Bildkompression

Einführung

- Was ist ein Bild?
  - Rechteckiges Array aus Pixeln
  - 3 Farbkomponenten pro Pixel
    - RGB (rot, grün, blau), oder
    - YUV (Y $\approx$ Luminanz $\approx$ Helligkeit; U, V $\approx$ Farbinformation)
  - meistens 8 bit pro Pixel und Farbkomponente
- hoher Anteil an Redundanz.
  Benachbarte Pixel haben mit großer Wahrscheinlichkeit gleiche oder ähnliche Farbe.
- Kompressionsfehler bis zu gewissem Grad akzeptierbar. ($\Rightarrow$ lossy compression)

Transform Coding

Transform Coding ist die bewährteste Methode zur Bildkompression. Sie besteht grob aus drei Schritten:

1. Transformation der Bilddaten.
   Ziel: Information soll sich auf einige wenige Koeffizienten konzentrieren.
2. Quantisierung
   Der Wertebereich der Koeffizienten wird vergröbert.
   $\Rightarrow$ Der Informationsgehalt wird verringert $\Rightarrow$ stärkere Kompression aber Qualitätsverlust.
3. Entropiekodierung
   - Huffman-Codes (oder ähnliche Verfahren)
     Häufige Werte (z.B. 0 ist nach der Quantisierung sehr häufig) bekommen kürzere Codes.
   - Häufigkeitsverteilung abhängig von benachbarten Koeffizienten
   - Differenzkodierung
   - etc. . . .

ad 1. Transformation der Bilddaten

Zur Transformation von Bilddaten (hier repräsentiert durch die Funktion f, f als Funktion vom Bildbereich auf Grau- bzw. Farbwerte) benötigt man einen Satz Basisfunktionen b_i, um einen entsprechenden Satz Koeffizienten c_i zu ermitteln, aus denen die urpsrüngliche Funktion auf folgende Weise zusammengesetzt werden kann:

$$f = c_1 b_1 + c_2 b_2 + c_3 b_3 + \cdots$$

Die Frage ist nun, wie die Koeffizienten c_i ermittelt werden. Dazu gibt es folgenden Satz aus der Mathematik:

> Wenn die b_i eine Orthonormalbasis darstellen, dann gilt:
>
> $$c_i = \langle f | b_i \rangle = \sum_t f(t) b_i(t)$$

Zur Erinnerung: b_i ist eine Orthonormalbasis, wenn:

$$\langle b_i | b_j \rangle = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

ad 1. Transformation der Bilddaten (2)

Die Komplexität dieser Vorgangsweise ist $O(n^2)$, d.h. die Anzahl der Multiplikationen, die benötigt werden, um alle c_i zu berechnen, ist von der Ordnung n^2, wobei n die Anzahl der Bildpixel ist. Meist wird zur Berechnung der Koeffizienten daher ein spezieller Algorithmus verwendet, der schneller ist.

Folgende Transformationen sind wichtig:

- **DCT** – Diskrete Kosinus-Transformation (Komplexität $O(n \log n)$)

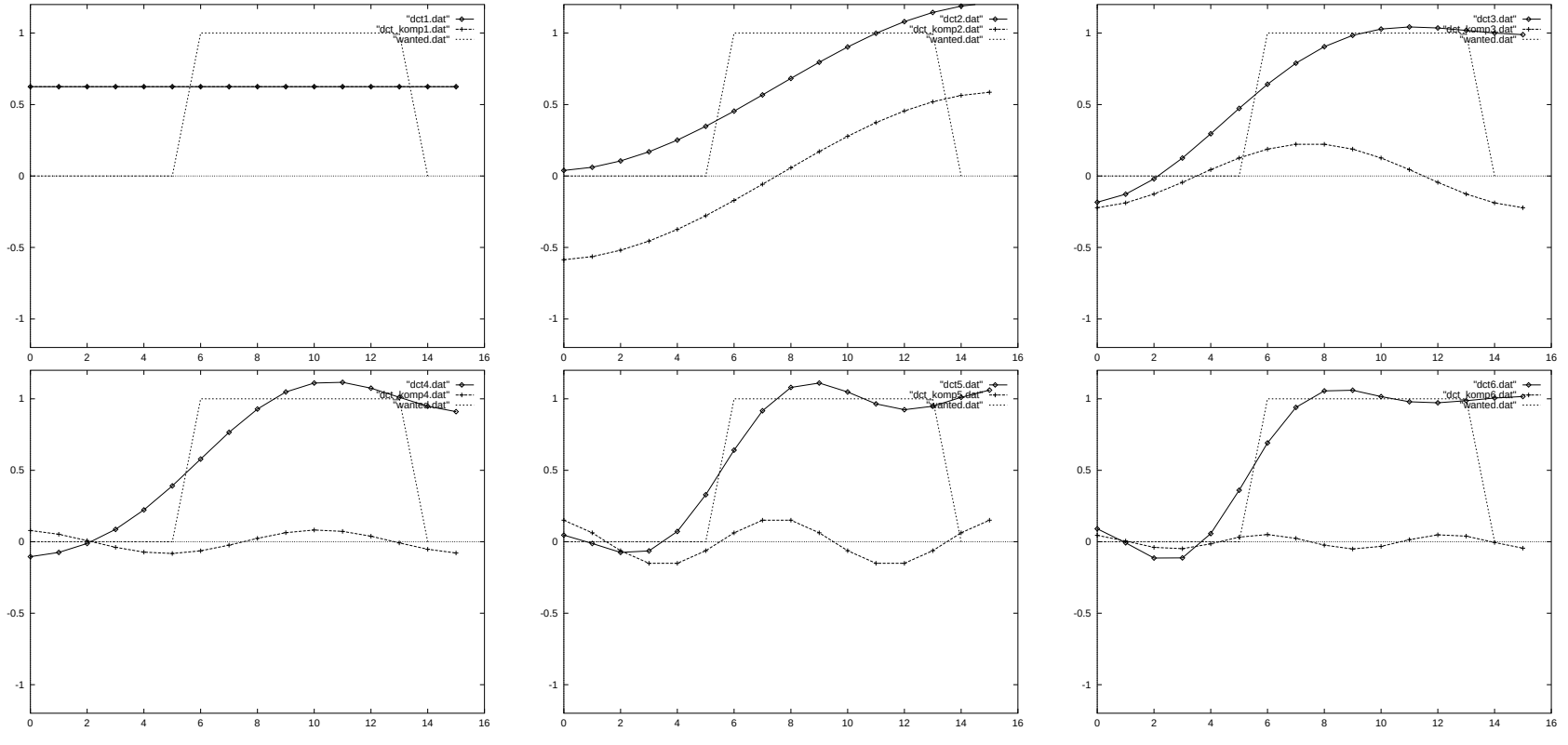
  $$c_i = A_i \sum_{t=0}^{n-1} f(t) \cos \frac{\pi(2t+1)i}{2n}$$

  wobei

  $$A_0 = \sqrt{\frac{1}{n}}, \quad A_i = \sqrt{\frac{2}{n}} \quad i \geq 1\,.$$

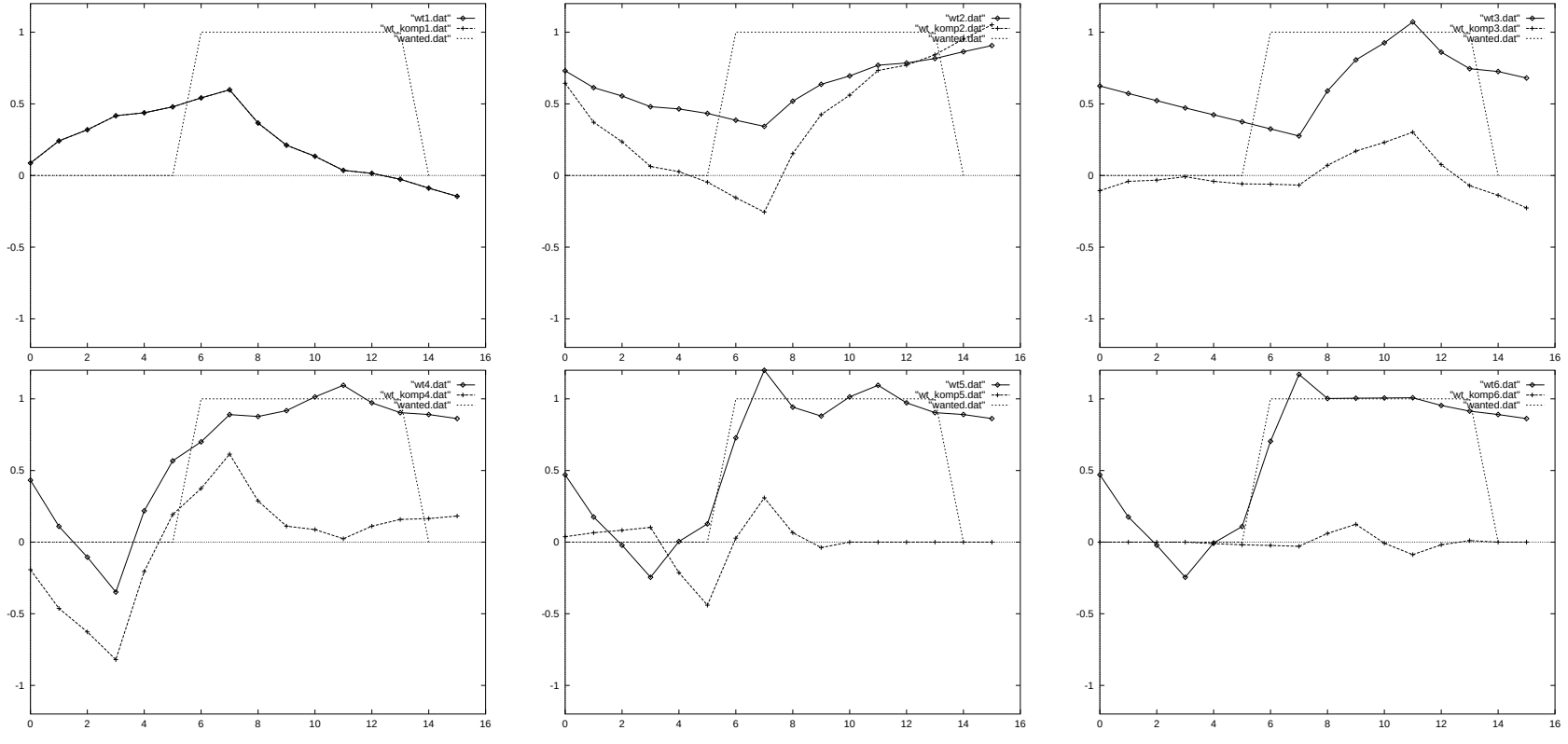
- **DWT** – Diskrete Wavelet-Transformation (Komplexität $O(n)$)
  Hier sind die Basisfunktionen nicht explizit gegeben, sondern durch den Transformationsalgorithmus gegeben.

DCT Veranschaulichung



Die ersten sechs Koeffizienten der DCT und ihr Einfluss auf die rekonstruierte Funktion.

WT Veranschaulichung



Die ersten sechs Koeffizienten der DWT und ihr Einfluss auf die rekonstruierte Funktion. Man beachte, dass vor allem die letzteren (höherfrequenten) Basisfunktionen (*Wavelets*) sich nicht über den ganzen Bereich erstrecken, sondern auch horizontal kleiner werden.

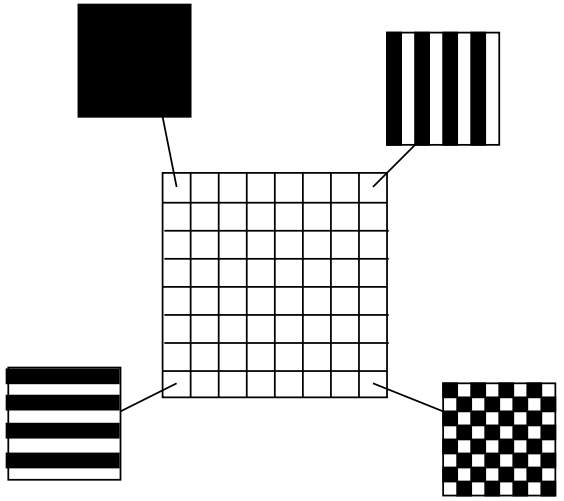
ad 1. Transformation der Bilddaten (3)

Ziel der Transformation ist es also, zu erreichen, dass die Größe der Koeffizienten mit dem Index abnimmt ($|c_1| \geq |c_2| \geq |c_3| \geq |c_4| \geq \cdots$).

Dies ist bei natürlichen Bildern meistens annähernd erfüllt.

Eine zweidimensionale Transformation wird meist durch aufeinanderfolgende horizontale und vertikale 1-D Transformationen durchgeführt.

Dadurch ergeben sich für die 2-D DCT eines 8×8-Blocks folgende 2-D Basisfunktionen:



ad 1. Transformation der Bilddaten – Beispiel

Einfaches Beispiel:

$b_1 = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$
$b_2 = (\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$
$b_3 = (\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2}, \frac{1}{2})$
$b_4 = (\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, -\frac{1}{2})$

Überprüfe, ob diese Basis orthonormal ist

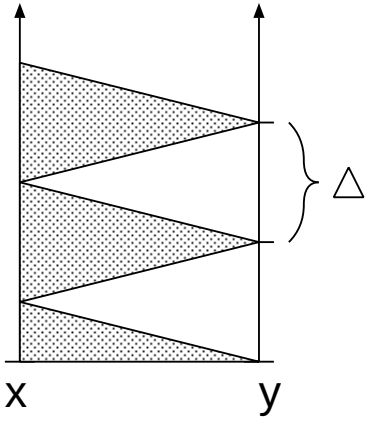
$f = (7, 5, 1, 3)$

$\Rightarrow c_1 = \langle (7, 5, 1, 3) | (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}) \rangle = \frac{7}{2} + \frac{5}{2} + \frac{1}{2} + \frac{3}{2} = \frac{16}{2} = 8$

weiters: $c_2 = 4, c_3 = 2, c4 = 0.$

daher: $c = \mathrm{Transform}(f) = (8, 4, 2, 0)$

ad 2. Quantisierung



Der Wertebereich der Bilddaten wird auf die Werte $0, \Delta, 2\Delta, 3\Delta, \ldots$ reduziert durch:

$$y = \mathrm{Quant}(x) = \mathrm{round}(\frac{x}{\Delta}) = \mathrm{round}(qx)$$

Dabei ist Δ das Quantisierungsdelta (Quantisierungsschrittweite), q heißt Quantisierungsfaktor. Je größer Δ (je kleiner q) desto größer der Informationsverlust.

Umkehrung: $\mathrm{Dequant}(y) = \Delta y = \frac{y}{q}$
Residuum (Residual):
$f(t) - \mathrm{Dequant}(\mathrm{Quant}(f(t))) \in (-\frac{\Delta}{2}, +\frac{\Delta}{2}]$
$\mathrm{MSE}(f,g) = \frac{1}{n}\sum_i (f(i) - g(i))^2$
$\mathrm{PSNR}(f,g) = 20 \log_{10} \frac{\mathrm{peak}}{\sqrt{\mathrm{MSE}(f,g)}}$
Hier ist peak der maximal mögliche Wert von $f(i)$. Daher auch PSNR = peak signal to noise ratio.

ad 2. Quantisierung – Beispiel

von vorher: zu quantisieren: $(8, 4, 2, 0)$

$\Delta = 3$

$\mathrm{Quant}(8, 4, 2, 0) = \mathrm{round}(\frac{8}{3}, \frac{4}{3}, \frac{2}{3}, 0) = \mathrm{round}(2.66, 1.33, 0.66, 0) = (3, 1, 1, 0)$

$\mathrm{Dequant}(3, 1, 1, 0) = (3 \cdot 3, 3 \cdot 1, 3 \cdot 1, 3 \cdot 0) = (9, 3, 3, 0)$

Residuum: $(8 - 9, 4 - 3, 2 - 3, 0 - 0) = (-1, 1, -1, 0)$

$\mathrm{MSE} = \frac{1}{4}((-1)^2 + 1^2 + (-1)^2 + 0^2) = \frac{3}{4}$

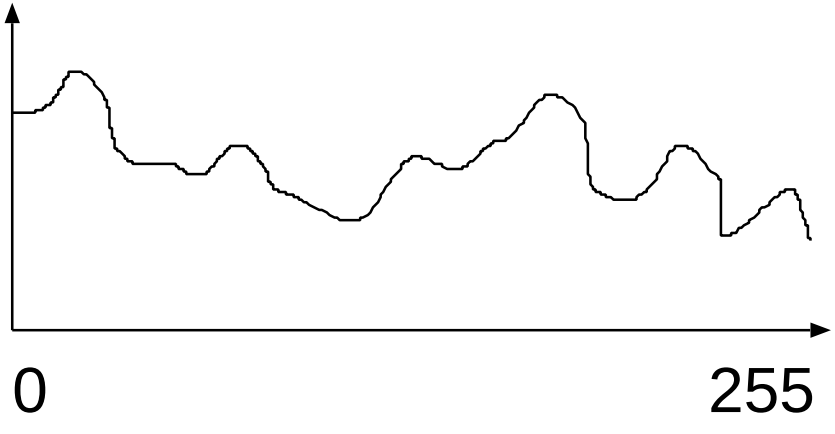
$\mathrm{PSNR} = 20 \log_{10} \frac{9}{\sqrt{\frac{3}{4}}} = 20.3$

Bei Bildern sind PSNR-Werte von 25 (schlecht) bis 40 (gut) üblich.

ad 3. Entropiekodierung

Durch die Transformation wird Information konzentriert. Das bedeutet für die Verteilung der transformierten (und quantisierten) Werte, dass kleine Werte häufiger sind:

Histogramm:



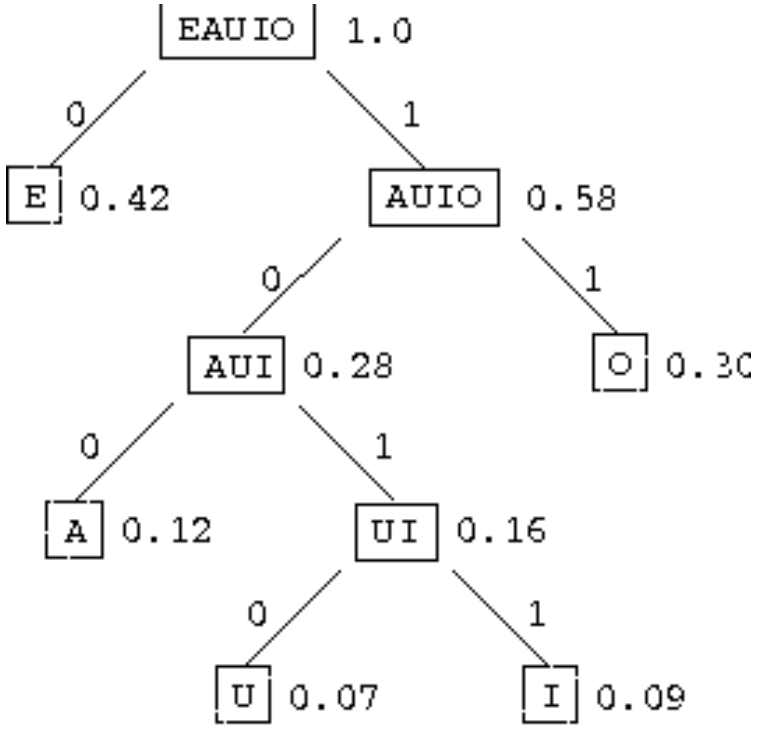
vor der Transformation

nach der Transformation

Dies kann man ausnutzen, indem häufigere Werte mit kürzerem Code versehen werden.

ad 3. Entropiekodierung – Huffman-Algorithmus

Jedem Symbol wird ein Code (Folge aus bits) zugeordnet, so dass kein Code ein Prefix eines anderen ist. Dies wird erreicht durch einen binären Baum mit den Symbolen an den Blättern. Verzweigungen nach links mit 0, nach rechts mit 1 kodieren. Unwahrscheinliche Symbole werden weiter vom Urpsrung entfernt angeordnet.
Die (bitweise) in einen **Bitstream** hintereinandergeschriebenen Codes können dann eindeutig aus diesem wieder rekonstruiert werden.
Nachteil: Bei z.B. nur zwei Symbolen muss man von einer 50:50 Wahrscheinlichkeitsverteilung ausgehen.



ad 3. Entropiekodierung – Beispiel

von vorher: zu kodieren: $(3, 1, 1, 0)$

Zuerst: straight forward

Symbol	Code
0	00
1	01
2	10
3	11

$\Rightarrow \text{code}(3, 1, 1, 0) = 11010100$ (8 bit)

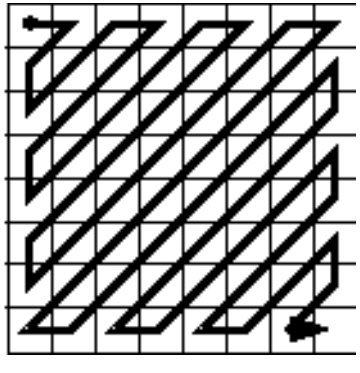
Jetzt mit Huffman-Code:

Symbol	Code
0	10
1	0
2	-
3	11

$\Rightarrow \text{code}(3, 1, 1, 0) = 110010$ (6 bit)

JPEG – wie es funktioniert

- Das Bild (genauer: jede Farb-Komponente) wird in 8×8-Pixel-Blöcke unterteilt. Diese werden DCT-transformiert. Der Grund dafür ist u.A., dass dadurch die Komplexität der DCT künstlich auf $O(n)$ reduziert wird.
- Es gibt einen eigenen Quantisierungsfaktor für jeden der 64 Koeffizienten (wählbar).
  $\Rightarrow 8 \times 8$ Quantisierungs-"Matrix".
- Die Koeffizienten eines Blocks werden in der Reihenfolge des Zigzag-Scan kodiert.



- Kodierung mittels Huffman-Codes
- Längere Folgen von 0-Werten in diesem Scan werden mit eigenen Symbolen (Codes) kodiert. Auch für Blockende (alle restlichen Koeffizienten 0) gibt es ein eigenes Symbol.
- DC-Koeffizienten (der ganz links oben) werden als Differenz zum vorigen Block kodiert.

JPEG2000 – neu, besser, komplizierter

- Verwendet die Wavelet-Transformation (statt DCT)
- Sukzessiv verfeinerte Quantisierung durch Bitplane-Coding
- Bittiefe auf Blockbasis steuerbar
- Optimierungsverfahren zum Ermitteln der besten Bittiefe für jeden Block (rate-distortion-optimization)
- Arithmetisches Kodieren statt Huffman-Codes
- Bitstrom für jeden Block wird in Packets unterteilt
- Über die Anordnung der Packets kann bestimmt werden, ob zuerst eine geringere Auflösung übertragen werden soll (resolution pregressive) oder zuerst schlechtere Qualität (layer progressive) usw. (component progressive, . . .)