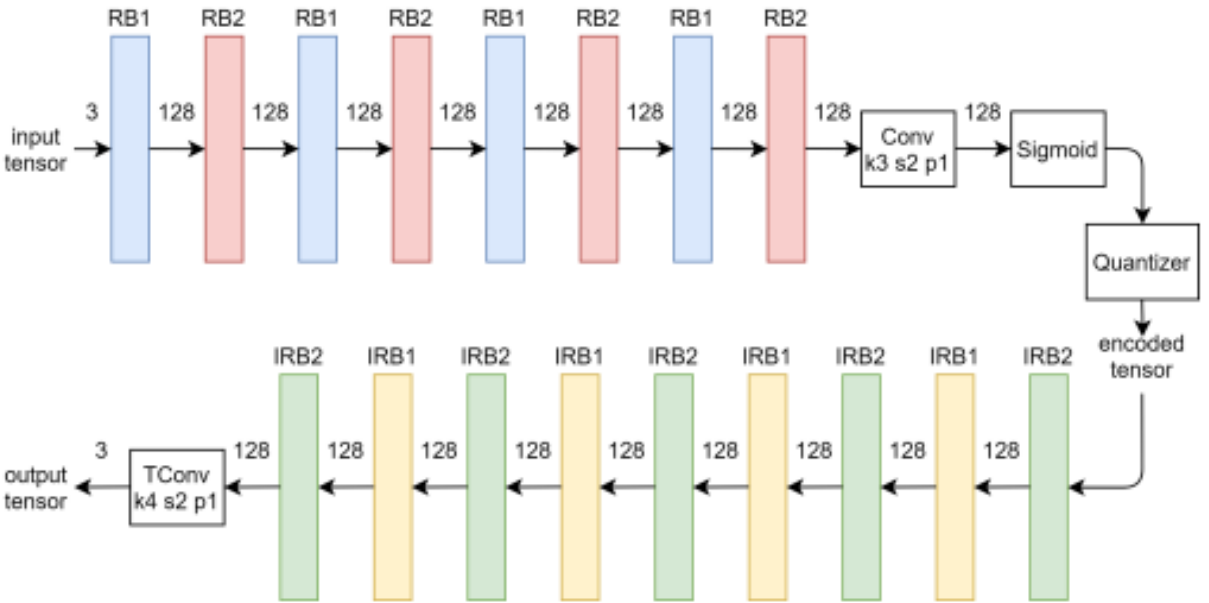
Compressing color images

- Human eye is less sensitive to changes in chroma
  - Separate RGB to intensity and chroma (YCrCb)
  - Down-sample chroma (Cr,Cb) with factor 2
  - Encode each channel separately (different quantization table for Y and Cr/Cb)
  - Using different quantization tables for chroma and intensity

Deep learning for image compression

- Auto-encoder architecture
  - Reduce spatial resolution
  - Increase channels
  - Learn to decode quantized latent representation
- Unsupervised training
  - Learn to reconstruct image
  - Train on real world images

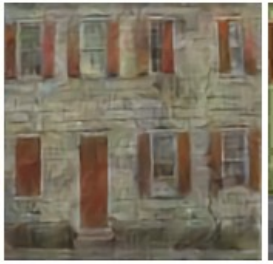
Compression architecture



Jan Pelicon, “Kompresija video posnetkov z nevronskimi mrežami”, 2020

Comparison

$n = 2$, $bpp = 0.150$,
$SS\text{-}SSIM = 0.57$,
$AEC_{noise} = 0$



$n = 2$, $bpp = 0.152$,
$SS\text{-}SSIM = 0.58$,
$AEC_{noise} = \sigma^2$

$Q = 3$, $bpp = 0.142$,
$SS\text{-}SSIM = 0.43$,
JPEG 4:2:0

$n = 16$, $bpp = 0.511$,
$SS\text{-}SSIM = 0.77$,
$AEC_{noise} = 0$

$n = 16$, $bpp = 0.503$,
$SS\text{-}SSIM = 0.70$,
$AEC_{noise} = \sigma^2$

$Q = 14$, $bpp = 0.498$,
$SS\text{-}SSIM = 0.70$,
JPEG 4:2:0