Document and Content Analysis

Lecture 2 — Document Compression

Faisal Shafait

Image Understanding and Pattern Recognition
DFKI & TU Kaiserslautern

2009/04/28

Motivation

Have you seen any files in a compressed format?

Motivation

Have you seen any files in a compressed format?

zip, JPEG, MPEG, MP3, ...

Motivation

Have you seen any files in a compressed format?

zip, JPEG, MPEG, MP3, ...

Do you know how they work?

Motivation

Have you seen any files in a compressed format?

zip, JPEG, MPEG, MP3, ...

Do you know how they work?

You should know after this lecture

Outline

Data vs. Information

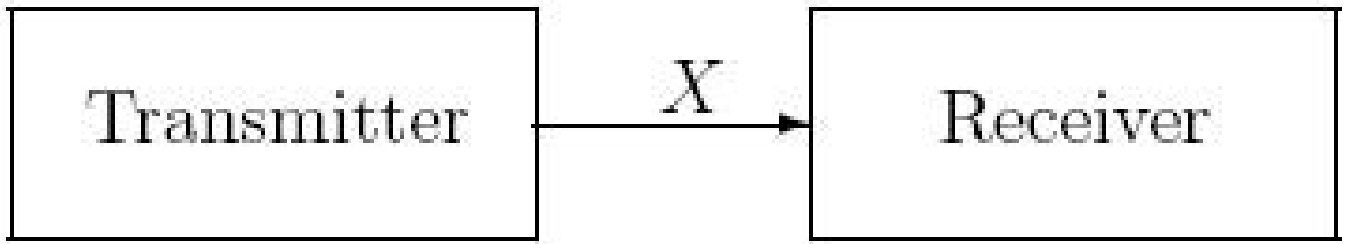
The same information can be encoded as data in many different forms, requiring different numbers of bits, e.g.

$$0080, 0090, 0070, 0070, 0050, 0010, 0090, 0070, 0080, 0090$$

- ASCII sequence
  "0080,0090,0070,0070,0050,0010,0090,0070,0080,0090"
  $\Rightarrow$ 49 bytes, 392 bits
- 32-bit integers: 80,90,70,70,50,10,90,70,80,90
  $\Rightarrow$ $10 \cdot 4$ bytes, 40 bytes, 320 bits
- ASCII: "8,9,7,7,5,1,9,7,8,9" $\Rightarrow$ 19 bytes, 152 bits
- ASCII: "8977519789" $\Rightarrow$ 10 bytes, 80 bits
- ASCII-hex: 2171a14ad $\Rightarrow$ 9 bytes, 72 bits
- binary: one 34 bit integer value $\Rightarrow$ 34 bits

Compression as Exchange of Information

Compression can be seen as an encoding/decoding problem:



where each side has a codebook to code or decode the message X. The codes can be bit-strings of arbitrary length. Often, we require that the codes should be uniquely decodable by looking at one bit at a time (prefix-codes).

Information and Coding Theory

Consider the message X of length n as a random process with i.i.d. (independent, identically distributed) events. The probability for an event i is $p(i)$.

Theorem

The information entropy of a message channel,

$$H := -\sum_i p(i) \log_2 p(i)$$

gives a lower bound on how many bits are needed to encode a message in the channel on the average.

In other words, no matter what the codebook is, a message of length n cannot become shorter than $n \cdot H$ bits per symbol on the average.
H doesn't have to be integer, and if we cannot send fractional bits, then we cannot reach the lower bound exactly.

Entropy as a measure of Information: Example

The message is $0, 1, 0, 2$.

- symbols are $0, 1$ and 2
- we assume $p(0) = 0.5$, $p(1) = p(2) = 0.25$.
- naive codebook: $0 = \mathbf{00}$, $1 = \mathbf{01}$, $2 = \mathbf{10}$
- The code is $\mathbf{00010010}$, i.e. 8 bits, or 2 bits per symbol
- entropy:

$$\begin{aligned} H &= -0.5 \cdot \log_2(0.5) - 0.25 \cdot \log_2(0.25) - 0.25 \cdot \log_2(0.25) \\ &= 0.5 \cdot 1 + 0.25 \cdot 2 + 0.25 \cdot 2 = 1.5 \end{aligned}$$

- A better codebook would be: $0 = \mathbf{0}$, $1 = \mathbf{10}$, $2 = \mathbf{11}$
- The total code is $\mathbf{010011}$, requiring 6 bits, i.e. we reach the theoretical bound of 1.5 bits per symbol.

Entropy as a measure of Information: Examples

The message $0, 1, 2, 3, 4, 5, 6, \ldots, 255$

- $p(i) = \frac{1}{256}$ for $i = 1, \ldots, 256$.
- Naive coding takes 8 bits per symbol.
- $H = -\sum_{i=0}^{255} \frac{1}{256} \log(\frac{1}{256}) = 8$
- So we cannot do better than naive 8-bit.

The message $0, 0, 0, 0, \ldots$.

- There is only one symbol, 0. $p(0) = 1$
- $H = 0$
- We don't have to send anything. We agree on a codebook where "nothing" is mapped to "0".
- problem ?

Variable Length Coding (VLC)

The codes we send do not have to be of identical length:

- original: 128 128 128 128 128 129 129 129 129 130 131 132
- usual 8-bit representation: $12 \cdot 8 = 96$ bits
- Entropy:
  - $p(0, \ldots, 127) = 0$
  - $p(128) = \frac{5}{12} = 0.416$
  - $p(129) = \frac{4}{12} = 0.333$
  - $p(130) = \frac{1}{12} \approx 0.083$
  - $p(131) = \frac{1}{12} \approx 0.083$
  - $p(132) = \frac{1}{12} \approx 0.083$
  - $p(133, \ldots, 255) = 0$

$$H = -\frac{5}{12} \cdot \log_2(\frac{5}{12}) - \frac{1}{3} \cdot \log_2(\frac{1}{3}) - 3 \cdot \frac{1}{12} \cdot \log_2(\frac{1}{12})$$
$$\approx 1.95$$

Utilizing if fewer than 256 values occur

- original: 128 128 128 128 128 129 129 129 129 130 131 132
- We have 5 distinct symbols, so we can use a 3 bit code:
  - 128 $\Rightarrow$ **000**
  - 129 $\Rightarrow$ **001**
  - 130 $\Rightarrow$ **010**
  - 131 $\Rightarrow$ **011**
  - 132 $\Rightarrow$ **100**
- 3 bits per symbol means $12 \cdot 3 = 36$ bits in total

Variable Length Coding (VLC) II

- In the previous example, 132 is uniquely identified after 1 bit:
  - $128 \Rightarrow \mathbf{000}$
  - $129 \Rightarrow \mathbf{001}$
  - $130 \Rightarrow \mathbf{010}$
  - $131 \Rightarrow \mathbf{011}$
  - $132 \Rightarrow \mathbf{1}$
- The message becomes
  $3+3+3+3+3+3+3+3+3+3+3+1 = 34$ bits
- Expected rate: $\frac{5}{12} \cdot 3 + \frac{4}{12} \cdot 3 + \frac{1}{12} 3 + \frac{1}{12} 3 + \frac{1}{12} \cdot 1 = 2.834$ bits per symbol

Variable Length Coding (VLC) III

- In general, it is better to code frequent symbols with short codes.
- Since 128 is more frequent than 132, it is better to use the single $\mathbf{1}$ for 128 and the longer codes for the less frequent values
  - $128 \Rightarrow \mathbf{1}$
  - $129 \Rightarrow \mathbf{000}$
  - $130 \Rightarrow \mathbf{001}$
  - $131 \Rightarrow \mathbf{010}$
  - $132 \Rightarrow \mathbf{011}$
- Expected rate: $\frac{5}{12} \cdot 1 + \frac{4}{12} \cdot 3 + \frac{1}{12} 3 + \frac{1}{12} 3 + \frac{1}{12} \cdot 3 = 2.168$
- Signal length:
  $1 + 1 + 1 + 1 + 1 + 3 + 3 + 3 + 3 + 3 + 3 + 3 = 26$ bits

Huffman Codes

- Huffman codes can be constructed for any given distribution
  - 128 $\Rightarrow$ **1**
  - 129 $\Rightarrow$ **01**
  - 130 $\Rightarrow$ **001**
  - 131 $\Rightarrow$ **0000**
  - 132 $\Rightarrow$ **0001**
- Expected rate: $\frac{5}{12} \cdot 1 + \frac{4}{12} \cdot 2 + \frac{1}{12} 3 + \frac{1}{12} 4 + \frac{1}{12} \cdot 4 = 2.0$
- Encoded signal: **111110101010100100000001** has 24 bits
- Lower bound from coding theorem is $12 \cdot 1.95 = 23.4$
  $\Rightarrow$ 24 is optimal.

Computing Huffman Codes

- compute probabilities of all symbols
- arrange symbols in descending order of probability
- construct a tree by combining two least probability nodes
- assign either 1 or 0 to each branch
- traverse the constructed tree from root node to any leaf to find the code for the leaf

Huffman Codes

- Huffman codes are optimal in the sense that no other code that maps symbols onto fixed bit-strings can create a shorter sequence.
- It doesn't mean that they perform closely to the entropy bound, e.g. for a binary signal $p(0) = 0.9999$, $p(1) = 0.0001$: $H \approx 0.0015$, but we need at least 1 bit per symbol.
- If we drop the requirement of mapping symbols onto fixed bit-strings, we can do better $\longrightarrow$ arithmetic encoding

Signal Compression by Entropy Coding

- Storing information (text, images, sound) can be performed more efficiently by using a good code, based on the entropy.
- We have achieved a compression ratio of $\frac{96}{24} = 4$ compared to raw 8-bit integers (PGM), just by using different symbols.
- Many compression utilities, e.g. WinZIP, rely on entropy codings, often by constructing Huffman codes.
- Huffman compression achieves rates of approx. 2 to 3 on English text, and 1 to 4 on images.
- Huffman coding doesn't have a fixed codebook. It depends on the data.
- In practice, the codebook and the length of the signal have to be transmitted together with the data!

Run-Length Encoding (RLE)

In many cases, neighboring data points are not statistically independent, but are strongly correlated (consider sending a fax). If the same values occur frequently next to each other, we can use *Run-Length-Encoding*:

- original: 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 $\Rightarrow$ 23 bit
- $p(0) = \frac{15}{23} = 0.65$, $p(1) = \frac{8}{23} = 0.35$, $H \approx 0.934$
- Run+Value: 6×0, 5×1, 5×0, 2×1, 5×0
- values alternate in fixed manner, only store run: 6 5 5 2 5
- naive 3-bit code: 15 bits

The result can be entropy-compressed:

- $p(2) = \frac{1}{6}$, $p(5) = \frac{3}{5}$, $p(6) = \frac{1}{10}$ $\quad H \approx 1.23$
- Huffman code: $5 \rightarrow \mathbf{1}, 2 \rightarrow \mathbf{00}, 6 \rightarrow \mathbf{01}$
- expected length: $\frac{1}{6} \cdot 2 + \frac{2}{3} \cdot 1 + \frac{1}{3} \cdot 2 = 1.333$ bits per sample
- total message: $\mathbf{0111001}$, 7 bits

Intermediate Summary

- Information is not the same as data. The same information can be expressed (**coded**) using more or less data.
- If we code a signal, the **entropy** of the coding alphabet is a **lower bound** on how many bits we will need per symbol (Shannon's Source Coding Theorem).
- The lower the entropy, the more efficient we can code.
- The more "peaked" the distribution, the lower its entropy.
- For each source, we can build a **Huffman-Code**. This is **optimal** (when...?).
- The Huffman code usually does not reach the entropy bound, but it is guaranteed to stay within 1 bit of it.
- Since entropy and coding efficiency depend on the **coding alphabet** distribution, we can achieve better compression by first transforming a signal, e.g. by RLE.

Making Use of Redundancy for Compression

Run-Length Encoding relies on redundancies between data samples. It is a general fact, that we can increase the compression ratio by utilizing data sample redundancy, e.g. by coding pairs of samples.
Take e.g. a pairs of data samples (i, j), and use the first data sample value as a predictor for the second, using the relation $p(i, j) = p(j|i)p(i)$:

$$\begin{aligned} H(i,j) &= -\sum_{i,j} p(i,j) \log_2 p(i,j) \\ &= -\sum_{i,j} p(j|i)p(i) \log_2 p(j|i) - \sum_{i,j} p(j|i)p(i) \log_2 p(i) \\ &= -\sum_{i} p(i) \sum_{j} p(j|i) \log_2 p(j|i) - \sum_{i} p(i) \log_2 p(i) \\ &= H(j|i) + H(i) \end{aligned}$$

- A signal of length N has $\frac{N}{2}$ pairs and requires at least $\frac{N}{2}\left(H(j|i) + H(i)\right)$ bits.
- If i and j are independent, $p(j|i) = p(j)$ and $H(j|i) = H(j)$, and we cannot gain anything.
- Otherwise, $H(j|i) < H(j)$, and the theoretical bound is lower than for coding the symbols one-by-one.

Differential Pulse Code Modulation (DPCM)

One method to use redundancy is to use the previous data sample as a linear predictor and encode $s_1, s_2 - s_1, s_3 - s_2, \ldots$.

- original: 128 128 128 128 128 129 129 129 129 130 131 132
- relative: 128 +0 +0 +0 +0 +0 +1 +0 +0 +1 +1 +1
  - $p(0) = \frac{7}{12} \approx 0.583$
  - $p(1) = \frac{4}{12} \approx 0.333$
  - $p(128) = \frac{1}{12} \approx 0.083$
- $H = -\frac{7}{12} \cdot \log_2(\frac{7}{12}) - \frac{4}{12} \cdot \log_2(\frac{4}{12}) - \frac{1}{12} \cdot \log_2(\frac{1}{12}) \approx 1.28$
- Huffman code: $0 \Rightarrow \mathbf{0}$, $1 \Rightarrow \mathbf{10}$, $128 \Rightarrow \mathbf{11}$.
- Expected rate: $\frac{7}{12} \cdot 1 + \frac{4}{12} \cdot 2 + \frac{1}{12} \cdot 2 = 1.415$
- Stored message: $\mathbf{1100000100010 1010}$, 17 bits

This method is also called *Differential Pulse Code Modulation* (DPCM). It improves the compression ratio, if the distribution of the difference values has a lower entropy than the sample values themselves.

Outline

Image as Data

An image is represented as a 2-D matrix, where each element is the intensity value at that location in the image

All techniques studied so far can be applied to image compression as well.

- All methods so far are reversible processes.
- The result is a *lossless* compression method. The decoded data is exactly identical to the original.

In images we can further exploit 2-dimensional spatial correlation of data.

Decorrelation of Image Data

Observation: A signal in which the samples are statistically independent cannot be coded more efficiently by combining symbols.

In image compression one searches for representations of the image data such that the individual samples are *uncorrelated*. The distribution of values should be as peaked as possible for as many coefficients as possible. Afterwards, applying entropy coding is very efficient.

How can we make image data uncorrelated while keeping the image information?

Motivation for unitary transforms

- Requirements of the transform:
  - *Energy compaction* to a small, relevant set of features for storage, transmission and analysis
  - *Decorrelation* of the image pixels
  - *Conservation of energy*
  - *Conservation of measure*, e.g. conservation of mean square differences between images
  - Features with small energy can be skipped (compression)
- Idea:
  - Represent image by weighted sum of *basis images*
  - Transform results (features) are the *weighting coefficients* within the sum
  - Basis images should be good representations of the input image content, so the optimal transform is image dependent

Discrete Cosine Transform (DCT)

- The DCT base functions do not depend on the data.
- They can be stored in the decoder or calculated on-the-fly.
- We only have to transmit the coefficients.

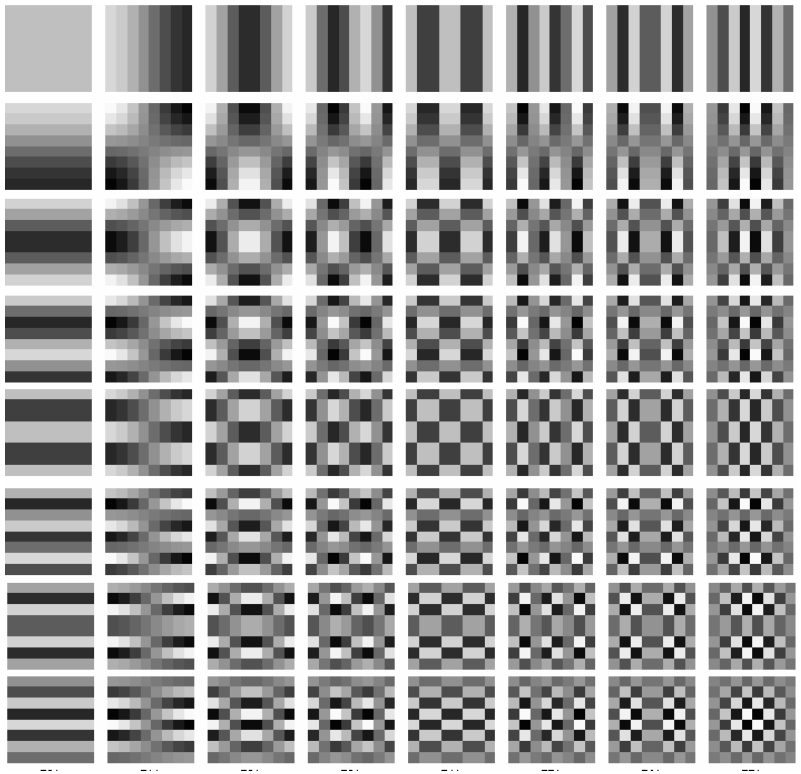
2D DCT

The 2D discrete cosine transform (DCT) of a an image $S(x, y)$ is computed as:

$$\mathcal{S}(u, v) = \frac{2}{N} K(u) K(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} S(x, y) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N}$$

where $K(0) = \frac{1}{\sqrt{2}}$ and $K(w) = 1$ for $w = 1, 2, ..., N-1$

DCT base images



The DCT is a separable transform related to the FFT and can be performed in $O(n \log n)$.

Example: 2D cosine transform of a 16x16 image

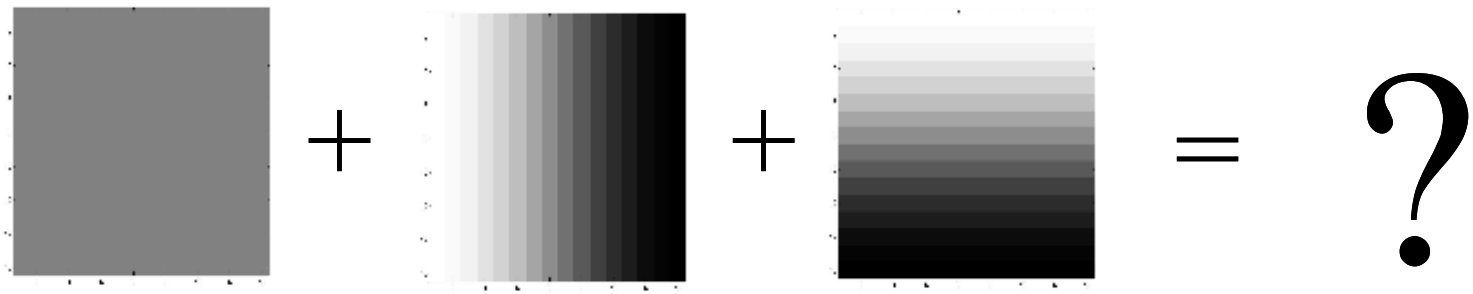


Figure: $\frac{1}{16}S(0,0) + \frac{1}{8\sqrt{2}}(S(0,1) + S(1,0))$

Example: 2D cosine transform of a 16x16 image (II)

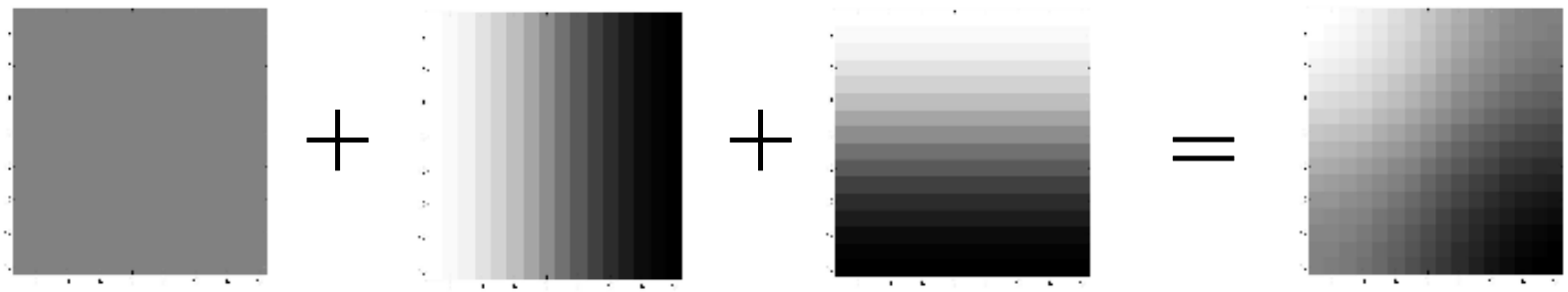


Figure: $\frac{1}{16}S(0,0) + \frac{1}{8\sqrt{2}}(S(0,1) + S(1,0))$

Example: 2D cosine transform of a 16x16 image (III)

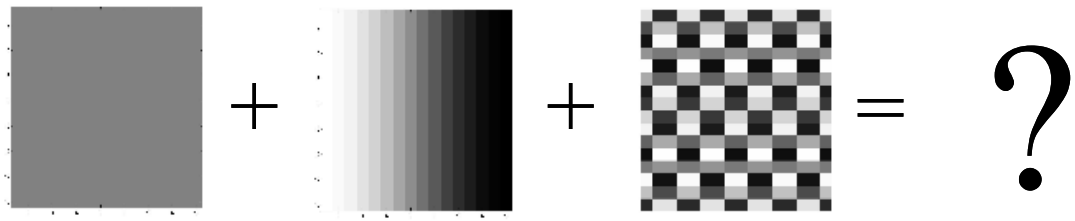


Figure: $\frac{1}{16}S(0,0) + \frac{1}{8\sqrt{2}}S(0,1) + \frac{1}{8}S(6,7)$

Example: 2D cosine transform of a 16x16 image (IV)

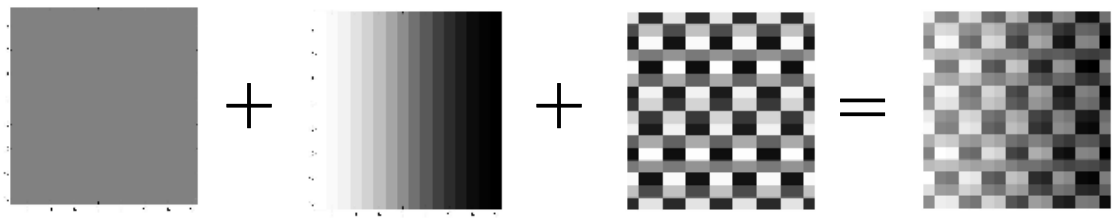


Figure: $\frac{1}{16}S(0,0) + \frac{1}{8\sqrt{2}}S(0,1) + \frac{1}{8}S(6,7)$

Example: 2D cosine transform of a 16x16 image (V)

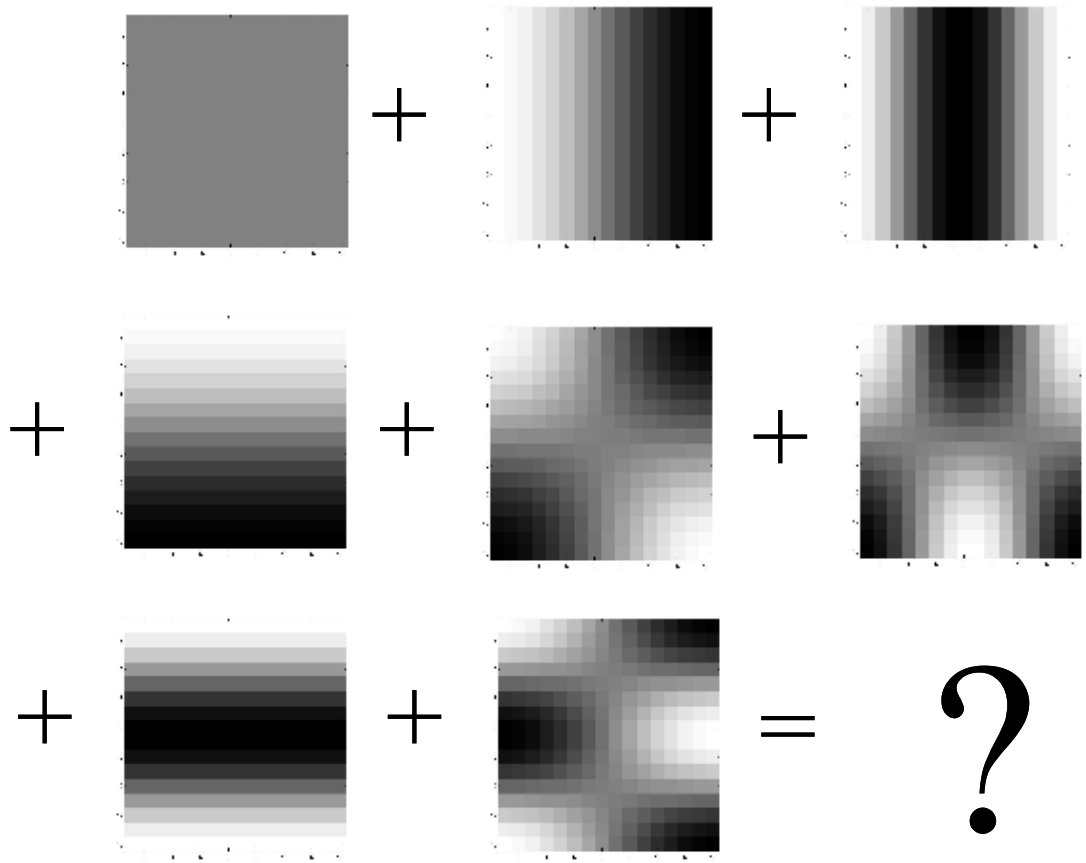


Figure: $\frac{1}{16}S(0,0) + \frac{1}{8\sqrt{2}}S(0,1) + \frac{1}{8\sqrt{2}}S(1,0) + \frac{1}{8}S(1,1) + \frac{1}{8\sqrt{2}}S(2,0) + \frac{1}{8\sqrt{2}}S(0,2) + \frac{1}{8}S(2,1) + \frac{1}{8}S(1,2)$

Example: 2D cosine transform of a 16x16 image (VI)

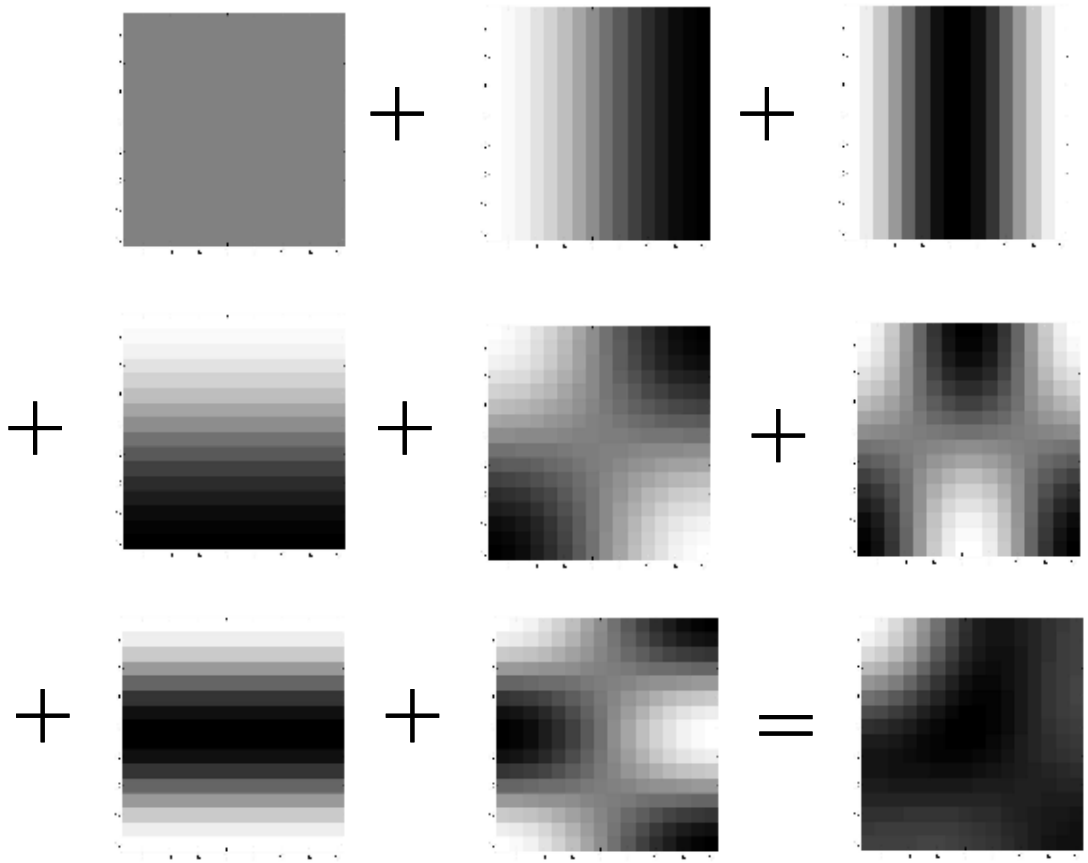


Figure: $\frac{1}{16}S(0,0) + \frac{1}{8\sqrt{2}}S(0,1) + \frac{1}{8\sqrt{2}}S(1,0) + \frac{1}{8}S(1,1) + \frac{1}{8\sqrt{2}}S(2,0) + \frac{1}{8\sqrt{2}}S(0,2) + \frac{1}{8}S(2,1) + \frac{1}{8}S(1,2)$

Simple Methods for Image Compression

With our knowledge so far, we can create simple methods for the compression of different kinds of images:

For binary images:

- Express the image by Run-Length-Encoding.
- Perform entropy coding of the run values.

This method is similar to the compression used for TIFF images and in fax machines.

For artificial images (e.g. computer graphics, screenshots)

- Create a predicted representation of the image data
- Perform entropy coding of the result.

This method is similar to the **PNG** format. PNG prediction is done vertically, each row being predicted by previous rows. In **lossless JPEG**, a pixel is predicted from its top, left and top-left neighbor.

Simple Methods for Image Compression

For natural images:

- Convert RGB input to YCbCr color space. The separate brightness and color channels in YCbCr are less correlated.
- Split the image (each color channel) into blocks of size 8×8.
- For each block, calculate the 2D-DCT coefficients.
- Arrange the coefficients into a $1D$-order (usually zig-zag).
- Perform entropy coding to the resulting coefficients.

This method achieves compression ratios close to 2. The structure is similar to JPEG encoding, but some crucial steps are missing.

Larger blocks than 8×8 improve the compression ratio but require more computing time and memory (DCT is $O(n \log n)$).

Instead of DCT, other transforms can be used as well, e.g. wavelets. Then usually the whole image is treated as a single block.

Outline

Lossless vs. Lossy Compression

- Lossless compression relied on the removal of redundancy.
- Lossy compression can use another powerful source of redundancy: the human visual system!
- We do not only reduce the amount of data, but we discard a part of the information, that does not change the human perception much.

The human visual system: Least Noticable Differences


The color depth was reduced from 8 to 7 bits per pixel per channel: 14% size reduction

The human visual system: High Frequencies


The highest frequencies were removed from the FFT spectrum: 10% size reduction

The human visual system: Color Resolution

The color channels in YCbCr were subsampled to half resolution: 50% size reduction

The human visual system: Resolution


The image was resized to 50% resolution:
75% size reduction.

JPEG compression

JPEG compression uses all the concepts we have seen:

- Conversion to YCbCr color space and reduction of resolution in the color components
- Blockwise 2D-DCT in each channel
- Quantization of the resulting DCT coefficients
- Predictive Encoding for the 1st quantized coefficent (DC)
- Run-Level Encoding for the other quantized coefficients
- Huffman Encoding of what is left

2D-DCT of 8×8 blocks

Luminance and Chrominance are arranged into 8×8 blocks. A value of 128 is subtracted from each entry and the resulting 8×8 block is transformed using the 2D-DCT. This yields an 8×8 block of coefficients.

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix} \longrightarrow \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}$$

$$\longrightarrow \begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}$$

The $(0, 0)$ (top-left) entry is called *DC*-coefficient, the other entries are called *AC*-coefficients.

Quantization of DCT-coefficients

The DCT coefficients are *quantized* by dividing them by a constant and rounding to the next integer. Each position in the block can have a different value (*quantizer*) to be divided by. The quantizers form quantization matrices for luma and chroma.

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	36	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

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

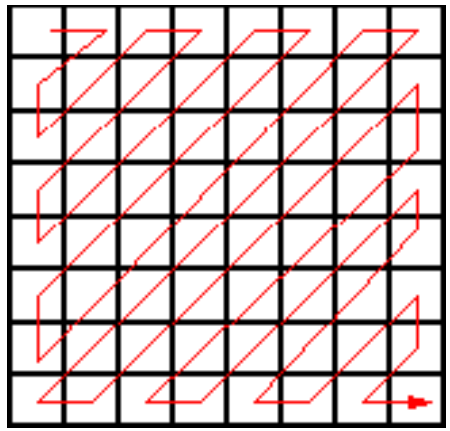
Quantizers are chosen with regards to the human visual system.

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Zigzag ordering

The 2D-block entries are read out in a zigzag order

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



This yields a 1D-array of size 64:
-26, -3, 0, -3, -2, -6, 2, -4, 1, -4, 1, 1, 5, 1, 2, -1, 1, -1, 2, 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

Run-Level Encoding

The DC coefficient is predicted from the previously processed block.

$$-26 \mapsto +2$$

The AC coefficients are Run-Length encoded using (*run*, *level*) codes. *run* is the number of zeros and *level* is the following coefficient value. There is a special code *EOB* for *end of block*.

-3, 0, -3, -2, -6, 2, -4, 1, -4, 1, 1, 5, 1, 2, -1, 1, -1, 2, 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

becomes

$(0, -3), (1, -3), (0, -2), (0, -6), (0, 2), (0, -4), (0, 1), (0, -4), (0, 1), (0, 1), (0, 5), (0, 1), (0, 2), (0, -1), (0, 1), (0, -1), (0, 2), (5, -1), (0, -1)$, *EOB*

Huffman Encoding

The Run-Level pairs are entropy encoded. The codebooks are not stored in the bitstream, but generated by a subroutine.

- For DC, larger absolute values have longer codewords.
- For AC, the codebook is specially designed for JPEG.

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
...	...	...
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
...	...	...
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111

JPEG decompression

JPEG decompression applies all encoding steps in backwards order:

- Decoding of Huffman Codes
- Conversion of Run-Level Codes to 1D array
- Addition of prediction to DC coefficient
- Dequantization of all coefficients
  $coeff[i] \quad \mapsto \quad q[i] \cdot coeff[i]$
- Arrangement of coefficients into 8×8 block
- Inverse 2D-DCT of each block
- Conversion to 4 luma and 2 chroma block to 4 RGB blocks.

The JPEG-standard only describes the decompression process! How exactly to come up with a valid bitstream is up to the compression software. (What are advantages of this approach?)

Results

original 24 bpp, 1:1

PNG: 14.7 bpp

JPEG: 1.6 bpp, 1:15

JPEG: 0.267 bpp, 1:90

JPEG: 0.114 bpp, 1:210

JPEG: 0.08 bpp, 1:300

JPEG achieves good visual results, often down to 0.1 bits per pixel.

Measuring Lossy Compression Performance

Lossy image compression methods are measured in a rate-distortion curve:

- By the choice of the quantizer matrix, we can reach virtually any compression ratio, but at the same time, more and more image information is discarded.
- To measure the image degradation, we use the mean square error (MSE) or the Peak-Signal-to-Noise-Ratio (PSNR):

$$MSE(I_{compr}, I_{orig}) = \frac{1}{N \cdot M} \sum_{x,y} (I_{compr}(x,y) - I_{orig}(x,y))^2$$

$$PSNR(I_{compr}, I_{orig}) = 10 \cdot \log_{10} \frac{255^2}{MSE(I_{comp}, I_{orig})}$$

- The PSNR depends strongly on the input image.

Problems of PSNR

- The PSNR is only meaningful when comparing exactly the same image information.
- It is an *objective* measure, based on signal processing theory.
- The human visual impression is *subjective*:
  – Different people judge the image quality differently.
  – Higher PSNR does not automatically mean better visual quality (even for the same person and the same image)
- Many other measures have been proposed, which should reflect the subjective quality better, but there is no established standard.
- When real subjective testing is crucial, human testing is necessary. Usually judging images on a *MOS (mean-opinion-score) scale*:

$$5 = \textit{excellent},\ 4 = \textit{good},\ 3 = \textit{fair},\ 2 = \textit{poor},\ 1 = \textit{bad}$$

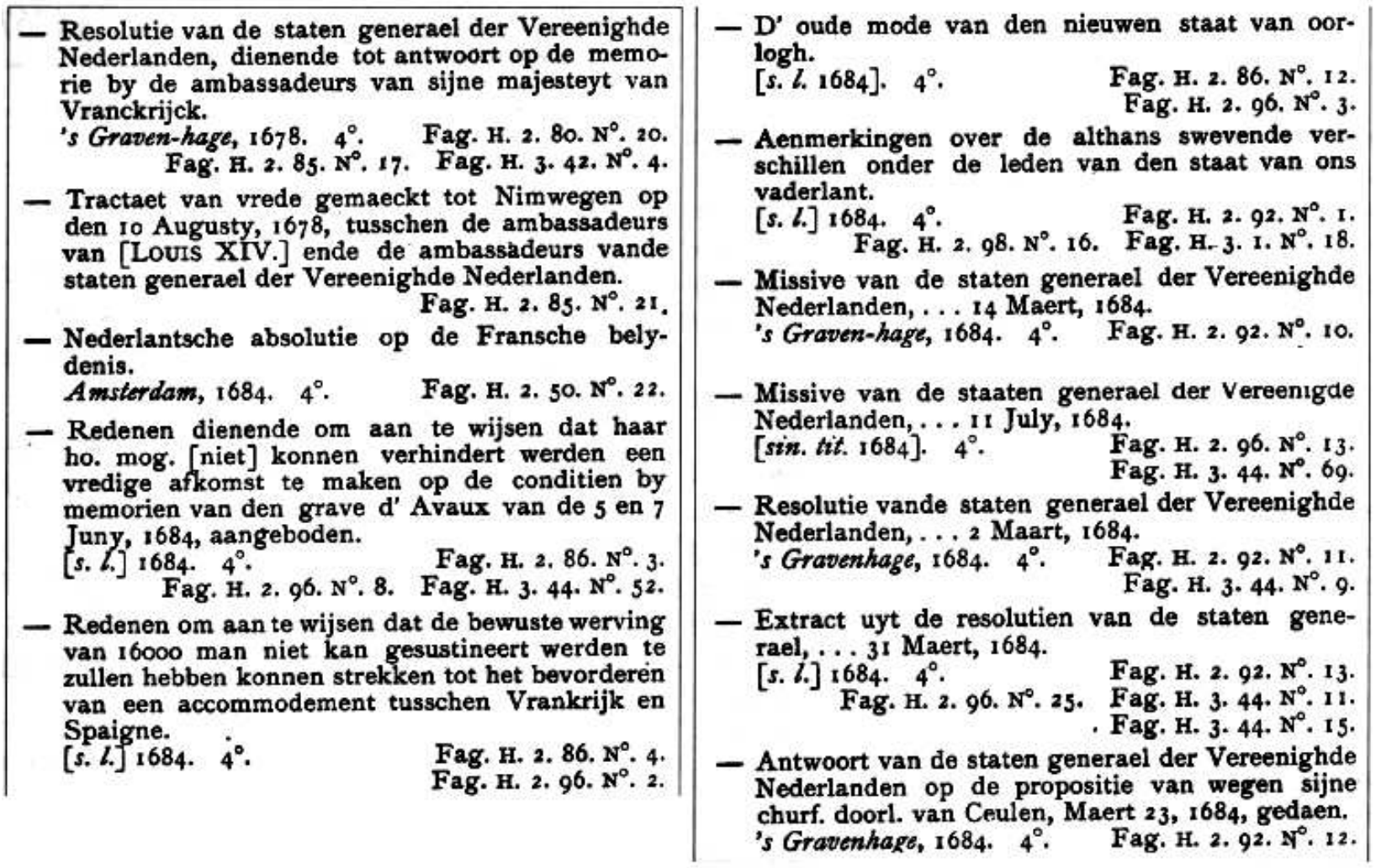
Outline

Document Image Properties

properties of document images (or textual images)
relevant for compression:

- binary images / high contrast
- repetition of *symbols*
- regular distribution of symbols / symbol distances

Example Image



— Resolutie van de staten generael der Vereenighde Nederlanden, dienende tot antwoort op de memorie by de ambassadeurs van sijne majesteyt van Vranckrijck.
's Graven-hage, 1678. 4°. Fag. H. 2. 80. N°. 20. Fag. H. 2. 85. N°. 17. Fag. H. 3. 42. N°. 4.

— Tractaet van vrede gemaeckt tot Nimwegen op den 10 Augusty, 1678, tusschen de ambassadeurs van [Louis XIV.] ende de ambassadeurs vande staten generael der Vereenighde Nederlanden.
Fag. H. 2. 85. N°. 21.

— Nederlantsche absolutie op de Fransche belydenis.
Amsterdam, 1684. 4°. Fag. H. 2. 50. N°. 22.

— Redenen dienende om aan te wijsen dat haar ho. mog. [niet] konnen verhindert werden een vredige afkomst te maken op de conditien by memorien van den grave d' Avaux van de 5 en 7 Juny, 1684, aangeboden.
[*s. l.*] 1684. 4°. Fag. H. 2. 86. N°. 3. Fag. H. 2. 96. N°. 8. Fag. H. 3. 44. N°. 52.

— Redenen om aan te wijsen dat de bewuste werving van 16000 man niet kan gesustineert werden te zullen hebben konnen strekken tot het bevorderen van een accommodement tusschen Vrankrijk en Spaigne.
[*s. l.*] 1684. 4°. Fag. H. 2. 86. N°. 4. Fag. H. 2. 96. N°. 2.

— D' oude mode van den nieuwen staat van oorlogh.
[*s. l.* 1684]. 4°. Fag. H. 2. 86. N°. 12. Fag. H. 2. 96. N°. 3.

— Aenmerkingen over de althans swevende verschillen onder de leden van den staat van ons vaderlant.
[*s. l.*] 1684. 4°. Fag. H. 2. 92. N°. 1. Fag. H. 2. 98. N°. 16. Fag. H. 3. 1. N°. 18.

— Missive van de staten generael der Vereenighde Nederlanden, . . . 14 Maert, 1684.
's Graven-hage, 1684. 4°. Fag. H. 2. 92. N°. 10.

— Missive van de staaten generael der Vereenigde Nederlanden, . . . 11 July, 1684.
[*sin. tit.* 1684]. 4°. Fag. H. 2. 96. N°. 13. Fag. H. 3. 44. N°. 69.

— Resolutie vande staten generael der Vereenighde Nederlanden, . . . 2 Maart, 1684.
's Gravenhage, 1684. 4°. Fag. H. 2. 92. N°. 11. Fag. H. 3. 44. N°. 9.

— Extract uyt de resolutien van de staten generael, . . . 31 Maert, 1684.
[*s. l.*] 1684. 4°. Fag. H. 2. 92. N°. 13. Fag. H. 2. 96. N°. 25. Fag. H. 3. 44. N°. 11. Fag. H. 3. 44. N°. 15.

— Antwoort van de staten generael der Vereenighde Nederlanden op de propositie van wegen sijne churf. doorl. van Ceulen, Maert 23, 1684, gedaen.
's Gravenhage, 1684. 4°. Fag. H. 2. 92. N°. 12.

Compression Approaches

approaches for compression

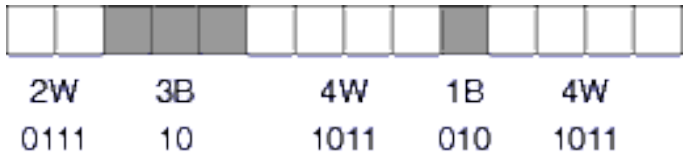
- standard-approaches like PNG/GIF (runs, repetitions)
- OCR and only save text and position
- fax compression (also used in TIFF, runs, similar lines)
- context-dependent arithmetic coding (JBIG)
- token-based compression and mixed raster format (DJVU)

JBIG=Joint Bi-level Image Experts Group
JPEG=Joint Photographic Experts Group

Fax Compression

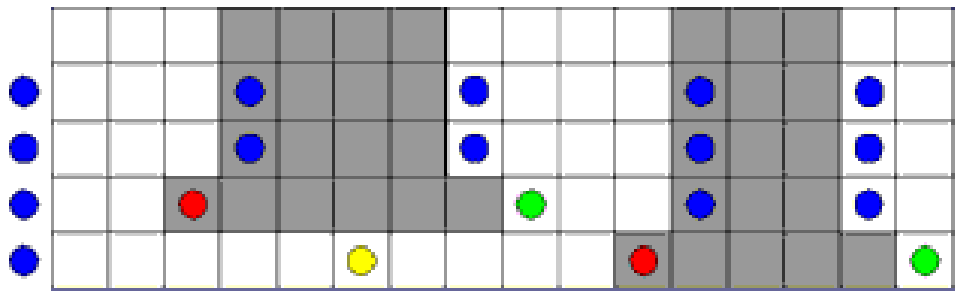
CCITT-G3

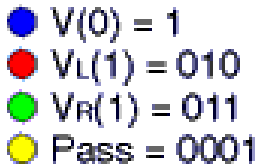
- current fax standard
- each line encoded separately (recover from errors)
- use Huffman coding for runs of pixels



CCITT-G4

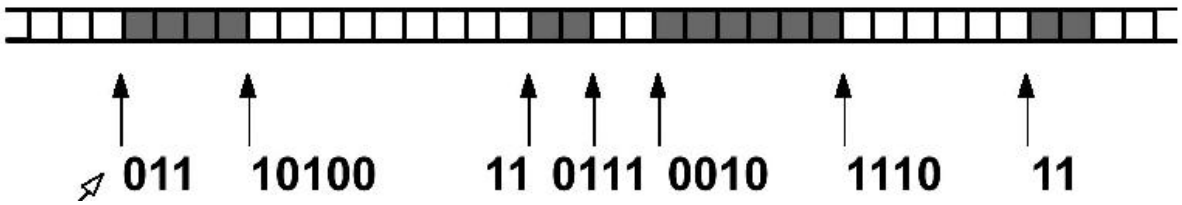
- used in TIFF-G4 compression
- lines can be encoded as in G3, or reference the previous line
- uses relative position of positions of color change





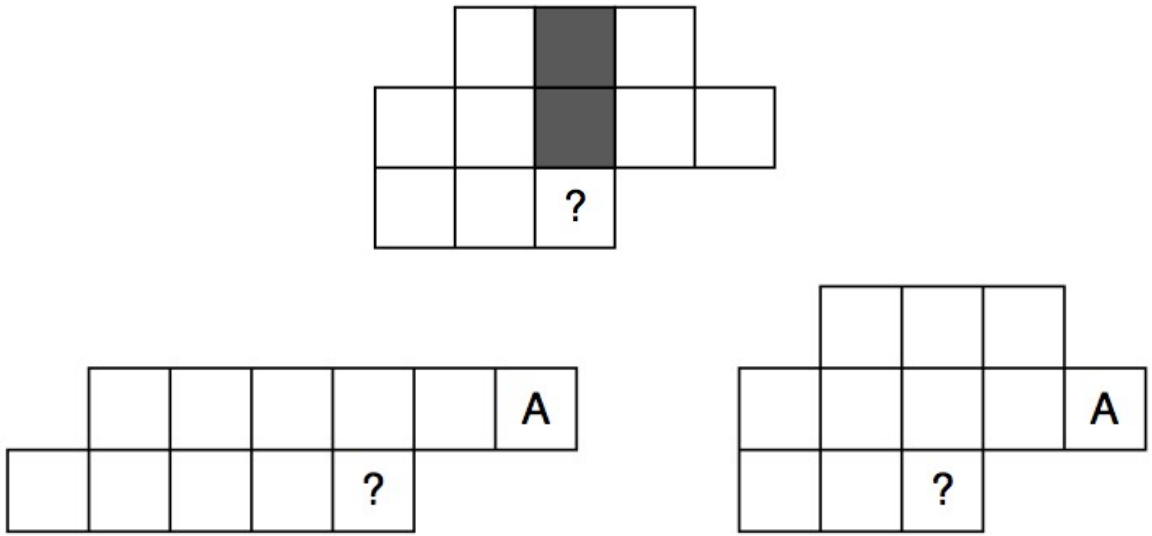
Resulting code:
...1111111111101001111100010100 11...

Fax Compression



Run Length	White Code Word	Black Code Word	Run Length	White Code Word	Black Code Word
0	00110101	0000110111	32	00011011	000001101010
1	000111	010	33	00010010	000001101011
2	0111	11	34	00010011	000011010010
3	1000	10	35	00010100	000011010011
4	1011	011	36	00010101	000011010100
5	1100	0011	37	00010110	000011010101
6	1110	0010	38	00010111	000011010110
7	1111	00011	39	00101000	000011010111
8	10011	000101	40	00101001	000001101100
9	10100	000100	41	00101010	000001101101
10	00111	0000100	42	00101011	000011011010
11	01000	0000101	43	00101100	000011011011
12	001000	0000111	44	00101101	000001010100
13	000011	00000100	45	00000100	000001010101
14	110100	00000111	46	00000101	000001010110
15	110101	000011000	47	00001010	000001010111

Context-Dependent Compression



- context allows a much better prediction of a pixel value
- context models have lower entropy and better compression

Example: JBIG (JBIG1)

Token-based Compression

- find all symbols (or tokens) $\rightarrow$ connected components
- construct a library of symbols (use repetition/similarity)
- compress library, sequence of symbols, and distances
- (lossless: compress difference image)

Example: JBIG2

Token-based Compression

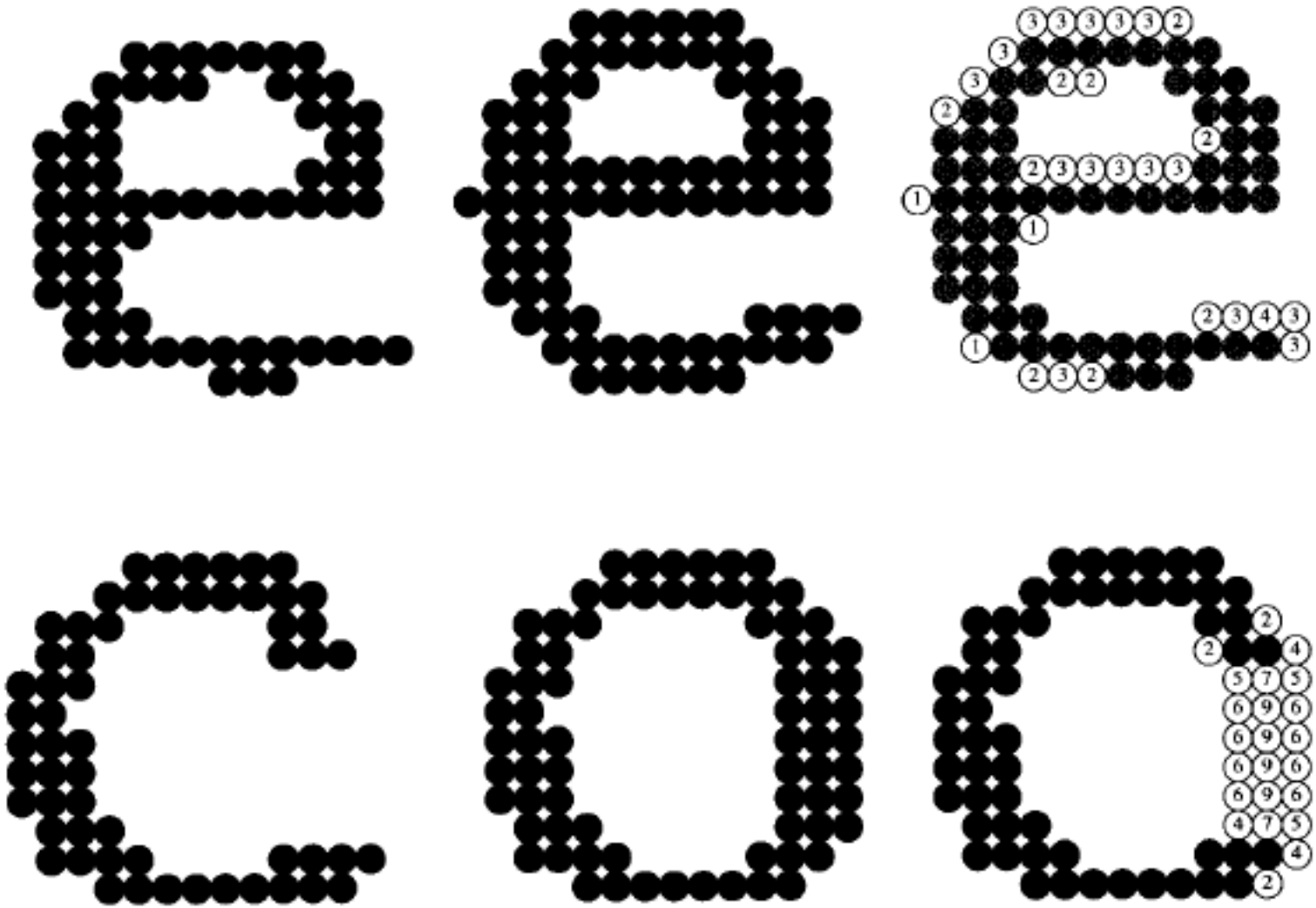
Construction of the library:

- for each token check if a *similar* token exists in the library, otherwise add it to the library
- needs a definition of similarity $\longrightarrow$ character recognition

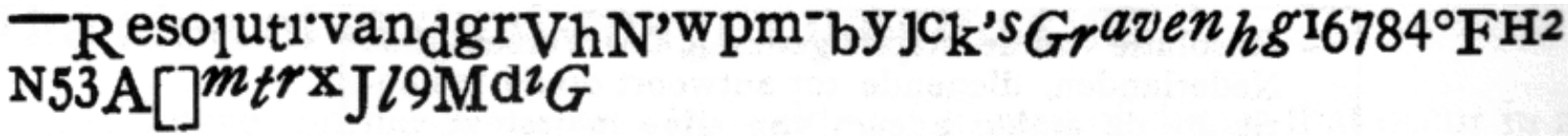
Example of (dis-)similarity:

- perform rough pre-selection (e.g. by size)
- normalize tokens with respect to center of mass
- one of
  - weighted pixel-wise XOR
  - Fourier coefficients
  - flexible image matching

Dissimilarity of Tokens



Library and Symbol Sequence



Symbols and Distances			
Δx	Δy	No.	Symbol
19	62	1	—
24	7	2	R
2	-1	3	e
3	0	4	s
4	0	5	o
2	-1	6	l
3	0	7	u
3	0	8	t
3	0	9	ı
-8	-25	10	.
7	25	3	e
15	0	11	v

— Resolutie van de staten generael der Vereenighde Nederlanden, dienende tot antwoort op de memorie by de ambassadeurs van sijne majesteyt van Vranckrijck.
's Graven-hage, 1678. 4°. Fag. H. 2. 80. N°. 20.
Fag. H. 2. 85. N°. 17. Fag. H. 3. 42. N°. 4.

— Tractaet van vrede gemaeckt tot Nimwegen op den 10 Augusty, 1678, tusschen de ambassadeurs van [Louis XIV.] ende de ambassadeurs vande staten generael der Vereenighde Nederlanden.
Fag. H. 2. 85. N°. 21.

— Nederlantsche absolutie op de Fransche bely-

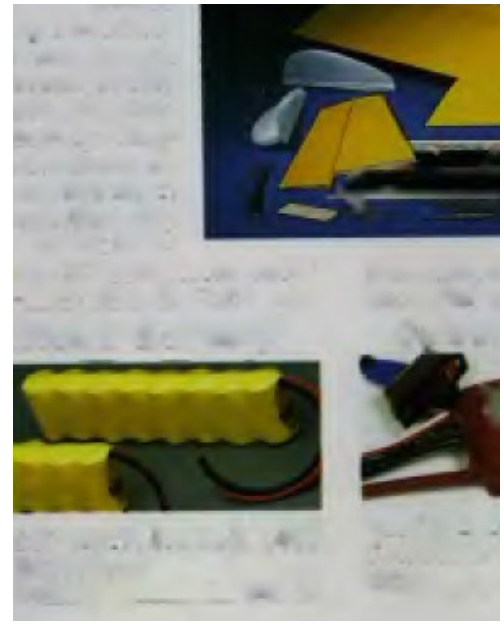
Mixed Raster Compression (MRC)

example: DJVU (also: JBIG2)

split image into layers and encode separately with appropriate method

- text layer $\longrightarrow$ token-based at about 300-400dpi
- text-color layer (usually black) $\longrightarrow$ image at about 25dpi
- background, images, texture $\longrightarrow$ image at about 100dpi

Mixed Raster Compression (MRC)



Summary of Image Compression

- Compression uses (different forms of) **redundancy**
- **Lossless compression** reduces data without losing information: entropy coding (Huffman) and run-level encoding.
- Symbols are better compressible if their distribution is strongly peaked.
- Pixel correlations can be used to make probability distributions more peaked: direct prediction or PCA/DCT/DWT
- **Lossy compression** achieves much higher compression ratios by discarding information.
- Properties of the human visual system are used to only remove "invisible" information:
  - Greyscale values or DCT coefficients are quantized.
  - High frequencies quantized stronger than low frequencies.
  - Color information is downscaled compared to brightness.

Audio Compression

How would you design an audio compression system (MP3)?