INF2310 – 29. mai 2012

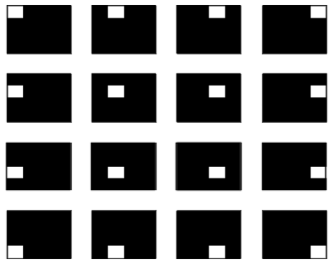
FORELESNING 16: REPETISJON

Denne delen:

- 2D diskret Fourier-transform (DFT)
- Kompresjon og koding
- Morfologiske operasjoner på binære bilder

Standardbasis for matriser

Eksempel: Standardbasis for 4x4



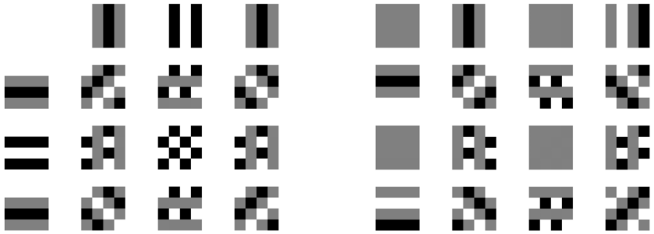
- Et gråtonebilde representeres vanligvis som et rutenett av gråtoneintensiteter
- Dette tilsvarer å bruke den såkalte *standardbasisen* for matriser
- Eksempel: 4x4-gråtonebilder
  - Standardbasisen er de 16 matrisene vist til venstre, der sort er 0 og hvit er 1
  - En vektet sum av disse matrisene kan unikt representere enhver 4x4-matrise/gråtonebilde

Undereksempel:

1	3	2	1
5	4	5	3
4	1	1	2
2	3	2	6

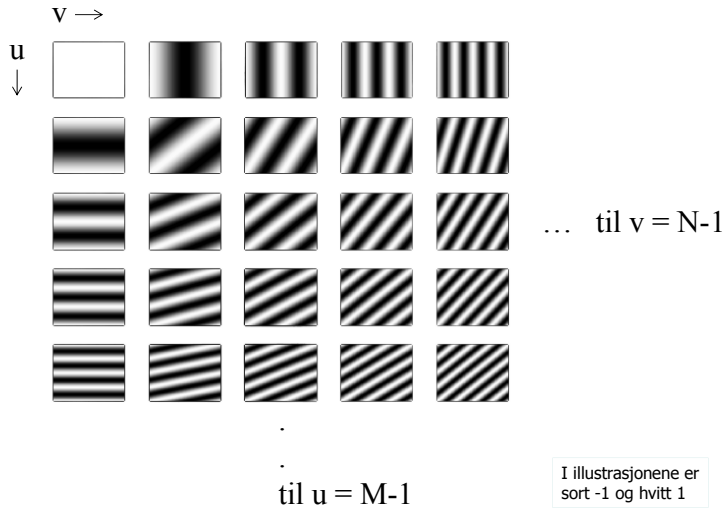
Alternativ basis

- Det finnes mange andre basiser for matriser
  - Muligheten til å unikt representere enhver matrise ligger i *basis*
- 2D DFT bruker én slik basis som er basert på sinuser og cosinuser med forskjellige frekvenser
  - Disse sinusene og cosinusene er faste for en gitt bildestørrelsen (MxN) og kan representeres som hver sin mengde av bilder:



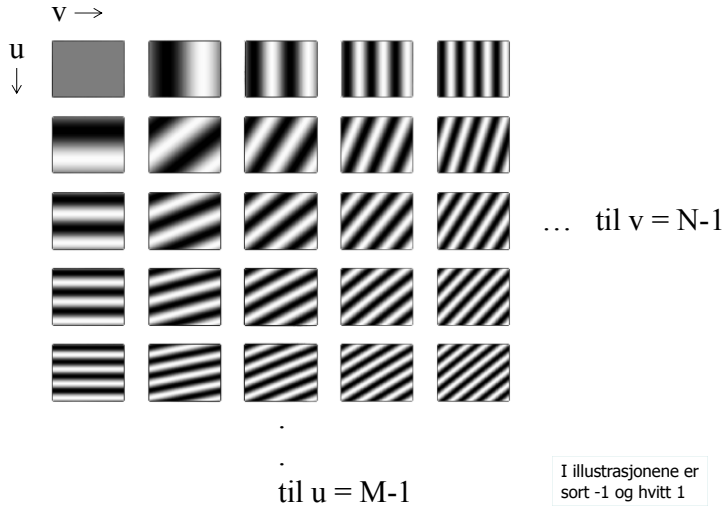
(i bildene er sort -1, grått er 0 og hvitt er 1)

Cosinus-bilder for større bilder



I illustrasjonene er sort -1 og hvitt 1

Sinus-bilder for større bilder

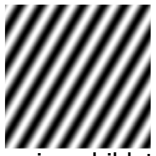


I illustrasjonene er sort -1 og hvitt 1

Beregning av 2D DFT for en gitt frekvens

- Frekvens (3,3) av 2D DFT-en av vårt tidligere eksempelbilde:
- real(F(3,3)) = sum(| 1 | 3 | 2 | 1 | | 5 | 4 | 5 | 3 | | 4 | 1 | 1 | 2 | | 2 | 3 | 2 | 6 | x | 1 | 0 | -1 | 0 | | 0 | -1 | 0 | 1 | | -1 | 0 | 1 | 0 | | 0 | 1 | 0 | -1 |) = -8
- imag(F(3,3)) = sum(| 1 | 3 | 2 | 1 | | 5 | 4 | 5 | 3 | | 4 | 1 | 1 | 2 | | 2 | 3 | 2 | 6 | x | 0 | 1 | 0 | -1 | | 1 | 0 | -1 | 0 | | 0 | -1 | 0 | 1 | | -1 | 0 | 1 | 0 |) = 3
- Altså er F(3,3) = -8+3i

Grunnleggende om 2D DFT

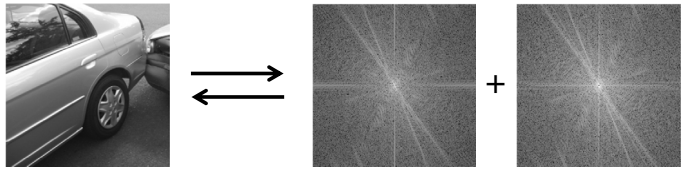
- real(F(u,v)) = sum( x )

  realdelen til 2D DFT-en av frekvensen (u,v) = sum(bildet punkt-multiplisert cosinus-bildet av frekvens (u,v))

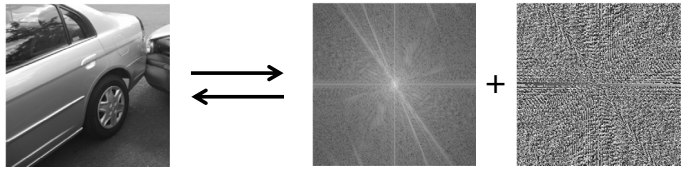
- Tilsvarende for imaginærdelen og sinus-bildet
  - F(u,v) er (generelt) et komplekst tall
  - F(u,v) er en vektet sum av *alle* gråtoneintensitetene i bildet
    - Hvert punkt av 2D DFT-en beskriver noe ved *hele* bildet og *ikke et bildepunkt*

2D DFT på polarform I

- Den visuelt intuitive informasjonen i 2D DFT er de store verdiene:



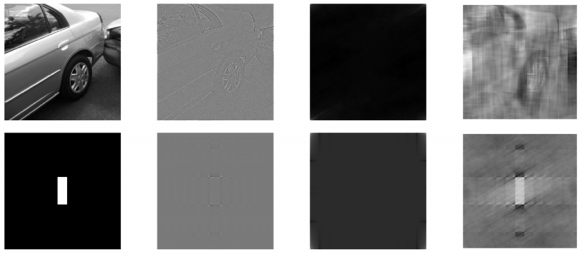
- Denne kan konsentreres ved å representere de komplekse tallene på polarform, dvs. ved amplitude og fase:



2D DFT på polarform II

- Magnituden av en 2D DFT kalles *Fourier-spekteret*
  - Beskriver hvilke frekvenser gråtonebildet inneholder
  - Sterkt knyttet til intuisjonen i Fourier-analyse
- For å rekonstruere det opprinnelige bildet er derimot fasen viktigere:

Alle bildene til høyre er lineært skalert til å fylle hele intensitets-intervallet

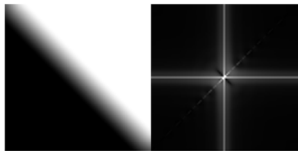


Rekonstruert ved bruk av	Original	Kun fasen	Kun spekteret	Den korrekte fasen og den andres spekter

Fourier-spektre og bildekanter

- Skarp bildekant:
  - Vektet sum av mange sin/cos med forskjellige frekvenser
  - Mange positive koeffisienter i Fourier-spekteret
  - Bredt bånd i Fourier-spekteret

- «Blurret» (alt. flytende) bildekant:
  - Vektet sum av færre sin/cos
  - Færre positive koeffisienter i Fourier-spekteret
  - Smalere bånd i Fourier-spekteret



Konvolusjonsteoremet

- Konvolusjonsteoremet består av to deler:

1) Når $\Leftrightarrow$ betegner at høyresiden er 2D DFT-en til venstresiden, er:

$$f(x,y) \star h(x,y) \iff F(u,v) \cdot H(u,v)$$

Sirkelkonvolusjon i bildedomenet $\Leftrightarrow$ Punktvis multiplikasjon i frekvensdomenet

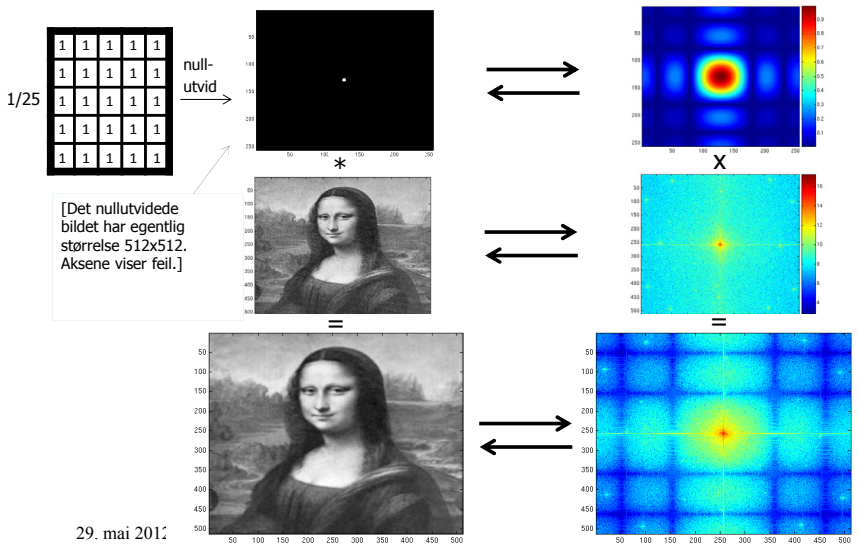
2) Det motsatte gjelder også:

$$f(x,y) \cdot h(x,y) \iff F(u,v) \star H(u,v)$$

Punktvis multiplikasjon i bildedomenet $\Leftrightarrow$ Sirkelkonvolusjon i frekvensdomenet

- Bildene *f* og *h* antatt å ha samme størrelse
  - Nullutvid det minste bildet slik at de får samme størrelse

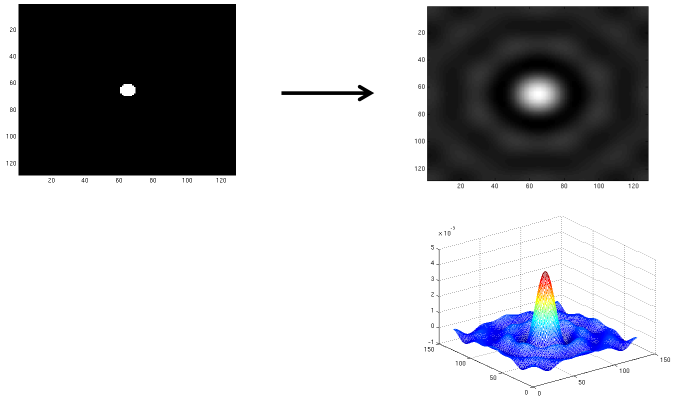
Eksempel: Middelverdifilteret



Anvendelse av konvolusjonsteoremet

- Design av lineære filtre med bestemte frekvensegenskaper
  - Designe filteret i frekvensdomenet slik at det får de ønskelige frekvensegenskapene
- Analyse av konvolusjonsfiltre
  - 2D DFT-en til et konvolusjonsfilter gir oss innblikk i hvordan filteret vil påvirke de forskjellig frekvenskomponentene
- Rask implementasjon av større konvolusjonsfiltre

Filterdesign i frekvensdomenet
Ideelt lavpassfilter og «ringing»



(trunkert sinc-funksjon)

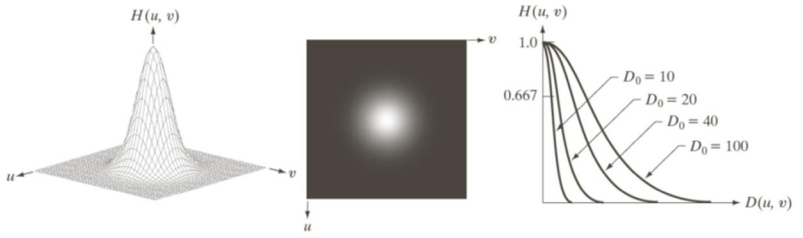
- Vi får en «ringing»-effekt i bildet
  - Husk også tommelfingerregelen:
    Smal/bred struktur i bildet ⇔ Bred/smal struktur i Fourier-spekteret

Eksempel: Ideelt lavpassfilter

Original D_0=0.2 D_0=0.3

Se på bildene i god nok oppløsning (du skal se stripe/ringing-effekter i de to til høyre)

Gaussisk lavpassfilter

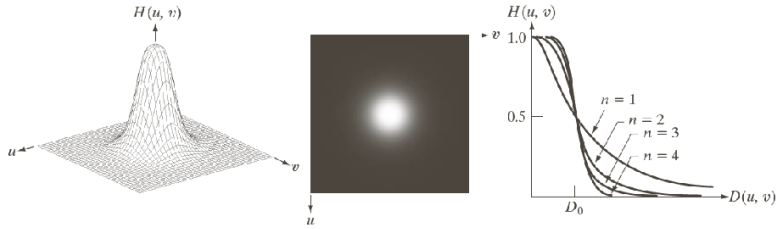


a b c

FIGURE 4.47 (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of D_0.

- 2D IDFT-en av et Gaussisk lavpassfilter er også Gaussisk
  - Ingen ringing i bildedomenet!

Butterworth lavpassfilter



a b c

FIGURE 4.44 (a) Perspective plot of a Butterworth lowpass-filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections of orders 1 through 4.

Eksempel: Butterworth lavpassfilter

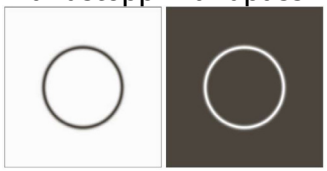
D_0=0.2

n=11 n=41 n=61

Andre typer filtre

- Høypassfilter: $H_{HP}(u,v) = 1 - H_{LP}(u,v)$
- Båndpass-/båndstoppfilter

Båndstopp Båndpass

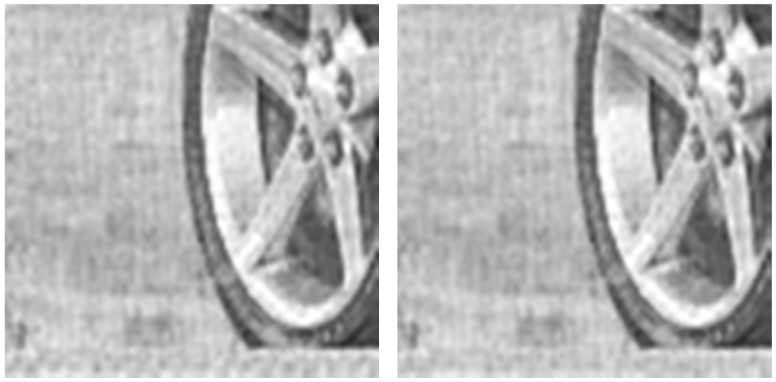


- Notch-filtre slipper igjennom eller stopper mindre predefinerte området i Fourier-spekteret
- Alle kan bruke de samme overgangene:
  - Ideelt, Butterworth, Gaussisk eller annen vindusfunksjon

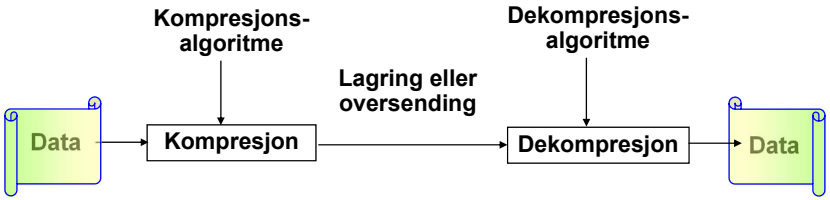
Wraparound-feil

- Periodisitetsegenskapen til 2D DFT gjør at sirkulær indeksering er implisitt antatt når vi filtrerer i frekvensdomenet
- For å behandle randproblemet annerledes må bildet utvides på ønskelig måte *før* 2D DFT-en av det beregnes
  - Filteret må da nullutvides til det utvidede bildet
- Dersom bildet har størrelse MxN og filteret mxn, må det utvidede bildet minst ha størrelse (M+m-1)x(N+n-1)

Eksempel: Wraparound-feil



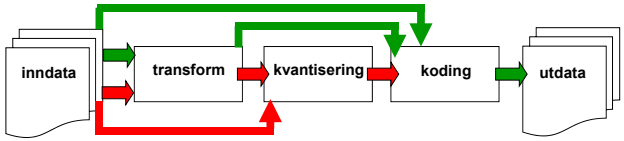
Kompresjon



- **Bildekompresjon** består i å pakke informasjonsinnholdet i bildet ved å ikke lagre redundant informasjon.
- Dataene **komprimeres**, deretter lagres eller overføres de.
- Når de senere skal leses, må vi **dekomprimere** dem.
- Koding er en del av kompresjon, men vi koder for å lagre effektivt, ikke for å hemmeligholde eller skjule informasjon.

Kompresjon

- Kompresjon kan deles inn i tre steg:
  - **Transform** - representer dataene mer kompakt.
  - **Kvantisering** - avrunding av representasjonen.
  - **Koding** - produksjon og bruk av kodebok.



- Kompresjons kan gjøres
  - **Eksakt / tapsfri** (”loss-less”) – følg de grønne pilene
    - Her kan vi rekonstruere den originale meldingen eksakt.
  - **Ikke-tapsfri** (”lossy”) – følg de røde pilene
    - Her kan vi ikke rekonstruere meldingen eksakt.
    - Resultatet kan likevel være ”godt nok”.
- Det finnes en mengde ulike metoder for begge kategorier kompresjon.

Melding, data og informasjon

- Vi skiller mellom data og informasjon:

- **Melding:** teksten, bildet eller signalet som vi skal lagre.
- **Data:** strømmen av biter som lagres på fil eller sendes.
- **Informasjon:** et mål som kvantifiserer mengden overraskelse/uventethet i en melding.
  - Et varierende signal har mer informasjon enn et monotont signal.
  - I et bilde: kanter rundt objekter har høyest informasjonsinnhold,
    - spesielt kanter med mye krumning.

Ulike typer redundans

- Psykovisuell redundans
  - Det finnes informasjon vi ikke kan se.
    - Kan kanskje subsample eller redusere antall bit per piksel.
- Interbilde redundans
  - Det er en viss likhet mellom nabobilder i en tidssekvens
    - Vi kan kode noen bilder i sekvensen og ellers bare differanser.
- Intersampel redundans
  - Nabopiksler ligner på hverandre eller er like.
    - Hver linje i bildet kan "run-length"-transformeres.
- Kodings-redundans
  - Gjennomsnittlig kodelengde minus et teoretisk minimum.
    - Velg en metode som er "grei" å bruke og gir liten kodingsredundans

Kompresjonsrate og redundans

- **Kompresjonsraten :**

$$CR = \frac{i}{c}$$

der i er antall bit pr. sampel originalt,
og c er antall bit pr. sampel i det komprimerte bildet.

- **Relativ redundans :**

$$R = 1 - \frac{1}{CR} = 1 - \frac{c}{i}$$

- **"percentage removed" :**

$$PR = 100\left(1 - \frac{c}{i}\right)\%$$

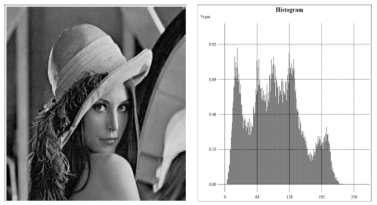
Differansetransform

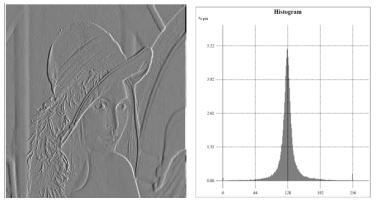
- Gitt en rad i bildet med gråtoner

  $f_1,.....f_N$, $0 \le f_i \le 2^b-1$.

- Transformer (reversibelt) til

$g_1 = f_1$, $g_2 = f_2-f_1$, ... , $g_N = f_N-f_{N-1}$





- Vi trenger nå b+1 biter hvis vi skal tilordne like lange binære koder til alle mulig verdier.
- I differansehistogrammet vil de fleste verdiene samle seg rundt 0.
- En naturlig bit-koding av differansene er ikke optimal.

Løpelengdetransform

- Ofte inneholder bildet objekter med lignende gråtoner,
  f.eks. svarte bokstaver på hvit bakgrunn.
- Vi kan benytte oss av kompresjonsteknikker
  som tar hensyn til at nabopiksler på samme rad ofte er like.
- Løpelengdetransform er reversibel.
- Først et eksempel:
  333333555555555544777777 (24 tall)
- Når tallet 3 forekommer 6 ganger etter hverandre,
  trenger vi bare lagre tallparet (3,6).
- Tilsammen trenger vi her 4 tallpar, (3,6), (5,10), (4,2), (7,6)
  altså 8 tall til å lagre hele sekvensen 24 tall ovenfor.
- Hvor mange biter vi bruker pr. tall, avhenger av den videre kodingen.

Entropi

- Hvis vi tar gjennomsnittet over alle symbolene s_i i alfabetet, får vi gjennomsnittlig informasjon pr. symbol.
- Entropi H:

$$H = \sum_{i=0}^{2^b-1} p(s_i) I(s_i) = -\sum_{i=0}^{2^b-1} p(s_i) \log_2(p(s_i))$$

- Entropien setter en nedre grense for hvor kompakt sekvensen kan representeres
  - Dette gjelder hvis vi bare koder hvert symbol for seg.

Huffman-koding

- Huffman-koding er en algoritme for variabel-lengde koding som er optimal under begrensningen at vi koder symbol for symbol.
- Er basert på at vi kjenner hyppigheten for hvert symbol
  - Dette betyr at vi må lage et histogram.
  - Ofte beregner vi sannsynlighetene
    - Men det holder at vi kjenner hyppighetene.

Eksempel - Huffman-koding

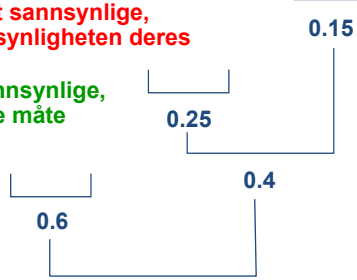
Gitt 6 begivenheter A, B, C, D, E, F med sannsynligheter

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0.3	0.3	0.13	0.12	0.1	0.05

Slå sammen de to minst sannsynlige,
Slå også sammen sannsynligheten deres

Finn de to som nå er minst sannsynlige,
og slå dem sammen på samme måte

Fortsett til det er bare to igjen

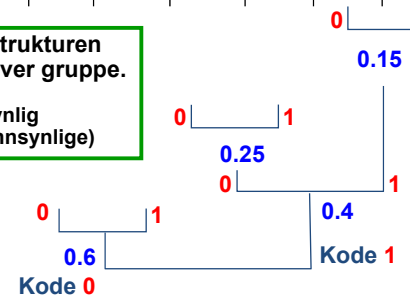


Eksempel - Huffman-koding

Gitt 6 begivenheter A, B, C, D, E, F med sannsynligheter

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0.3	0.3	0.13	0.12	0.1	0.05

Gå baklengs gjennom strukturen og tilordne 0 eller 1 til hver gruppe.
(F. eks.
kode 0 til den mest sannsynlig
og kode 1 til den minst sannsynlige)



Eksempel på Lempel-Ziv-Welch I

- Anta at alfabetet består av a, b og c som tilordnes kodene 1, 2 og 3.
  La meldingen være ababcbababaaaaabab (18 symboler)
  sender: ny frase = sendt streng pluss neste usendte symbol
  mottaker: ny frase = nest siste streng pluss første symbol i sist tilsendte streng

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

- Vi mottar en kode som ikke finnes i listen.
- Kode 8 ble laget som "ba+?" , og nå sendes kode 8.
- Altså må vi ha "?" = "b" => 8 = ba + b = bab.

Eksempel på Lempel-Ziv-Welch II

Ser	Sender	Senders liste	Mottar	Tolker	Mottakers liste
		a=1,b=2,c=3			a=1, b=2, c=3
a	1	ab=4	1	a	
b	2	ba=5	2	b	ab=4
ab	4	abc=6	4	ab	ba=5
c	3	cb=7	3	c	abc=6
ba	5	bab=8	5	ba	cb=7
bab	8	baba=9	8	bab	bab=8
a	1	aa=10	1	a	baba=9
aa	10	aaa=11	10	aa	aa=10
aa	10	aab=12	10	aa	aaa=11
bab	8		8	bab	aab=12

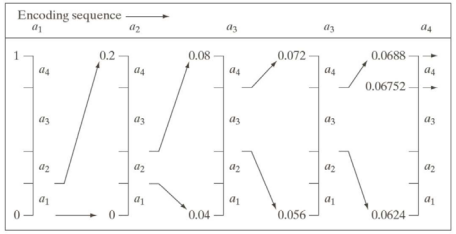
- Kode 10 ble laget idet 1 = a ble sendt, som 10 " a+?". Så sendes "10".
- Da må vi ha 10="a+a" ="aa".
- Istedenfor 18 tegn er det sendt 10 koder.
- 5 av 12 koder ble ikke brukt.

Aritmetisk koding

- Tapsfri kompresjonsmetode
  - Gir ofte litt bedre kompresjonsrate enn Huffman-koding.
- Er en variabel-lengde entropikoding.
- Metoden produserer ikke kodeord for enkeltsymboler.
- Metoden koder en sekvens av symboler til et tall n $(0.0 \leq n < 1.0)$.
- Dermed kan man få en kortere kode enn med Huffman-koding, fordi man ikke lenger er begrenset av at hvert symbol skal tilordnes et kodeord med et heltalls antall biter.
- Resulterer i et bitforbruk per symbol som er nær entropien.
- Er generelt bedre (målt i bitforbruk per symbol)
  - når sannsynlighetsmodellen samsvarer med de virkelige forekomstene av symboler
  - jo lenger symbolsekvensen er

AK: Dele opp i stadig mindre intervaller

- Anta at vi har en 5-tegns symbolsekvens $a_1a_2a_3a_3a_4$
- Anta at symbolsannsynlighetene er lik forekomsthyppighetene;
  $P(a_1) = P(a_2) = P(a_4) = 0{,}2$ og $P(a_3) = 0{,}4$



- a_1 ligger i intervallet [0, 0.2)
- a_1a_2 ligger i intervallet [0.04, 0.08)
- $a_1a_2a_3$ ligger i intervallet [0.056, 0.072)
- $a_1a_2a_3a_3$ ligger i intervallet [0.0624, 0.0688)
- $a_1a_2a_3a_3a_4$ ligger i intervallet [0.06752, 0.0688)

AK: Finne en minst mulig binærrepresentasjon av intervallet

- Finn kortest mulig N=0.$c_1c_2c_3$... innenfor [0.6, 0.7).
- Når c_i er 0 eller 1 og n ≥ k så gjelder følgende:

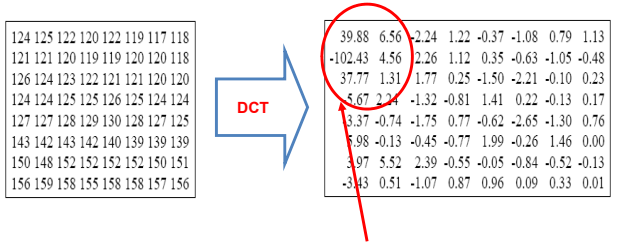
  $c_k*2^{-k} + \ldots + c_n*2^{-n} < 2^{-(k-1)}$
- Ergo:

  N = 0.**1...** => 0.5 ≤ N < 1
  N = 0.**10...** => 0.5 ≤ N < 0.75
  N = 0.**100...** => 0.5 ≤ N < 0.625
  N = 0.**101...** => 0.625 ≤ N < 0.75
- Vi kan kode intervallet vårt med det binære kommatallet **.101$_2$** (ekvivalent med 0.625_{10}), altså med bare 3 biter.
  - I noen situasjoner vil vi kreve at øvre og nedre grense er innenfor intervallet; i dette eksempelet vil $.1010_2$ brukes i et slikt tilfelle:

    N = 0.**1010...** => 0.625 ≤ N < 0.6875

Ikke-tapsfri JPEG-kompresjon I

1. Hver bildekanal deles opp i blokker på 8x8 piksler, og hver blokk i hver kanal kodes separat.
2. Dersom intensitetene er gitt uten fortegn; trekk fra 2^{b-1} der 2^b er antall gråtoner
   - Gjør at forventet gjennomsnittlig gråtone er 0
   - Eksempel: For intensitetsintervallet [0, 255]; trekk 128 fra alle pikselverdiene.
3. Hver blokk transformeres med 2D DCT (diskret cosinus-transform)

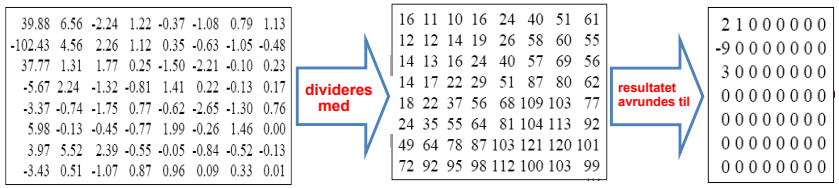


   - Mye av informasjonen i de 64 pikslene samles i en liten del av de 64 DCT-koeffisientene; nemlig de i øverste, venstre hjørne

Ikke-tapsfri JPEG-kompresjon II

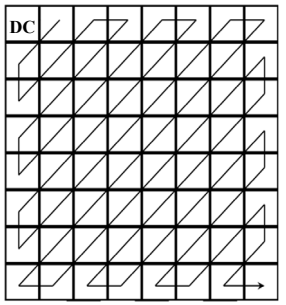
JPEG-kompresjonen fortsetter med at:

4. 2D DCT-koeffisientene
   a) skaleres med en vektmatrise og
   b) kvantiseres til heltall.



Ikke-tapsfri JPEG-kompresjon III

DC- og AC-koeffisientene behandles nå separat;



- Sikk-sakk-skanning ordner AC-koeffisientene i en 1D-følge.
  - Absoluttverdien av koeffisientene vil stort sett avta utover i følgen.
  - Mange koeffisienter er null.
- Løpelengde-transform av koeffisientene
- Huffman-koding eller aritmetisk koding av løpelengdene.
  - Både predefinerte og egendefinerte Huffman-kodebøker tillates.

Ikke-tapsfri JPEG-kompresjon IV

DC- og AC-koeffisientene
behandles nå separat;

- For hver kanal samles DC-koeffisientene fra alle blokkene.
- Disse er korrelerte og blir derfor differansetransformert.
- Differansene Huffman-kodes eller aritmetisk kodes.

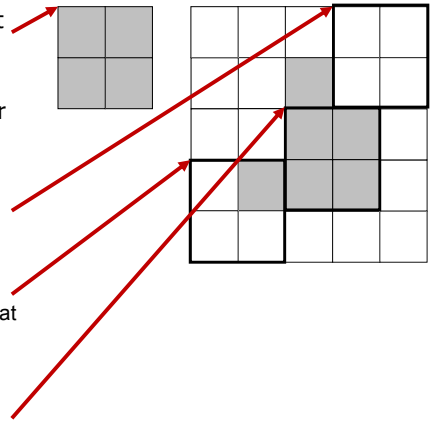
Ikke-tapsfri JPEG: Blokk-artefakter

- Blokk-artefaktene øker med kompresjonsraten.

- Øverst: kompresjonsrate = 25
- Nederst: kompresjonsrate = 52

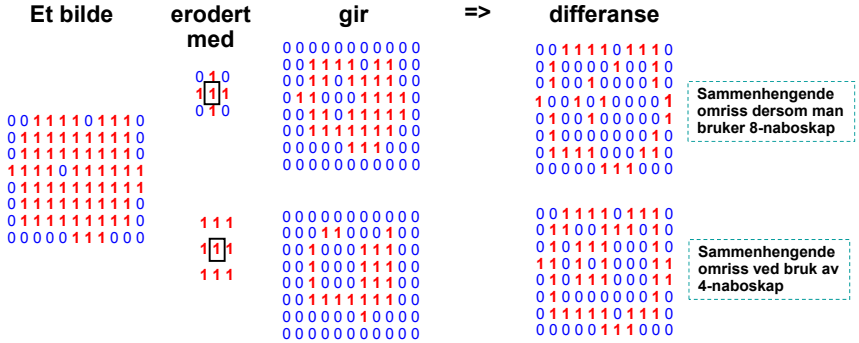
Morfologi: Tre sentrale begrep

- Et **strukturelement** for et binært bilde er en liten matrise av piksler
- Når vi fører strukturelementet over det binære bildet vil vi finne:
  - Posisjoner der strukturelementet ikke overlapper objektet.
  - Posisjoner der strukturelementet delvis overlapper objektet, vi sier at elementet **treffer** objektet.
  - Posisjoner der strukturelementet ligger inni objektet, vi sier at elementet **passer** i objektet.



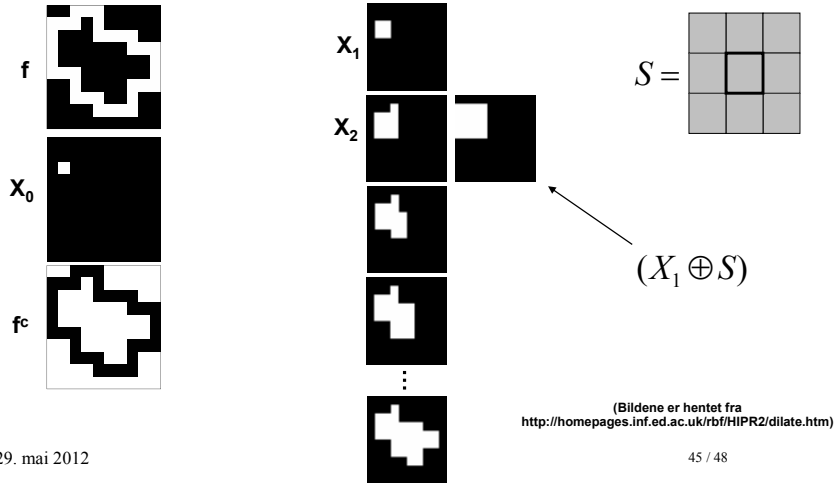
Kantdeteksjon ved erosjon

- Erodering fjerner piksler langs omrisset av en region.
- Vi kan finne kanten av regionene i bildet ved å subtrahere et erodert bilde fra originalbildet: **g = f - (f ⊖ S)**
- Strukturelementet avgjør 4- eller 8-naboskap:



Region-fylling med dilasjon

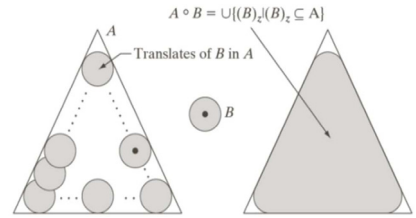
- La X_0 inneholde et punkt i regionen som skal fylles.
- Deretter iterer over følgende: $X_k = (X_{k-1} \oplus S) \cap f^c$



(Bildene er hentet fra http://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm)

Geometrisk tolkning av åpning

- Tenk at strukturelementet definerer størrelsen og formen til spissen av en tusjpenn.
- Det er bare tillatt å fargelegge innenfor objekter.
  - Et par detaljer: Man må holde tusjen vinkelrett på tegneflaten og med samme rotasjon som strukturelementet.
- Åpningen er resultatet av å fargelegge så mye man har lov til.
- For runde strukturelementer: Konvekse hjørner blir avrundet, konkave hjørner beholdes rette.
  - Akkurat som ved dilasjon (skyldes at en åpning avsluttes med en dilasjon).



(Deler av figur 9.8 i DIP)

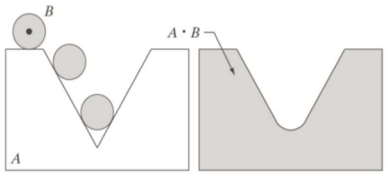
Åpning er idempotent:

$$(f \circ S) \circ S = f \circ S$$

dvs. at gjentatte anvendelser med samme strukturelement gir ingen endring.

Geometrisk tolkning av lukking

- Vi kan benytte samme metafor som for åpning:
  - Strukturelementet definerer størrelsen og formen til spissen av en tusjpenn.
  - Man holder tusjen vinkelrett tegneflaten og fargelegger så mye man har lov til.
- Denne gangen er det bare tillatt å fargelegge utenfor objekter.
  - En detalj: Denne gangen skal tusjen holdes speilvendt (180° rotert) av strukturelementet.
- Lukkingen er det som ikke fargelegges.
  - Denne gangen fargelegger vi altså bakgrunnen, sist fargela vi forgrunnen.
- For runde strukturelementer: Konkave hjørner blir avrundet, konvekse hjørner beholdes rette.
  - Akkurat som ved erosjon (skyldes at en lukking avsluttes med en erosjon).



(Deler av figur 9.9 i DIP)

Også lukking er idempotent:

$$(f \bullet S) \bullet S = f \bullet S$$

Filtrering ved åpning og lukking

FIGURE 9.11
(a) Noisy image. (b) Structuring element. (c) Eroded image. (d) Opening of A. (e) Dilation of the opening. (f) Closing of the opening. (Original image courtesy of the National Institute of Standards and Technology.)