

Kompresjon og koding II

Kap. 7.4-7.7 og 8.10 (4th Ed.), Kompendium: 18.7.3 og 18.8-18.9, og QOI spesifikasjonen

Aasan og Tan

November 8, 2023

Table of contents

1 **Basistransformer**

2 **Prediktiv koding og LZ-transformer**

3 **Kompresjonsalgoritmer i Praksis**

Basistransformer

Cosinustransform I

Fra forelesningene om Fouriertransform:

- Signaler kan dekomponeres i sinusoidal basis.
- cos: symmetrisk; sin: antisymmetrisk.
- Euler: $e^{j\pi} = \cos\pi + j\sin\pi \Rightarrow$ kompleks verdimengde $\mathbb{C}$.
- Ofte glissen representasjon for naturlige bilder.
- Antar periodiske randverdier.
- Fourier basis er **seperabelt** for to (eller fler) dimensjoner;

$$\tilde{X} = \mathcal{F}X\mathcal{F}^{\mathsf{T}}$$

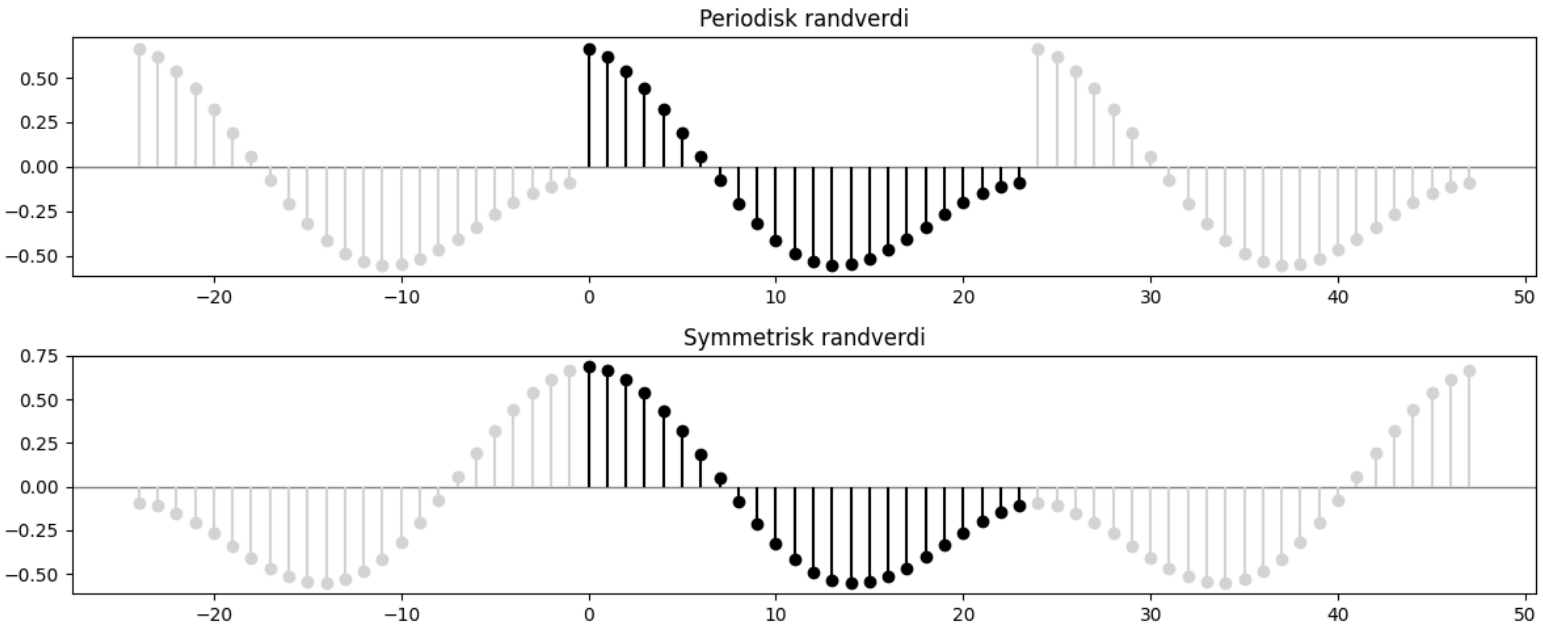
For reelle signaler $f : [0, 1]^d \to \mathbb{R}$ vil mange av koeffesientene i DFT være like. Med reell DFT (`rfft`) fjerner vi alle disse symmetriske koeffesientene i utregningene.

Spørsmål: Er koeffisienter i $\mathbb{C}$ optimalt for kompresjon?

Cosinustransform II

Idé: Bruk kun cos koeffisienter!

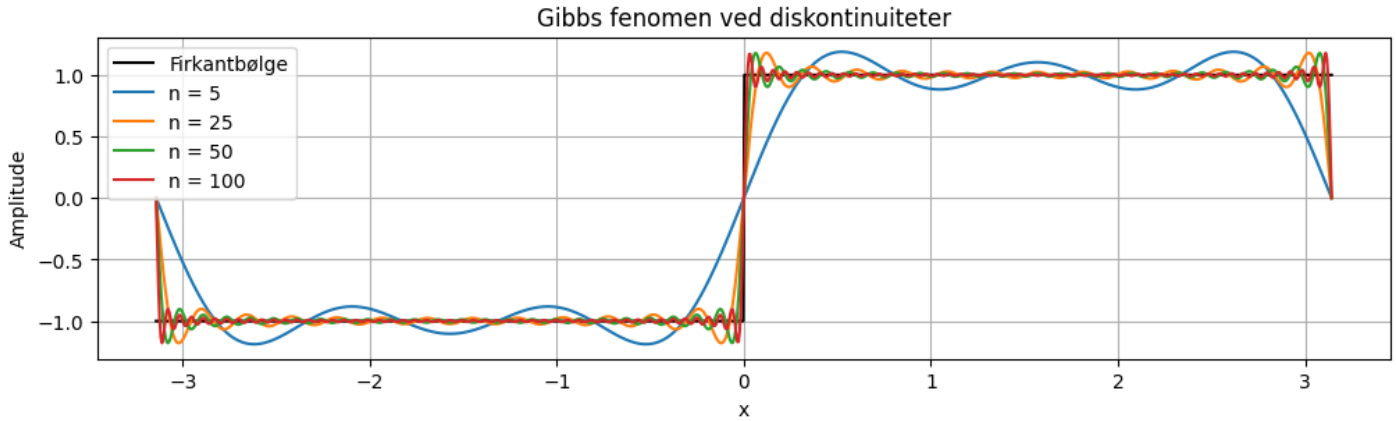
- Ignorér komplekse antisymmetriske komponenter fra sin.
- Antar symmetriske randverdier og konstant fase.



Spørsmål: Hvilken effekt har dette for dekomposisjonen?

Cosinustransform III

Gibbs fenomen: Mange artifakter i høyfrekvenskomponenter som sprer energiinnholdet i dekomposisjonen, og gir **mindre kompakt energispredning.**



Periodisk



Symmetrisk



Cosinustransform IV

Finnes flere varianter av DCT. Mest vanlig (DCT-II):

$$\tilde{x}_k = \sum_{n=0}^{N-1} x_n \cos\left(k\pi(n+1/2)/N\right), \tag{1}$$

for $k = 0\ldots, N-1$. Kan gjøres effektivt med FFT (Makhoul, 1980):

1. Uvid signalet x med full refleksjonsrandutvidelse:

$$z_n = \begin{cases} x_n, & \text{for } 0 \leq n < N; \\ x_{2n-1-n}, & \text{for } N \leq n < 2N. \end{cases}$$

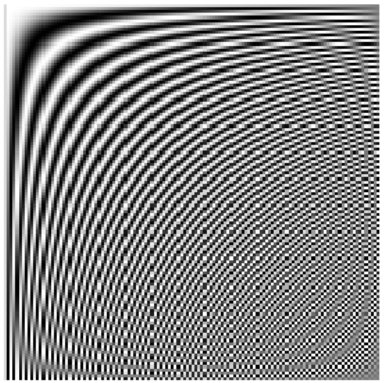
2. Ta FFT: $\tilde{z}_k = (Fz)_k$ for $0 \leq k < N$.
3. Ta realdelen av faseskift: $\tilde{x}_k = \mathfrak{Re}\left(\tilde{z}_k \cdot \exp\left(-\frac{j\pi k}{2n}\right)\right)$.

Får kompleksitet $\mathcal{O}(n \log n)$ med lik kompleksitet for inverstransform.
Merk: Standardform for DCT-II er ikke normalisert til å bli ortonormal.

Cosinustransform V

Som DFT er DCT også seperabel.

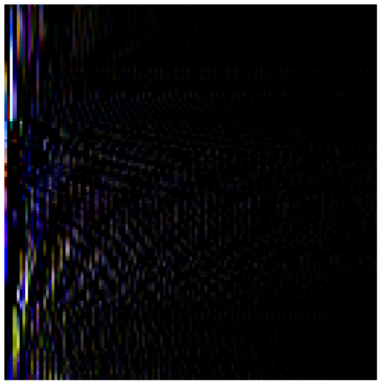
Basismatrise DCT-II: C



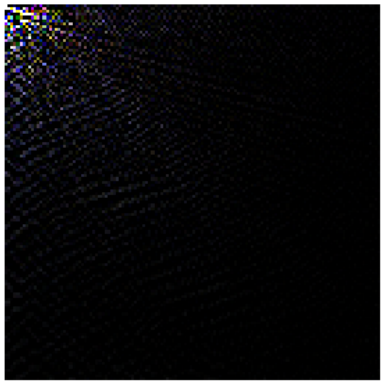
Originalbilde: X



Radvis DCT-II: XC^{T}



Full DCT-II: CXC^{T}



- Koeffisientene er veldig energikompakte, og gir en svært glissen representasjon med mange verdier nær null.
- For DCT trengs ikke skift av frekvensdomenet til senter, siden vi ikke regner med de andre kvadrantene.
- For naturlige bilder vil koeffisientene ha høyest amplitude nær origo (øverst til venstre).

Cosinustransform VI

Oppsummering DCT:

- gir kun reelle koeffisienter,
- antar symmetrisk randutvidelse av signaler og bilder,
- har ofte bedre energikompakthet, gir mer glisne koeffesienter,
- har ingen fasekomponent, siden vi antar symmetri.
- får $\mathcal{O}(n \log n)$ kompleksitet via FFT.

Hvis DCT er så flott, hvorfor ikke bruke DCT for filtrering?

- glissenhet er ofte ikke spesielt nyttig for filterdesign,
- mindre presis, mangler faseinformasjon,
- reell DFT krever i praksis færre utregninger.

DCT har mange fordeler i kompresjon og representasjon, som ikke nødvendigvis er overførbart til filtrering.

Diskrete Transformer I

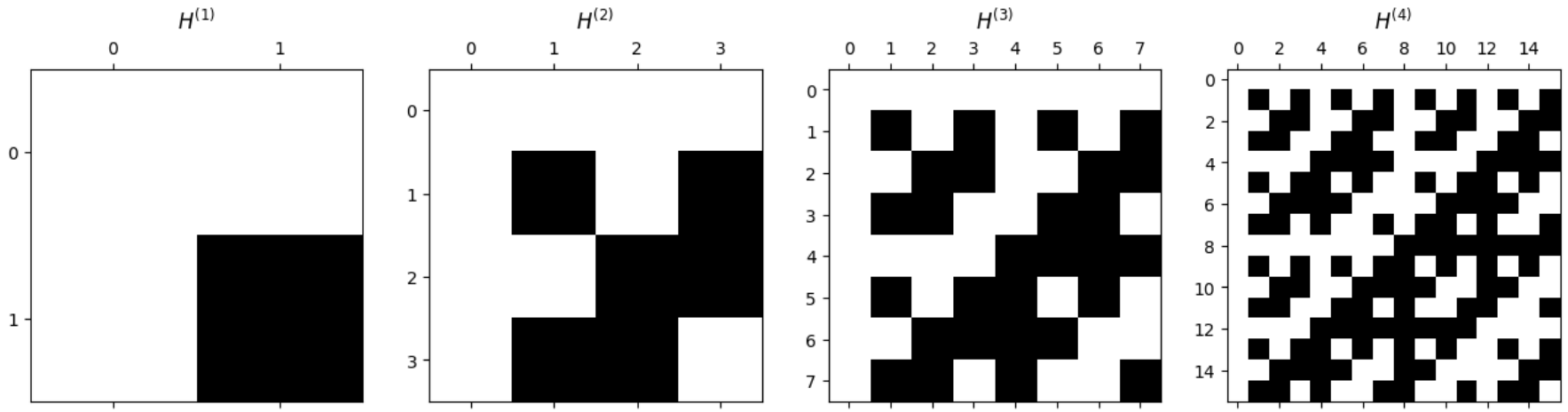
Fra Gibbs-fenomen: Hvor naturlig er det å dekomponere digitale bilder med kontinuerlige bølger?

Idé: Dekomponer bilder ved bruk av stegfunksjoner eller firkantbølger.

La $W^{(0)} = 1$. Familien med blokkmatriser på formen

$$W^{(2n)} = \begin{bmatrix} W^{(n)} & W^{(n)} \\ W^{(n)} & -W^{(n)} \end{bmatrix} \tag{2}$$

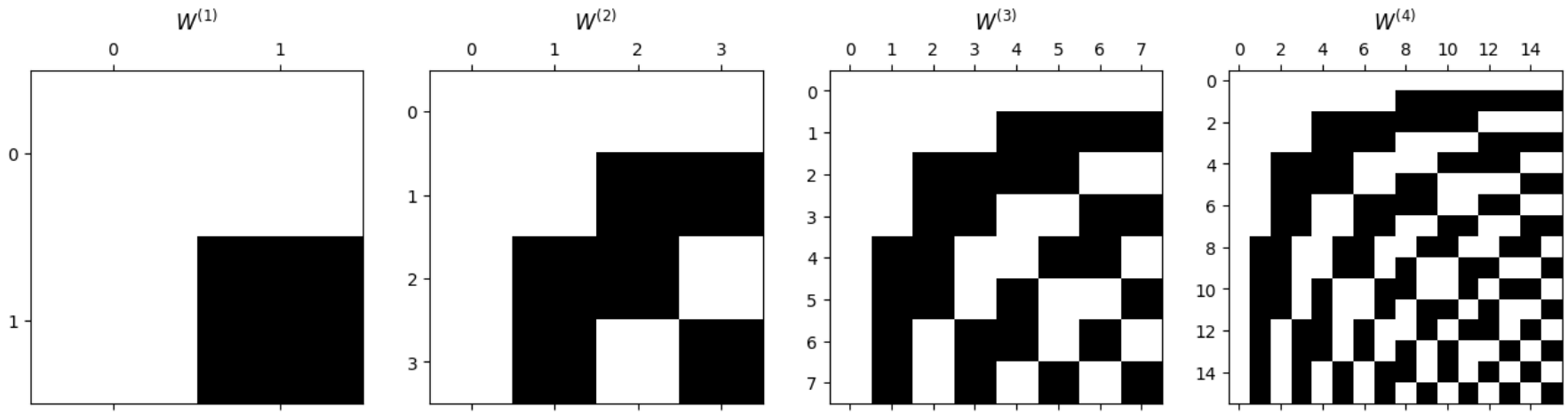
kalles **Walsh-Hadamard transform** på Hadamard form.



Diskrete Transformer II

Egenskaper til Walsh-Hadamard transformen (WHT);

- Fourier for Boolske grupper, lik DCT når kvantisert til $\{-1, 1\}$.
- Sentral brikke som en av de universale logiske portene (quantum logic gates) for kvantedatamaskiner.
- Bruker kun addisjon og subtraksjon, ingen multiplikasjoner.
- $W^{(n)}$ er symmetrisk og ortogonal $\Rightarrow (W^{(n)})^2 = I^{(n)}$ (involusjon)



Figur: Ved å sortere rader på antall nullkryssninger, får vi Walsh formen.

Diskrete Transformer III

Globale transformasjoner (DFT / DCT / WHT) dekomponerer ved hjelp av basisfunksjoner som opererer over hele bildet samtidig.

Begrensninger:

- De gir ikke lokal informasjon om frekvensinnhold – vi vet ikke nøyaktig hvor i bildet frekvenskomponentene oppstår.
- De er ikke optimalisert for å håndtere anisotropiske (retningssensitive) eller ikke-stasjonære egenskaper i bildedata, som kan variere over ulike områder og skalaer.
- Det er vanskelig å tilpasse dem til å fange opp lokale særegenheter, som skarpe kanter eller lokale teksturmønstre.

Problemstilling: Hvordan konstruere en transformasjon som er mer dynamisk og kan tilpasses bildeinformasjon på flere skalaer og detaljnivåer?

Diskrete Transformer IV

Wavelet-transformasjoner dekomponerer bilder på flere oppløsningsnivåer, og er spesielt nyttige for skarpe kanter og lokale teksturer. Et komplett basissett genereres ved dilatasjon (skalering) og translasjon av en waveletfunksjon.

Haar Wavelet-funksjonen er den enkleste mulige wavelet-funksjonen og er definert ved

$$\Psi(t) = \begin{cases} 1 & \text{for } 0 \leq t < \frac{1}{2}, \\ -1 & \text{for } \frac{1}{2} \leq t < 1, \\ 0 & \text{ellers.} \end{cases}$$

Den tilhørende skaleringsfunksjonen er gitt ved

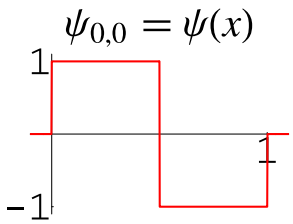
$$\Phi(t) = \begin{cases} 1 & \text{for } 0 \leq t < 1, \\ 0 & \text{ellers.} \end{cases}$$

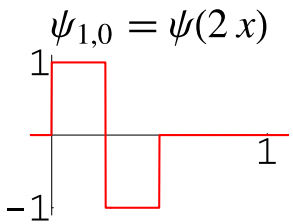
Diskrete Transformer V

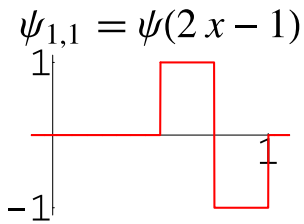
Haar-wavelet transformen dekomponerer signaler med basisen

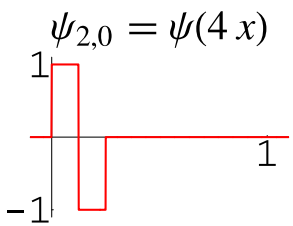
$$\psi_{j,k}(t) = \psi(2^j t - k) \tag{3}$$

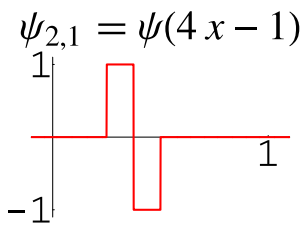
der n indikerer dilatasjon og k indikerer translasjon.

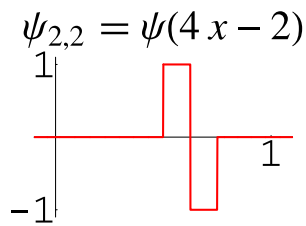


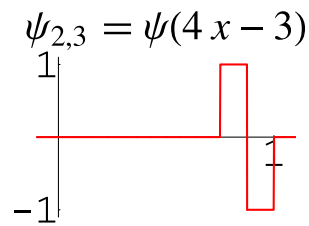












Diskrete Transformer VI

Eksempel

For et signal med lengde $n = 2^2 = 4$ har vi skaleringsfunksjon

$$\varphi(t) = [1, 1, 1, 1]$$

De resterende radene representerer waveletfunksjonen $\psi(t)$ ved forskjellige skalaer og posisjoner, og gir detaljnivået i dekomposisjonen:

$$\psi_{0,0}(t) = [1, 1, -1, -1]$$

$$\psi_{1,0}(t) = [1, -1, 0, 0]$$

$$\psi_{1,1}(t) = [0, 0, 1, -1]$$

Så Haar matrisen H_4 kan skrives som:

$$H^{(4)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

Merk: Som med DCT er standardform her ikke ortonormal.

Diskrete Transformer VII

Som Walsh-Hademard kan Haar matrisene defineres rekursivt ved

$$H^{(2n)} = \begin{bmatrix} H^{(n)} \otimes [1,1] \\ I^{(n)} \otimes [1,-1] \end{bmatrix} \tag{4}$$

der $I^{(n)}$ er $n \times n$ identitetsmatrisen og $\otimes$ er Kronecker-produktet, gitt ved

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix} \tag{5}$$

Intuisjon: Haar-wavelet transformen dekomponerer signaler hierarkisk ved hjelp av enkle lavpass ($[1,1]$) og høypass ($[1,-1]$) filtre.

Akkurat som DFT / DCT / WHT er DWT også separabelt.

DWT er mye brukt i **videokompresjon**.

Diskrete Transformer VIII

- Beregningslette transformer som gir glisne representasjoner.
- Bruker firkantbølger og stegfunksjoner, diskret natur.
- Walsh-Hademard Transform (WHT):
  - Kan sees på som Fourier transform over $(\mathbb{Z}/2\mathbb{Z})^n$.
  - Sentral i kvanteberegninger.
- Diskret Wavelet transform (DWT):
  - **Haar-wavelet**: Enkleste DWT, velegnet til digitale signaler.
  - Dekomponerer signal i flere oppløsningsnivåer, grunnlaget til *multiresolusjonsanalyse*.
  - Bruker en wavelet funksjon ψ og en skaleringsfunksjon φ.
  - Ofte brukt i videokompresjon pga. lokaliseringsevne.
- WHT og Haar-DWT er definert rekursivt ved

$$W^{(2n)} = \begin{bmatrix} W^{(n)} & W^{(n)} \\ W^{(n)} & -W^{(n)} \end{bmatrix}; \quad H^{(2n)} = \begin{bmatrix} H^{(n)} \otimes [1,1] \\ I^{(n)} \otimes [1,-1] \end{bmatrix}.$$

Merk: Generell wavelet transform har flere bruksområder, og er også mye brukt i filterdesign.

Prediktiv koding og LZ-transformer

Prediktiv Koding I

Differansetransform: *Utnytte kontinuitet; nabopiksler er ofte like.*
Radvis: $\delta(1, y) = f(1, y), \delta(x + 1, y) = f(x + 1, y) - f(x, y)$.

Alternativt: "Predikerer" neste piksel er lik forrige piksel, og koder feilen.
Vi kan også lage mer avanserte modeller for prediksjon.

Definisjon (Autoregressiv Modell)

En p-ordens autoregressiv modell $\mathrm{AR}(p)$ er gitt ved

$$x^{(t)} = \beta^{(0)} + \beta^{(1)} x^{(t-1)} + \beta^{(2)} x^{(t-2)} + \cdots + \beta^{(p)} x^{(t-p)} + \delta^{(t)}, \quad (6)$$

der $x^{(t)}$ er piksel i nåværende posisjon, $x^{(t-1)}, \ldots, x^{(t-p)}$ er tidligere pikselverdier, $\beta^{(1)}, \ldots, \beta^{(p)}$ er vekter for modellen, $\beta^{(0)}$ er et konstantledd, og $\delta^{(t)}$ er et støyledd som representerer prediksjonsfeilen.

Prediktiv Koding II

La oss definere en $\mathrm{AR}(1)$ modell med $\beta_0 = 0$ og $\beta_1 = 1$.
Dette gir

$$x^{(t)} = \beta^{(1)} x^{(t-1)} + \delta^{(t)} = x^{(t-1)} + \delta^{(t)}$$
$$\delta^{(t)} = x^{(t)} - x^{(t-1)}$$

Innsikt: Differensetransform er en enkel $\mathrm{AR}(1)$ modell.

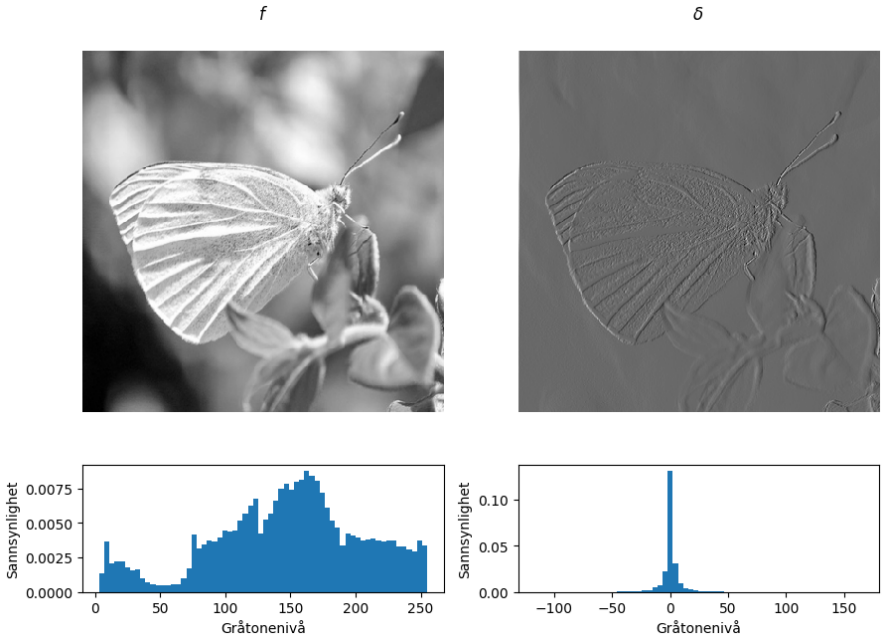
AR modeller kan sees på som **kausal konvolusjon**, der vi ser på et naboskap rundt pikselen rundt den vi ønsker å predikere, og regner ut vekter som gjør at prediksjonsfeilen $\delta^{(t)}$ blir så liten som mulig.

Utregningen av vektene gjøres gjerne med minste kvadraters metode eller andre optimeringsmetoder.

Svært avhengig av **skannerekkefølge** samt hvilke tidligere verdier vi skal velge å inkludere. Trenger ikke nødvendigvis velge forgjengere sekvensielt!

Prediktiv Koding III

Prediktiv koding og differansekoding har som resultat at mange verdier får veldig liten magnitude - ofte Laplace fordeling.
Har **a priori** kunnskap om sannsynlighetene til prediksjonsfeilene.



Spørsmål: Finnes det kodingsmodeller som utnytter dette?

Golomb-koding: Anta verdier med eksponensielt avtakende frekvens.
Ønsker: Små feil $\Rightarrow$ liten bitlengde b og store feil $\Rightarrow$ lengre bitlengde.

Prediktiv Koding IV

Golomb-koding:

1. Velg m som best representerer frekvenser i prediksjonsfeil.
2. For en gitt verdi x, del x med m for å finne kvotienten q og resten r.
3. Kod kvotienten q som en unær kode, det vil si q enere etterfulgt av en null.
4. Kod resten r enten som en fast lengde kode eller en optimal prefikskode, avhengig av m.

Matematisk Representasjon:

$$\text{Kodeord for } x = (\underbrace{1, 1, \ldots, 1}_{q \text{ ganger}}, 0) \oplus \text{Kode}(r)$$

Eksempel: Hvis $m = 3$ og verdien $x = 10$, får vi $q = 3$ (kvotient) og $r = 1$ (rest). Kodeordet blir dermed $1110 \oplus \text{Kode}(1)$.

Prediktiv Koding V

Oppsummering:

- **Prediktiv koding**: utnytte interpikselredudansen i feilene, som ofte har en mer kompakt og komprimerbar representasjon.
  - Autoregressiv modell: lineær prediksjon, få parametere.
  - Kan bruke mer avansert modell, men koster gjerne mer.
  - Avhengig av skannerekkefølge og definisjon av kausalt naboskap.
  - Kan både gjøres tapsfri eller ikke-tapsfri.
- **Golomb-koding**: effektivt når verdiene følger geometrisk fordeling, eksponensielt avtakende.
  - Velegnet for kombinasjon med prediktiv koding.
  - Kan også brukes for løpelendger. Mange små løp gir ofte geometrisk fordeling.
  - For Laplace trenger vi ekstra bit for fortegn $\pm$.
  - Enkel utregning, rask kompresjon og dekompresjon.
  - Til sammenlikning: Huffmann / Aritmetisk er optimal med få eller ingen antakelser om fordeling.

Fra Prediksjon til Ordbok

I prediktiv koding forsøker vi å predikere nye verdier basert på tidligere observerte verdier.

Noen begrensninger:

- Predikerer stort sett én verdi av gangen.
- Standard AR: Statisk prediksjon, antar stasjonæritet.
- Avhengig av verdibasert prediksjon, ikke symbolbasert.
  - Inkompatibel med tekst (uten vesentlige modifikasjoner).

Idé: Prediker med **oppslag** som prediktivt verktøy.

Istedenfor å bruke en statisk modell for prediksjon, kan vi benytte dynamiske ordbøker som oppslagsverk for å koble tilbake til hele sekvenser som allerede har forekommet i dataene.

*Algoritmer som bruker dette kalles ofte **Lempel-Ziv (LZ)** transformer.*

Lempel-Ziv-Welch (LZW) transform

Utnytter mønstre i meldingen.

- Ser på samforekomster av symboler
- Lar symbolsekvenser få egne koder
- Reduserer hovedsakelig intersample/interpiksel-redundans

Sender og mottaker bygger opp en liste av koder for symbolsekvenser

- ... dette gjør de separat. Siden listen kan bygges opp unikt trenger man ikke sende eller lagre den.
- Trenger kun et standard alfabet for symbolene

Regel for å bygge opp listen:

- Sender: Ny streng = siste sendte streng + neste usendte symbol
- Mottaker: Ny streng = nest siste streng + første symbol i sist tilsendte streng

LZW transform er best vist med et eksempel!

LZW transform eksempel I

Anta vi har et standard alfabet $\mathcal{A} = \{a, b, c\}$ med kodene 0,1,2 respektivt.
Melding vi skal sende: ababcbababaaaaabab

Vi går gjennom meldingen slik:

Ser	Sender	Senders liste	Mottar	Tolker	Mottakers liste
		a=0, b=1, c=1			a=0, b=1, c=1
a	0	ab=3	0	a	
b	1	ba=4	1	b	ab=3
ab	3	abc = 5	3	ab	ba=4
c	2	cb=6	2	c	abc=5
ba	4	bab=7	4	ba	cb=6
bab	7	baba=8	7		

Hva skal mottaker tolke når vi mottar kode 7? Denne er ikke i listen!
Vet at 7 = ba + ? i følge mottakers regel.
Siden sist tilsendte streng er 7 må ?=b. Derfor er 7 = bab.

LZW transform eksempel II

Fortsettelse av meldingen ababcbababaaaaabab:

Ser	Sender	Senders liste	Mottar	Tolker	Mottakers liste
		a=0, b=1, c=1			a=0, b=1, c=1
a	0	ab=3	0	a	
b	1	ba=4	1	b	ab=3
ab	3	abc = 5	3	ab	ba=4
c	2	cb=6	2	c	abc=5
ba	4	bab=7	4	ba	cb=6
bab	7	baba=8	7	bab	bab=7
a	0	aa=9	0	a	baba=8
aa	9	aaa=10	9	aa	aa=9
aa	9	aab=11	9	aa	aaa=10
bab	7		7	bab	aab=1

Møter samme problem når mottaker får koden 9 for første gang her.
Siden kode 9 er siste tilsendte streng må det siste symbolet i kode 9 være a.
Ved mottakers regel har vi at 9 = aa.

I stedet for å sende 18 symboler har vi sendt 10 koder.
5 av 12 koder som ble laget ble ikke brukt.

LZW transform oppsummering

- LZW kodene er som regel gitt i naturlig binærkode
- Komprimerer tekstfiler med en faktor på ca. 2
- **LZW-transform er mye brukt:**
  - Brukt i Unix-kommandoen `compress` fra 1984
  - Brukt i GIF-formatet fra 1987
  - Kan også bli brukt i TIFF- of PDF-formatet
- **LZW-kodene kan kodes videre** e.g. med Huffmann-koding
- Man kan redusere antall mulige koder ved å begrense listen:
  - Sett maks antall koder til f.eks. 2^{b+1}. Etter dette lagres ikke flere koder.
  - Ha en prodedyre for å slette koder som brukes sjeldent. Prosedyrene må være **faste** slik at mottaker sletter samme koder som sender.

Kompresjonsalgoritmer i Praksis

Quite OK Image Format (QOI) I

Ny tapsfri bildekompresjonsmetode fra 2021 (Dominic Szablewski).
Brukes på RBG/RGBA-bilder. Vi antar RGB bilde i denne forelesningen.

Bruker i hovedsak 4 forskjellige metoder for å kode pikslene:

- Løpelengde-kode (hvis forrige pikselverdi repeterer seg)
- Differansen med forrige pikselverdi (hvis forskjellen er liten)
- Indeks til en tabell over tidligere observerte piksler – en form for LZ-transform.
- RGB-verdiene av pikselen (dersom ingen av de øvre metodene passet)

Bildet blir skannet radvis i standard rekkefølge (ventre til høyre, topp til bunn).
Hver piksel blir kun behandlet én gang.

- QOI sammenligner nåværende piksel og forrige piksel.
- I tillegg lagres et fargeindeks-array for tidligere observerte pikselverdier. Arrayet har en fast lengde, som gir en *buffer* på hvor langt tilbake man leter etter samme pikselverdi. Indeksposisjonen blir regnet med en hashfunksjon basert på RGB verdiene: indeks $= (r \cdot 3 + g \cdot 5 + b \cdot 7)$ % `array_lengde`

Quite OK Image Format (QOI) II

QOI regel 1: Hvis nåværende piksel har samme RGB verdier som forrige piksel: Finn løpelengden (antall repetisjoner av pikselverdien). Løpelengden lagres med 8 biter = 1 byte.

QOI_OP_RUN

Byte[0]							
7	6	5	4	3	2	1	0
1	1	run					

De 2 første bitene er reservert for en tag.
Løpelengden blir lagret med 6 biter.
Maks løpelengde: $2^6 = 64$

QOI regel 2: Hvis pikselverdien matcher med en verdi i fargeindeks-arraet over tidligere observerte pikselverdier. Indeksen blir lagret med 8 biter.

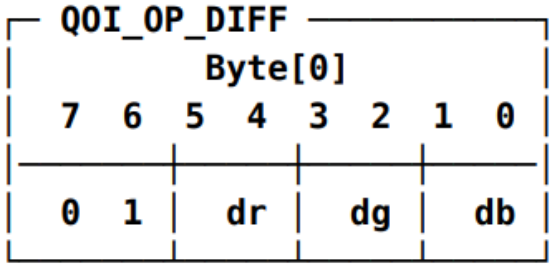
QOI_OP_INDEX

Byte[0]							
7	6	5	4	3	2	1	0
0	0	index					

De 2 første bitene er reservert for en tag.
indeks $= (r \cdot 3 + g \cdot 5 + b \cdot 7)\ \%\ 64$
Lengden på fargeindeks-arrayet er $2^6 = 64$ siden vi setter av 6 biter til å lagre indeksen.

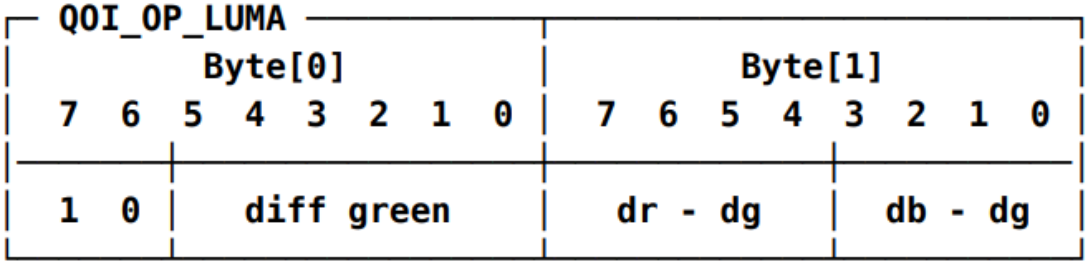
Quite OK Image Format (QOI) III

QOI regel 3: Hvis differansen av RGB-verdiene mellom nåværende piksel og forrige piksel er mellom -2 og 1. Differansen blir lagret med 8 biter.



De 2 første bitene er reservert for en tag.
Differansene `dr`, `dg`, `db` for hver RGB-verdi blir lagret med 2 biter hver.
Krever at $-2 \leq \texttt{dr} \leq 1$, $-2 \leq \texttt{dg} \leq 1$, og $-2 \leq \texttt{db} \leq 1$

QOI regel 4: Hvis differansen av RGB-verdiene mellom nåværende piksel og forrige piksel er tilstrekkelig små.



$-32 \leq \texttt{dg} \leq 31$
$-8 \leq \texttt{dr-dg} \leq 7$
$-8 \leq \texttt{db-dg} \leq 7$

Differansen blir lagret med 16 biter.
De 2 første bitene er reservert for en tag.
Differansen `dg` blir lagret med 6 biter. Differansene `dr-dg`, `db-dg` blir lagret med 4 biter hver.

Quite OK Image Format (QOI) IV

QOI regel 5: Hvis nåværende piksel ikke tilfredsstilte noen av de andre reglene, lagres hele pikselverdien med 16 biter

QOI_OP_RGB

Byte[0]								Byte[1]	Byte[2]	Byte[3]
7	6	5	4	3	2	1	0	7 .. 0	7 .. 0	7 .. 0
1	1	1	1	1	1	1	0	red	green	blue

De første 8 bitene er reservert for en tag.
Hver RGB-verdi blir lagret med 8 biter.

QOI oppsummering:

- 5 forkjellige koder er brukt for å lagre pikslene
- Hver kode blir identifisert med en tag
- Hver kode bruker enten 8 eller 16 biter $\rightarrow$ representert med *hele* bytes.

Portable Network Graphics (PNG) I

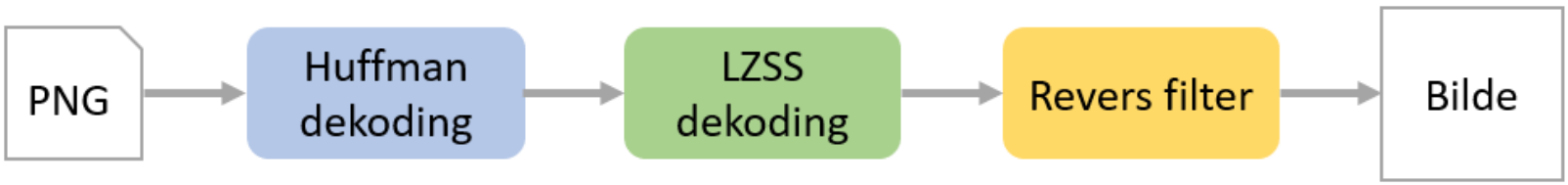
PNG-kompresjon kan deles deles i tre steg:



Filter: 5 forskjellige filtre: SUB, UP, AVG, PAETH, eller INGEN
Lempel–Ziv–Storer–Szymanski (LZSS): En Lempel-Ziv modell.
Huffman koding: En variant av Huffman koding utviklet for LZSS

Vi skal ikke gå dypt i detaljene i disse stegene. LZSS-varianten av Huffmann-koding er veldig komplisert og derfor ikke pensum.

Dekompresjon er å gjøre prosessen i revers:



Portable Network Graphics (PNG) II

PNG kort oppsummert:

- RGB-kanalene blir behandlet separat.
- For å velge hvilket filter som er best for hver rad i bildet regnes det ut en *score* for hvert filter. Dette krever at hver rad behandles 5 ganger, bare for å velge filter!
- LZSS bruker et “sliding window” til å identifisere tidligere observerte verdier, og koder disse med (indeks, løpelengde) dersom løpelengden er 3 eller mer.
- Fler heuristikker og filtre gjør PNG-komprimering ganske tregt!
- Dekomprimering tar kortere tid enn komprimering
- Huffman-kodingen i PNG er veldig innviklet

Oppsummering: Tapsfri kompresjon

- QOI behandler hver pikselverdi kun én gang, mens PNG behandler dem flere ganger.
- QOI behandler RGB-kanalene samlet, mens PNG behandler dem separat.
- QOI har $\sim$20-50 ganger raskere enkoding enn PNG
- QOI har $\sim$3-4 ganger raskere dekoding enn PNG
- Ca. samme kompresjonsrate; PNG litt bedre
- Spesifikasjonen til QOI er enkel (kun 1 side), mens spesifikasjonen til PNG er veldig lang. Sjekk https://qoiformat.org/qoi-specification.pdf og https://www.w3.org/TR/png/
- Detaljer rundt PNG er ikke pensum.

JPEG kompresjon I

Den mest brukte ikke-tapsfrie kompresjonsalgoritmen for bilder!

Utnytter flere redudanser (Interpiksel [I], Koding [K], Psykovisuell [P])

Steg for steg:

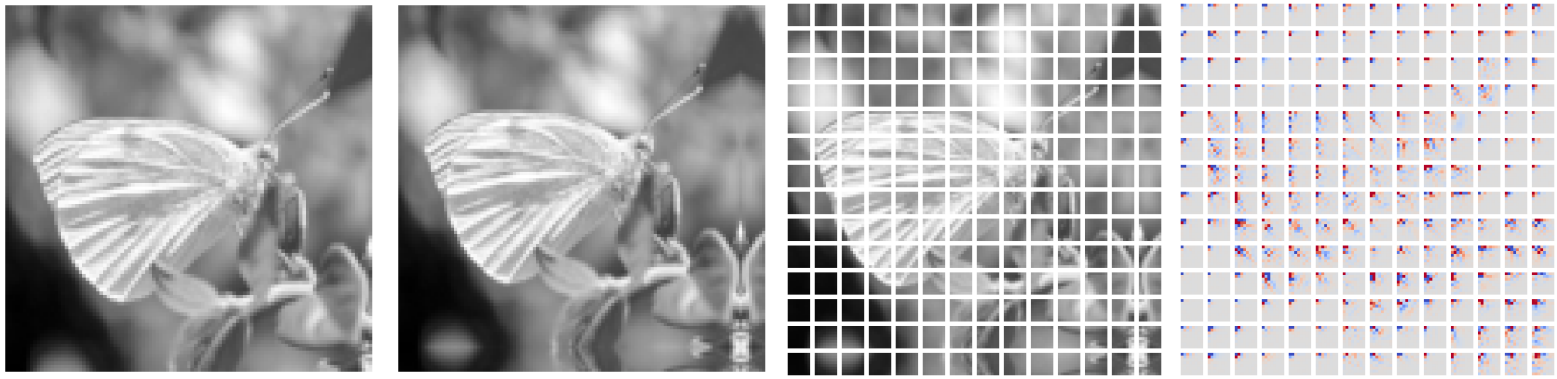
1. YCbCr Fargerom [K][P]
2. Kromatisk fargesubsampling [I][P]
3. Blokkvis Cosinustransform [I][P]
4. Koeffisientkvantisering [K]
5. Kontuerlig diagonal skannerekkefølge og RLE [I][K]
6. Huffman koding [K]

Steg 3–6: Denne forelesningen, *antar gråtonebilde.*
Steg 1–2: Gjelder kun fargebilder, utsetter til fargeromsforelesningen.

JPEG Kompresjon II

Steg 3:

1. Randutvid med refleksjon så høyde og bredde er delelig med 8.
2. Inndel bildet i 8×8 ikke-overlappende blokker.
3. Subtraher 2^{b-1} fra hver blokk og ta DCT.



Steg 4:

1. Bestem en kvantiseringsmaske[1] basert på ønsket kvalitetsnivå.
2. Kvantiser verdiene ved å dele DCT koeffisientene med masken.
3. De resulterende verdiene kvantiseres til 8-bitdybde.

[1] Disse er gjerne forhåndsberegnet, men inkluderes i header for kompresjonsfilen.

JPEG Kompresjon III

Steg 5:

1. Sorter verdiene i hver blokk med kontinuerlig diagonal skannerekkefølge.
   - Husk: Forventet magnitude til DCT-koeffisientene er oftest høyere nær origo.
2. Ta løpelengdetransformen (RLE) til de sorterte verdiene.
   - Kontinuerlig diagonal skannerekkefølge skanner forventet høyeste magnitude først, og forventet laveste magnitude sist.
   - Løpelengdetransformen vil da gjerne ha flere løp med nuller etter kvantisering.

Steg 6:

1. Slå sammen data fra hver blokk i standard skannerekkefølge.
2. Kod resluterende data med Huffman koding.

JPEG Kompresjon IV

JPEG kompresjon:

- Mest brukte ikke-tapsfri kompresjon for bilder.
- Viser tydelig hvordan flere redudanskomponenter kan benyttes for å komprimere bilder.
- **For årets kurs**: Ikke basis for oblig 2, *eksamensrelevant*!
  - Mange steg, kan virke litt vanskelig å huske alle stegene.
  - Viktigst med **grunnleggende forståelse**.
- **JPEG2000**: Senere versjon, benytter Wavelet transform og prediktiv koding.
- Varianter benyttes i videoenkoding, gjerne i sammenheng med prediktiv koding.

Aasan og Tan

Kompresjon og koding II

Kap. 7.4-7.7 og 8.10 (4th Ed.),
Kompendium: 18.7.3 og 18.8-18.9,
og QOI spesifikasjonen

