INF 2310 – Digital bildebehandling

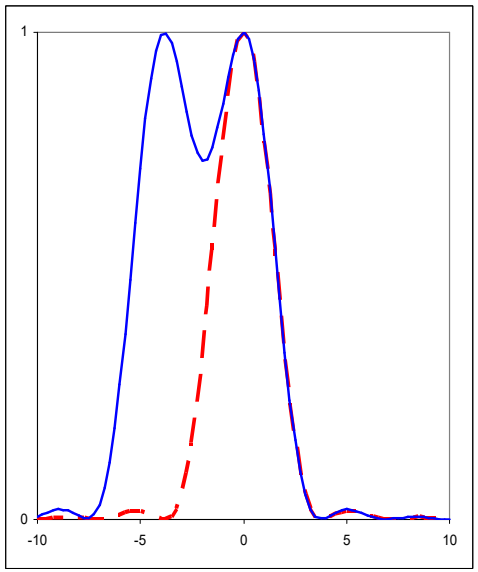
Oppsummering FA, mai 2020:

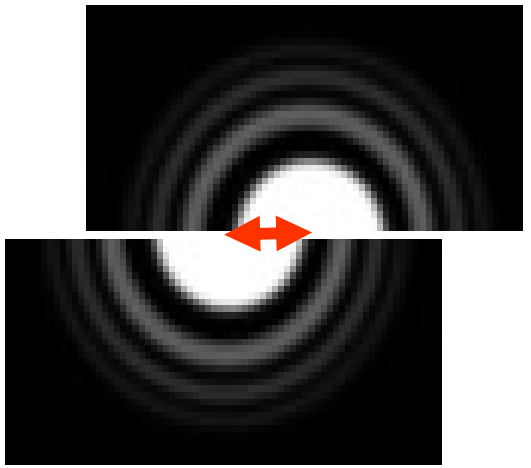
Innledning	F1
Sampling og kvantisering	F2
Segmentering ved terskling	F6
Filtrering i bildedomenet	F7, F8
Morfologiske operasjoner	F11
Kompresjon og koding	F12, F13
Farger og fargerom	F14

Rayleigh-kriteriet

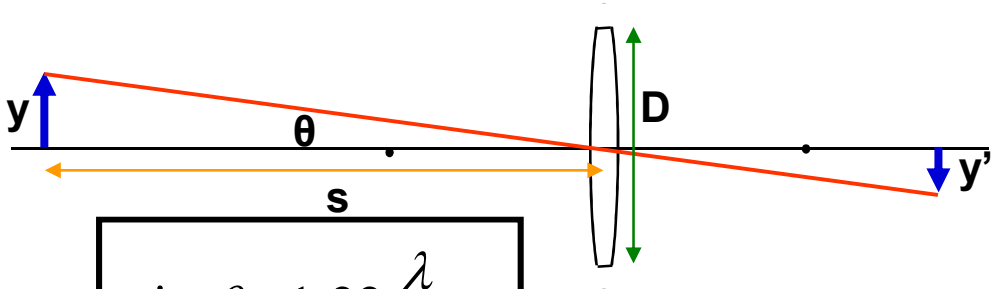
- To punkt-kilder kan adskilles hvis de ligger slik at sentrum i det ene diffraksjonsmønstret faller sammen med den første mørke ringen i det andre.
  - Vinkelen mellom dem er da gitt ved

    $\sin \theta = 1.22 \lambda / D$ radianer.
  - Dette er "Rayleigh-kriteriet".
  - *Vi kan ikke se detaljer som er mindre enn dette.*





Hvor små detaljer kan en linse oppløse?



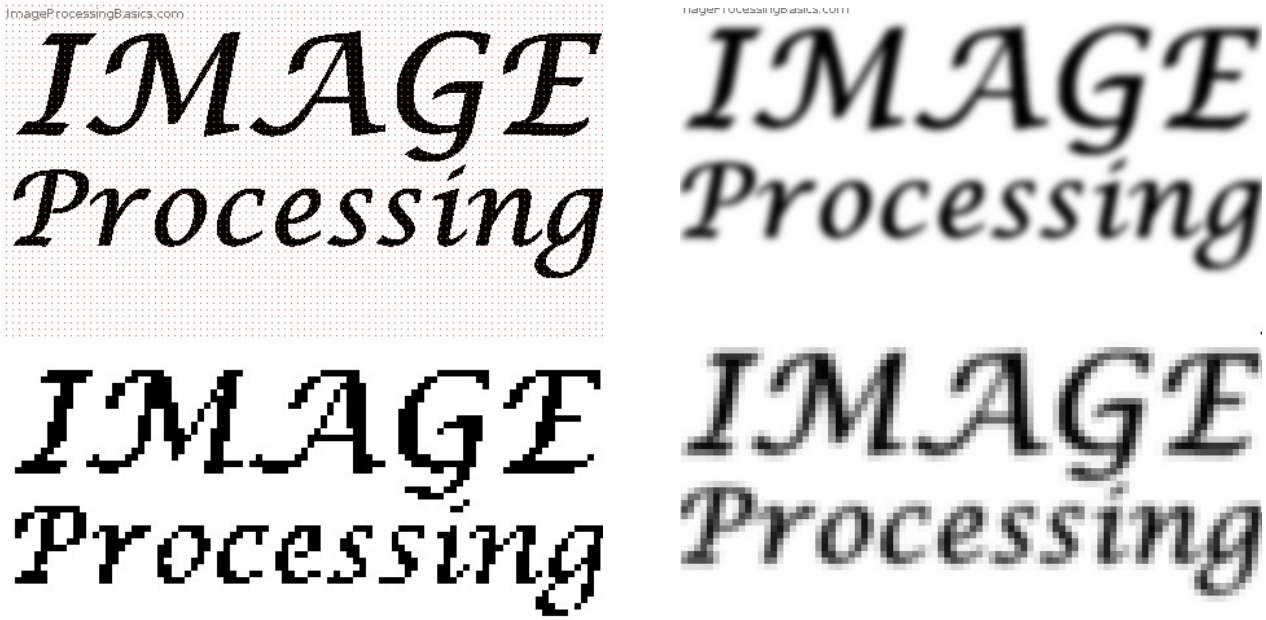
- Vinkeloppløsningen er gitt ved $$\sin\theta = 1.22\frac{\lambda}{D}$$
- Tangens til vinkelen θ er gitt ved $$tg(\theta) = \frac{y}{s}$$
- For små vinkler er sin(θ) = tg(θ) = θ, når vinkelen θ er gitt i radianer.
- **=> Den minste detaljen vi kan oppløse:** $$\frac{y}{s} = 1.22\frac{\lambda}{D} \Rightarrow y = 1.22\frac{s\lambda}{D}$$

Samplingsteoremet (Shannon/Nyquist)

- Anta at det kontinuerlige bildet er båndbegrenset, dvs. det inneholder ikke høyere frekvenser enn f_{max}
- Det kontinuerlige bildet kan rekonstrueres fra det digitale bildet dersom samplingsraten $f_s=1/T_s$ er større enn 2 f_{max} (altså $T_s < ½T_0$)
- 2 f_{max} kalles Nyquist-raten
- I praksis oversampler vi med en viss faktor for å kunne få god rekonstruksjon

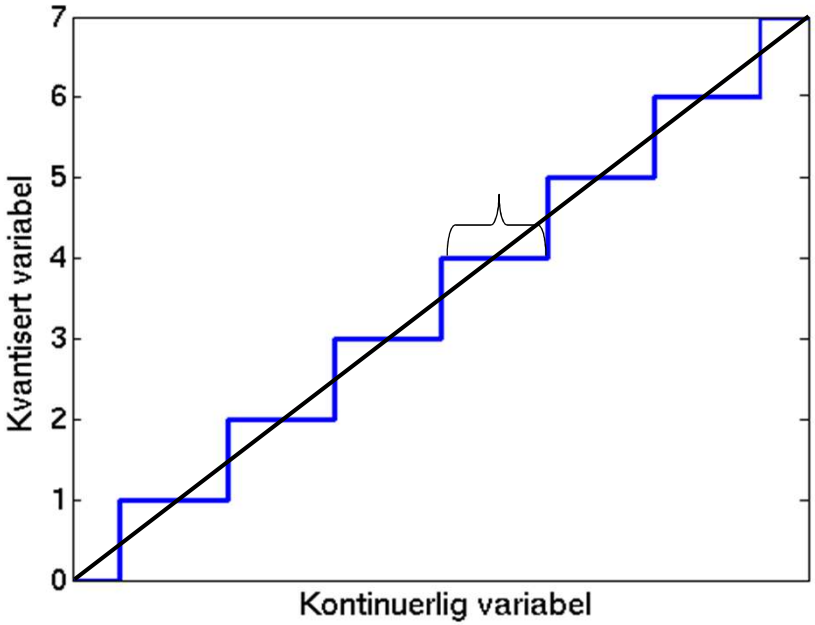
Anti-aliasing

- Ved *anti-aliasing* fjerner/demper vi de høyere frekvensene i bildet **før** vi sampler



Kvantisering

- Hvert piksel lagres vha. *n* biter
- Pikselet kan da inneholde heltallsverdier fra 0 til 2^n-1
- Eks 3 biter:



Kvantiseringsfeil

- Kvantiseringsfeil
  - Summen av hver piksels avrundingsfeil
  - Kan velge intervaller og tilhørende rekonstruksjonsintensiteter for å minimere denne => Ikke nødvendigvis uniform fordeling

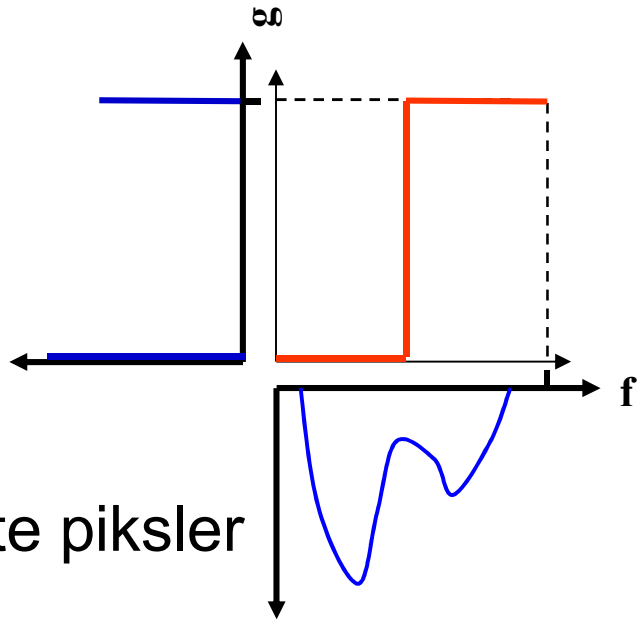
- Sentrale stikkord:
  - Lagringsplass
  - Behov for presisjon/akseptabelt informasjonstap
  - Hardware-kompleksitet, eller fysiske begrensninger

- Merk: Fremvisning og videre analyse av det kvantiserte bildet kan stille ulike krav til presisjon

Terskling – en gråtonetransform

- Hvis vi har grunn til å anta at objektene f.eks. er lysere enn bakgrunnen, så kan vi sette en terskel T og lage oss et binært ut-bilde g(x,y) ved mappingen:

$$g(x,y) = \begin{cases} 0 \text{ hvis } & f(x,y) \le T \\ 1 \text{ hvis } & f(x,y) > T \end{cases}$$



- Da har vi fått et ut-bilde g(x,y) med bare to mulige verdier.
- Med riktig valg av T vil nå de fleste piksler med g(x,y)=1 være objekt-piksler.

Klassifikasjonsfeil ved terskling

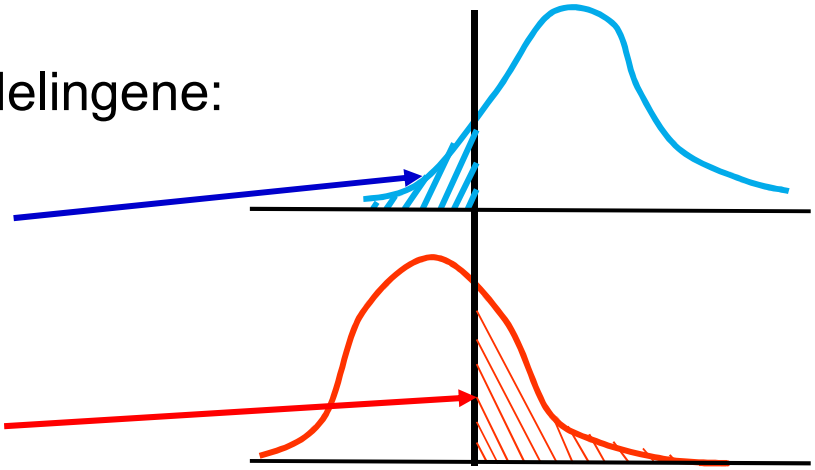
- Anta at histogrammet er en sum av to fordelinger *b(z)* og *f(z)*, *b* og *f* er ***normaliserte*** bakgrunns- og forgrunns-histogrammer.
- La *F* og *B* være ***a priori sannsynlighet*** for bakgrunn og forgrunn
- Det normaliserte histogrammet til bildet kan da skrives

$$p(z) = B \cdot b(z) + F \cdot f(z)$$

- Sannsynlighetene for å feilklassifisere et piksel, gitt en terskelverdi, finner vi fra de normaliserte fordelingene:

$$E_B(t) = \int_{-\infty}^{t} f(z)dz$$

$$E_F(t) = \int_{t}^{\infty} b(z)dz$$



Finn den T som minimerer feilen

$$E(t) = F\int_{-\infty}^{t} f(z)dz + B\int_{t}^{\infty} b(z)dz$$

- Deriverer E(t) mhp. t vha. Leibnitz regel for derivasjon av integraler.
- Setter den deriverte lik 0 og får:

VIKTIG !!!

$$\frac{dE(t)}{dt} = 0 \quad \Rightarrow \quad F \cdot f(T) = B \cdot b(T)$$

- Merk at dette er en generell løsning som gir minst feil.
- Det er ingen restriksjoner mht. fordelingene *b* og *f* !!

En enkel tersklings-algoritme

- Start med terskel-verdi t = middelverdien til alle pikslene i bildet.
  - Finn middelverdien ($\mu_1(t)$) av alle piksler som er mørkere enn terskelen
  - Finn middelverdien ($\mu_2(t)$) av alle piksler som er lysere enn terskelen.
- La ny terskel-verdi være $$t = \frac{1}{2}\left(\mu_1(t) + \mu_2(t)\right)$$
- Gjenta de to punktene ovenfor til terskelen ikke flytter seg mer.
- Dette kalles Ridler og Calvard's metode
- Dette gjøres i algoritmen på side 742 i GW.
  - Hvilke betingelser må være oppfylt for at metoden skal virke?
  - Når vil denne metoden svikte?

Otsu's metode - motivasjon

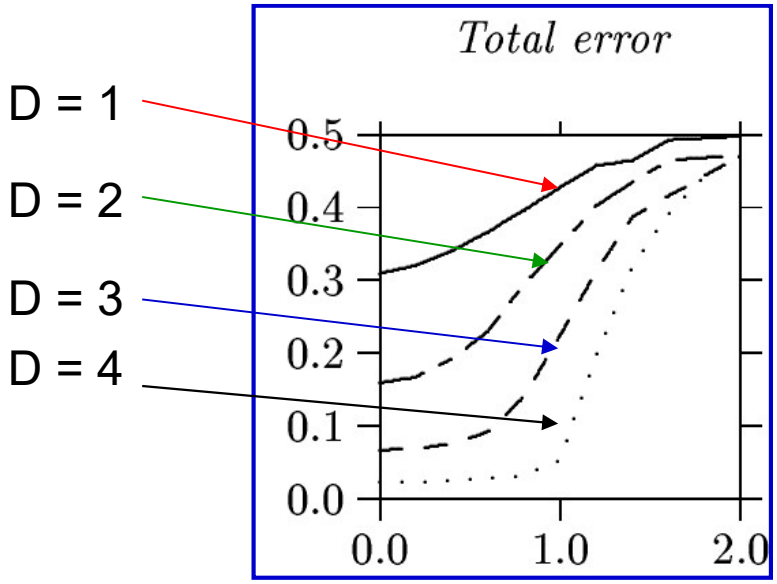
- Anta at vi har et gråtonebilde med G gråtoner, med normalisert histogram p(i).
- Anta at bildet inneholder to populasjoner av piksler, slik at pikslene innenfor hver populasjon er noenlunde like, mens populasjonene er forskjellige.
- **Målsetting:**

  Vi vil finne en terskel T slik at hver av de to klassene som oppstår ved tersklingen blir mest mulig homogen, mens de to klassene bli mest mulig forskjellige.

  - **variansen i hver av de to klassene er minst mulig.**
  - **avstanden mellom middelverdiene er størst mulig.**

Effekten av *a priori* sannsynlighet

- Total tersklingsfeil mot $\log_{10}(P_1/P_2)$ for fire verdier av $\mu_2-\mu_1 = D\sigma$:



- Feilen øker raskt ved $\log_{10}(P_1/P_2) \approx 1$
- => Otsu's metode bør bare brukes

  når $0.1<P_1/P_2<10$.
- Det samme gjelder for Ridler & Calvard.
- Det finnes gode alternativer !

”Minimum feil” terskling

- Kittler og Illingworth (1985) beregner et kriterium for alle mulige terskelverdier:

$$J(t) = 1 + 2\left[P_1(t)\ln\sigma_1(t) + P_2(t)\ln\sigma_2(t)\right] - 2\left[P_1(t)\ln P_1(t) + P_2(t)\ln P_2(t)\right]$$

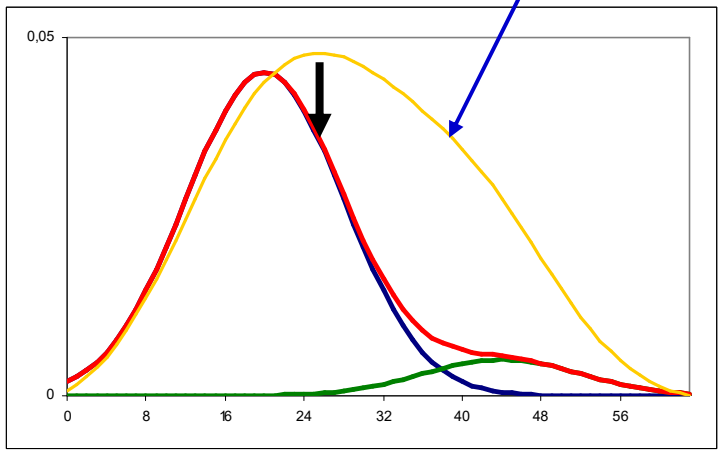
- For hver t-verdi estimeres alle fem parametrene.
- Beregn J(t) for alle t og finn ***minimum***, eller finn løsning iterativt.
- Kriterie-funksjonen har lokale minima ved endene av gråtoneskalaen.
- En uheldig start-verdi i et iterativt søk kan gi meningsløs terskelverdi.
- Bruk Otsu’s terskelverdi som start-verdi i et iterativt søk.

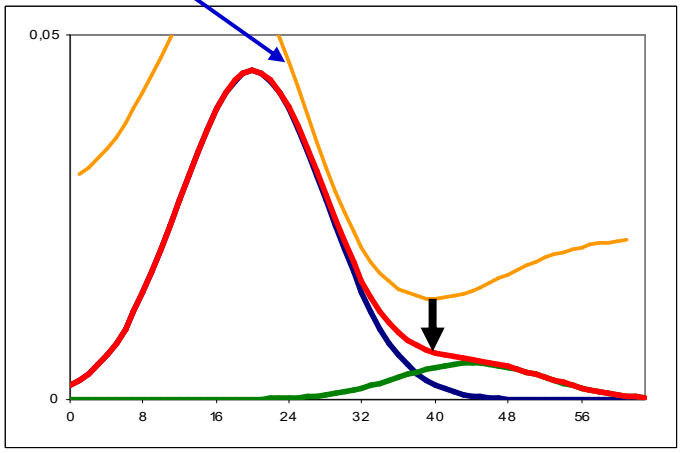
En sammenligning

- For B =0.9, F =0.1, μ_B = 20, μ_F= 44, σ_F = σ_B = 8:

- Otsu's terskel (venstre) gir signifikant feilterskling.
- Kittler og Illingworth's terskel (høyre) er OK.

Kriteriefunksjoner: $\sigma_B^2(t)$ og $J(t)$





Filtrering: 2-D konvolusjon

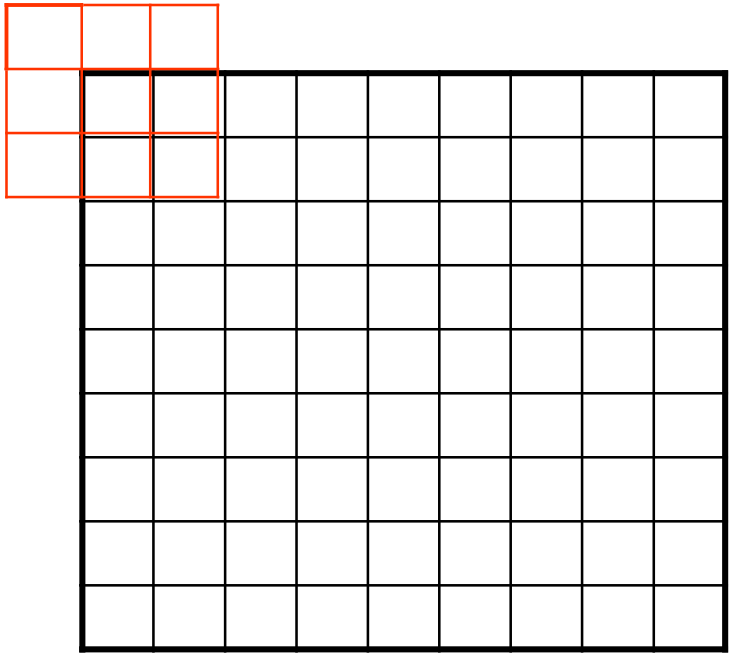
$$g(x,y) = \sum_{j=x-w_1}^{x+w_1} \sum_{k=y-w_2}^{y+w_2} h(x-j, y-k) f(j,k)$$

- For å regne ut resultatet av en konvolusjon for posisjon (*x,y)* :
  - Roter konvolusjonsfilteret 180 grader
  - Legg filteret over bildet slik at origo overlapper posisjon (x,y) i bildet.
  - Multipliser hver vekt i filteret med underliggende pikselverdi.
  - **Summen av produktene gir verdien for *g(x,y)* i posisjon (*x,y)*.**

- For å regne ut resultatet for alle posisjoner:
  Flytt filteret piksel for piksel og gjenta operasjonene over.
- Vi bruker notasjonen g = h * f

Praktiske problemer

- Kan ut-bildet ha samme piksel-representasjon som inn-bildet?
- Trenger vi et mellom-lager?
- Hva gjør vi langs bilde-randen?



- Anta at bildet er M x N piksler
- Anta at filteret er m x n

  ($m=2m_2+1$, $n=2n_2+1$)
- Uberørt av rand-effekt:

  (M-m+1)x(N-n+1)

  3x3: (M-2)x(N-2)

  5x5: (M-4)x(N-4)

Hva gjør vi langs randen?

Alternativer:

1. Sett g(x,y)=0
2. Sett g(x,y)=f(x,y)
3. Trunker ut-bildet
4. Trunker konvolusjons-masken h
5. Utvid bildet ved ”reflected indexing”
6. ”Circular indexing”

Et lite tips om konvolusjon

- Når vi konvolverer et filter med et bilde:
  - Er vi interessert i å lage et nytt bilde med samme størrelse som input-bildet.
  - Vi bruker en av teknikkene fra forrige foil.

- Når vi konvolverer en filter-kjerne med en annen filter-kjerne:
  - Vi vil lage effektiv implementasjon av et stort filter ved å kombinere enkle, separable filtre.
  - Vi beregner resultatet for alle posisjoner der de to filter-kjernene gir overlapp.

Egenskaper ved konvolusjon

- Kommutativ

  $f*g = g*f$

- Assosiativ

  $(f*g)*h = f*(g*h)$

- Distributiv

  $f*(g+h) = (f*g) + (f*h)$

- Assosiativ ved skalar multiplikasjon

  $a(f*g) = (af)*g = f*(ag)$

- Kan utnyttes i sammensatte konvolusjoner !

Lavpass-filtre

- Slipper gjennom lave frekvenser, og demper eller fjerner høye frekvener.
  - Høye frekvenser = skarpe kanter, støy, detaljer.
- Effekt: ”blurring” eller utsmøring av bildet
- Utfordring: bevare kanter
  samtidig som homogene områder glattes.

Ikke-uniformt lavpass-filter

- Uniforme lavpass-filtre kan implementeres raskt.

- Ikke-uniforme filtre, for eksempel:
  - 2D Gauss-filter:

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

  - Parameter σ er standard-avviket(bredden)
  - Filterstørrelse må tilpasses σ

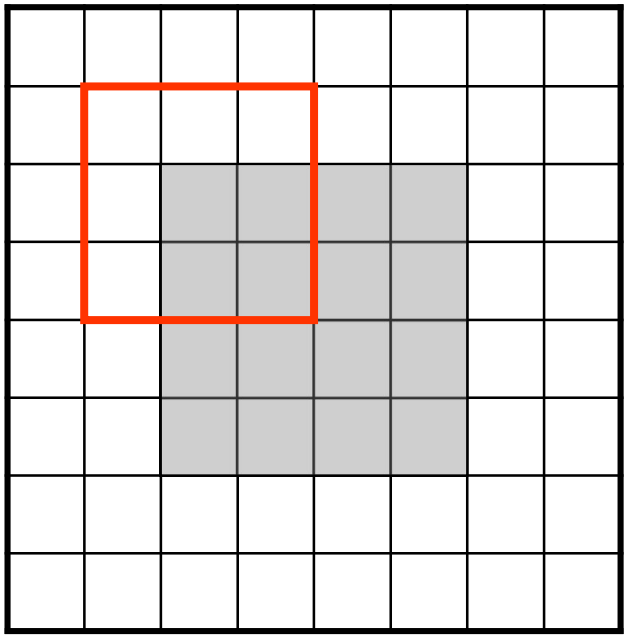
Rang-filtrering

- Vi lager en en-dimensjonal liste av alle piksel-verdiene innenfor vinduet.
- Vi sorterer listen i stigende rekkefølge.
- Responsen i (x,y) er pikselverdien i en bestemt posisjon i listen, eller en veiet sum av en bestemt del av listen.
- Dette er ikke-lineære filtre.

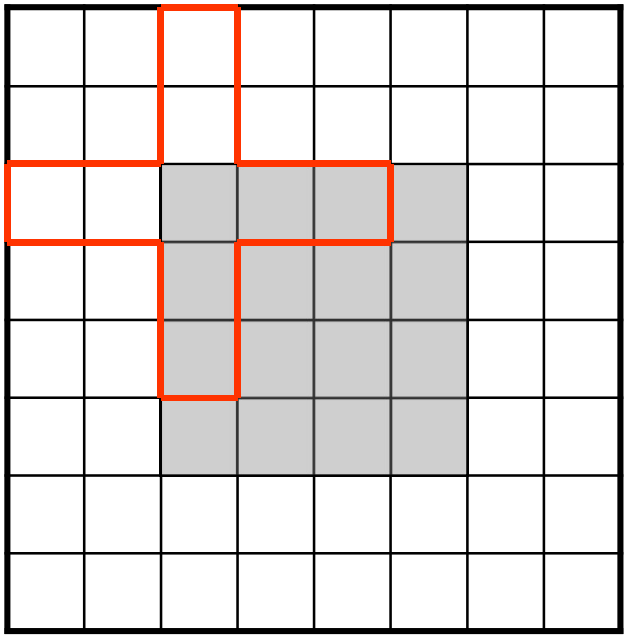
Median-filter

- g(x,y) = median av verdiene i et vindu rundt inn-pikslet.
- Median = den midterste verdien i sortert liste.
- Vindu: kvadrat, rektangel, pluss.
- Rask implementasjon kan gjøres vha. histogram, med histogram-oppdatering etter hvert som vinduet flyttes.
- Et av de mest brukte kant-bevarende støy-filtre.
- Spesielt godt til å fjerne impuls-støy (”salt og pepper”)
- Problemer:
  - Tynne kanter kan forsvinne
  - Hjørner kan rundes av
  - Objekter kan bli litt mindre
- Valg av vindus-størrelse og form er viktig!

Median og hjørner



Med kvadratisk vindu
rundes hjørnet av



Med ”pluss”-vindu
bevares hjørnet

Høypass-filtere

- Slipper gjennom høye frekvenser.
- Demper eller fjerner lav-frekvente variasjoner.
- Effekt:
  - Fjerner langsomt varierende bakgrunn
  - Framheve kanter, linjer og skarpe detaljer.

Høypass-filtre

- Et høypass-filter må ha positive vekter i midten, og negative vekter lenger ut.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Vi lar summen av vektene være null
- Hvis vi lar middelverdien av ut-bildet bli null, må noen deler av ut-bildet være <0.
- Det er ingen god ide å benytte $|g(x,y)|$.
- For framvisning, skaler $g(x,y)$ og legg til en konstant slik at vi får positive pikselverdier.

Gradient-operatorer

- Asymmetrisk 1D-operator:

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

- Symmetrisk 1D-operator:

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

- Roberts-operatoren (også kalt Roberts kryssgradient-operator):

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

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

G&W angir filtermasker som skal brukes til korrelasjon.

Filtrene vil derfor avvike med en 180 graders rotasjon.

Gradient-operatorer

- Prewitt-operatoren:

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

- Sobel-operatoren:

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

- Frei-Chen-operatoren:

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

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

G&W angir filtermasker som skal brukes til korrelasjon.

Filtrene vil derfor avvike med en 180 graders rotasjon.

g_x, g_y og gradient-magnituden, G

- Vi finner de horisontale kantene:
  - Beregn $g_x(x,y)=h_x * f(x,y)$
- Vi finner de vertikale kantene:
  - Beregn $g_y(x,y)=h_y * f(x,y)$
- Beregn gradient-magnitude og retning:

$$G(x,y) = \sqrt{g_x^2(x,y) + g_y^2(x,y)}$$ **Gradient-magnitude**

$$\theta(x,y) = \tan^{-1}\left(\frac{g_y(x,y)}{g_x(x,y)}\right)$$ **Gradient retning**

Laplace-operatoren

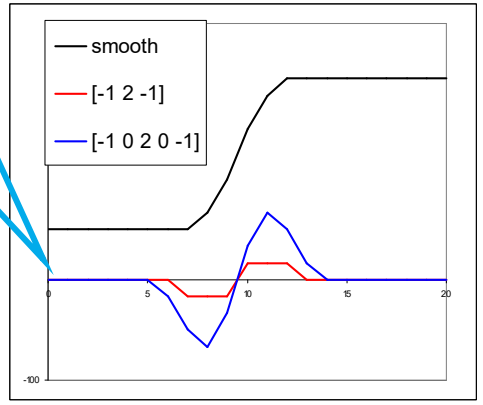
- Laplace-operatoren er gitt ved:

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

Den endrer fortegn der f(x,y) har et vendepunkt.

$|\nabla^2 f|$ har to ekstremverdier per kant

$\nabla^2 f = 0$ markerer kant-posisjon.



- Kantens eksakte posisjon er nullgjennomgangen.
- Dette gir ikke brede kanter.
- Vi finner bare magnitude, ikke retning.

Flere Laplace-operatorer

- Merk at Laplace-operatorene kan uttrykkes som senter-verdi minus en (relativt veiet) middelverdi over et lokalt naboskap.
- 1D $\nabla^2 f(i) = -f(i-1,j) + 2f(i,j) - f(i+1-j) = 3f(i) - \sum_{j=i-1}^{i+1} f(j)$
- 2D "pluss" $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$
- 2D "kvadrat" $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Laplace vs. Sobel

Sobel-filtrert
=> bred kant

Laplace-filtrert
=> dobbelt-kant

Fra Laplace til LoG

- Vi gjorde gradient-operatorene støy-robuste ved å bygge inn en lavpassfiltrering.
  Eksempel: Sobel-operator

$$h_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad h_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

- Vi kan gjøre det samme med Laplace-operatoren
  - Vi bruker et Gauss-filter G

$$\nabla^2 * (f * G) = \left(\nabla^2 * G\right) * f = LoG * f$$

  - Der LoG er resultatet av å anvende

    Laplace-operatoren på en Gauss-funksjon.

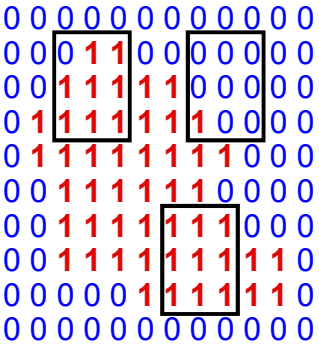
Cannys algoritme

1. Lavpassfiltrer med Gauss-filter (med gitt σ).
2. Finn gradient-magnituden og gradient-retningen.
3. Tynning av gradient-magnitude ortogonalt på kant.
   - F.eks.: Hvis en piksel i gradient-magnitude-bildet har en 8-nabo i eller mot gradient-retningen med høyere verdi, så settes pikselverdien til 0.
4. Hysterese-terskling (to terskler, T_h og T_l):
   a. Merk alle piksler der $g(x,y) \geq T_h$
   b. For alle piksler der $g(x,y) \in [T_l, T_h)$:
      - Hvis (4 eller 8)-nabo til en merket piksel, så merkes denne pikselen også.
   c. Gjenta fra trinn b til konvergens.

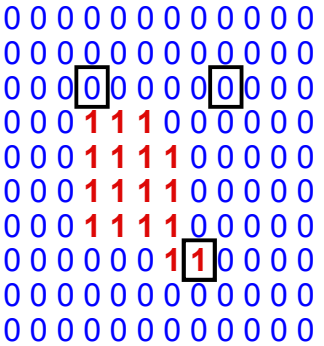
Erosjon: Passer strukturelementet til det binære bildet?

- Vi flytter strukturelementet rundt over et binært bilde.
- Strukturelementet **passer** i posisjonen (x,y) i bildet hvis alle elementer ≠ 0 i strukturelementet overlapper en pikselverdi ≠ 0 i bildet.

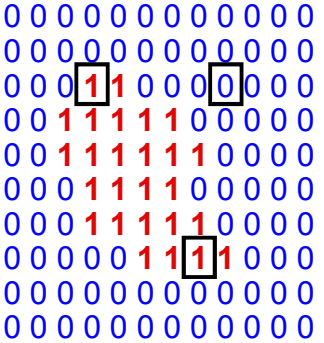
Et bilde



To forskjellige struktur-elementer



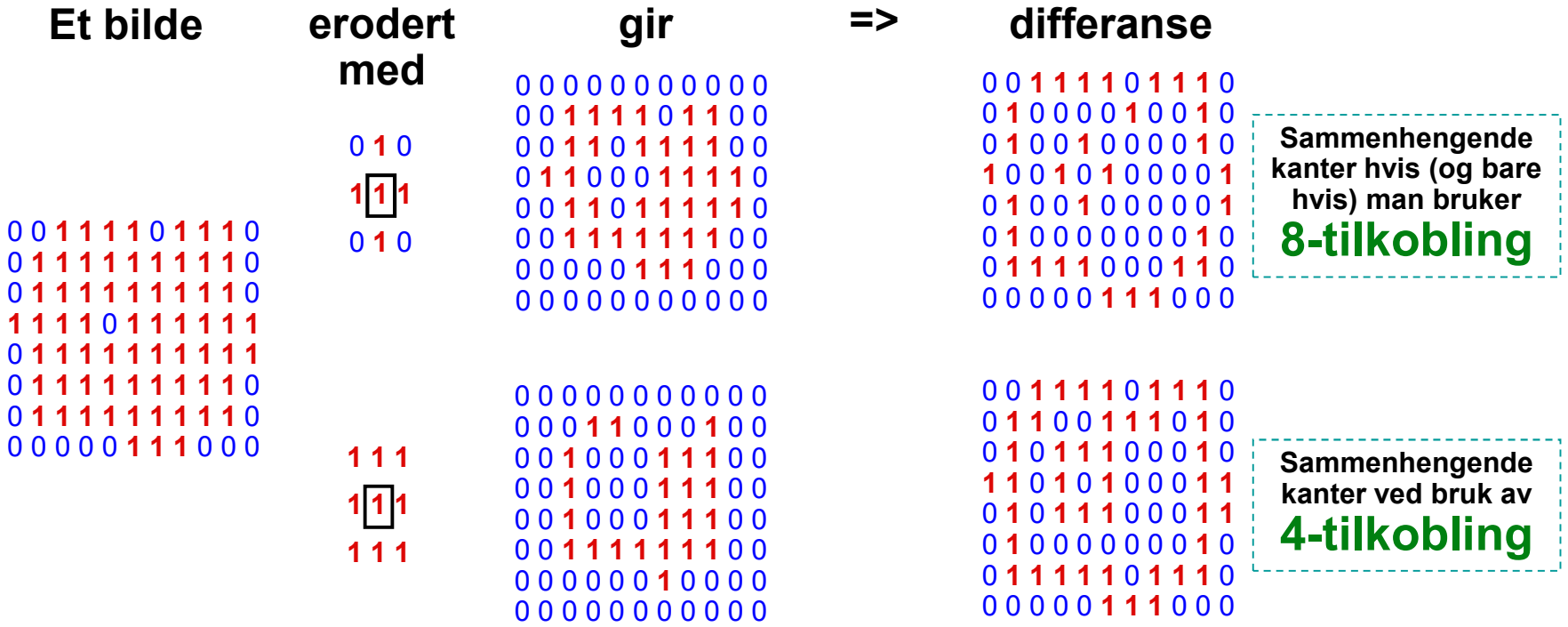
To forskjellige resultater



- Ignorere pikselverdier som overlapper 0 i strukturelementet.
  - 0 markerer jo «ikke er med i naboskapet».
- Anta at piksler utenfor bildet er 0.

Anvendelse av erosjon: Kantdeteksjon

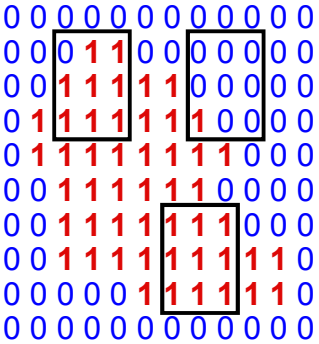
- Erodering fjerner piksler langs omrisset av et objekt.
- Vi kan finne kantene av objektene i bildet ved å subtrahere et erodert bilde fra originalbildet: **g = f - (f ⊖ S)**
- Det benyttede strukturelementet avgjør kantens tilkoblingstype:



Dilasjon: Treffer strukturelementet det binære bildet?

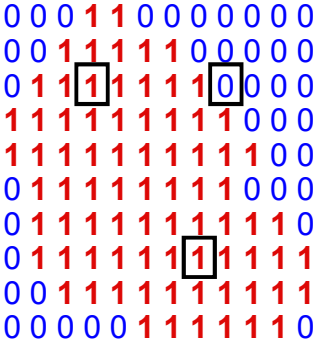
- Vi flytter strukturelementet rundt over et binært bilde.
- Strukturelementet **treffer** i posisjonen (x,y) i bildet hvis et element ≠ 0 i strukturelementet overlapper en pikselverdi ≠ 0 i bildet.
- Her reflekterer vi (roterer 180°) strukturelementet før vi flytter det rundt.

Et bilde



To forskjellige struktur-elementer

To forskjellige resultater

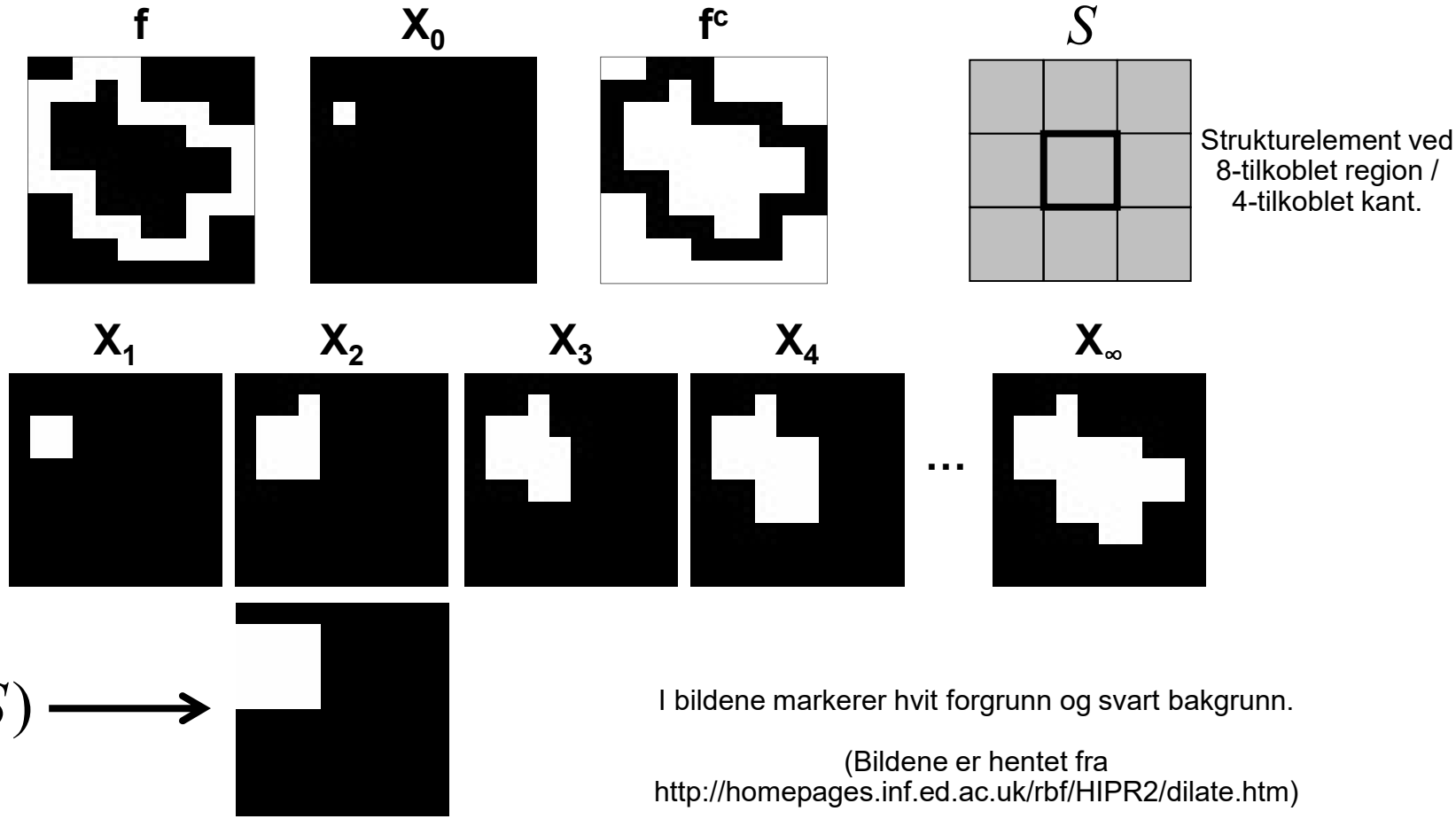


Fortsatt vil vi:

- Ignorere pikselverdier som overlapper 0 i strukturelementet.
- Anta at piksler utenfor bildet er 0.

Anvendelse av dilasjon: Region-fylling

- La X_0 inneholde et punkt i regionen som skal fylles.
- Iterativt beregn $X_k = (X_{k-1} \oplus S) \cap f^c$ inntil konvergens:



I bildene markerer hvit forgrunn og svart bakgrunn.

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

Dualitet

- **Dilasjon og erosjon er duale** med hensyn til komplementering og reflektering (180° rotasjon), dvs. at dilasjon og erosjon kan uttrykkes ved hverandre:

$$f \oplus S = \left(f^c \ \theta \ \hat{S}\right)^c$$

$$f \ \theta \ S = \left(f^c \oplus \hat{S}\right)^c$$

- For å dilatere f med symmetrisk S kan vi erodere komplementet til f med S, og ta komplementet av resultatet.
  - Tilsvarende for å erodere.
- => Dilasjon og erosjon kan utføres av samme prosedyre, forutsatt at vi kan rotere et strukturelement 180° og finne komplementet til et binært bilde.

et bilde	komplementet
0 0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1
0 0 1 1 0 0 1 1 1 0 0	1 1 0 0 1 1 0 0 0 1 1
0 0 0 1 1 1 0 0 1 0 0	1 1 1 0 0 0 1 1 0 1 1
0 0 0 1 0 0 0 0 1 0 0	1 1 1 0 1 1 1 1 0 1 1
0 0 0 0 1 0 0 0 1 0 0	1 1 1 1 0 1 1 1 0 1 1
0 0 0 0 0 1 0 0 1 0 0	1 1 1 1 1 0 1 1 0 1 1
0 0 0 0 0 0 1 0 1 0 0	1 1 1 1 1 1 0 1 0 1 1
0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1
dilatert med	**erodert med**
0 1 0	0 1 0
1 1 1	1 1 1
0 1 0	0 1 0
gir	**gir**
0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1
0 0 1 1 0 0 1 1 1 0 0	1 1 0 0 1 1 0 0 0 1 1
0 1 1 1 1 1 1 1 1 1 0	1 0 0 0 0 0 0 0 0 0 1
0 0 1 1 1 1 1 1 1 1 0	1 1 0 0 0 0 0 0 0 0 1
0 0 1 1 1 1 0 1 1 1 0	1 1 0 0 0 0 1 0 0 0 1
0 0 0 1 1 1 0 1 1 1 0	1 1 1 0 0 0 1 0 0 0 1
0 0 0 0 1 1 1 1 1 1 0	1 1 1 1 0 0 0 0 0 0 1
0 0 0 0 0 1 1 1 1 1 0	1 1 1 1 1 0 0 0 0 0 1
0 0 0 0 0 0 1 0 1 0 0	1 1 1 1 1 1 0 1 0 1 1
0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1

og disse bildene er komplementære.

De to matrisene til høyre er 1 utenfor randen.

Dilasjon: Andre egenskaper

- **Dilasjon er kommutativ.**

$$f \oplus S = S \oplus f$$

  - Selv om det er en konvensjon at første operand er bildet og andre er strukturelementet, så har dette altså ingen betydning.

- **Dilasjon er assosiativ.**

$$f \oplus (S_1 \oplus S_2) = (f \oplus S_1) \oplus S_2$$

  - Hvis S kan dekomponeres, dvs. at S er S_1 dilatert med S_2, kan vi spare en del regnetid, spesielt hvis S_1 og S_2 er én-dimensjonale. Eksempel:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Erosjon: Andre egenskaper

- Erosjon er **IKKE** kommutativ:

$$f \ \theta \ S \neq S \ \theta \ f$$

- Erosjon er heller **IKKE** assosiativ, men
  suksessiv erosjon av bildet f med A og så med B
  er ekvivalent med erosjon av bildet f med A **dilatert** med B:

$$(f \ \theta \ \mathrm{A}) \ \theta \ \mathrm{B} = f \ \theta \ (\mathrm{A} \oplus \mathrm{B})$$

  - Passer det med denne tidligere påstanden?
    «Hvis s_2 er formlik s_1,
    men dobbelt så stort,
    så er f ɵ s_2 ≈ (f ɵ s_1) ɵ s_1»

Åpning

- **Erosjon** av et bilde **fjerner** alle strukturer
  som ikke kan inneholde strukturelementet,
  og «**krymper**» alle andre strukturer.
- Hvis vi **dilaterer** resultatet av en erosjon med samme strukturelement,
  vil de strukturene som «**overlevde**» erosjonen bli omtrentlig **gjenskapt**.
- Dette er en **morfologisk åpning**;

$$f \circ S = \left(f \;\theta\; \mathrm{S}\right) \oplus S$$

- Navnet kommer av at operasjonen kan skape en åpning (et mellomrom)
  mellom to strukturer som bare henger sammen ved en tynn «bro»,
  uten å krympe disse to strukturene i noen betydelig grad.
  - Bare erosjon kan også skape en slik åpning/mellomrom,
    men vil også krympe begge strukturene.

Lukking

- **Dilasjon** av et bilde **utvider** strukturer,
  fyller i **hull og innbuktninger** i omrisset.
- Hvis vi **eroderer** resultatet av en dilasjon med samme strukturelement, vil strukturene **stort sett** få **gjenskapt** sin opprinnelige størrelse og form, men **hull og innbuktninger** som ble fylt igjen ved dilasjonen vil **ikke gjenoppstå**.
- Dette er en **morfologisk lukking**;

$$f \bullet S = \left(f \oplus \mathrm{S}\right) \theta \ S$$

- Navnet kommer av at operasjonen kan lukke en åpning mellom to strukturer som bare er adskilt med et lite gap, uten at de to strukturene vokser i noen betydelig grad.
  - Bare dilasjon kan også lukke en slik åpning, men vil også forstørre begge strukturene.

Dualitet mellom åpning og lukking

- **Lukking** er en **dual** operasjon til **åpning** med hensyn til komplementering og reflektering (180° rotering), og omvendt:

$$f \bullet S = (f^c \circ \hat{S})^c \qquad f \circ S = (f^c \bullet \hat{S})^c$$

- Lukking kan utføres ved å komplementere bildet,
  åpne det med det speilvendte (180° rotere) strukturelementet,
  og ta komplementet av resultatet.
  - Tilsvarende for åpning.
- Vi kan altså utføre begge operasjonene med kode bare for den ene, hvis vi har kode for å speilvende og komplementere et binært bilde.

- **Lukking** er en **ekstensiv** transformasjon (piksler legges til).
- **Åpning** er en **antiekstensiv** transformasjon (piksler fjernes).

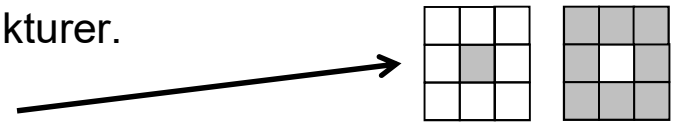
$$f \circ S \subseteq f \subseteq f \bullet S$$

«Hit-or-miss»-transformasjonen

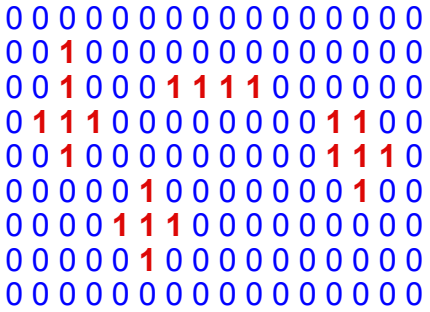
- Tilbake til den opprinnelige situasjonen: Bilde f og strukturelement S
- Men strukturelementet S er nå definert ved et par $[S_1, S_2]$ av binære strukturelementer som ikke har noen felles elementer.
- «Hit-or-miss»-transformasjonen av f med $S = [S_1, S_2]$ er definert som:

$$f(*)\,S = f(*)\,[S_1, S_2] = (f \ominus S_1) \cap (f^C \ominus S_2)$$

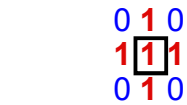
- En **forgrunnspiksel** i ut-bildet **oppnås** kun hvis:
  - **S_1 passer forgrunnen** rundt pikselen **og**
  - **S_2 passer bakgrunnen** rundt pikselen.
- Kan brukes til å finne/behandle bestemte mønstre i et bilde, f.eks. til å:
  - Finne bestemte strukturer.
  - Fjerne enkeltpiksler.
  - Benyttet i "tynning"



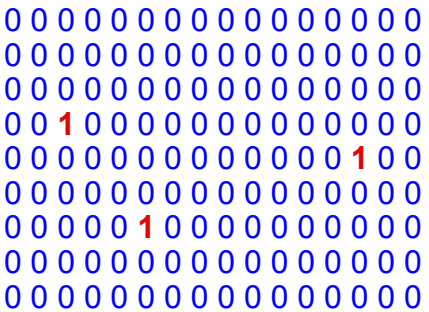
Eksempel: «Hit-or-miss»



Et bilde A

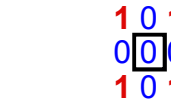


Strukturelement S_1

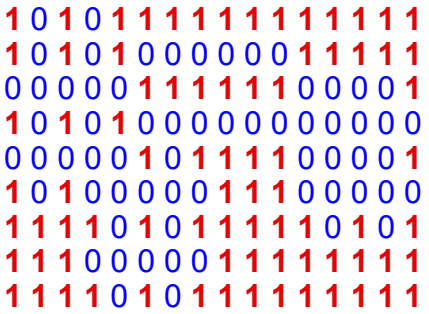


Resultat etter erosjon med S_1

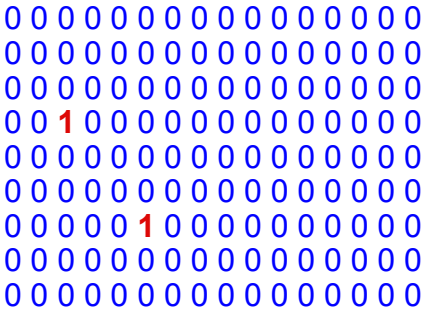
A^c - komplementet til bildet
(er 1 utenfor randen)



Strukturelement S_2



A^c erodert med S_2

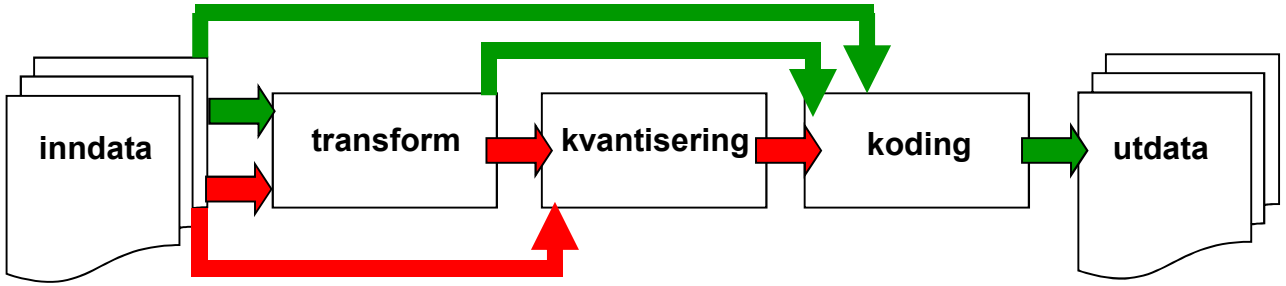


«Hit-or-miss»-resultatet

Logisk AND av de to delresultatene

Kompresjon og koding

- 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.

Entropi

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

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

- 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.

Huffman-koding

- Huffman-koding er en algoritme for variabel-lengde koding som er **optimal** under begrensningen at vi **koder symbol for symbol**.
  - Med *optimal* menes her minst mulig kodings-redundans.
- Antar at vi kjenner hyppigheten for hvert symbol.
  - Enten spesifisert som en modell.
    - Huffman-koden er da optimal hvis modellen stemmer.
  - Eller så kan vi bruke symbol-histogrammet til sekvensen.
    - Huffman-koden er da optimal for sekvensen.
    - Ofte bruker vi sannsynlighetene i stedet, men vi kunne likegodt benyttet hyppighetene.

Huffman-koding: Algoritmen

Gitt en sekvens med N symboler:

1. Sorter symbolene etter sannsynlighet,
   slik at de minst sannsynlige kommer sist.
2. Slå sammen de to minst sannsynlige symbolene til en gruppe,
   og sorter igjen etter sannsynlighet.
3. Gjenta 2 til det bare er to grupper igjen.
4. Gi koden 0 til den ene gruppen
   og koden 1 til den andre.
5. Traverser innover i begge gruppene og
   legg til 0 og 1 bakerst i kodeordet
   til hver av de to undergruppene.

Eksempel: Huffman-koding

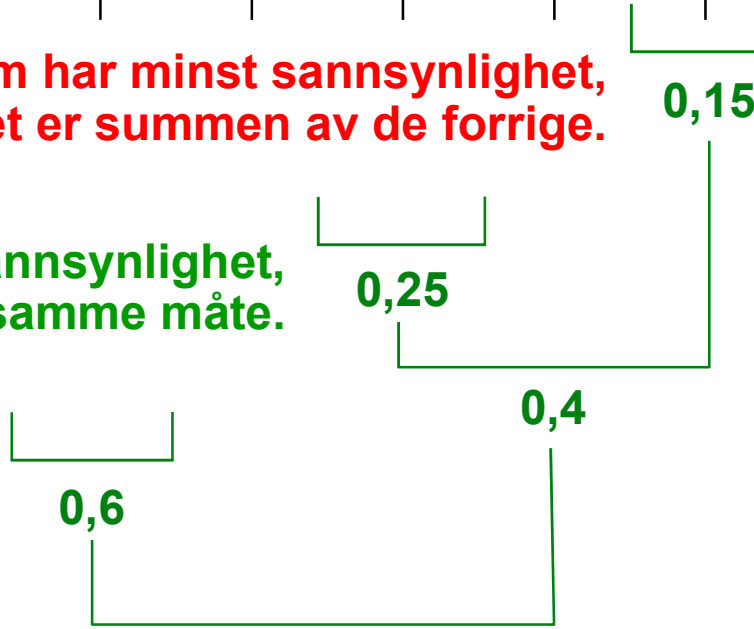
- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

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.

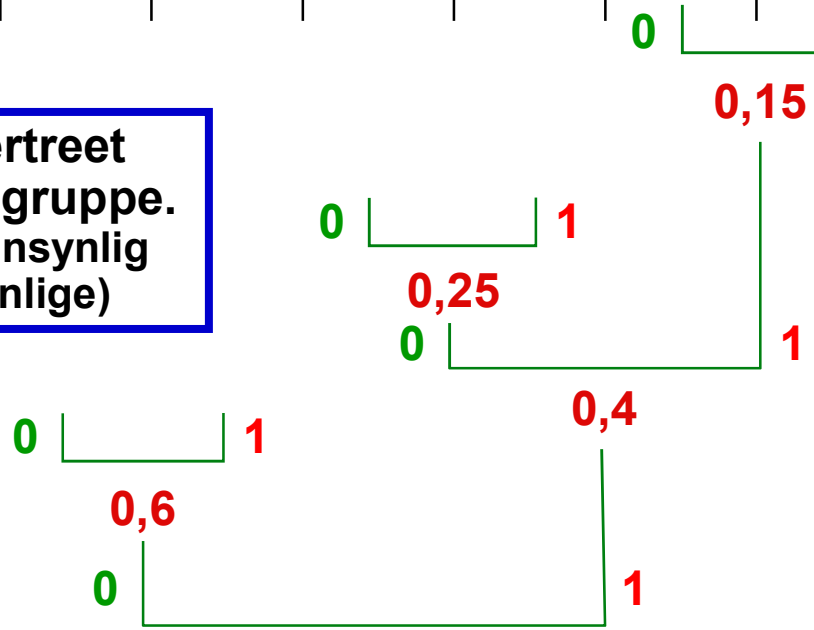


Eksempel: Huffman-koding

- 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)



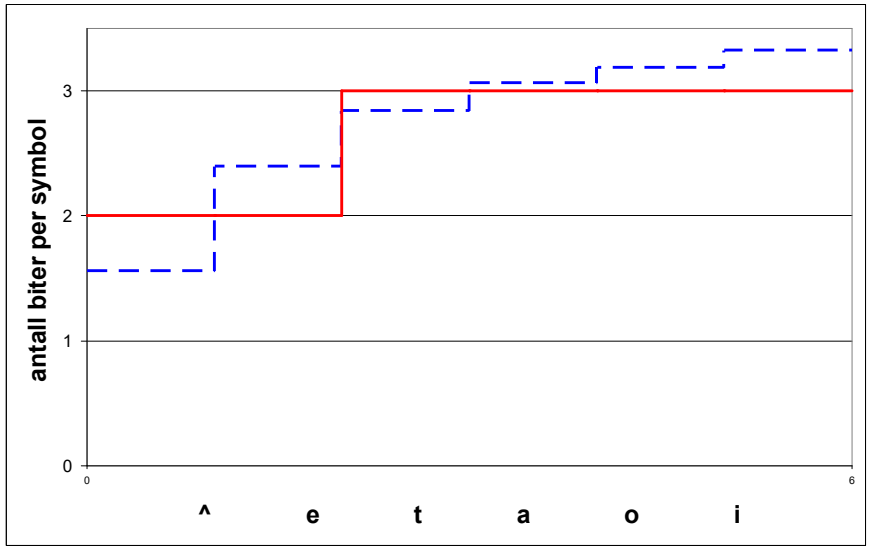
Ideell og faktisk kodeord-lengde

- Hvis gjennomsnittlig antall biter per symbol, c, skal være lik entropien, H, så må:

$$c = \sum_{i=0}^{G-1} b_i p_i = -\sum_{i=0}^{G-1} p_i \log_2 p_i = H$$

- Informasjonsinnholdet I(s_i) i hendelsen s_i angir altså den **ideelle binære kodeordlengden** for symbol s_i: $b_i = I(s_i) = \log_2 \frac{1}{p_i}$

- Plotter den ideelle lengden på kodeordene (vist i blått) sammen med de faktiske kodeordlengden (vist i rødt) for forrige eksempel, får vi:



Når gir Huffman-koding ingen kodingsredundans?

- Den **ideelle binære kodeordlengden** for symbol s_i er:

$$b_i = -\log_2(p_i)$$

- Siden **bare heltalls kodeordlengder er mulig**,
  er det bare når $p_i = \frac{1}{2^k}$
  for et heltall k som dette kan tilfredsstilles.

- Eksempel: Hvis meldingen har sannsynlighetene:

Symbol	s_0	s_1	s_2	s_3	s_4	s_5
Sannsynlighet	0,5	0,25	0,125	0,0625	0,03125	0,03125
Huffman-kodeord	0	10	110	1110	11110	11111

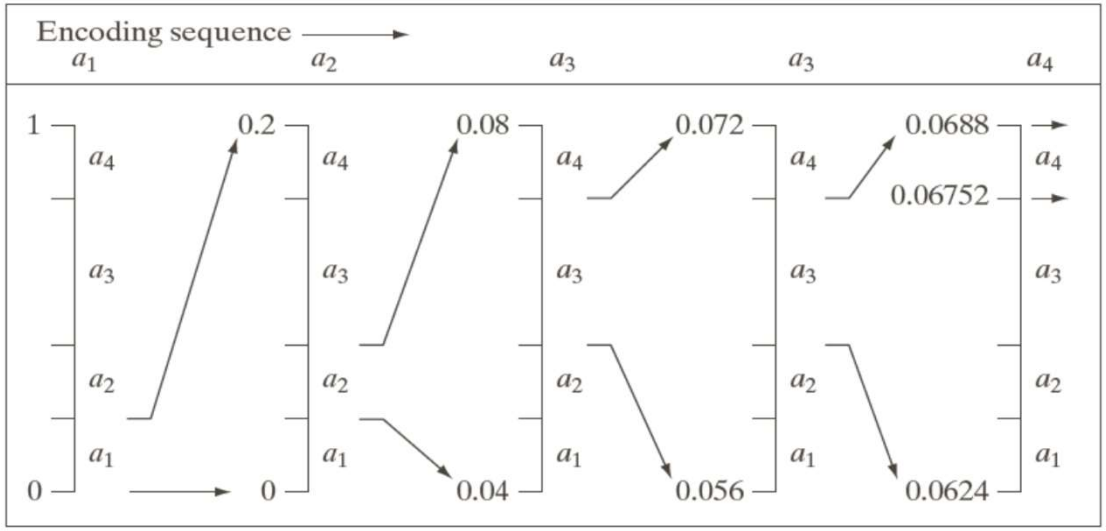
er gjennomsnittlig bitforbruk per symbol etter Huffman-koding:

$$c = 1{,}9375 = H$$

der H er entropien. Altså får vi **ingen kodingsredundans**!

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)

AK: Kodingseksempel

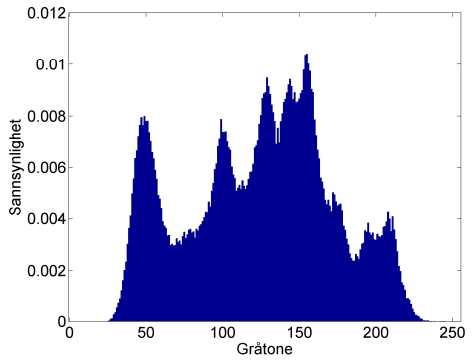
- Modell: Alfabet [a, b, c] med sannsynligheter [0,6, 0,2, 0,2].
- Hvilket delintervall av [0, 1) vil entydig representere meldingen **acaba** ?
  - **a** ligger i intervallet [0, 0,6).
    - «Current interval» har nå en bredde på 0.6.
  - **ac** ligger i intervallet [0+0,6*0,8, 0+0,6*1) = [0,48, 0,6).
    - Intervallbredden er nå 0,12 (= produktet 0,6*0,2).
  - **aca** ligger i intervallet [0,48+0,12*0, 0,48+0,12*0,6) = [0,48, 0,552).
    - Intervallbredden er 0,072 (= produktet 0,6*0,2*0,6).
  - **acab** er i [0,48+0,072*0,6, 0,48+0,072*0,8) = [0,5232, 0,5376).
    - Intervallbredden er 0,0144 (= produktet 0,6*0,2*0,6*0,2).
  - **acaba** er i [0,5232+0,0144*0, 0,5232+0,0144*0,6) = [0,5232, 0,53184).
    - Intervallbredden er nå 0,00864 (= produktet 0,6*0,2*0,6*0,2*0,6).
- Et tall i intervallet, f.eks. 0,53125, vil entydig representere **acaba**, forutsatt at mottakeren har den samme modellen og vet når man skal stoppe.

Differansetransform

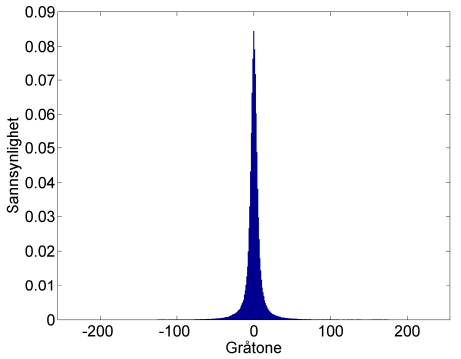
- Utnytter at horisontale nabopiksler ofte har ganske lik gråtone.

- Gitt en rad i bildet med gråtoner:
  $f_1, \ldots, f_N$ der $0 \le f_i \le 2^b-1$
- Transformer (reversibelt) til
  $g_1 = f_1,\ g_2 = f_2-f_1,\ \ldots,\ g_N = f_N-f_{N-1}$

- Merk at: $-(2^b-1) \le g_i \le 2^b-1$
  - Må bruke b+1 biter per g_i hvis vi skal tilordne like lange kodeord til alle mulig verdier.
- De fleste differansene er nær 0.
  - Naturlig binærkoding av differansene er ikke optimalt.



Entropi ≈ 7,45 => CR ≈ 1,1



Entropi ≈ 5,07 => CR ≈ 1,6

Løpelengde-transform

- Ofte inneholder bildet objekter med lignende gråtoner.
- Løpelengde-transformen (eng.: *run-length transform*)
  utnytter når horisontale nabopiksler har samme gråtone.
  - Merk: Krever ekte likhet, ikke bare «omtrent like».
- Løpelengde-transformen er reversibel.
- Hvis pikselverdiene til en rad er:
  333333555555555544777777 (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: ababcbababaaaaabab (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		

– Fra ny-streng-oppskriften vet vi at kode 7 ble laget ved: ba + ?

– Siden kode 7 nå sendes, må: ? = b => 7 = ba + b = bab

Ikke-tapsfri JPEG-kompresjon

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 intensitetsverdier.
   - Gjør at forventet gjennomsnittlig pikselverdi er omtrent 0.
   - Eks.: Intensitetsintervallet [0, 255]; 128 trekkes fra alle pikselverdiene.

3. Hver blokk transformeres med 2D DCT (diskret cosinus-transform).

124	125	122	120	122	119	117	118
121	121	120	119	119	120	120	118
126	124	123	122	121	121	120	120
124	124	125	125	126	125	124	124
127	127	128	129	130	128	127	125
143	142	143	142	140	139	139	139
150	148	152	152	152	152	150	151
156	159	158	155	158	158	157	156

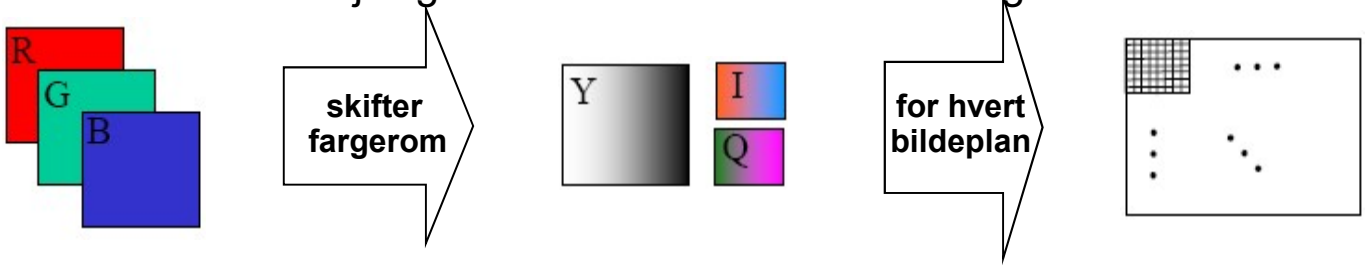
2D DCT

39.88	6.56	-2.24	1.22	-0.37	-1.08	0.79	1.13
-102.43	4.56	2.26	1.12	0.35	-0.63	-1.05	-0.48
37.77	1.31	1.77	0.25	-1.50	-2.21	-0.10	0.23
-5.67	2.24	-1.32	-0.81	1.41	0.22	-0.13	0.17
-3.37	-0.74	-1.75	0.77	-0.62	-2.65	-1.30	0.76
5.98	-0.13	-0.45	-0.77	1.99	-0.26	1.46	0.00
3.97	5.52	2.39	-0.55	-0.05	-0.84	-0.52	-0.13
-3.43	0.51	-1.07	0.87	0.96	0.09	0.33	0.01

   - Mye av informasjonen i de 64 pikslene samles i en liten del av de 64 2D DCT-koeffisientene; nemlig de i øverste, venstre hjørne.

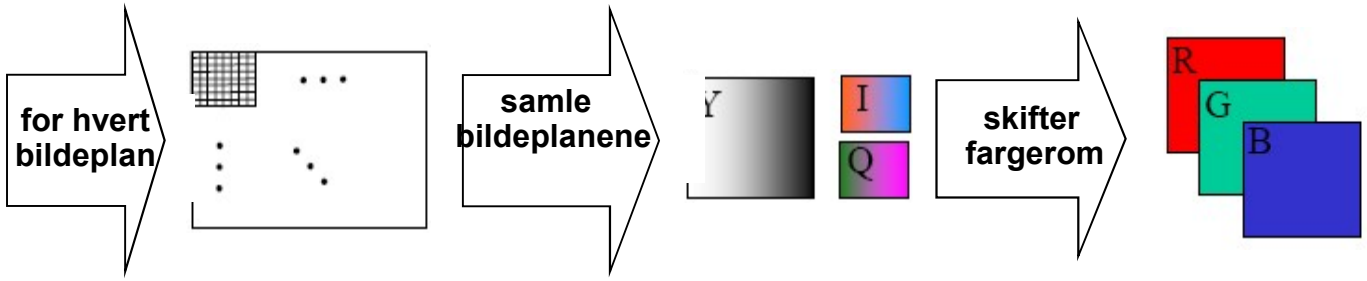
Ikke-tapsfri JPEG-kompresjon av fargebilde

- Skifter fargerom for å separere lysintensitet fra kromasi.
  - Stemmer bedre med hvordan vi oppfatter et fargebilde.
    - Lysintensiteten er viktigere enn kromasi for oss.
  - Kan også gi lavere kompleksitet i hver kanal.
- Nedsampler (vanligvis) kromasitet-kanalene.
  - Typisk med en faktor 2 i begge retninger.
- Hver bildekanal deles opp i blokker på 8x8 piksler, og hver blokk kodes separat som før.
  - Kan bruke forskjellige vektmatriser for intensitet- og kromasitet-kanalene.



Ikke-tapsfri dekompresjon av fargebilde

- Alle dekomprimerte 8x8-blokker i hver bildekanal samles til en matrise for den bildekanalen.
- Bildekanalene samles til et fargebilde.
- Vi skifter fargerom fra den brukte fargemodellen til:
  - til RGB for fremvisning, eller til CMYK for utskrift.



  - Kan få 8×8-blokkartefakter i intensitet.
  - Ved en faktor 2 nedsampling i hver retning av kromasitet-kanalene kan vi få 16×16 piksels blokkartefakter i kromasi («fargene»).

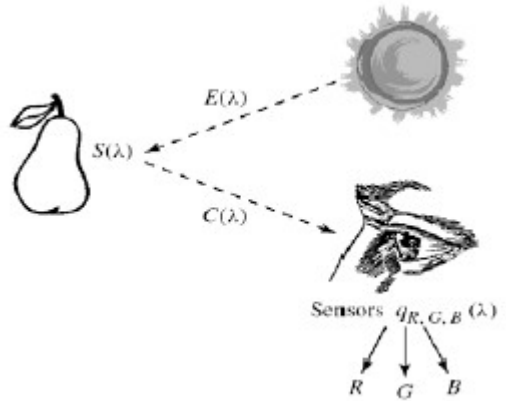
Blokk-artefakter

- Blokk-artefaktene øker med kompresjonsraten.

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

Fargerom: Tre integraler gir RGB

- Lys fra en kilde med spektralfordeling E(λ)
  - treffer et objekt med spektral refleksjonsfunksjon S(λ).
  - Reflektert lys detekteres av tre typer tapper med spektral lysfølsomhetsfunksjon $q_i(\lambda)$.



- Tre analoge signaler kommer ut av dette:

$$R = \int E(\lambda)\, S(\lambda)\, q_R(\lambda)\, d\lambda$$

$$G = \int E(\lambda)\, S(\lambda)\, q_G(\lambda)\, d\lambda$$

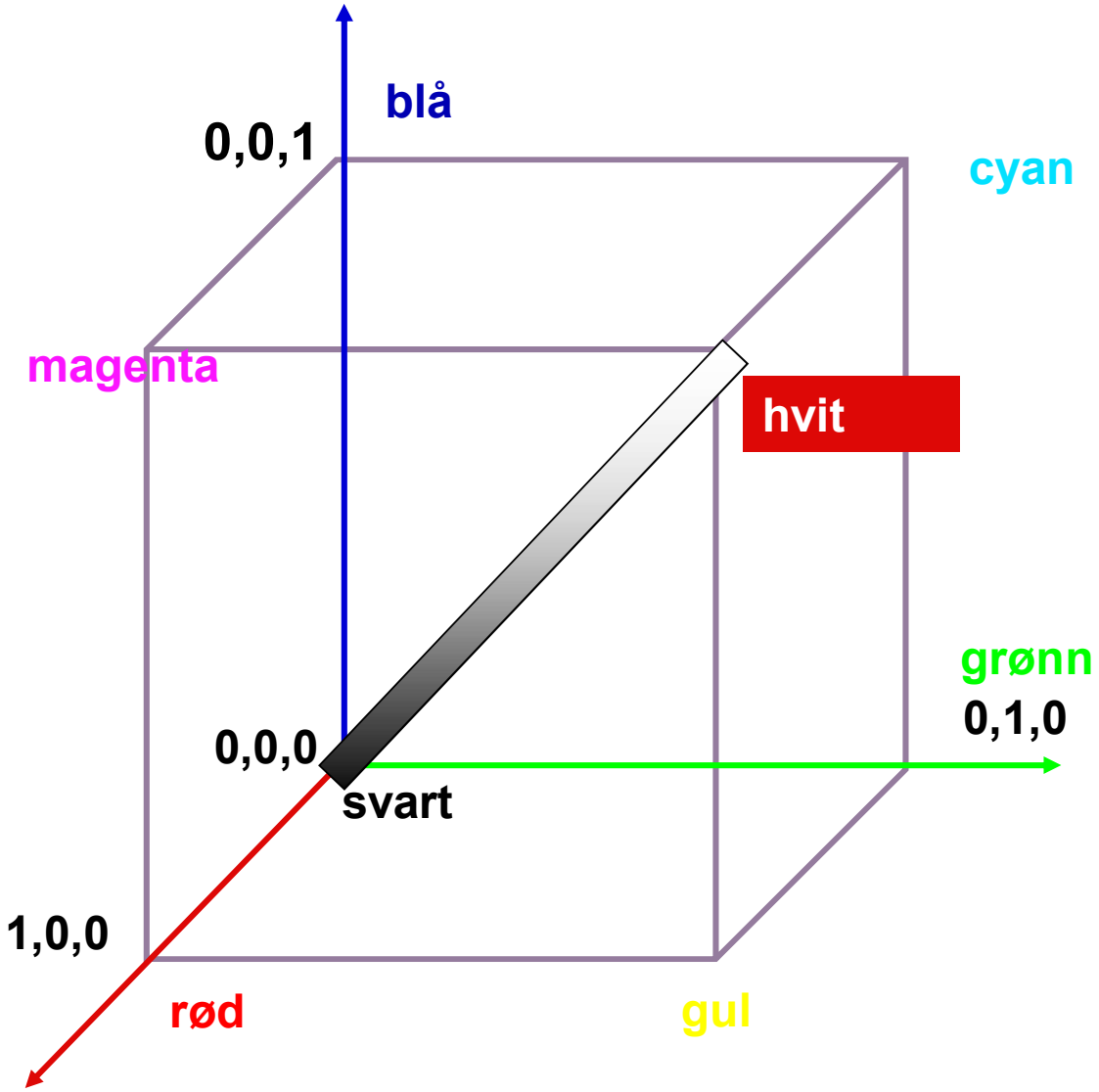
$$B = \int E(\lambda)\, S(\lambda)\, q_B(\lambda)\, d\lambda$$

Beskrivelse av farger

- En farge kan beskrives på forskjellige måter (fargerom)
  - RGB
  - HSI (Hue, Saturation, Intensity)
  - CMY (Cyan, Magenta, Yellow)
  - pluss mange flere …..
- HSI er viktig for hvordan vi beskriver og skiller farger.
  - I – Intensitet: hvor lys eller mørk er den
  - S – saturation/metning: hvor ”sterk” er fargen
  - H – dominerende farge (bølgelengde)
  - H og S beskriver sammen fargen og kalles kromatisitet

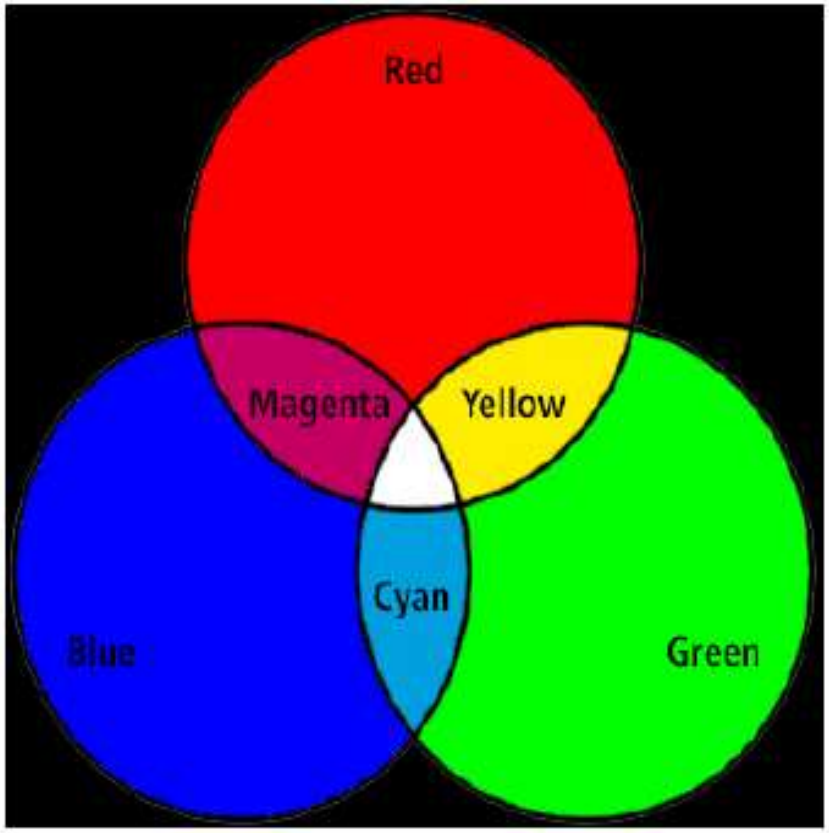
RGB-kuben

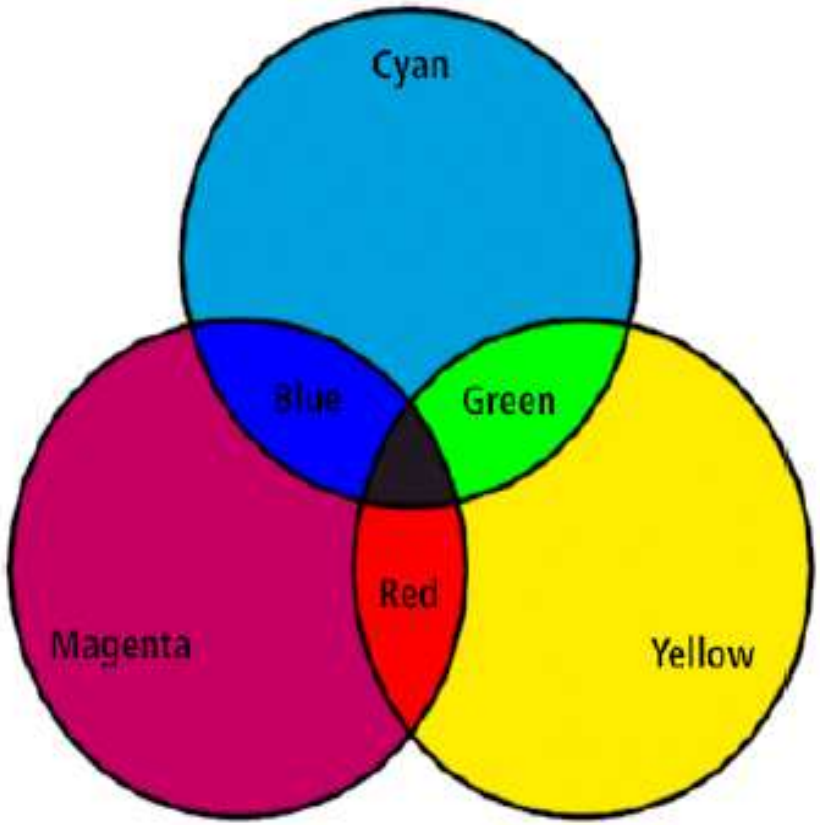
Gråtonebilder:
r=g=b



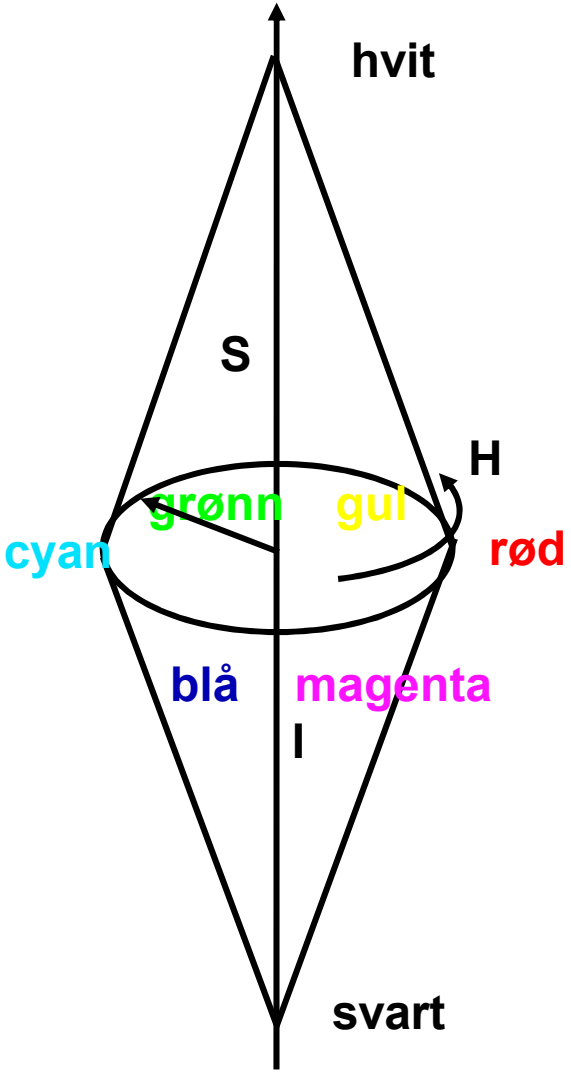
RGB og CMY

- RGB og CMY er i prinsippet sekundærfarger for hverandre.





Hue, Saturation, Intensity (HSI)



- Hue: ren farge - gir bølgelengden i det elektromagnetiske spektrum.

- H er vinkel og ligger mellom 0 og 2π:
  Rød: H=0, grønn: H= $2\pi/3$, blå= $4\pi/3$,
  gul: H=$\pi/3$, cyan= π, magenta= $5\pi/3$

- Hvis vi skalerer H-verdiene til 8-bits:
  Rød: H=0, grønn: H= 85, blå= 170,
  gul: H=42, cyan= 127, magenta= 213.

Takk for oppmerksomheten !

- Det har vært en fornøyelse å forelese for dere!
- Vi har gitt dere
  - 14 forelesninger, 2 obliger og masse ukeoppgaver
  - 630 lysark (pluss repetisjonene)
- Noe vil dere finne at dere har bruk for…
- Men det er sikkert noe dere ikke kan …

  - og noe dere ikke vet at dere ikke kan …

  - og mye dere kan som dere ikke har fra oss …

"As we know - there are known knowns.
There are things we know we know.
We also know - there are known unknowns.
That is to say
we know there are some things - we do not know.
But there are also unknown unknowns,
the ones we don't know – that we don't know."

D.H. Rumsfeld, DoD news briefing, — Feb. 12, 2002.

Ken Musgrave, "Blessed State" (1988)

- Vi står fortsatt til disposisjon, helt fram til like før eksamen 3/6.
- *Takk for følget, og lykke til med eksamen !*