

---

# INF2310 – Digital bildebehandling

## **FORELESNING 15**

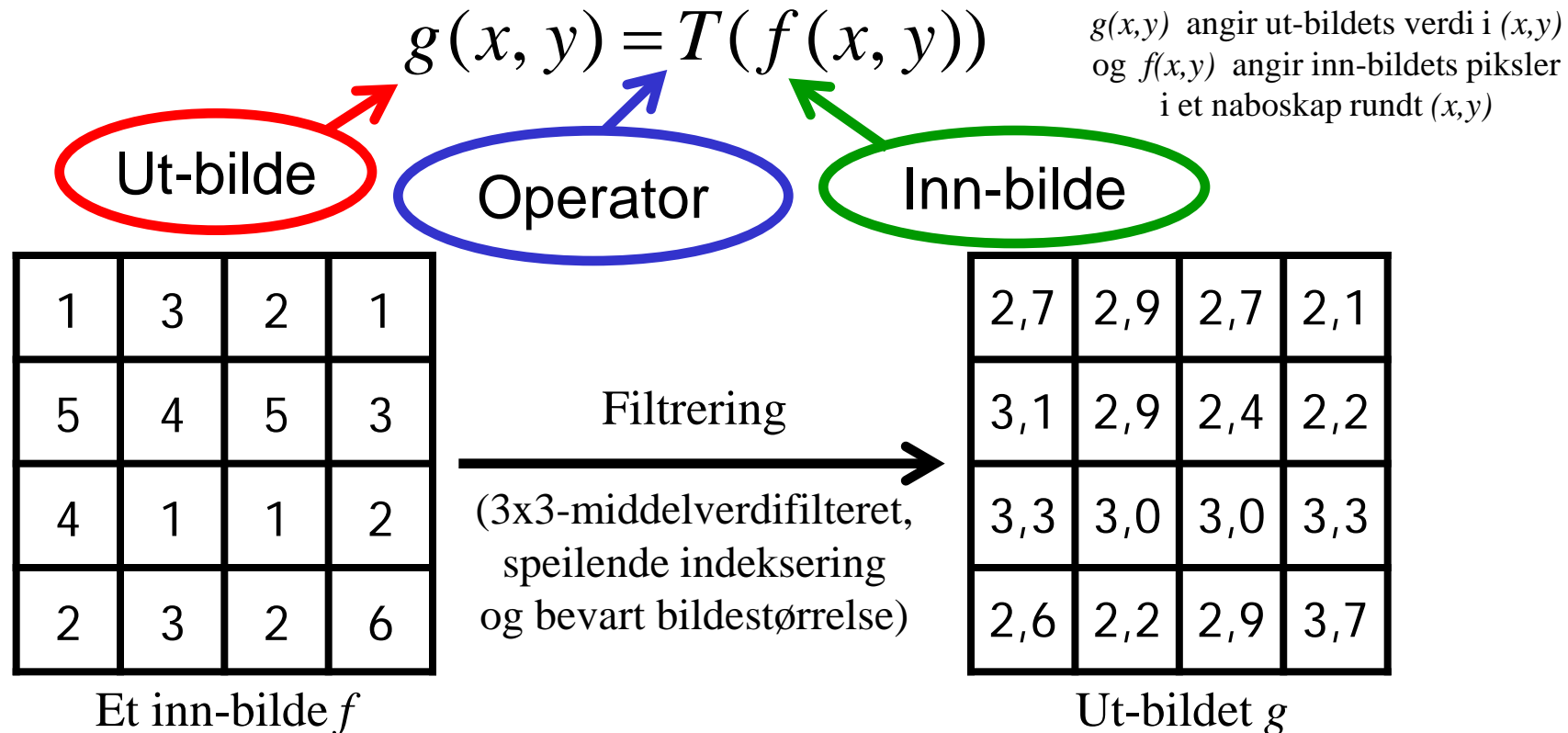
### **REPETISJON**

Andreas Kleppe

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

# Filtrering i billedomenet

- Anvendelsen av en **operator** som beregner ut-bildets verdi i hver piksel  $(x,y)$  ved bruk av inn-bildets piksler i et **naboskap** rundt  $(x,y)$ .



# Lavpassfiltre

---

- Slipper gjennom lave frekvenser og demper eller fjerner høye frekvenser.
  - Lav frekvenser = trege variasjoner, store trender.
  - Høye frekvenser = skarpe kanter, støy, detaljer.
  - ... mye mer om frekvens i Fourier-forelesningene.
- Effekt: **Glattning**/utsmøring/«blurring» av bildet.
- Typiske mål: Fjerne støy, finne større objekter.
- **Utfordring**: **Bevare kanter**.

# Middelverdifilter (lavpass)

---

- Beregner middelveidien i naboskapet.
  - Alle vektene er like.
  - Vektene summerer seg til 1.
    - Gjør at den lokale middelveidien bevares.
- Størrelsen på filteret avgjør graden av glatting.
  - Stort filter: mye glatting (utsmørt bilde).
  - Lite filter: lite glatting, men kanter bevares bedre.

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\frac{1}{49} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

# Gauss-filter (lavpass)

- For heltallsverdier av  $x$  og  $y$ , la:

$$h(x, y) = A \exp\left(-\frac{(x^2 + y^2)}{2\sigma^2}\right)$$

- $A$  settes slik at summen av vektene blir 1.
- $N(0, \sigma^2 I_2)$  med alternativ skalering  $A$ .

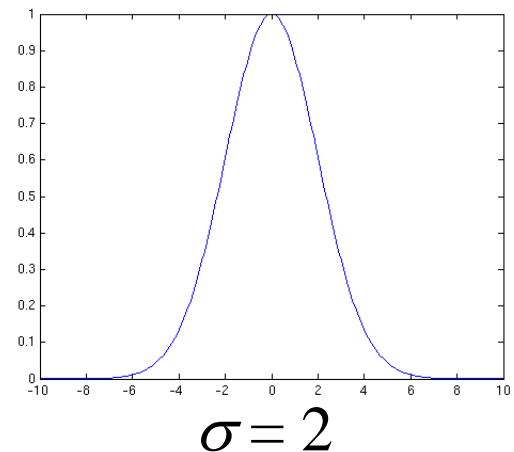
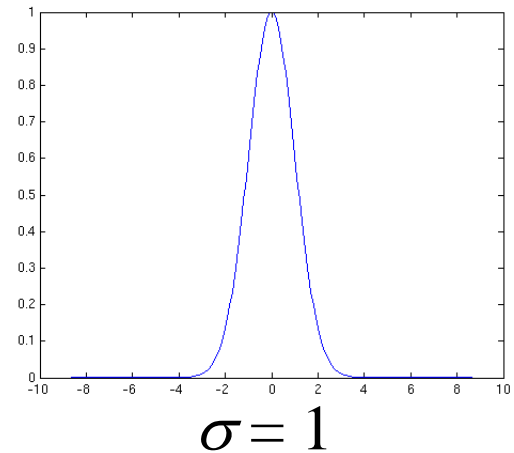
- **Ikke-uniformt lavpassfilter.**

- Parameteren  $\sigma$  er standardavviket og avgjør graden av glatting.


- Lite  $\sigma \Rightarrow$  lite glatting.
- Stort  $\sigma \Rightarrow$  mye glatting.

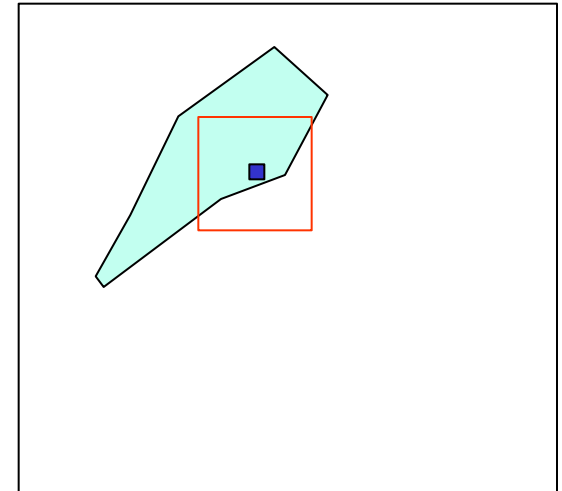
- Filterstørrelsen  $n \times n$  må tilpasses  $\sigma$ .

- Et Gauss-filter glatter *mindre* enn et middelværdifilter av samme størrelse.



# Kant-bevarende støyfiltrering

- Ofte lavpassfiltrerer vi for å **fjerne støy**, men ønsker samtidig å **bevare kanter**.
- Det finnes et utall av «kantbevarende» filtre.
- Men det er et system:
  - Tenker at vi har flere piksel-populasjoner i naboskapet rundt  $(x,y)$ , f.eks. to: 
  - Sub-optimalt å bruke alle pikslene.
- Vi kan sortere pikslene:
  - Radiometrisk (etter pikselverdi)
  - Både geometrisk (etter pikselposisjon) og radiometrisk



# Median-filter (lavpass)

---

- $g(x,y)$  = median pikselverdi i naboskapet rundt  $(x,y)$ .
- Median = den midterste verdien i den sorterte listen.
  - Så et medianfilter er et rangfilter der vi velger midterste posisjon i den sorterte 1D-listen.
- Et av de mest brukte kant-bevarende støyfiltrene.
- Spesielt god mot impuls-støy («salt-og-pepper-støy»).
- Ikke uvanlig med ikke-rektangulære naboskap, f.eks. +
- Problemer:
  - Tynne linjer kan forsvinne.
  - Hjørner kan rundes av.
  - Objekter kan bli litt mindre.
- Valg av størrelse og form på naboskapet er viktig!

# Middelverdi eller median?

---



Inn-bilde med tydelig  
salt-og-pepper-støy



Etter  
middelverdifiltrering



Etter  
medianfiltrering



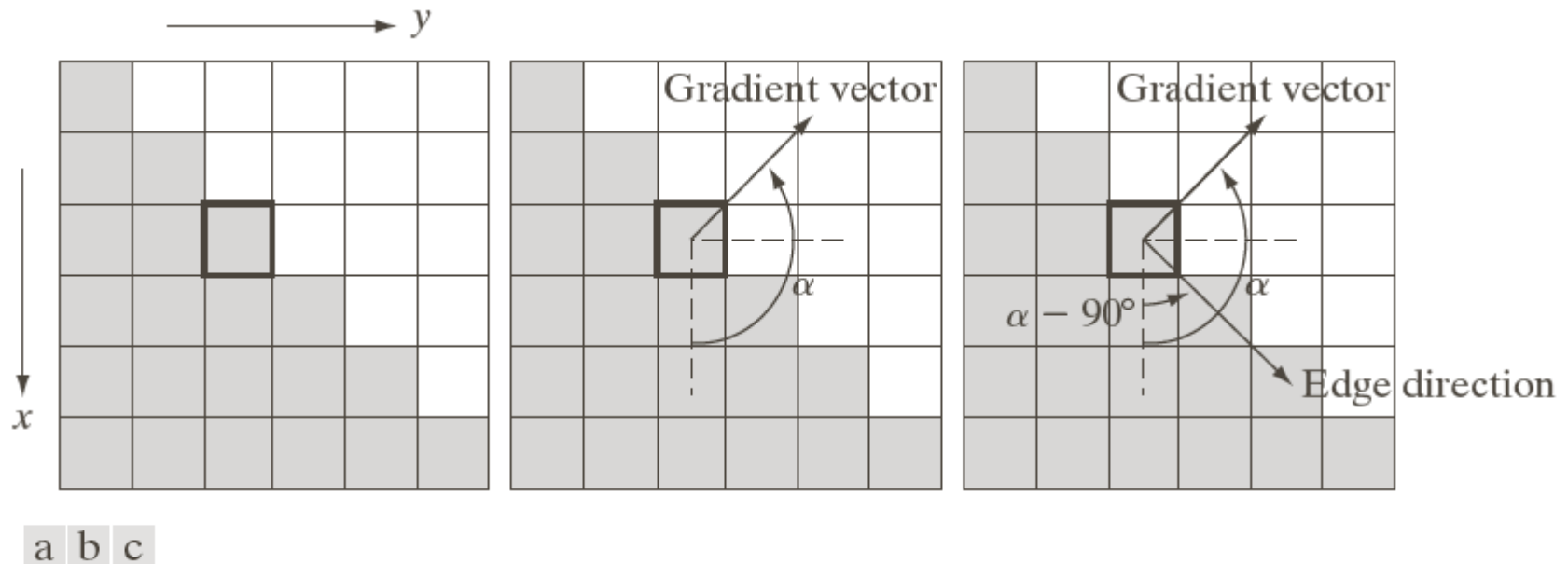
# Høypassfiltre

---

- Slipper gjennom høye frekvenser, og demper eller fjerner lave frekvenser.
  - Typisk fjernes den aller laveste frekvensen helt, dvs. at homogene områder får ut-verdi 0.
- Effekt:
  - Demper langsomme variasjoner, f.eks. bakgrunn.
  - Fremhever skarpe kanter, linjer og detaljer.
- Typiske mål: «Forbedre» skarpheten, detektere kanter.
- **Q**: Hva skjer med støy?

# Gradient $\perp$ Kant

- **Gradienten** peker i retningen der funksjonen **øker mest** og **kanten** går **vinkelrett på gradienten**.



**FIGURE 10.12** Using the gradient to determine edge strength and direction at a point. Note that the edge is perpendicular to the direction of the gradient vector at the point where the gradient is computed. Each square in the figure represents one pixel.

# Gradient-operatorer

- Prewitt-operatoren:

$$h_x(i, j) = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad h_y(i, j) = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

- Sobel-operatoren:

$$h_x(i, j) = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad h_y(i, j) = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

- Frei-Chen-operatoren:

$$h_x(i, j) = \begin{bmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{bmatrix} \quad h_y(i, j) = \begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix}$$

PS: Vi angir konvolusjonsfiltre i den tanke at de skal brukes til konvolusjon.

G&W angir filtermasker som skal brukes til korrelasjon.

Filtrene vil derfor avvike med en 180 graders rotasjon.

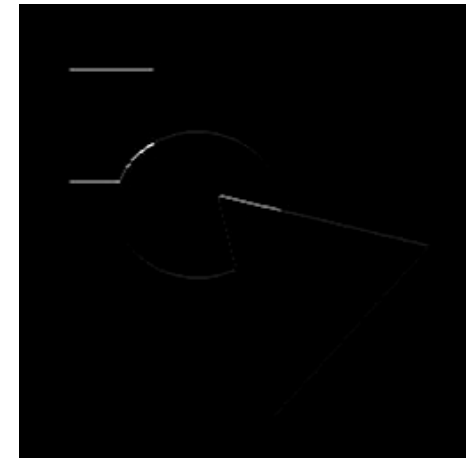
# Eksempel: Gradient-beregning med Sobel-operatoren



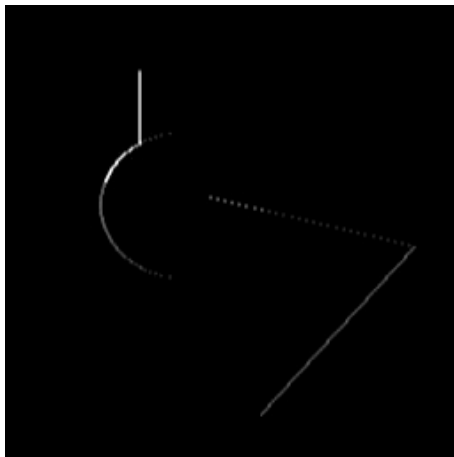
Inn-bilde  $f$



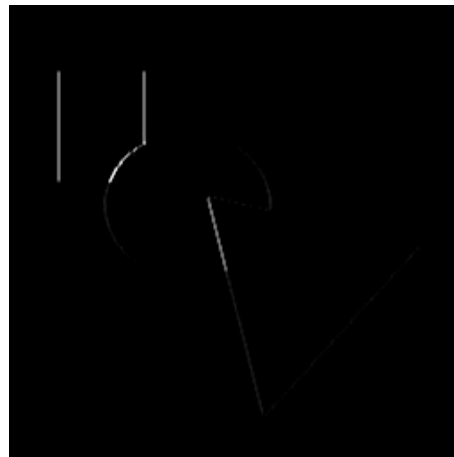
$g_x = f * h_x$



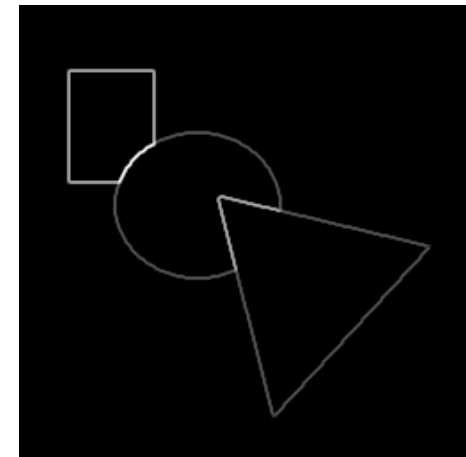
$g_x^2$



$g_y = f * h_y$



$g_y^2$



$(g_x^2 + g_y^2)^{1/2}$

Merk:  
Hvert bilde  
er skalert  
ved å dele  
på sitt  
maksimum.  
De negative  
verdiene i  
 $g_x$  og  $g_y$   
er satt til 0.

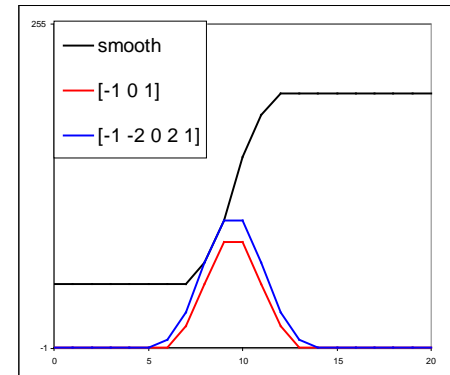
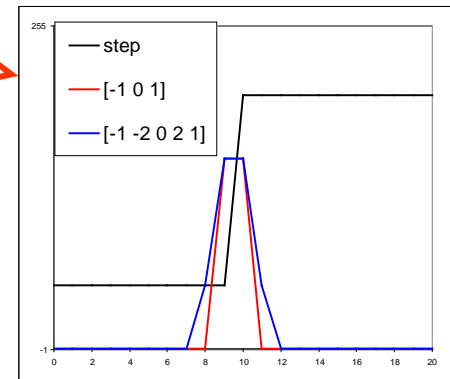
# Gradient til kant-deteksjon

- Gradient-magnituden har «bred respons», men vi ønsker eksakt, tynn kant.

- For en steg-kant:
  - Merk: Bredden på responsen er avhengig av størrelsen på filteret.

- For en bred kant (glattet med  $[1\ 2\ 3\ 2\ 1]/9$ ):
  - Merk: Bredden på responsen er avhengig av bredden på kanten.

- **Maksimumet er likt og fornuftig lokalisert!**
  - Bruke den andrederiverte til å finne maksimumene?



PS: Filtrene til venstre er brukt til korrelasjon.

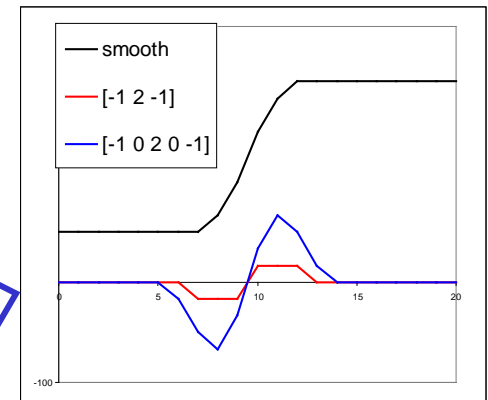
# Laplace-operatoren

- Laplace-operatoren er gitt ved:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

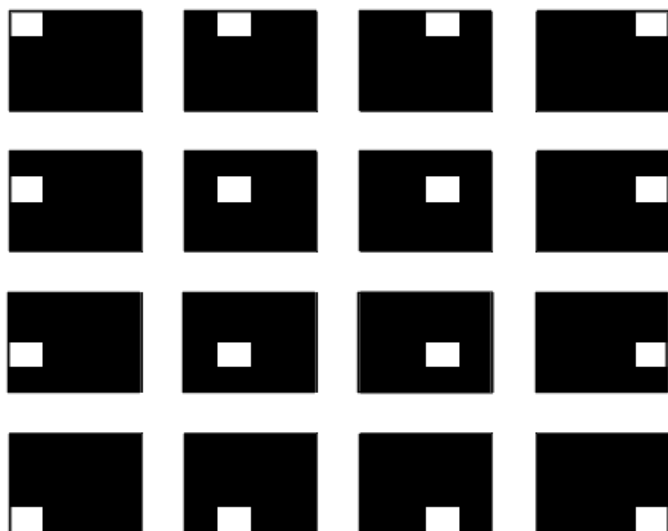
- Den endrer fortegn der  $f$  et vendepunkt.
- $\nabla^2 f = 0$  markerer kant-posisjon.
- $|\nabla^2 f|$  har to ekstremverdier per kant; på starten og på slutten av kanten.
  - Derfor brukte vi den tidligere til å forbedre bildeskarpheiten!

- Kantens eksakte posisjon er nullgjennomgangen.
- Dette gir tynne kanter.
- Vi finner bare kant-posisjoner, ikke kant-retninger.



# Standardbasis for matriser

## Eksempel: Standardbasis for 4x4



- Et gråtonebilde representeres vanligvis som en matrise av gråtoneintensiteter.
- Dette tilsvarer å bruke den såkalte *standardbasisen* for matriser.
- Eksempel: 4x4-gråtonebilder
  - Standardbasisen er de 16 4x4-matrisene vist til venstre.
  - En vektet sum av disse matrisene kan unikt representere enhver 4x4-matrise/gråtonebilde.

## Undereksempel:

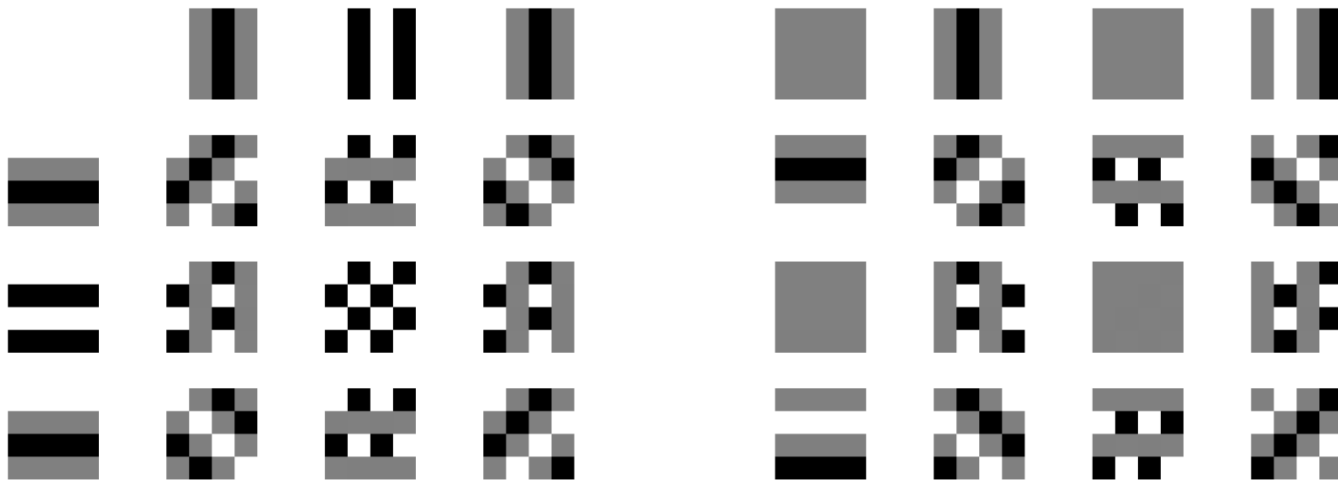
1	3	2	1
5	4	5	3
4	1	1	2
2	3	2	6

$$= 1 * \begin{matrix} \blacksquare & & & \\ & \blacksquare & & \\ & & \blacksquare & \\ & & & \blacksquare \end{matrix} + 3 * \begin{matrix} & \blacksquare & & \\ & & \blacksquare & \\ & & & \blacksquare \\ & & & & \blacksquare \end{matrix} + \dots + 6 * \begin{matrix} & & & \\ & & & \\ & & & \\ & & & \blacksquare \end{matrix}$$

I bildene er  
sort 0 og hvitt 1.

# Alternativ basis

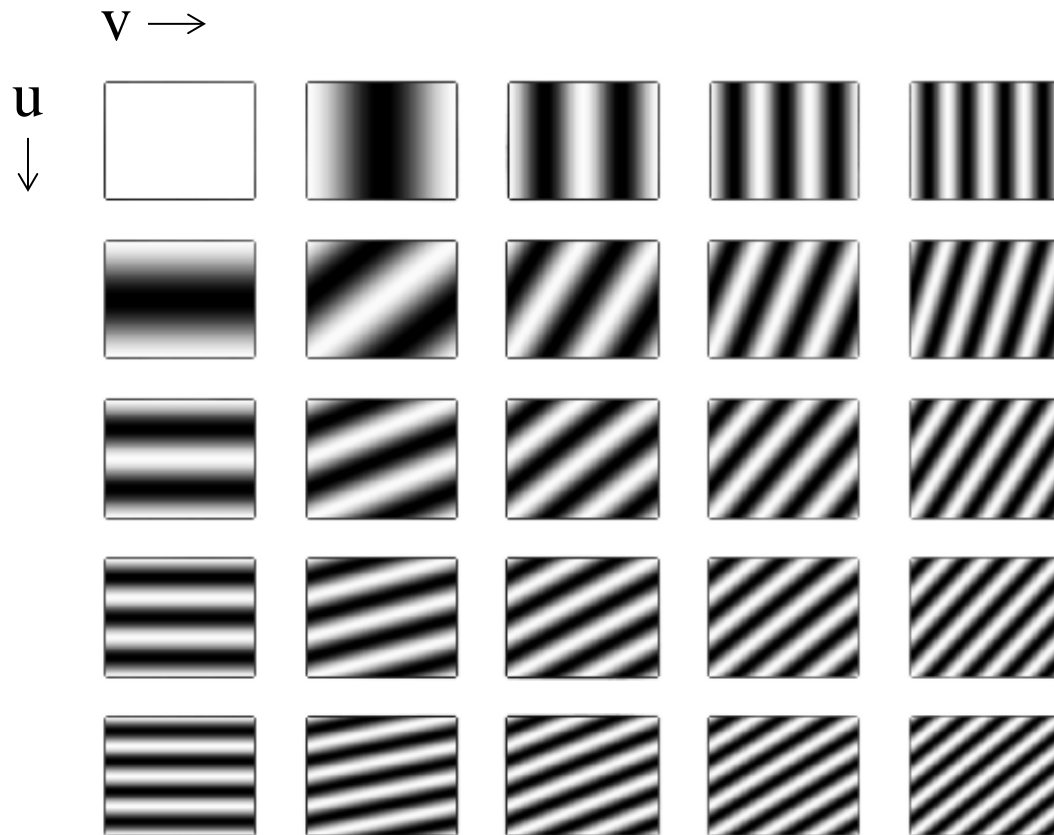
- Det finnes mange andre basiser for matriser.
  - Muligheten til å **unik** representere **enhver** matrise ligger i *basis*.
- **2D DFT** bruker én slik basis som er basert på **sinuser og cosinuser med forskjellige frekvenser**.
  - Disse sinusene og cosinusene er faste for en gitt bildestørrelse ( $M \times N$ ) og kan representeres som hver sin mengde av  $MN$   $M \times N$ -bilder;



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



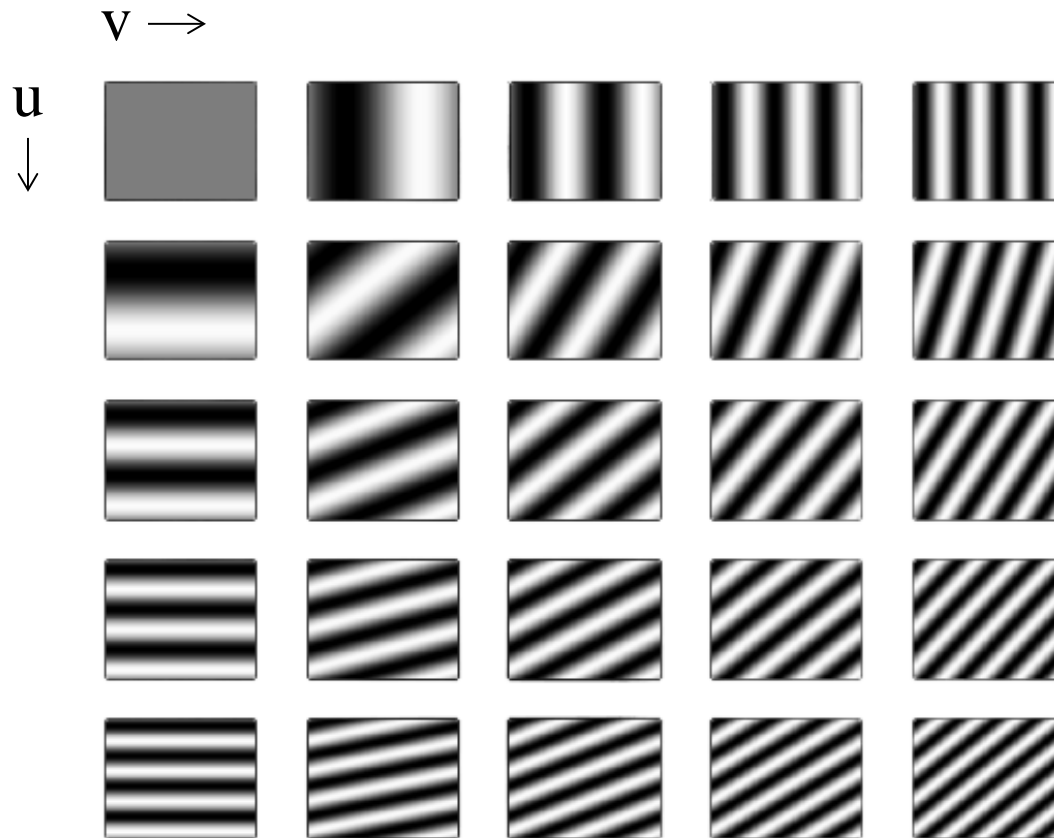
# Cosinus-bilder for større bilder



•  
•  
til  $u = M-1$

I bildene er  
sort -1 og hvitt 1.

# Sinus-bilder for større bilder



•  
•  
til  $u = M-1$

I bildene er  
sort -1 og hvitt 1.

# Beregning av 2D DFT for en gitt frekvens

- Koeffisienten til 2D DFT av tidligere eksempelbilde for frekvens (0,1):

- $\text{real}(F(0,1)) = \text{sum}(\begin{array}{|c|c|c|c|} \hline 1 & 3 & 2 & 1 \\ \hline 5 & 4 & 5 & 3 \\ \hline 4 & 1 & 1 & 2 \\ \hline 2 & 3 & 2 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline 1 & 0 & -1 & 0 \\ \hline 1 & 0 & -1 & 0 \\ \hline 1 & 0 & -1 & 0 \\ \hline 1 & 0 & -1 & 0 \\ \hline \end{array}) = 2$

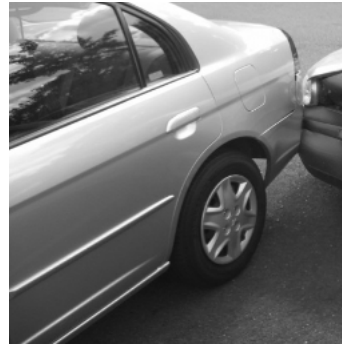
- $\text{imag}(F(0,1)) = \text{sum}(\begin{array}{|c|c|c|c|} \hline 1 & 3 & 2 & 1 \\ \hline 5 & 4 & 5 & 3 \\ \hline 4 & 1 & 1 & 2 \\ \hline 2 & 3 & 2 & 6 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|} \hline 0 & -1 & 0 & 1 \\ \hline 0 & -1 & 0 & 1 \\ \hline 0 & -1 & 0 & 1 \\ \hline 0 & -1 & 0 & 1 \\ \hline \end{array}) = 1$

- Altså er  $F(0,1) = 2+j$ .

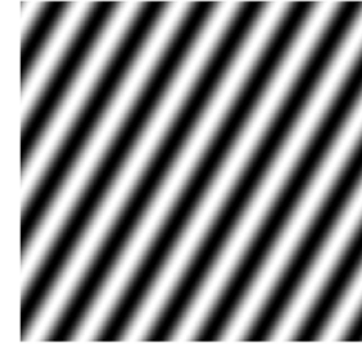
# Grunnleggende om 2D DFT

- $$\text{real}(F(u,v)) = \text{sum}(\text{bildet} \times \text{cosinus-bildet for frekvens (u,v)})$$

realdelen til 2D DFT-en i frekvens (u,v)



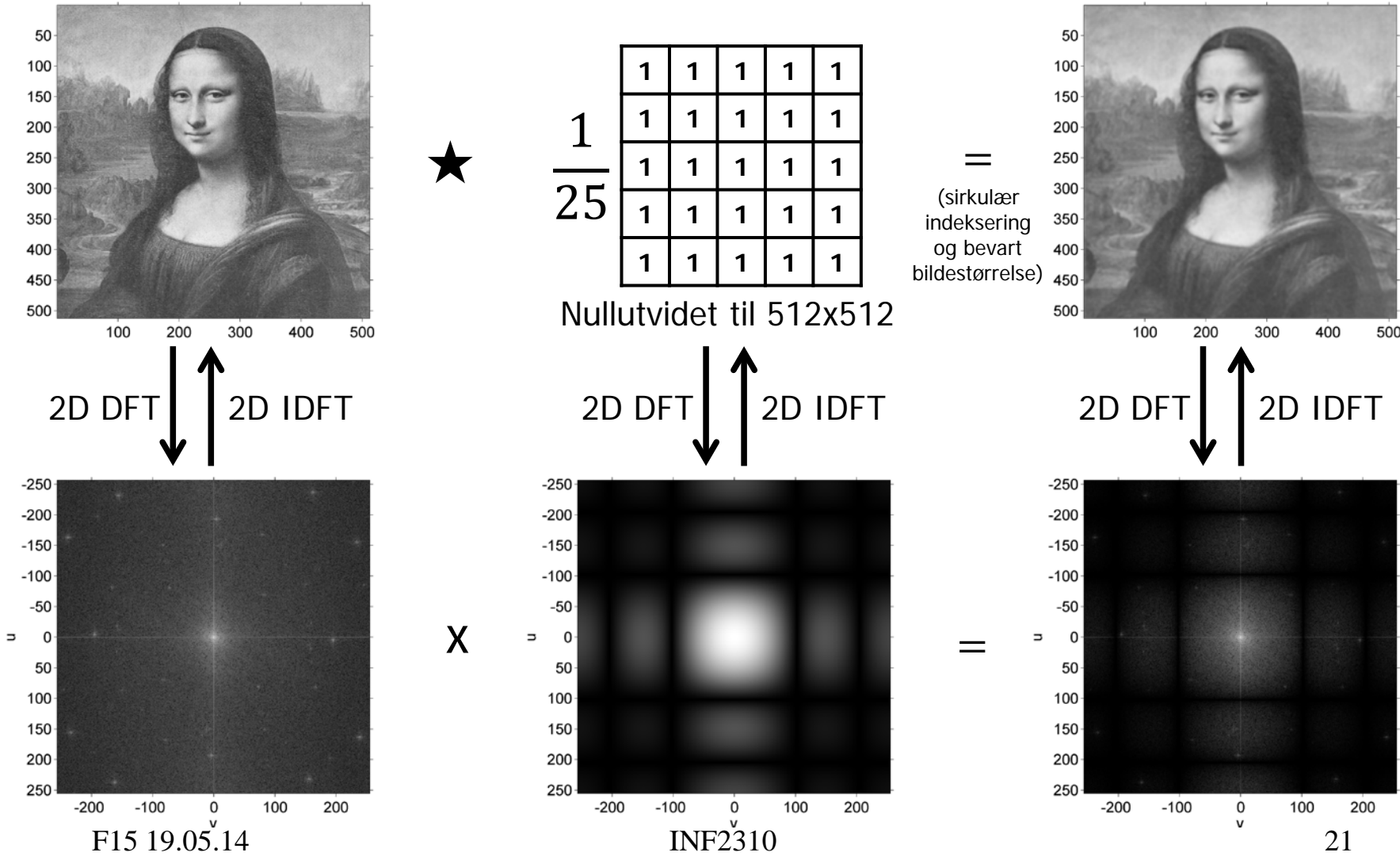
x



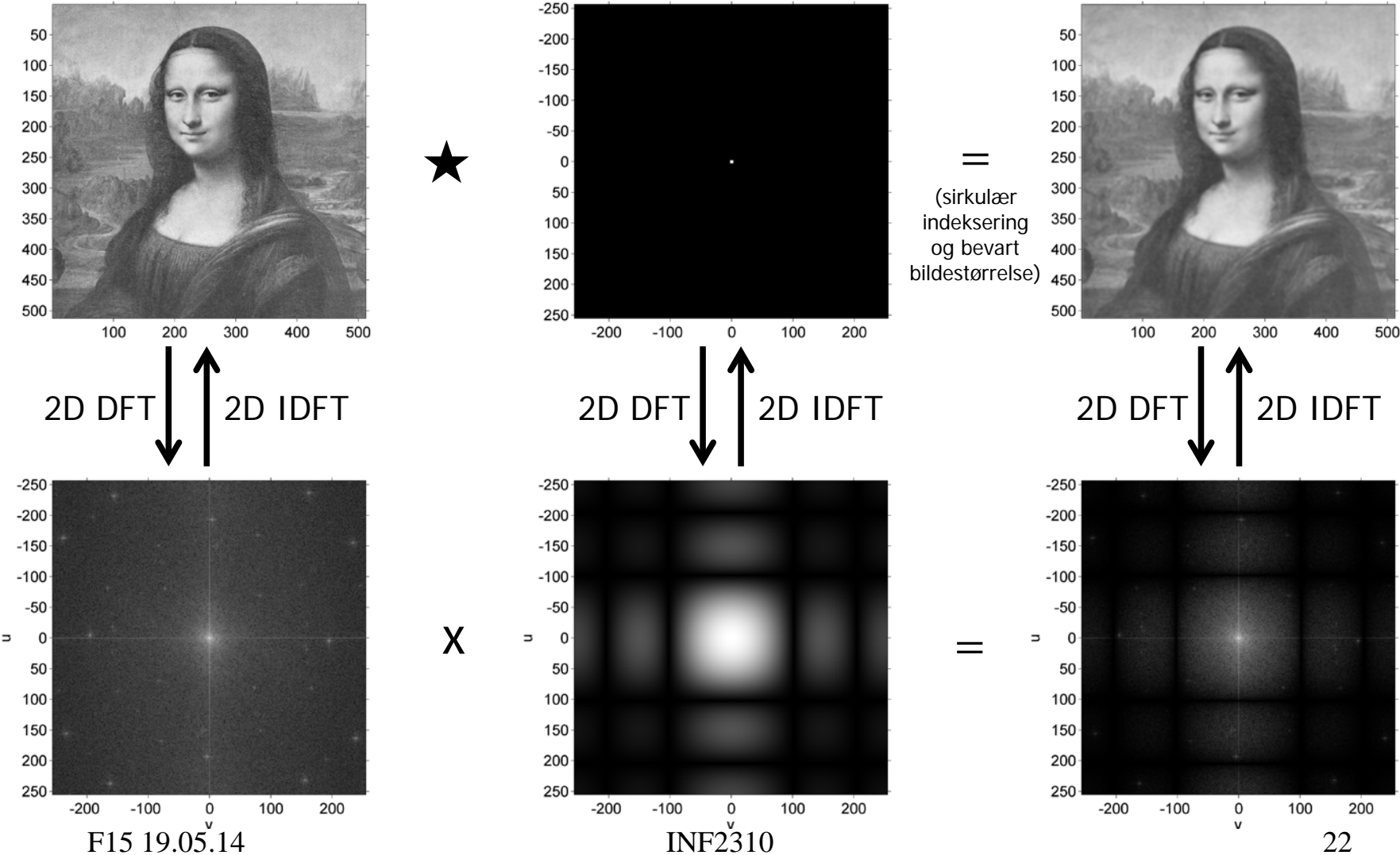
)

- Tilsvarende for imaginærdelen og sinus-bildet.
- Hvert punkt** i 2D DFT-en beskriver altså noe ved **hele bildet**.

# Eksempel: 5x5-middelverdifiltrering og konvolusjonsteoremet



# Eksempel: 5x5-middelverdifiltrering og konvolusjonsteoremet



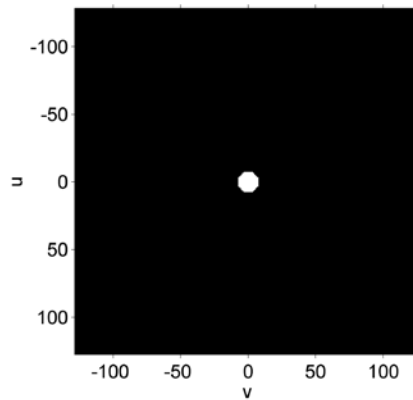
# Anvendelse av konvolusjonsteoremet

---

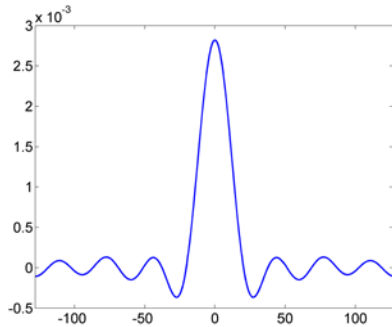
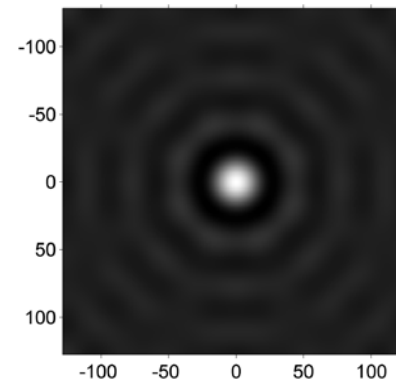
- Design av konvolusjonsfiltre med bestemte frekvenssegenskaper.
  - Designe konvolusjonsfilteret i Fourier-rommet slik at vi har bedre kontroll på dets frekvenssegenskaper.
- Analyse av konvolusjonsfiltre.
  - 2D DFT-en til et konvolusjonsfilter gir oss innblikk i hvordan filteret vil påvirke de forskjellige frekvenskomponentene.
- Rask implementasjon av større konvolusjonsfiltre.

# Filter-design i Fourier-domenet

## Romlig representasjon av ideelt lavpassfilter



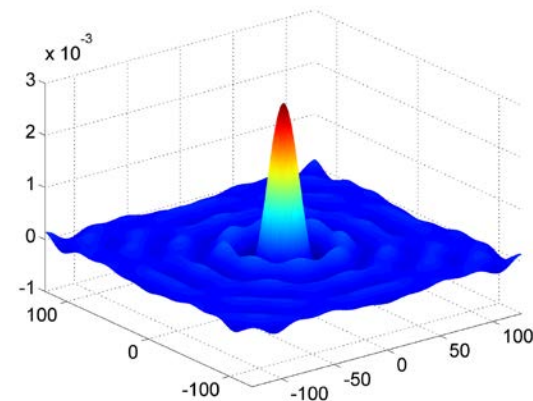
2D IDFT →



Den romlige representasjonen er en trunkert sinc-funksjon. Sinc-funksjonen er definert som:

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

og  $\text{sinc}(0) = 1$ .



- Vi får *ringing* i bildet.
  - Husk også tommelfingerregel fra forrige forelesning:  
Smal/bred struktur i bildet  $\leftrightarrow$  Bred/smål struktur i Fourier-spekteret



# Eksempel: Ideelt lavpassfilter

---



Original



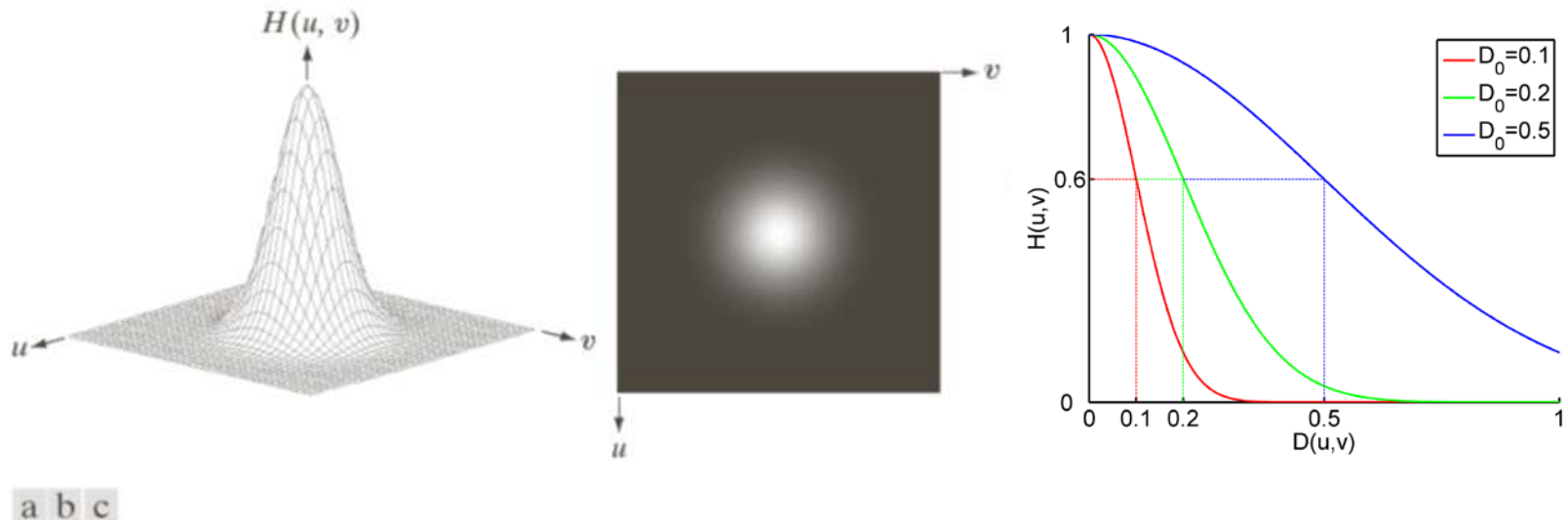
Filtrert med  $D_0 = 0,2$



Filtrert med  $D_0 = 0,3$

I god nok oppløsning kan striper/ringinger sees ut fra markante kanter i de to filtrerte bildene. Det er dette vi kaller *ringing*.

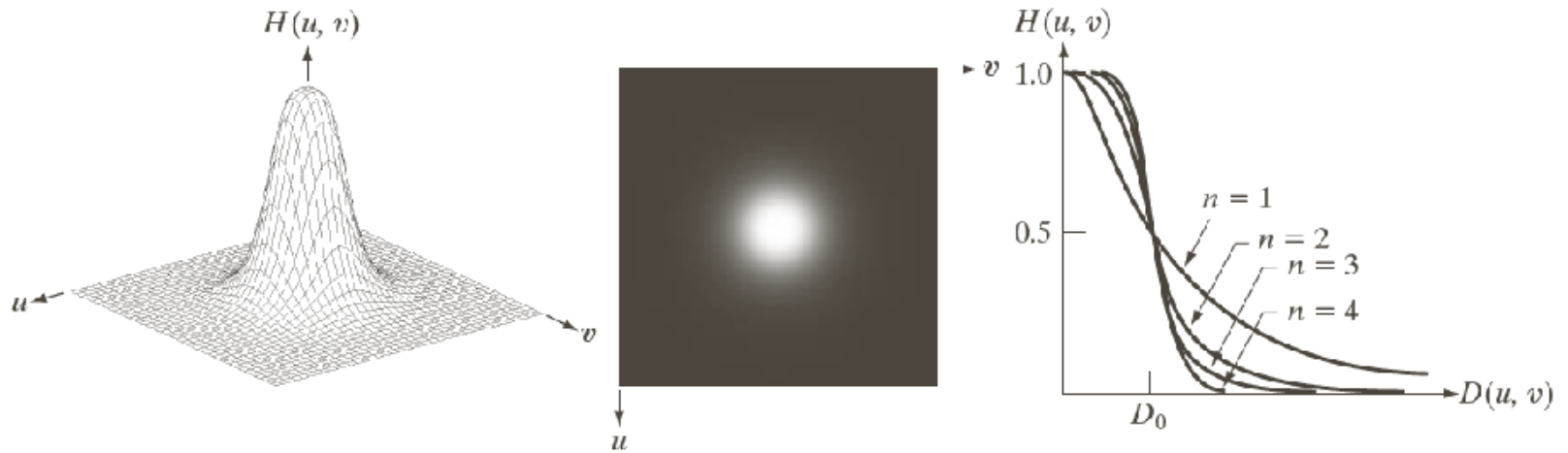
# Gaussisk lavpassfilter



**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  $D_0$ .

Husk tommelfingerregelen:  
Smal/bred struktur i bildet  $\Leftrightarrow$  Bred/smal struktur i Fourier-spekteret

# Butterworth lowpassfilter

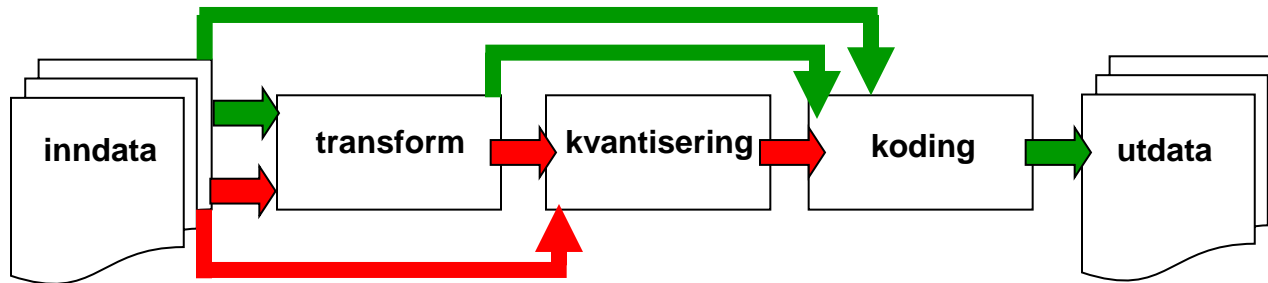


a b c

**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.

# Kompresjon


- Kompresjon kan deles inn i tre steg:
  - **Transform** - representer bildet mer kompakt.
  - **Kvantisering** - avrund representasjonen.
  - **Koding** - produser og bruk en kodebok.



- Kompresjon kan gjøres:
  - **Eksakt / tapsfri** (eng.: *lossless*) – følg de grønne pilene.
    - Kan da eksakt rekonstruere det originale bildet.
  - **Ikke-tapsfri** (eng.: *lossy*) – følg de røde pilene.
    - Kan da (generelt) ikke eksakt rekonstruere bildet.
    - Resultatet kan likevel være «godt nok».
  - Det finnes en mengde ulike metoder innenfor begge kategorier.

# Ulike typer redundans

---

- **Psykovisuell** redundans.  Mer generelt: **Irrelevant informasjon**: Unødvendig informasjon for anvendelsen, f.eks. for visuell betraktning av hele bildet.
  - Det finnes informasjon vi ikke kan se.
    - Eksempler på enkle muligheter for å redusere redundansen: Subsample eller redusere antall biter per piksel.
- **Interbilde**-redundans.
  - Likhet mellom nabobilder i en tidssekvens.
    - Eks.: Lagre noen bilder i tidssekvensen og ellers bare differanser.
- **Intersampel**-redundans.
  - Likhet mellom nabopiksler.
    - Eks.: Hver linje i bildet kan løpelengde-transformeres.
- **Kodings**-redundans.
  - Enkeltsymboler (enkeltpiksler) blir ikke lagret optimalt.
  - Gitt som gjennomsnittlig kodelengde minus et teoretisk minimum.
    - Velg en metode som er «grei» å bruke og gir liten kodingsredundans.

# Entropi

- Gjennomsnittlig informasjonsinnhold i sekvensen, også kalt gjennomsnittlig informasjon per symbol, er da:

$$H = \sum_{i=0}^{G-1} p_i I(s_i) = - \sum_{i=0}^{G-1} p_i \log_2 p_i$$

Hvis  $p(s_i)=0$  lar vi det tilhørende entropibidraget,  $0 \log_2 0$ , være 0.

- H er entropien til sekvensen av symbolene.
- **Entropien setter en nedre grense for hvor kompakt sekvensen kan representeres.**
  - Gjelder bare hvis vi koder hvert symbol for seg.

# Eksempel: Huffman-koding

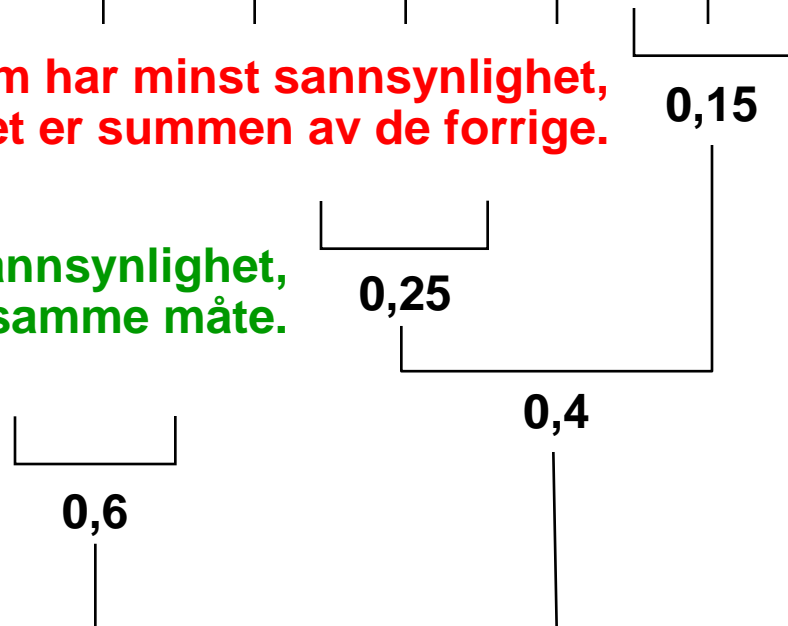
- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05

**Slå sammen de to gruppene som har minst sannsynlighet, Den nye gruppens sannsynlighet er summen av de forrige.**

**Finn de to som nå har minst sannsynlighet, og slå dem sammen på samme måte.**

**Fortsett til det er bare to igjen.**

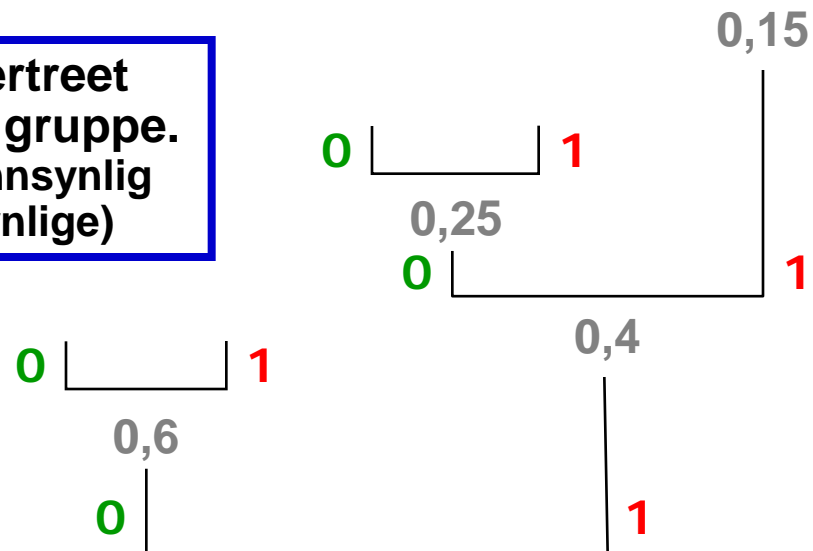


# Eksempel: Huffman-koding

- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05

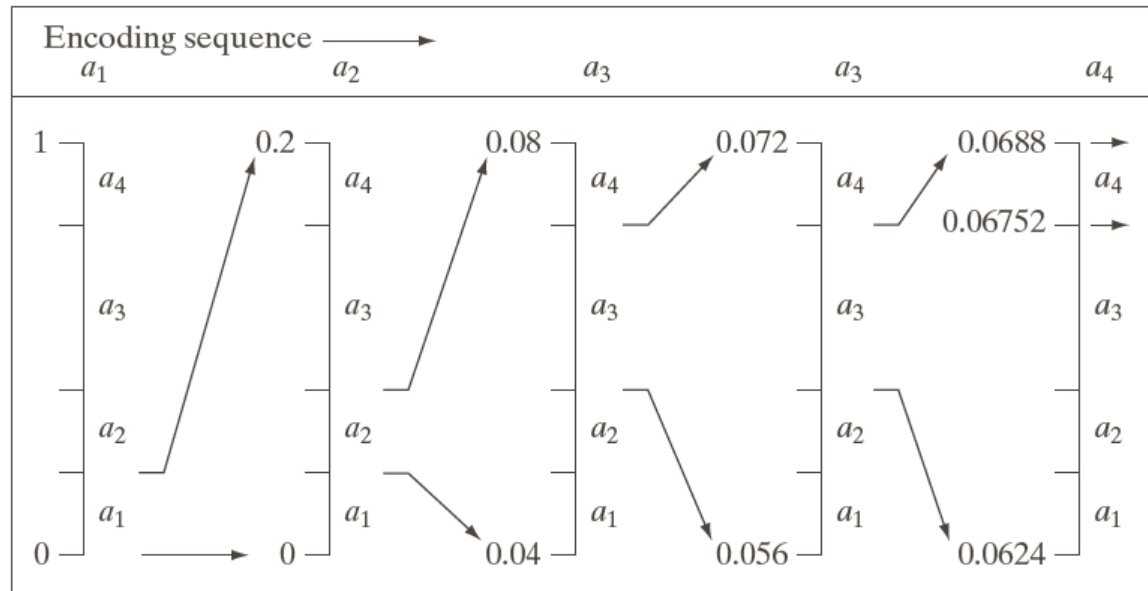
**Gå baklengs gjennom binærtreet og tilordne 0 eller 1 til hver gruppe. (F. eks. kode 0 til den mest sannsynlig og kode 1 til den minst sannsynlige)**





# Eksempel: Aritmetisk koding

- Sannsynlighetsmodell:  $P(a_1)=P(a_2)=P(a_4)=0,2$  og  $P(a_3)=0,4$
- Melding/symbolsekvens:  $a_1a_2a_3a_3a_4$



- $a_1$  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  $[0,06752, 0,0688)$

# Representasjon av intervall

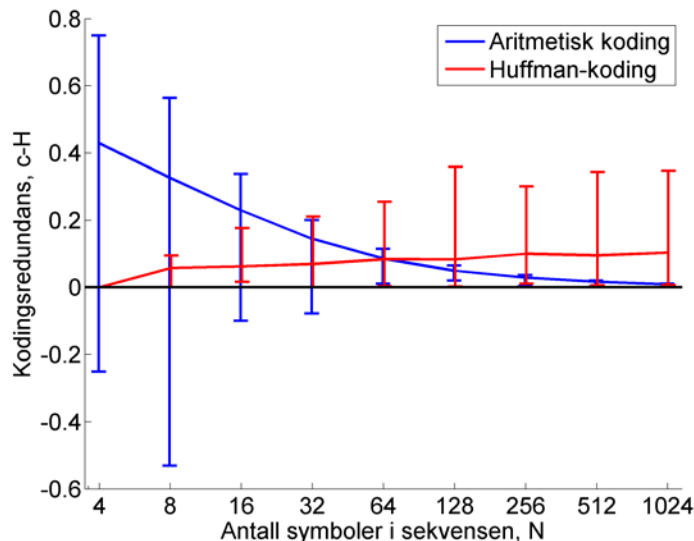
- Eksempel: Intervallet er  $[0,5232, 0,53184)$ .
  - Finner første forskjellig bit i binærrepresentasjonen:
    - Skriver  $0,5232_{10}$  på binærform:

$2 * 0,5232 = 1,0464$	$\Rightarrow d_1 = 1$ , rest = 0,0464
$2 * 0,0464 = 0,0928$	$\Rightarrow d_2 = 0$ , rest = 0,0928
$2 * 0,0928 = 0,1856$	$\Rightarrow d_3 = 0$ , rest = 0,1856
$2 * 0,1856 = 0,3712$	$\Rightarrow d_4 = 0$ , rest = 0,3712
$2 * 0,3712 = 0,7424$	$\Rightarrow d_5 = 0$ , rest = 0,7424
    - Skriver  $0,53184_{10}$  på binærform:

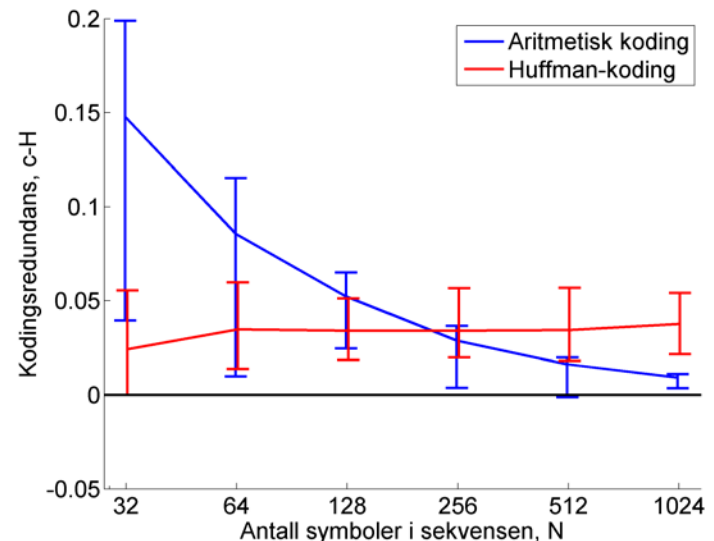
$2 * 0,53184 = 1,06368$	$\Rightarrow d_1 = 1$ , rest = 0,06368
$2 * 0,06368 = 0,12736$	$\Rightarrow d_2 = 0$ , rest = 0,12736
$2 * 0,12736 = 0,25472$	$\Rightarrow d_3 = 0$ , rest = 0,25472
$2 * 0,25472 = 0,50944$	$\Rightarrow d_4 = 0$ , rest = 0,50944
$2 * 0,50944 = 1,01888$	$\Rightarrow d_5 = 1$ , rest = 0,01888
  - Siden den siste resten til den øvre grensa er større enn 0 må  $0,10001_2 = 0,53125_{10}$  være et tall i intervallet.
  - $0,53125_{10}$  er det tallet i intervallet med kortest binær-representasjon.

# Aritmetisk koding vs Huffman-koding

- Aritmetisk koding: Bedre kompresjon jo lenger symbolsekvensen er.
- Huffman-koding: Bedre kompresjon jo flere symboler i alfabetet.
- Aritmetisk koding komprimerer typisk litt bedre enn Huffman-koding, men er mer regnekrevende å utføre.
  - For vanlige bilder, dvs. med relativt få symboler i alfabetet og (potensielt sett) mange symboler i sekvensen.



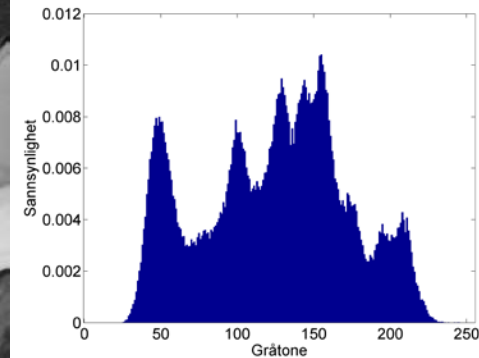
Med  $G=4$  symboler i alfabetet.



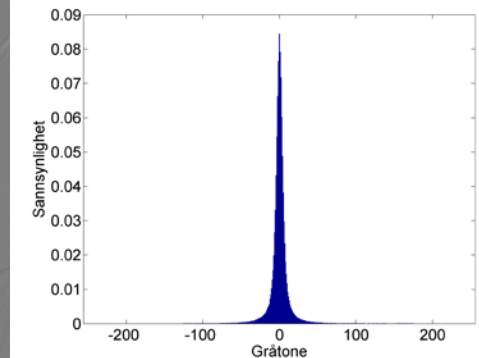
Med  $G=20$  symboler i alfabetet.

# Differansetransform

- Utnytter at horisontale nabopiksler ofte har ganske lik gråtone.
- Gitt en rad i bildet med gråtoner:  $f_1, \dots, f_N$  der  $0 \leq f_i \leq 2^b - 1$
- Transformer (reversibelt) til  $g_1 = f_1, g_2 = f_2 - f_1, \dots, g_N = f_N - f_{N-1}$
- Merk at:  $-(2^b - 1) \leq g_i \leq 2^b - 1$ 
  - Må bruke  $b+1$  biter per  $g_i$  hvis vi skal tilordne like lange kodeord til alle mulig verdier.
- Ofte er de fleste differansene nær 0.
  - Naturlig binærkoding av differansene er ikke optimalt.



Entropi  $\approx 7,45 \Rightarrow CR \approx 1,1$



Entropi  $\approx 5,07 \Rightarrow CR \approx 1,6$

# Løpelengde-transform

---

- Ofte inneholder bildet objekter med lignende gråtoner, f.eks. svarte bokstaver på hvit bakgrunn.
- Løpelengde-transformen (eng.: *run-length transform*)  
**utnytter når horisontale nabopiksler har samme gråtone.**
  - Merk: Krever ekte likhet, ikke bare omtrent like.
  - Løpelengde-transformen komprimerer bedre ettersom kompleksiteten i bildet blir mindre.
- Løpelengde-transformen er reversibel.
- Hvis pikselverdiene til en rad er:  
33333355555555544777777 (24 tall)
- Så starter løpelengde-transformen fra venstre og finner tallet 3 gjentatt 6 ganger etter hverandre, og returnerer derfor tallparet (3,6). **Formatet er: (tall, løpelengde)**
- For hele sekvensen vil løpelengdetransformen gi de 4 tallparene:  
(3,6), (5,10), (4,2), (7,6) (merk at dette bare er 8 tall)
- Kodingen avgjør hvor mange biter vi bruker for å lagre tallene.

# Eksempel: LZW-transform

- Alfabetet: a, b og c med koder 0, 1 og 2, henholdsvis.
- Meldingen: ababcbabababababababab (18 symboler)
- LZW-sender: ny streng = **sendt streng** **pluss** **neste usendte symbol**
- LZW-mottaker: ny streng = **nest siste streng** **pluss** **første symbol i sist tilsendte streng**

Ser	Sender	Senders liste	Mottar	Tolker	Mottakers liste
		a=0, b=1, c=2			a=0, b=1, c=2
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		

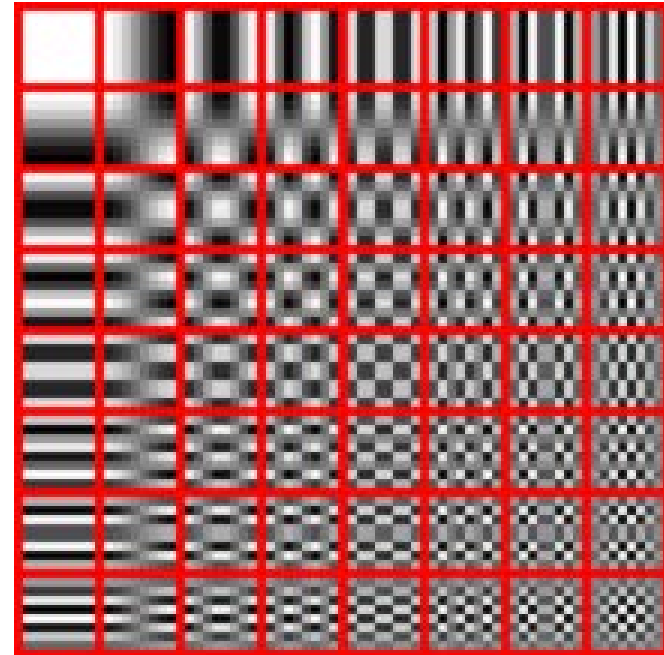
- » Vi mottar kode 7, men denne koden finnes ikke i listen!
- » Fra ny-streng-oppskriften vet vi at kode 7 ble laget ved: ba + ?
- » Siden kode 7 nå sendes, må: ? = b => 7 = ba + b = bab

# 2D diskret cosinus-transform

- Grunnpilaren i ikke-tapsfri JPEG-kompresjon er 2D DCT:

$$F(u, v) = \frac{2}{\sqrt{MN}} c(u)c(v) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cos\left[\frac{\pi u}{M}\left(x + \frac{1}{2}\right)\right] \cos\left[\frac{\pi v}{N}\left(y + \frac{1}{2}\right)\right], \quad c(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{hvis } \xi = 0 \\ 1 & \text{ellers} \end{cases}$$

- Sterkt relatert til 2D DFT.
- I JPEG transformerer vi 8x8-blokker så vi bruker bare de 64 «8x8-cosinus-bildene»:
  - For hvert bilde vi går til høyre eller ned så økes den tilhørende frekvenskomponenten med 0,5.
  - Husk: I 2D DFT økte frekvenskomp. med 1, som lagde par med like cosinus-bilder.
  - Husk: I 2D DFT hadde vi også noen sinus-bilder.
  - 2D DCT-koeffisientene beregnes analogt med det vi gjorde for 2D DFT; summere punktproduktet mellom 8x8-blokken og hvert «cosinus-bilde».
  - 2D DCT beregnes hurtig ved å forhåndsberegnes de 64 «8x8-cosinus-bildene».



# Rekonstruksjonsfeil i gråtonebilder

- JPEG-kompresjon kan gi **8×8-piksels blokk-artefakter**, **glatting** og **ringinger**.
- Avhengig av vektmatrisen
  - som bestemmer hvor mange koeffisienter som lagres, og hvor presist disse lagres.



F15 19.05.14



INF2310



# Blokk-artefakter

---

- Blokk-artefaktene øker med kompresjonsraten.



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

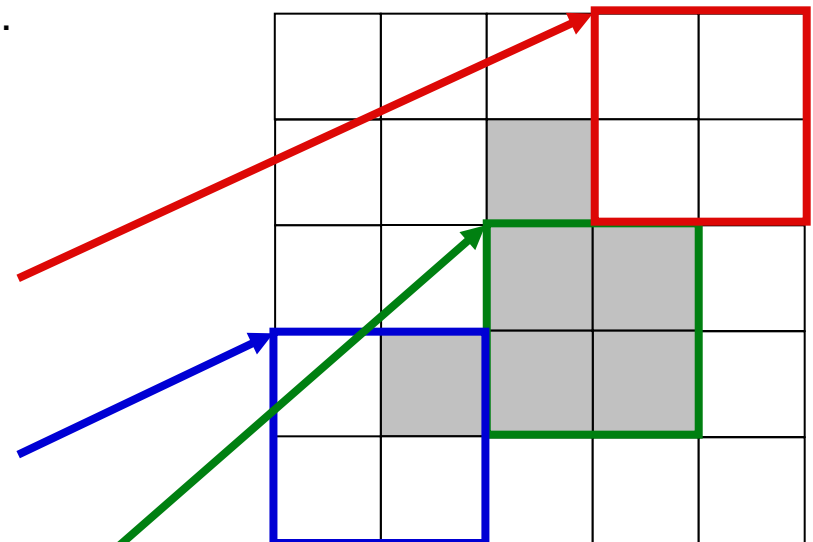
# Morfologi: Tre sentrale begrep

- Et **strukturelement** for et binært bilde er et **naboskap**.

- Typisk definert ved en binær matrise der 1 markerer med i naboskapet.

- 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.



I figurene markerer grått med i mengden (forgrunns piksel), og hvitt ikke med (bakgrunns piksel).

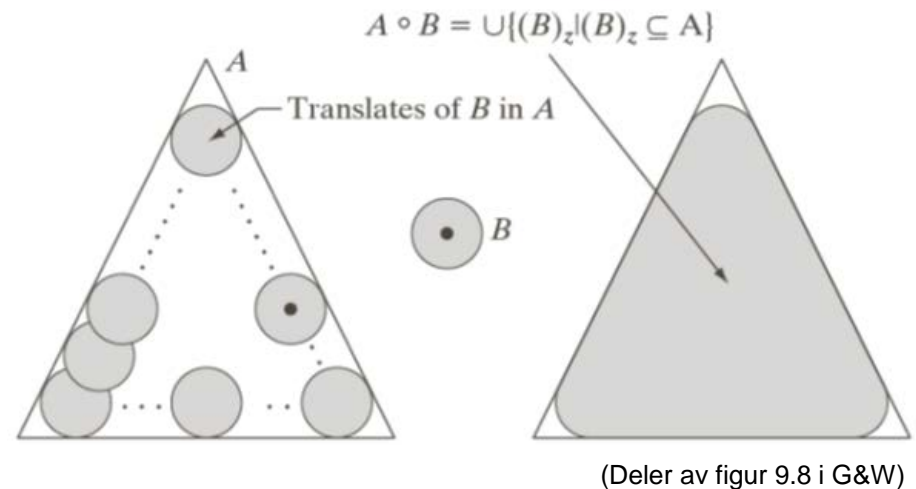
# Anvendelse av erosjon: Kantdeteksjon

- Erodering fjerner piksler langs omrisset av et objekt.
- Vi kan finne kantene av objektene i bildet ved å subtrahere et erodert bilde fra originalbildet:  $g = f - (f \ominus S)$
- Det benyttede strukturelementet avgjør kantens tilkoblingstype:

Et bilde	erodert med	gir	=>	differanse	
<pre> 001111011110 011111111110 011111111110 111101111111 011111111111 011111111110 011111111110 00000111000           </pre>	<pre> 010 111 010           </pre>	<pre> 000000000000 001111011100 001101111100 011000111110 001101111110 001111111110 000001110000 000000000000           </pre>	=>	<pre> 001111011110 010000100100 010010000010 100101000001 010010000001 010000000010 011110001110 000001110000           </pre>	<div style="border: 1px dashed green; padding: 5px;"> <p>Sammenhengende kanter hvis (og bare hvis) man bruker <b>8-tilkobling</b></p> </div>
	<pre> 111 111 111           </pre>	<pre> 000000000000 000110001000 001000111100 001000111100 001000111100 001111111110 000000100000 000000000000           </pre>	=>	<pre> 001111011110 011001111010 010111100010 110101000011 010111100011 010000000010 011111011110 000001110000           </pre>	<div style="border: 1px dashed green; padding: 5px;"> <p>Sammenhengende kanter ved bruk av <b>4-tilkobling</b></p> </div>

# Geometrisk tolkning av åpning

- Tenk at strukturelementet definerer **størrelsen og formen til spissen av en tusjenn.**
- 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 enhver åpning avsluttes med en dilasjon).



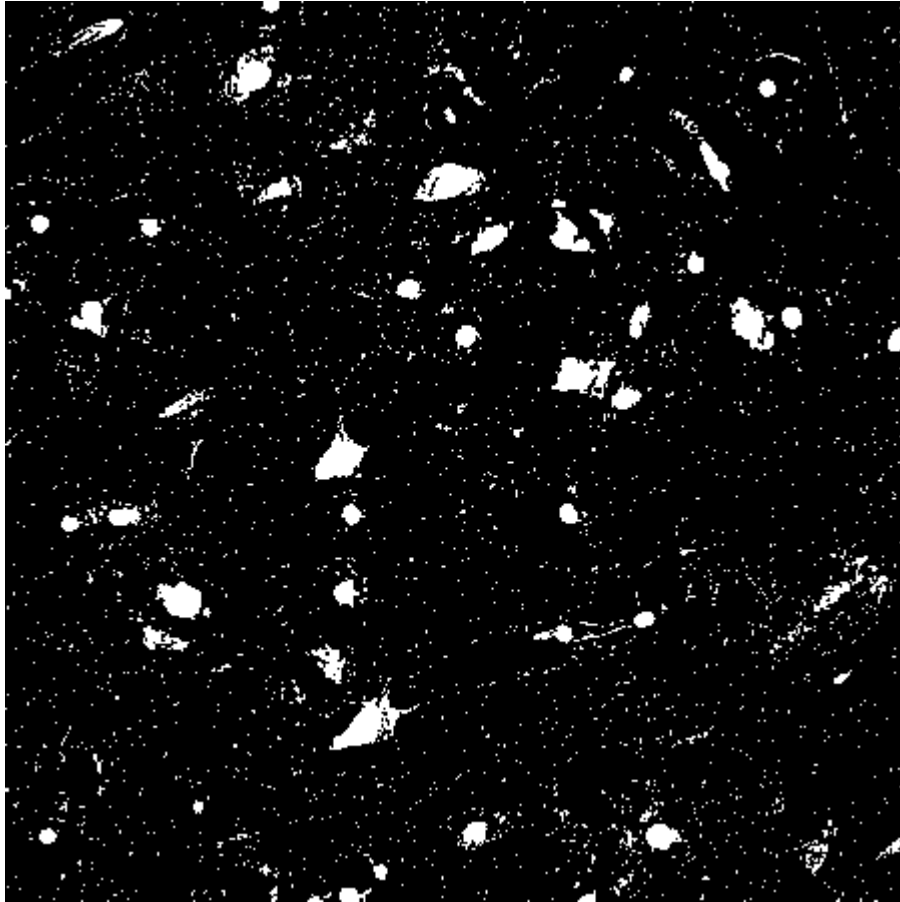
Åpning er **idempotent**:

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

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

# Eksempel: Støyfjerning med åpning

---



Åpning med et (7x7) sirkulært strukturelement

I bildene markerer hvit forgrunn og svart bakgrunn.  
(Bildene er hentet fra  
<http://homepages.inf.ed.ac.uk/rbf/HIPR2/>)

# Geometrisk tolkning av lukking

□ Vi kan benytte **samme metafor** som for åpning:

- Strukturelementet definerer størrelsen og formen til spissen av en tusjpen.
- Man holder tusjen vinkelrett på tegneflaten og fargelegger så mye man har lov til.

□ **Denne gangen** er det bare tillatt å **fargelegge utenfor objekter**.

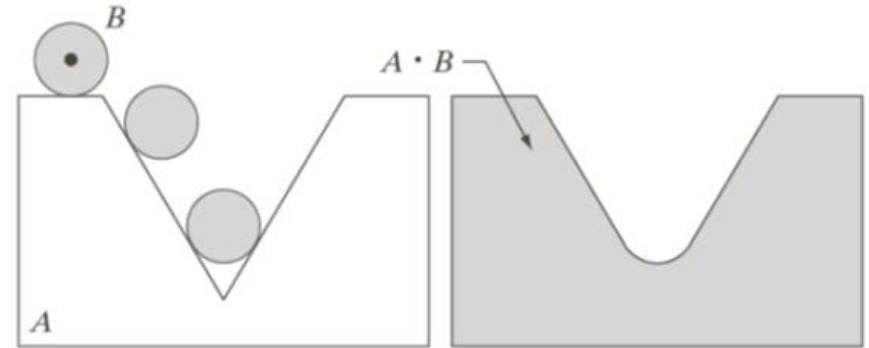
- En detalj: Denne gangen skal tusjen holdes speilvendt ( $180^\circ$  rotert) av strukturelementet.

□ **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 enhver lukking avsluttes med en erosjon).



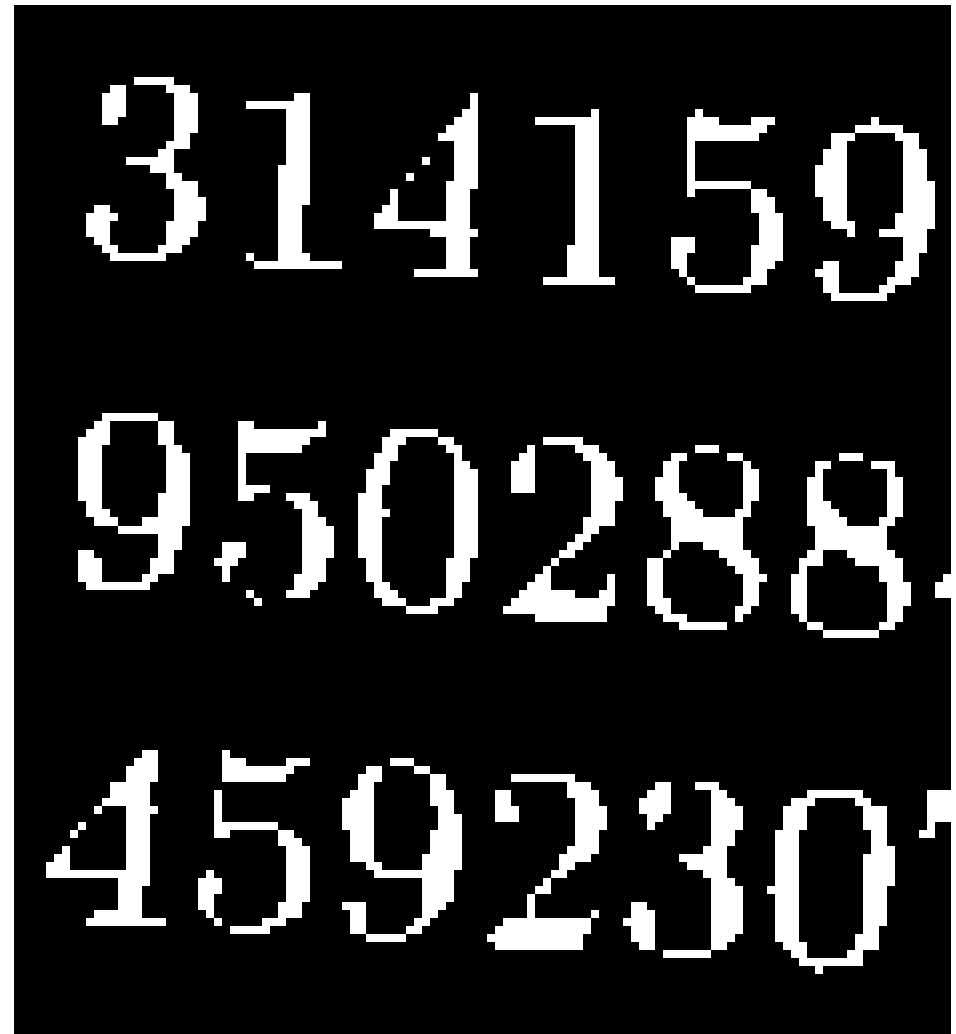
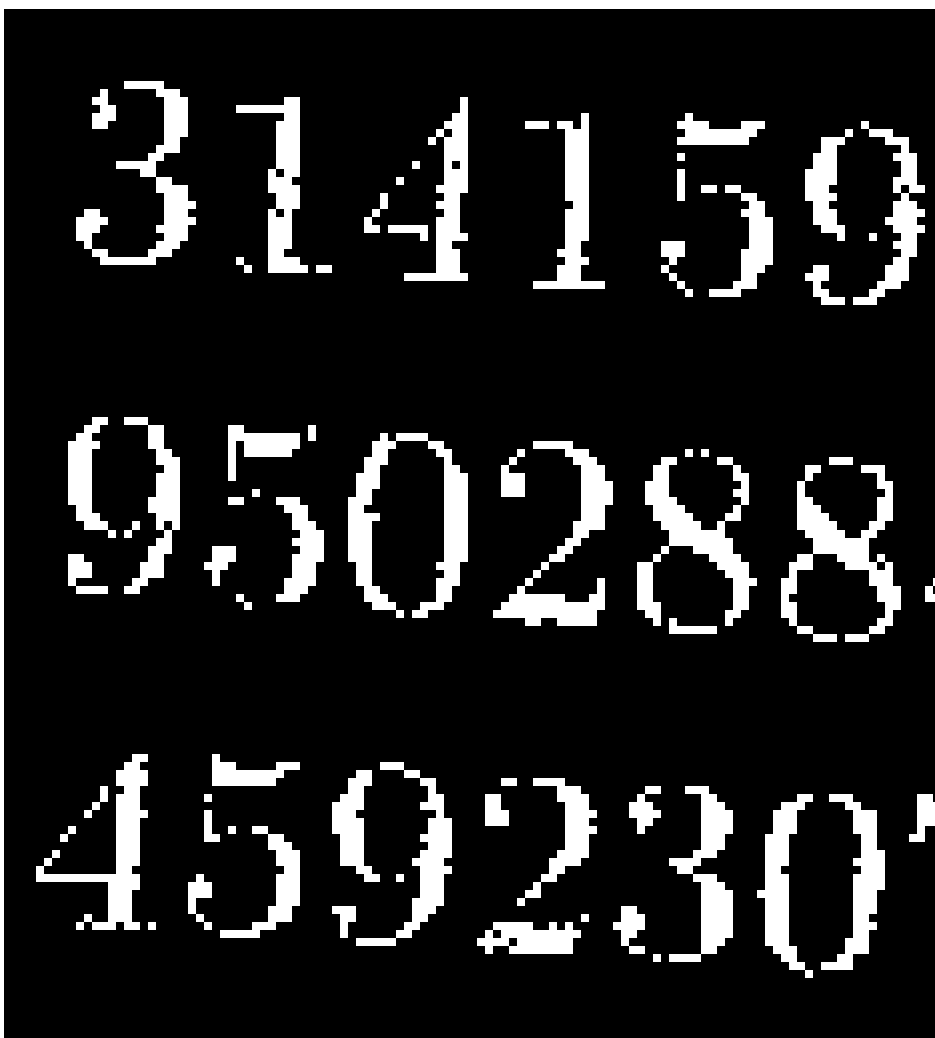
(Deler av figur 9.9 i G&W)

Også lukking er **idempotent**:

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

# Eksempel: Filtrering ved lukking

---



Lukking med et (3x3) kvadratisk strukturelement

# Eksempel: «Hit-or-miss»

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 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
    
```

Et bilde A

```

0 1 0
1 1 1
0 1 0
    
```

Strukturelement  
 $S_1$

```

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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 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 0
0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    
```

Resultat etter erosjon med  $S_1$

```

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1
1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1
1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1
1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
    
```

$A^c$  - komplementet til bildet  
(er 1 utenfor randen)

```

1 0 1
0 0 0
1 0 1
    
```

Strukturelement  
 $S_2$

```

1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1
1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1
1 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1
1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1
    
```

$A^c$  erodert med  $S_2$

```

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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 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
0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    
```

«Hit-or-miss»-resultatet

Logisk AND av de  
to delresultatene