IN2070 - Oppsummering

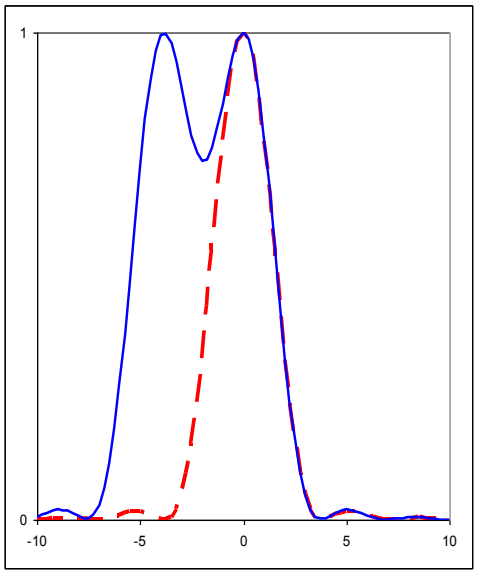
19. mai 2021

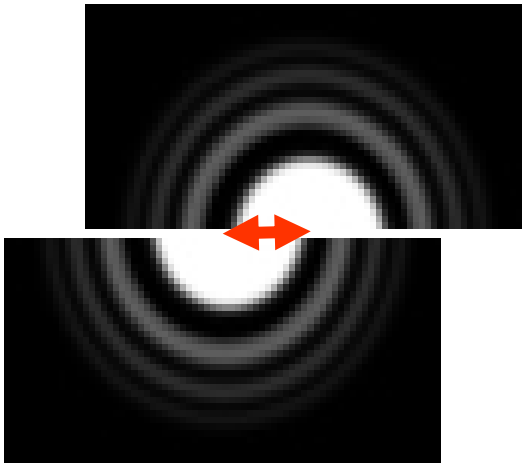
- Sampling og kvantisering
- Geometriske operasjoner
- Gråtonemapping
- Histogrambaserte operasjoner
- Segmentering
- Filtrering i bildedomenet
- Filtrering i frekvensdomenet
- Kompresjon og koding
- Morfologi
- Farger

Rayleigh-kriteriet

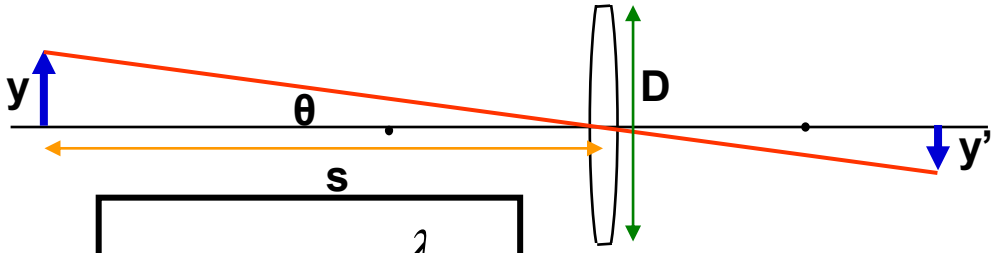
- To punkt-kilder kan adskilles hvis de ligger slik at sentrum i det ene diffraksjonsmønstret faller sammen med den første mørke ringen i det andre.
  - Vinkelen mellom dem er da gitt ved

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





Hvor små detaljer kan en linse oppløse?



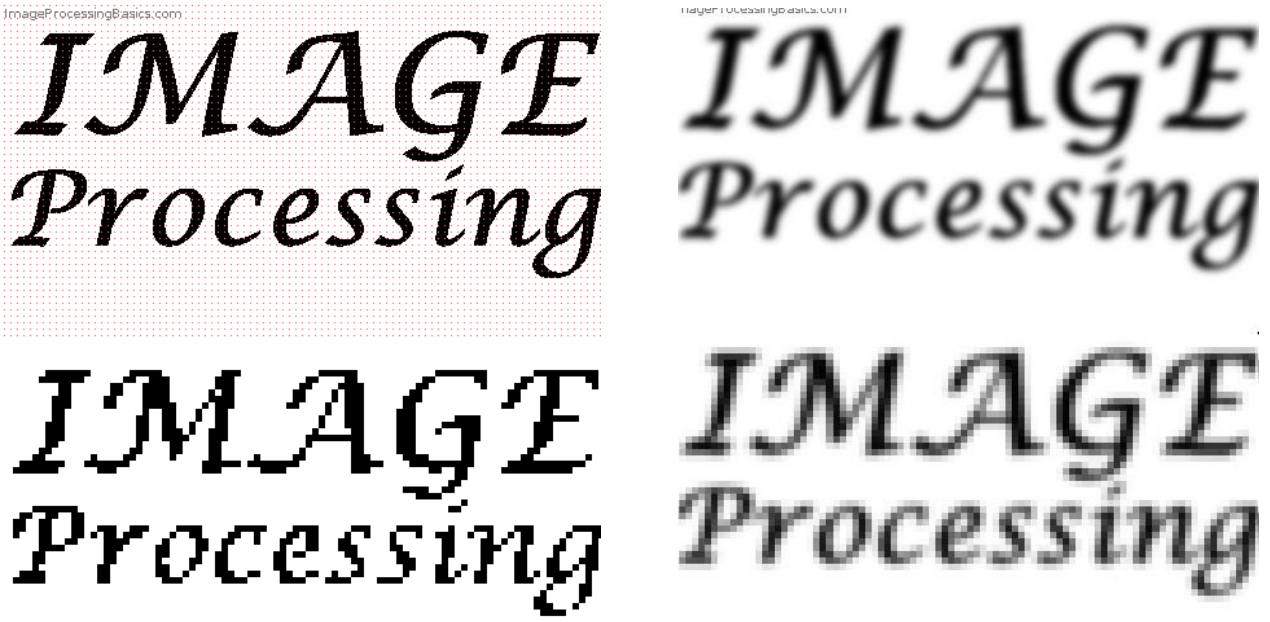
- Vinkeloppløsningen er gitt ved $$\sin\theta = 1.22\frac{\lambda}{D}$$
- Tangens til vinkelen θ er gitt ved $$tg(\theta) = \frac{y}{s}$$
- For små vinkler er sin(θ) = tg(θ) = θ, når vinkelen θ er gitt i radianer.
- **=> Den minste detaljen vi kan oppløse:** $$\frac{y}{s} = 1.22\frac{\lambda}{D} \Rightarrow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

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

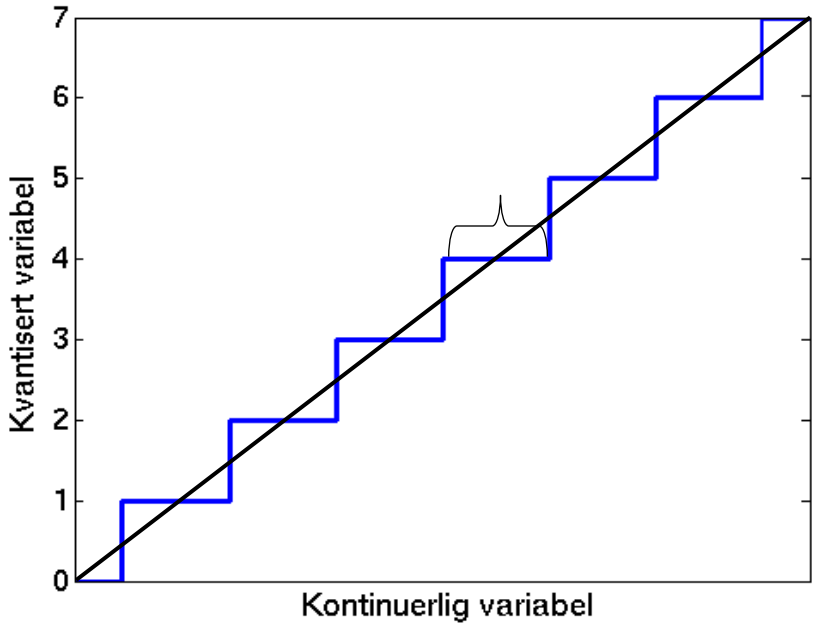
Anti-aliasing

- 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

Geometriske operasjoner: Affine transformasjoner

- Transformasjon av koordinatene (x, y) til (x', y'),

$$x' = T_x(x, y)$$
$$y' = T_y(x, y)$$

- Affine transformasjoner beskrives ved

$$x' = a_0 x + a_1 y + a_2$$
$$y' = b_0 x + b_1 y + b_2$$

- På matriseform:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{eller} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ b_0 & b_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_2 \\ b_2 \end{bmatrix}$$

Geometriske operasjoner: Oppsummering

- Transformasjoner av pikslenes koordinater

Geometriske operasjoner: Oppsummering

- Transformasjoner av pikslenes koordinater
- Resampling
  - Forlengs-mapping
  - Baklengs-mapping

Geometriske operasjoner: Oppsummering

- Transformasjoner av pikslenes koordinater
- Resampling
  - Forlengs-mapping
  - Baklengs-mapping
- Interpolasjonsmetoder
  - Nærmeste nabo interpolasjon
  - Bilineær interpolasjon
  - Høyere-ordens interpolasjon

Geometriske operasjoner: Oppsummering

- Transformasjoner av pikslenes koordinater
- Resampling
  - Forlengs-mapping
  - Baklengs-mapping
- Interpolasjonsmetoder
  - Nærmeste nabo interpolasjon
  - Bilineær interpolasjon
  - Høyere-ordens interpolasjon
- Bruk av geometriske operasjoner til å samregistrere bilder

Lineær kontrastendring

- Lineær/affin strekking

$$T[i] = ai + b$$
$$g(x, y) = af(x, y) + b$$

- **a** regulerer kontrasten, og **b** ”lysheten”
- **a**>1: mer kontrast
- **a**<1: mindre kontrast
- **b**: flytter alle gråtoner **b** nivåer
- Negativer: **a**=-1 , **b**=maxverdi for bildetype

Oppsummering

- Gråtonehistogrammer
- Lineær/affin transform
  - Forstå effekten av parametrene *a* og *b*
  - Hvordan sette *a* og *b*?
    - Eksplisitt
    - Mappe ett intensitetsintervall til et annet
    - Bestemme ønsket lyshet (μ_T) og kontrast (σ_T)
      - Jfr. standardisering av lyshet og kontrast
- Ikke-lineære, parametriske transformer
  - Logaritmisk, eksponentiell, ”gamma”, stykkevis lineær
  - Hva gjøres med kontrasten i de mørke og lyse delene av bildet etter slike transformer
  - Tegn skisse av funksjonene og se Δf mot Δg (lokalt stigningstall)

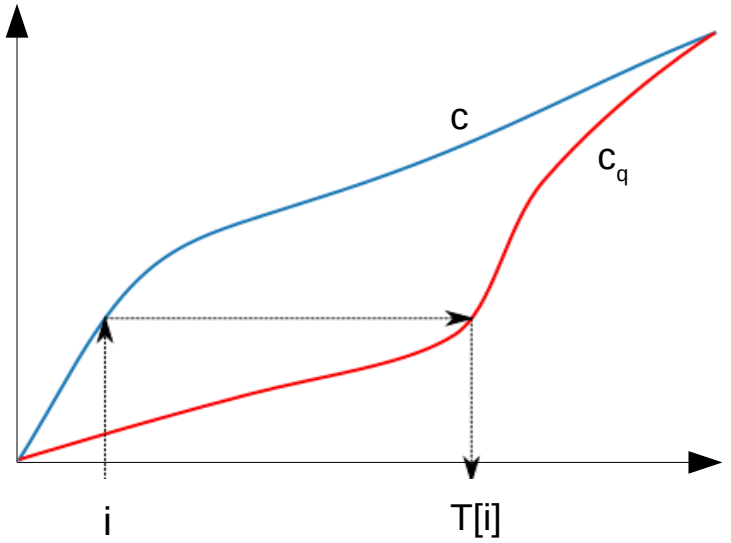
Histogramtilpasning I/II

- Histogramutjevning gir (tilnærmet) flatt histogram
- Kan spesifisere annen form på resultathistogrammet:

1. Gjør histogramutjevning på innbildet, finn s=T(i)
2. Spesifiser ønsket nytt histogram g(z)
3. Finn den transformen T_g som histogramutjevner g(z) og inverstransformen T_g^{-1}
4. Inverstransformer det histogramutjevnede bildet fra punkt 1 ved $z=T_g^{-1}(s)$

Histogramtilpasning II/II

- Alternativ tankegang: Flytt søylene på det *kumulative* histogrammet slik at vi får et nytt *kumulativt* histogram som ligner på det ønskede
- For hver i har vi en søylehøyde c(i) ..
- .. finn så T[i] slik at søylehøyden på det ønskede kumulative histogrammet, c_q, blir tilnærmet likt c(i)



$$c(i) \approx c_q(T[i])$$

Finn beste T[i] for hver i

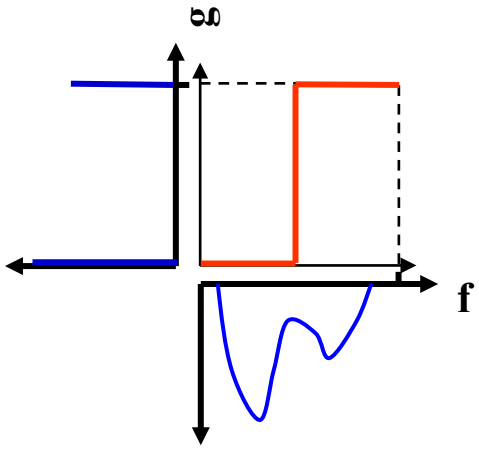
Sentrale temaer i dag

- Histogramtransformasjoner
  - Histogramutjevning
  - Histogramtilpasning
- Standardisering av histogram for billedserier
  - Fjerne effekten av variasjoner i avbildningsforhold (døgnvariasjon, lysforhold, sensorbytte, støv etc)
  - Ikke lurt med histogramtilpasning hvis histogram-formen inneholder informasjon som (senere) skal/kan benyttes
  - Alternativ til standardisering av bilder med lineær transform
- Lokal gråtone-transformasjon
  - Samme kontrast og «lyshet» over hele bildet
  - Beregn og benytt transformene på lokalt vindu rundt hver piksel
  - Kontrastbegrensning (både for lineær strekking og histogramutjevning)
  - Regnekrevende

Dagens verktøy: Terskling

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

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



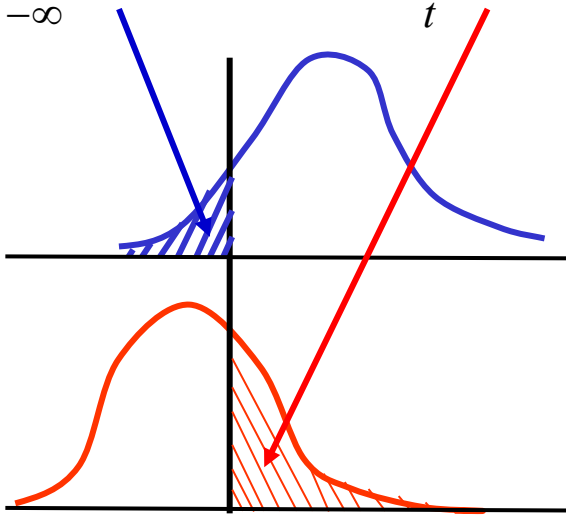
- Da har vi fått et ut-bilde g(x,y) med bare to mulige verdier.
- Vi tolker nå piksler med g(x,y)=1 som objekt-piksler.
- Vi har gjort en global pikselvis *klassifikasjon* basert på pikselintensitet alene

Klassifikasjonsfeil ved terskling

- Andelen feilklassifiserte piksler:
  - Andelen forgrunnspiksler klassifisert som bakgrunnspiksler pluss andelen bakgrunnspiksler klassifisert som forgrunn

- For en gitt terskel t:

$$E(t) = F\int_{-\infty}^{t} p_2(z)dz + B\int_{t}^{\infty} p_1(z)dz$$



(Benytter ofte kontinuerlige variable, da våre histogrammodeller ofte er definert for slike, jfr. normalfordelingen)

Finn den T som minimerer feilen

$$E(t) = F\int_{-\infty}^{t} p_2(z)dz + B\int_{t}^{\infty} p_1(z)dz$$

- E(t) vil alltid ha et minimum der kurvene for forgrunns- og bakgrunnshistogrammer krysser hverandre (hvorfor?)
- Kan også sette den deriverte lik 0 og vi får:

VIKTIG!

$$\frac{dE(t)}{dt} = 0 \Rightarrow F \cdot p_2(t) = B \cdot p_1(t)$$

- Merk at dette er en generell løsning som gir minst feil.
- Det er ingen restriksjoner mht. fordelingene p_1 og p_2!

Ridler og Calvards metode: Iterativ tilpasning av likeformede Gausser

- Anta to Gauss-fordelinger med forventninger μ_1 og μ_2, og med $\sigma_1 \approx \sigma_2$, og anta $F \approx B$

- 1) Vi kjenner ikke den sanne μ_1 og μ_2 så vi starter med å gjette på en løsning
- 2) Beregner så en terskel ved $T=(\mu_1+\mu_2)/2$
- 3) Basert på denne terskelen, finner vi ny μ_1 og μ_2 som henholdsvis middelverdien til pikslene under og over terskelen

  → Dess mindre modellen passer med dataene, dess mer vil μ_1 og μ_2 endre seg

- Gjenta derfor 2) og 3) til den nye terskelen ikke endrer verdi (T konvergerer)

- Dette er en enkel og rask måte å finne parametrene μ_1 og μ_2 som gir fordelinger som tilnærmet best "passer" histogrammet vårt, jfr. lysark 9
  - Husk at vi antar $\sigma_1 \approx \sigma_2$ og $F \approx B$
- Algoritmen beskrives i DIP s. (746) (også kjente under navnet *k-means*)

Otsus metode: Likeformet Gausstilpasning direkte søk

- **Bakgrunnstankegang**: Anta at bildet inneholder to populasjoner av piksler, slik at pikslene innenfor hver populasjon er noenlunde like, mens populasjonene er forskjellige.

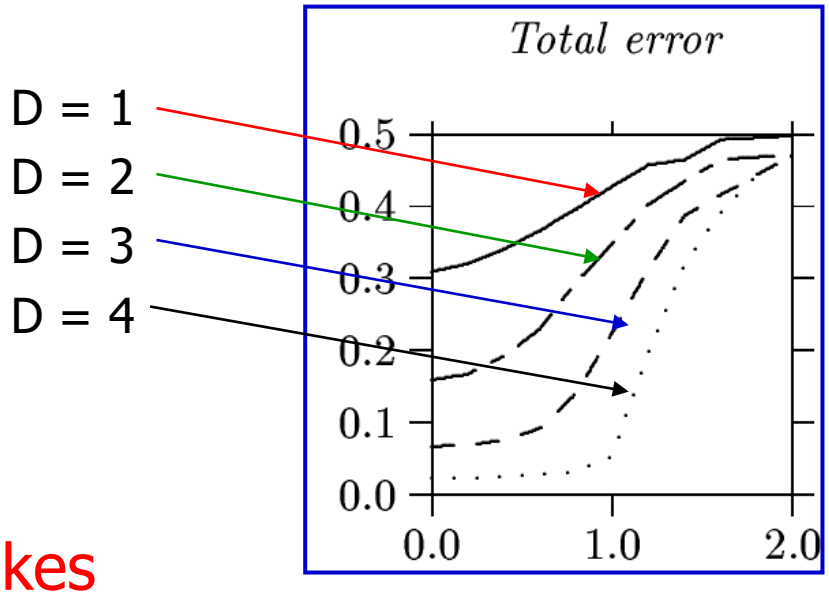
- **Målsetting:**

  Let igjennom alle gråtonene, og finn en terskel T slik at hver av de to klassene som oppstår ved tersklingen blir mest mulig homogene, mens de to klassene bli mest mulig forskjellige.

  – Klassene er homogene:
    variansen i hver av de to klassene er minst mulig.
  – Separasjonen mellom klassene er stor:
    avstanden mellom middelverdiene er størst mulig.

Effekten av *a priori* sannsynlighet

- Total tersklingsfeil mot $\log_{10}(P_1/P_2)$
  for fire verdier av μ_2-μ_1 = Dσ:



- Feilen øker raskt ved
  $\log_{10}(P_1/P_2) \approx 1$
- => Otsus metode bør bare brukes
  når $0.1<P_1/P_2<10$.
- Det samme gjelder for Ridler & Calvard.

Oppsummering terskling

- Generelle histogramfordelinger og klassifikasjonsfeil
  - Har vi h_1 og h_2 har vi alt: forgrunn der $h_2(i)>(h_1)$ altså der $F*p_2(i) > B*p_1(i)$
  - Terskling og (lokale) klassifikasjonsfeilminima der $h_1(i)=h_2(i)$
- To vanlige globale tersklingsalgoritmer:
  - Begge antar og tilpasser likeformede «Gausser»
  - Ridler og Calvards metode (iterativ, gjentatt klassifikasjon til stabile μ_1 og μ_2)
  - Otsus metode (prøv alle terskler, søk maksimal separasjon)
- Ulik apriori sannsynlighet og bruken av kantinformasjon
- Effektene av ”støy” og bruken av lavpassfilter
- Flernivå-terskling
- Lokale adaptive metoder
  - Håndtere mangel på lokal bimodalitet

Filtrering: 2-D konvolusjon

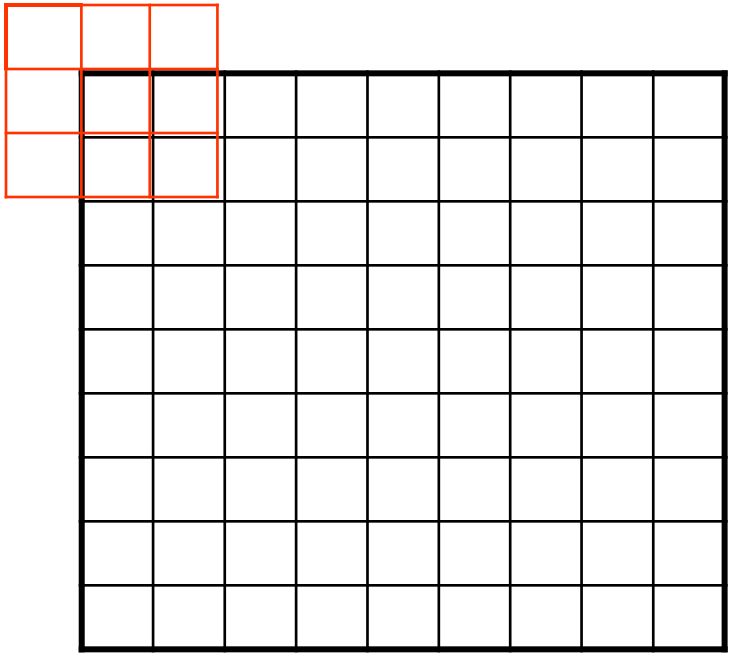
$$g(x,y) = \sum_{j=x-w_1}^{x+w_1} \sum_{k=y-w_2}^{y+w_2} h(x-j, y-k) f(j,k)$$

- 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.

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

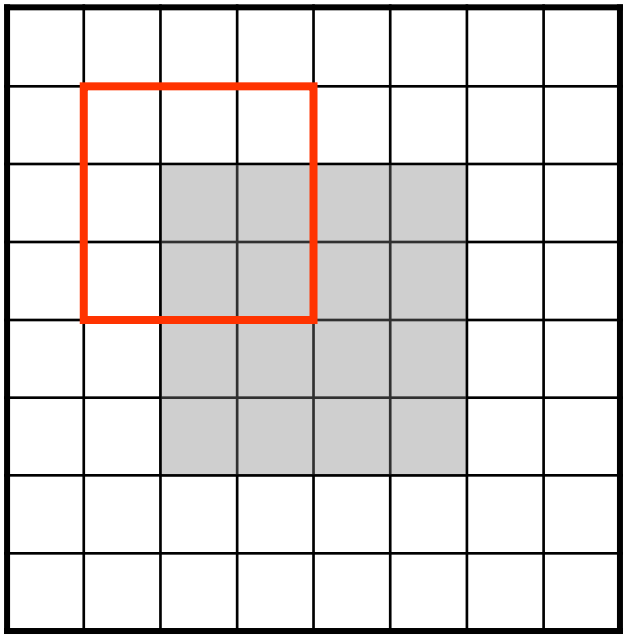
Rang-filtrering

- 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.

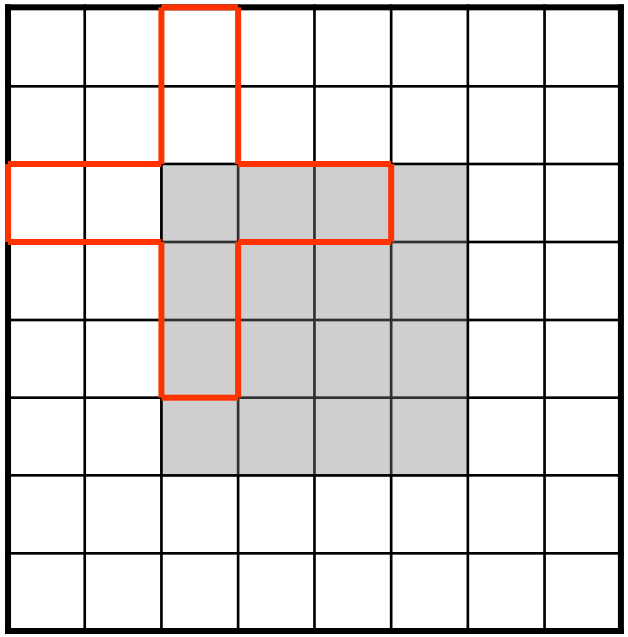
Median-filter

- 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

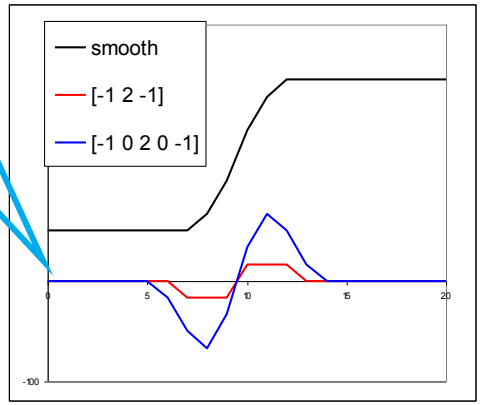
- Laplace-operatoren er gitt ved:

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

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

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

  - 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.

Fouriertransformasjon (del 1): Grunnidé

Digitale gråtonebilder av størrelse $M \times N$ kan representeres ved en vektet sum av disse $M \times N$ sinus- og cosinus-bildene:

$$\cos\left(2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right) \qquad \sin\left(-2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right)$$

for frekvensene

$$u = 0, 1, \ldots, M-1$$
$$v = 0, 1, \ldots, N-1$$

Fouriertransformasjon (del 1): Oppsummering

- Sinusfunksjoner
  - Frekvens/periode
  - Amplitude
  - Fase
  - Dekomponere $A\sin(\theta+\phi)$ i en $\sin$- og $\cos$-komponent
  - 1D og 2D varianter

Fouriertransformasjon (del 1): Oppsummering

- Sinusfunksjoner
  - Frekvens/periode
  - Amplitude
  - Fase
  - Dekomponere $A\sin(\theta+\phi)$ i en $\sin$- og $\cos$-komponent
  - 1D og 2D varianter
- Diskret Fourier-transformasjon
  - Bilde beskrevet med $\cos$- og $\sin$-basisbilder
  - Representasjon ved komplekse tall
    $\rightarrow$ $\cos$- og $\sin$-ledd som hhv. reell- og imaginær-komponent
  - Implisitt periodisitet i bildet
    $\rightarrow$ Utslag i diskontinuitet, "ekstra" frekvenser
  - Fremvisning av spekteret $|F(u,v)|$
  - Observasjoner

Fouriertransformasjon (del 2): Konvolusjonsteoremet

Konvolusjon i bildedomenet $\Leftrightarrow$ elementvis multiplikasjon i frekvensdomenet,

$$f * h \Leftrightarrow F \odot H$$

Motsatt gjelder også:

$$f \odot h \Leftrightarrow F * H,$$

altså elementvis multiplikasjon i bildedomenet $\Leftrightarrow$ konvolusjon i frekvensdomenet

Vi snakker om sirkelkonvolusjon her!

Fouriertransformasjon (del 2): Oppsummering

- Konvolusjonsteoremet: Sirkelkonvolusjon i bildedomenet er ekvivalent med elementvis multiplikasjon i frekvensdomenet, og omvendt

Fouriertransformasjon (del 2): Oppsummering

- Konvolusjonsteoremet: Sirkelkonvolusjon i bildedomenet er ekvivalent med elementvis multiplikasjon i frekvensdomenet, og omvendt
- Anvendelser:
  - Design av filtre i frekvensdomenet
    - Lavpass, høypass, båndpass, båndstopp, notch
    - I praksis: la filteret H være symmetrisk om nullfrekvens. Konjugert symmetri, $H(u, v) = H^*(-u, -v)$ gir reelle utbilder
    - Myke overganger $\rightarrow$ redusere ringing
  - Analyse av konvolusjonsfiltre gjennom frekvensrespons
  - Rask implementasjon av større konvolusjonsfiltre

Fouriertransformasjon (del 2): Oppsummering

- Konvolusjonsteoremet: Sirkelkonvolusjon i bildedomenet er ekvivalent med elementvis multiplikasjon i frekvensdomenet, og omvendt
- Anvendelser:
  - Design av filtre i frekvensdomenet
    - Lavpass, høypass, båndpass, båndstopp, notch
    - I praksis: la filteret H være symmetrisk om nullfrekvens. Konjugert symmetri, $H(u, v) = H^*(-u, -v)$ gir reelle utbilder
    - Myke overganger $\rightarrow$ redusere ringing
  - Analyse av konvolusjonsfiltre gjennom frekvensrespons
  - Rask implementasjon av større konvolusjonsfiltre
- Vindusfunksjoner på bilder før transformasjonen

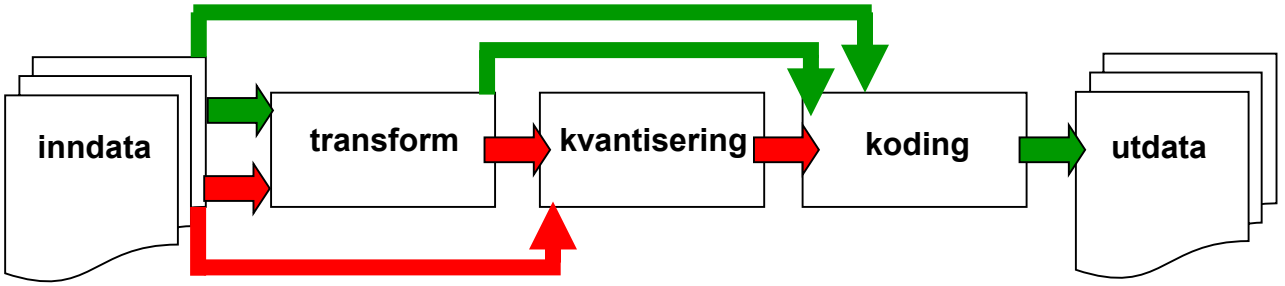
Fouriertransformasjon (del 2): Oppsummering

- Konvolusjonsteoremet: Sirkelkonvolusjon i bildedomenet er ekvivalent med elementvis multiplikasjon i frekvensdomenet, og omvendt
- Anvendelser:
  - Design av filtre i frekvensdomenet
    - Lavpass, høypass, båndpass, båndstopp, notch
    - I praksis: la filteret H være symmetrisk om nullfrekvens. Konjugert symmetri, $H(u, v) = H^*(-u, -v)$ gir reelle utbilder
    - Myke overganger $\rightarrow$ redusere ringing
  - Analyse av konvolusjonsfiltre gjennom frekvensrespons
  - Rask implementasjon av større konvolusjonsfiltre
- Vindusfunksjoner på bilder før transformasjonen

$\rightarrow$ Redusere bidrag langs aksene i frekvensspekteret

Kompresjon

- Kompresjon kan deles inn i tre steg:
  - **Transform** - representér bildet mer kompakt.
  - **Kvantisering** - avrund representasjonen.
  - **Koding** - produsér 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 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.**
  - Men dette gjelder bare hvis vi koder hvert symbol for seg.

10 slides om 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 kan like godt benytte 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

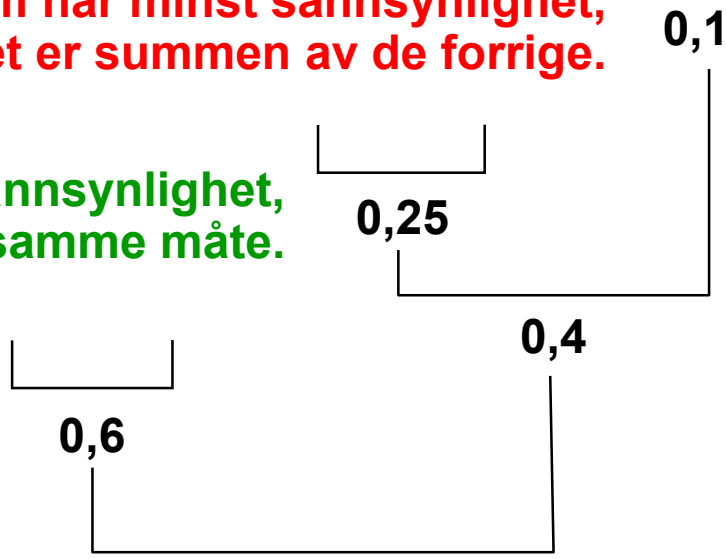
- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

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.

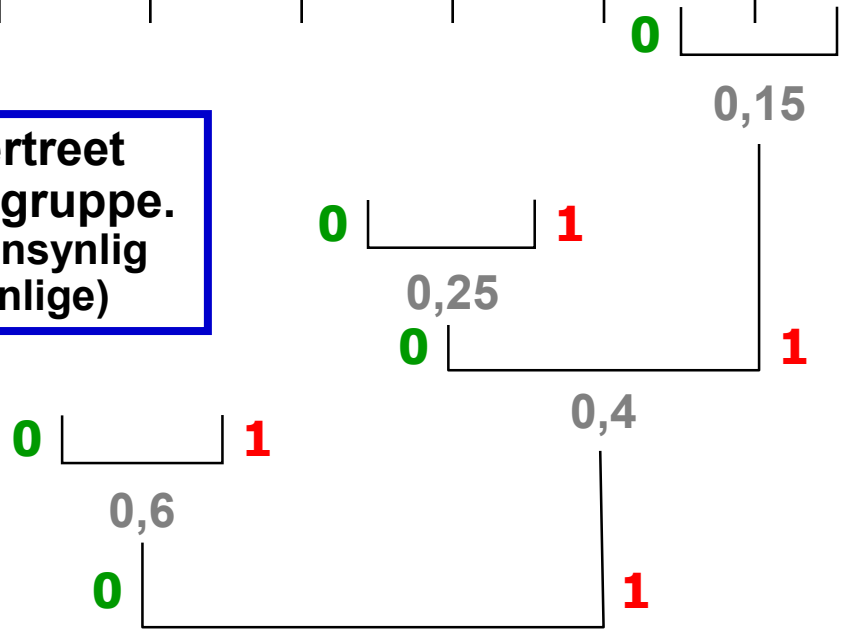


Eksempel: Huffman-koding

- 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

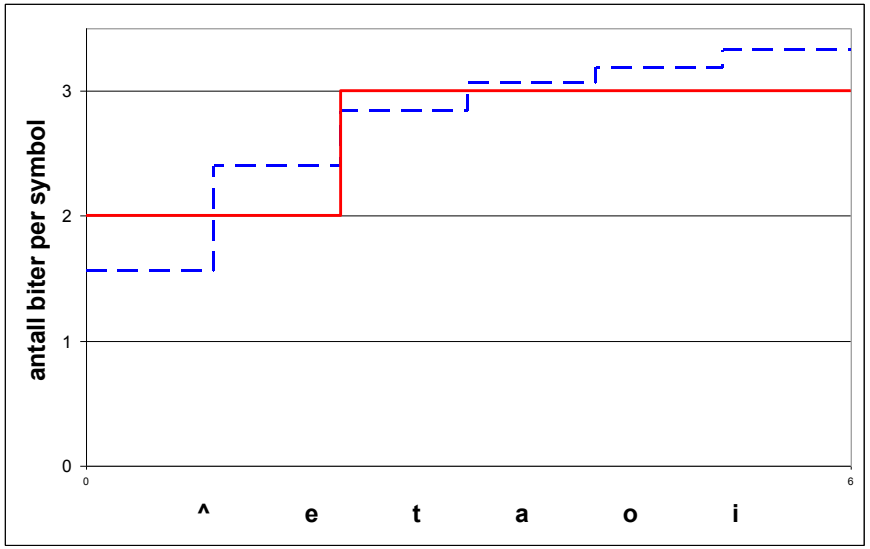
- Hvis gjennomsnittlig antall biter per symbol, c, skal være lik entropien, H, så må:

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

- 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

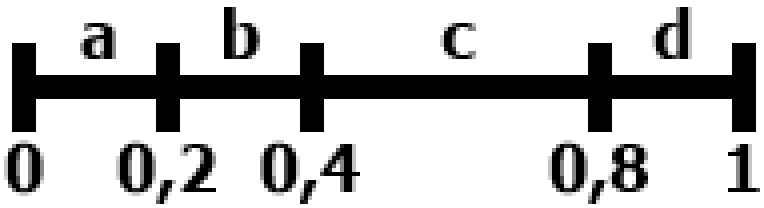
er gjennomsnittlig bitforbruk per symbol etter Huffman-koding:

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

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

Aritmetisk koding: Grunntanke

- Symbolsannsynlighetene summerer seg til 1.
- Dermed definerer de en **oppdeling av intervallet [0, 1)**.
  - Hvert delintervall representerer ett symbol.

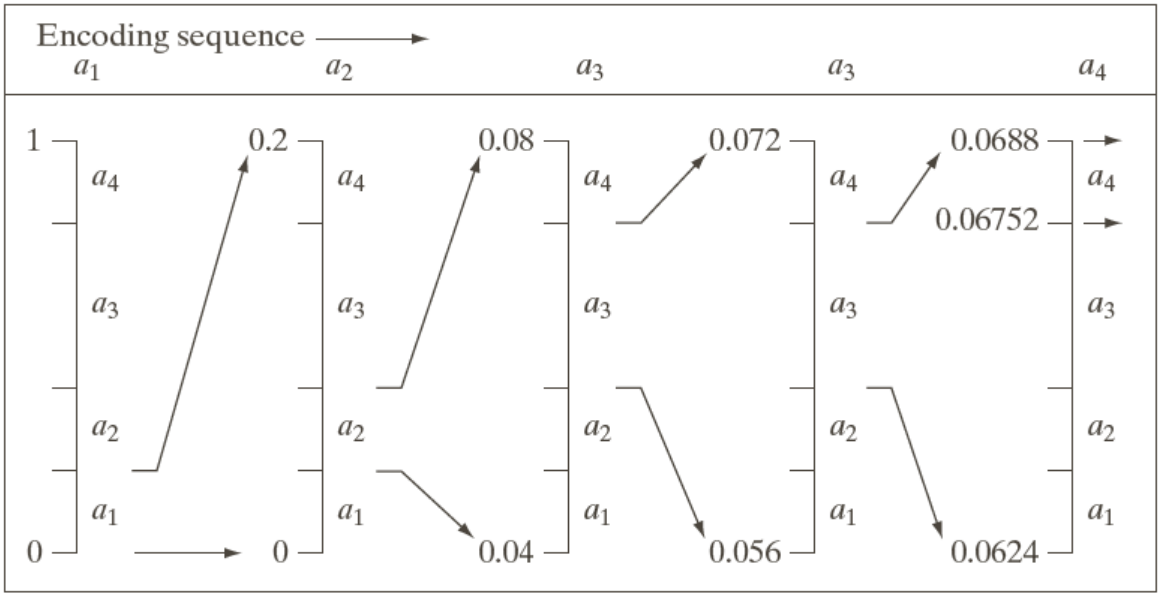


- Har vi to symboler etter hverandre, kan vi **oppdele intervallet som representerer det første symbolet**.
  - Hvert delintervall representerer symbolparet; det første symbolet etterfulgt av ett symbol.

- Tilsvarende for flere symboler etter hverandre.
- Resultat: Et halvåpent delintervall av [0, 1).
- Finner så en bitsekvens som representerer intervallet.

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

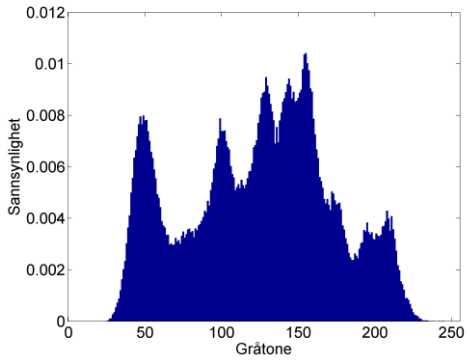
- Modell: Alfabet [a, b, c] med sannsynligheter [0,6, 0,2, 0,2].
- Hvilket delintervall av [0, 1) vil entydig representere meldingen **acaba** ?

- **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).

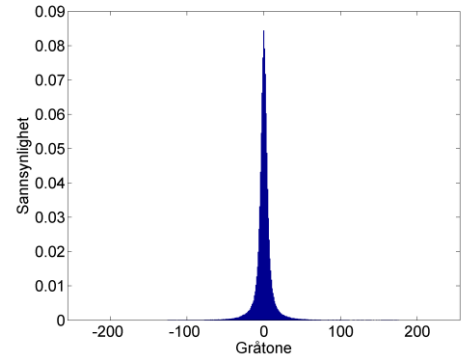
- Et tall i dette intervallet, f.eks. 0,53125, vil entydig representere **acaba**, forutsatt at mottakeren har den samme modellen og vet når å stoppe.

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.
- Bildet f gjenskapes ved transformen:
  $f_1 = g_1,\ f_2 = g_2+f_1,\ \ldots,\ f_N = g_N+f_{N-1}$



Entropi ≈ 7,45 => CR = b/c ≈ 1,07



Entropi ≈ 5,07 => CR = b/c ≈ 1,58

Løpelengde-transform

- Ofte inneholder bildet objekter med lignende gråtoner, f.eks. svarte bokstaver på hvit bakgrunn.
- Løpelengde-transformen (eng.: *run-length transform*) **utnytter at 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:

  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		

» 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

Ikke-tapsfri JPEG-dekompresjon

- Differansene fra den originale blokken er **små**!

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

-

123	122	122	121	120	120	119	119
121	121	121	120	119	118	118	118
121	121	120	119	119	118	117	117
124	124	123	122	122	121	120	120
130	130	129	129	128	128	128	127
141	141	140	140	139	139	138	137
152	152	151	151	150	149	149	148
159	159	158	157	157	156	155	155

=

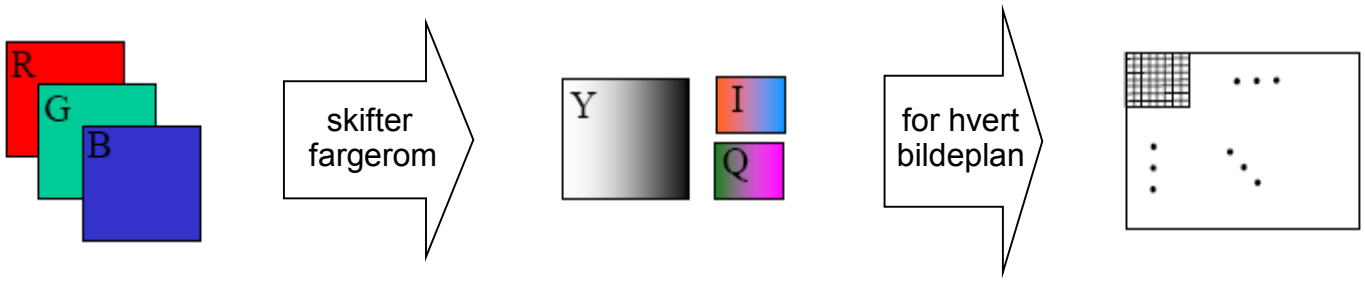
1	3	0	-1	-2	-1	-2	-1
0	0	-1	-1	0	2	2	0
5	3	3	3	3	3	3	3
0	0	2	3	4	4	4	4
-3	-3	-1	0	2	0	-1	-2
2	1	3	2	1	0	1	2
2	-4	1	1	2	3	1	3
-3	0	0	-2	-1	2	2	1

- De er **likevel ikke 0**.
- Det kan bli gjort forskjellig feil på nabopiksler, spesielt dersom de tilhører forskjellige blokker.
  - Kompresjon / dekompresjon kan derfor gi **blokk-artefakter**; rekonstruksjonsfeil som gjør at vi ser at bildet er blokk-inndelt.

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 (normalt) 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.

Intet fargerom er spesifisert i del 1 (1992) av JPEG-standarden. En senere del, del 5 (2009), spesifiserer filformatet JFIF (JPEG File Interchange Format). Her brukes fargemodellen $Y'C_BC_R$



Blokk-artefakter

- Blokk-artefaktene øker med kompresjonsraten.

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

Erosjon

- Plassér strukturelementet S slik at origo overlapper med posisjon (x, y) i innbildet f og beregn utbildet g ved:

$$g(x,y) = \begin{cases} 1, & \text{hvis } S \textbf{ passer } f \\ & \text{i posisjonen } (x,y) \\ 0, & \text{ellers} \end{cases}$$

- Vi skriver erosjon av et bilde f med strukturelement S som: $f \ominus S$

0	1	0	0	0	0	0	1	1	0	0
1	1	1	0	0	0	1	1	1	1	0
0	1	1	1	0	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	0	1	1	1	0
0	1	1	1	1	0	0	0	1	1	1
0	0	1	1	0	0	0	0	0	1	0

Erodert med

1	1	1
1	1	1
1	1	1

gir

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	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	1	0
1	1	1
0	1	0

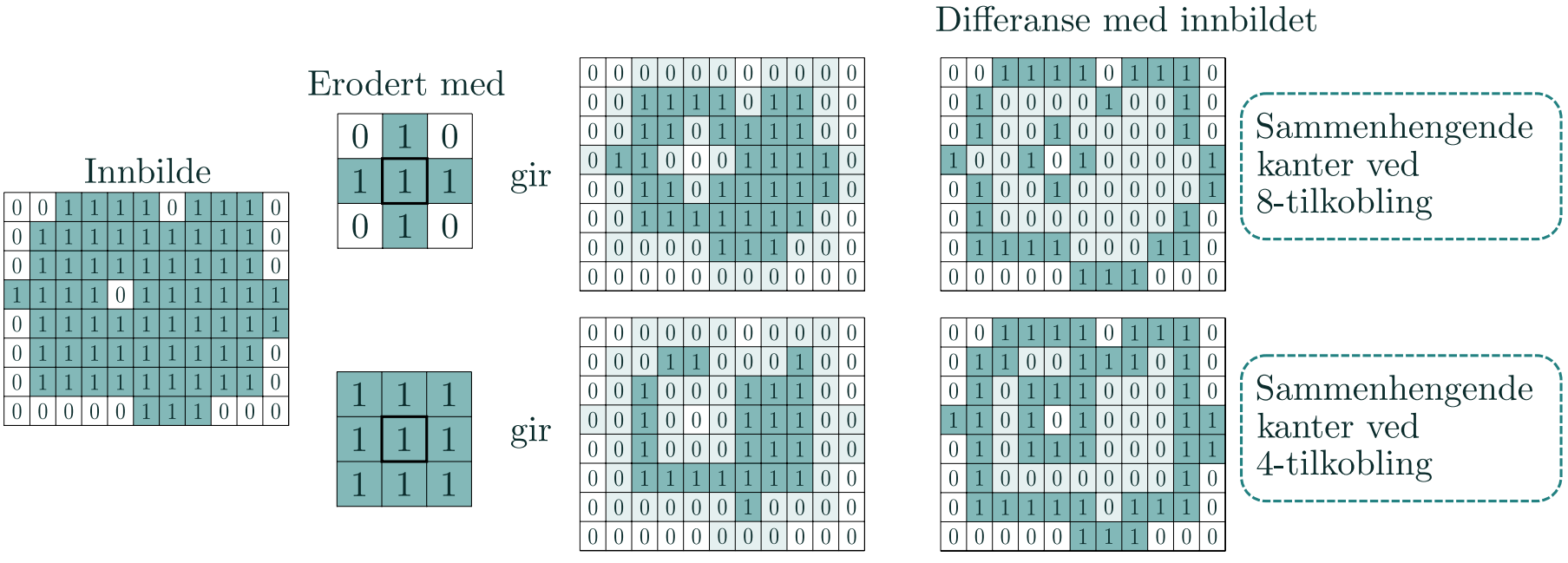
gir

0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	1	0	0
0	0	1	0	0	0	1	1	0	0	0
0	0	0	1	0	1	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0
0	0	0	1	1	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0

Anvendelse av erosjon: kantdeteksjon

Erodering fjerner piksler langs omrisset av et objekt

- Kantene til objektene i bildet kan finnes ved $g = f - (f \ominus S)$
- Strukturelementet S bestemmer kantens tilkoblingstype:



Erosjon: Andre egenskaper

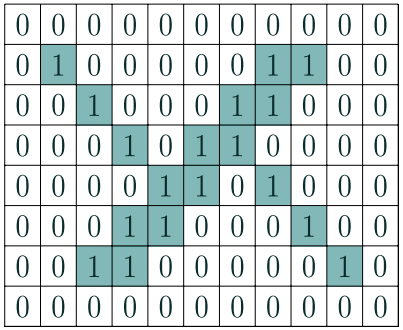
- Erosjon er ikke kommutativ: $f \ominus S \neq S \ominus f$
- Erosjon er ikke assosiativ, men har heller $(f \ominus S_1) \ominus S_2 = f \ominus (S_1 \oplus S_2)$

Dilasjon (dilatasjon)

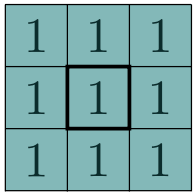
- Rotér S 180 grader for å få $\hat{S}$ og plasser det slik at origo overlapper posisjon (x, y) i innbildet f og beregn utbildet g ved

$$g(x,y) = \begin{cases} 1, & \text{hvis } \hat{S} \textbf{ treffer } f \text{ i posisjonen } (x,y) \\ 0, & \text{ellers} \end{cases}$$

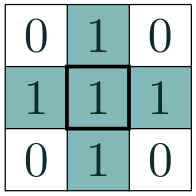
- Dilasjon av et bilde f med strukturelement S skrives som: $f \oplus S$



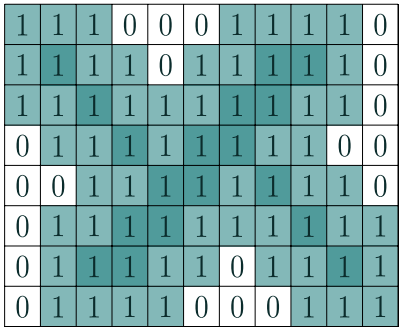
Dilatert med

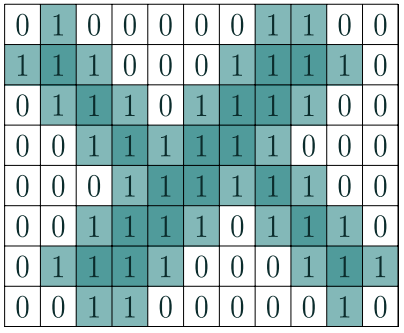


gir



gir

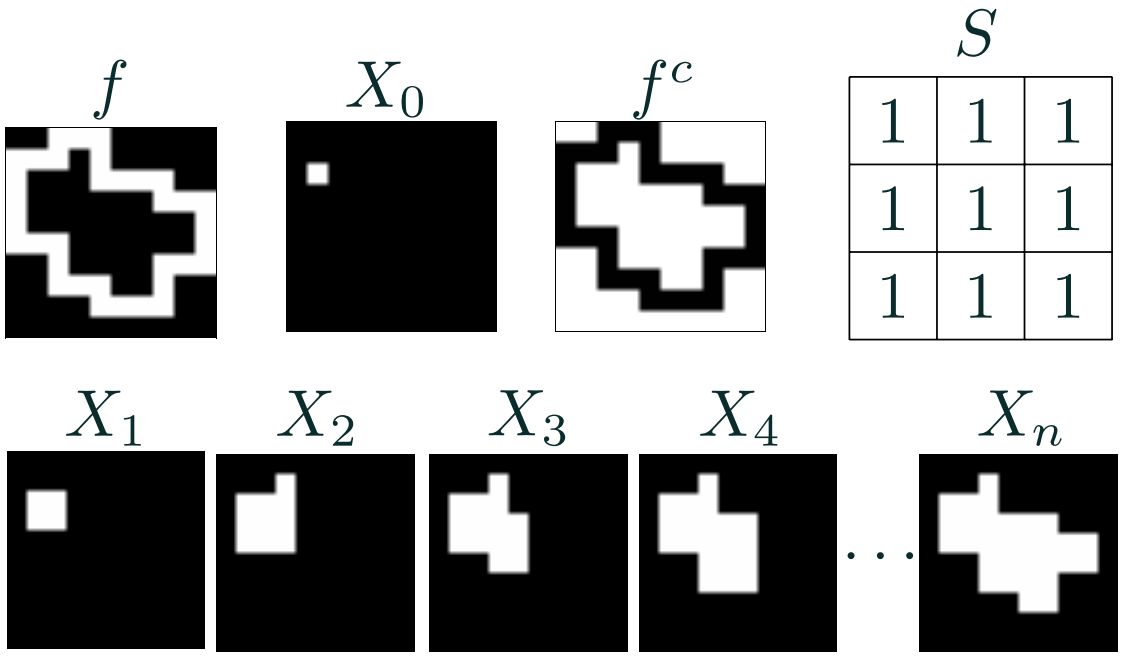




Anvendelse av dilasjon: region-fylling

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



Her er hvit forgrunn og svart bakgrunn

Bildene er hentet fra `denne siden` (siden beskriver også dilasjon)

Dilasjon: Andre egenskaper

- Dilasjon er *kommutativ*: $f \oplus S = S \oplus f$
- Dilasjon er *assosiativ*: $f \oplus (S_1 \oplus S_2) = (f \oplus S_1) \oplus S_2$

Dualitet

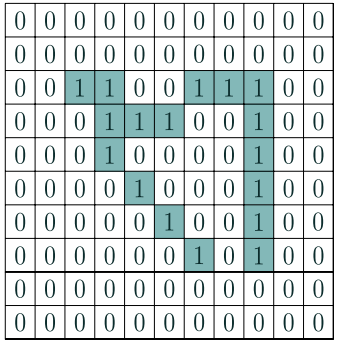
- Dilasjon og erosjon er **duale**, dvs at vi kan uttrykke dem ved hjelp av hverandre:

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

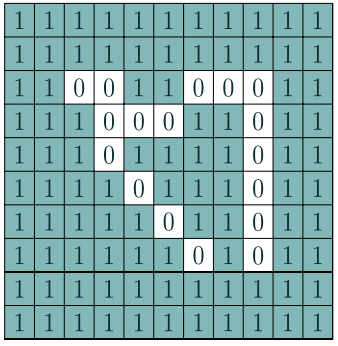
$$f \ominus S = \left(f^c \oplus \hat{S}\right)^c$$

- $\Rightarrow$ Vi kan utføre dilasjon og erosjon ved samme prosedyre så lenge vi kan rotere S 180 grader og finne komplementet til et binært bilde!

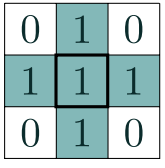
Bildet f



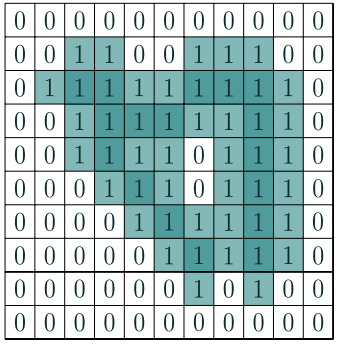
Komplementet f^c



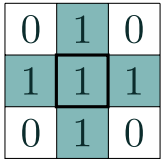
Dilatert med



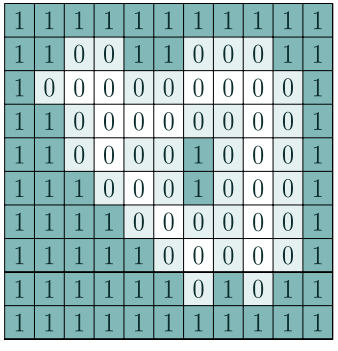
gir



Erodert med



gir



Åpning

- **Erosjon**: fjerner alle strukturer som ikke kan inneholde strukturelementet og krymper andre strukturer
- Dilasjon av resultatet med samme strukturelement kan vi omtrentlig gjenskape de strukturene som "overlevde" erosjonen
- **Morfologisk åpning**: $f \circ S = (f \ominus S) \oplus S$
- Åpning i form av at operasjonen kan skape en åpning mellom to strukturer som henger sammen ved en tynn linje/"bro" uten å krympe strukturene i betydelig grad

Lukking

- **Dilasjon**: Utvider strukturer, fyller hull og innbuktninger i omriss
- Erosjon av resultatet med samme strukturelement vil strukturene stort sett få gjenskapt sin opprinnelige størrelse og form, men uten hull og innbuktninger som ble fylt ved dilasjon
- **Morfologisk lukking**: $f \bullet S = (f \oplus S) \ominus S$
- Lukking i form av at operasjonen kan lukke en åpning mellom to strukturer som er skilt ved et lite gap, uten av strukturene i seg selv vokser i betydelig grad

Dualitet mellom åpning og lukking

- Lukking er en dual operasjon til åpning:

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

- Kan utføre begge operasjonene med kode bare for den ene, hvis vi kan speilvende og komplementere et binært bilde!
- **Lukking** er *ekstensiv* transformasjon (piksler legges til)
- **Åpning** er *antiekstensiv* transformasjon (piksler fjernes)

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

“Hit-or-miss”-transformasjonen

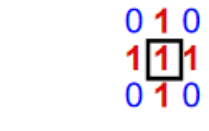
- Har bilde f og strukturelement S
- Lar S være definert ved $\{S_1, S_2\}$, som er to strukturelementer som har ikke noe til felles
- **“Hit-or-miss”-transformasjonen:**

$$f \circledast S = f \circledast \{S_1, S_2\} = (f \ominus S_1) \cap (f^c \ominus S_2)$$

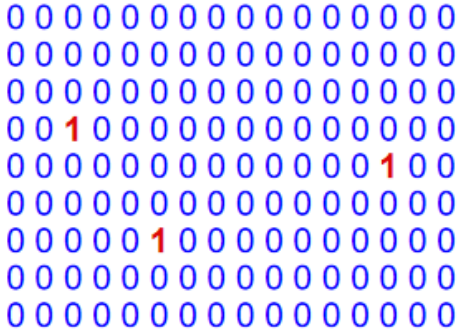
- Vi får en forgrunnspiksel i utbildet bare hvis
  - S_1 passer forgrunnen rundt pikselen **og**
  - S_2 passer bakgrunnen rundt pikselen
- Kan brukes til finne/behandle visse mønstre i et bilde, f.eks ved
  - Finne bestemte strukturer
  - Fjerne enkeltpiksler
  - Brukt til tynning

Eksempel: “Hit-or-miss”

Et bilde A

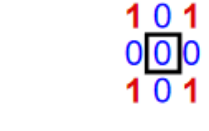


Strukturelement S_1



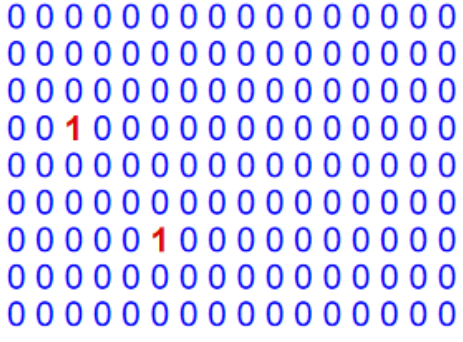
Resultat etter erosjon med S_1

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



Strukturelement S_2

A^c erodert med S_2

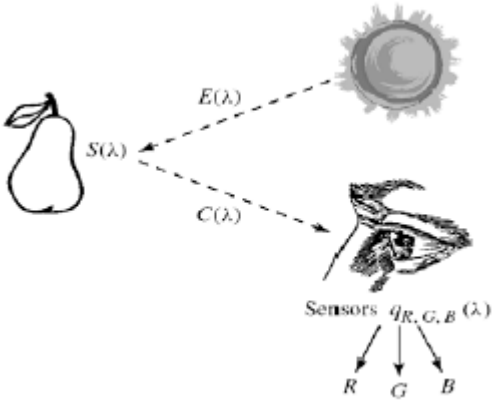


«Hit-or-miss»-resultatet

Logisk AND av de
to delresultatene

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:

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

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

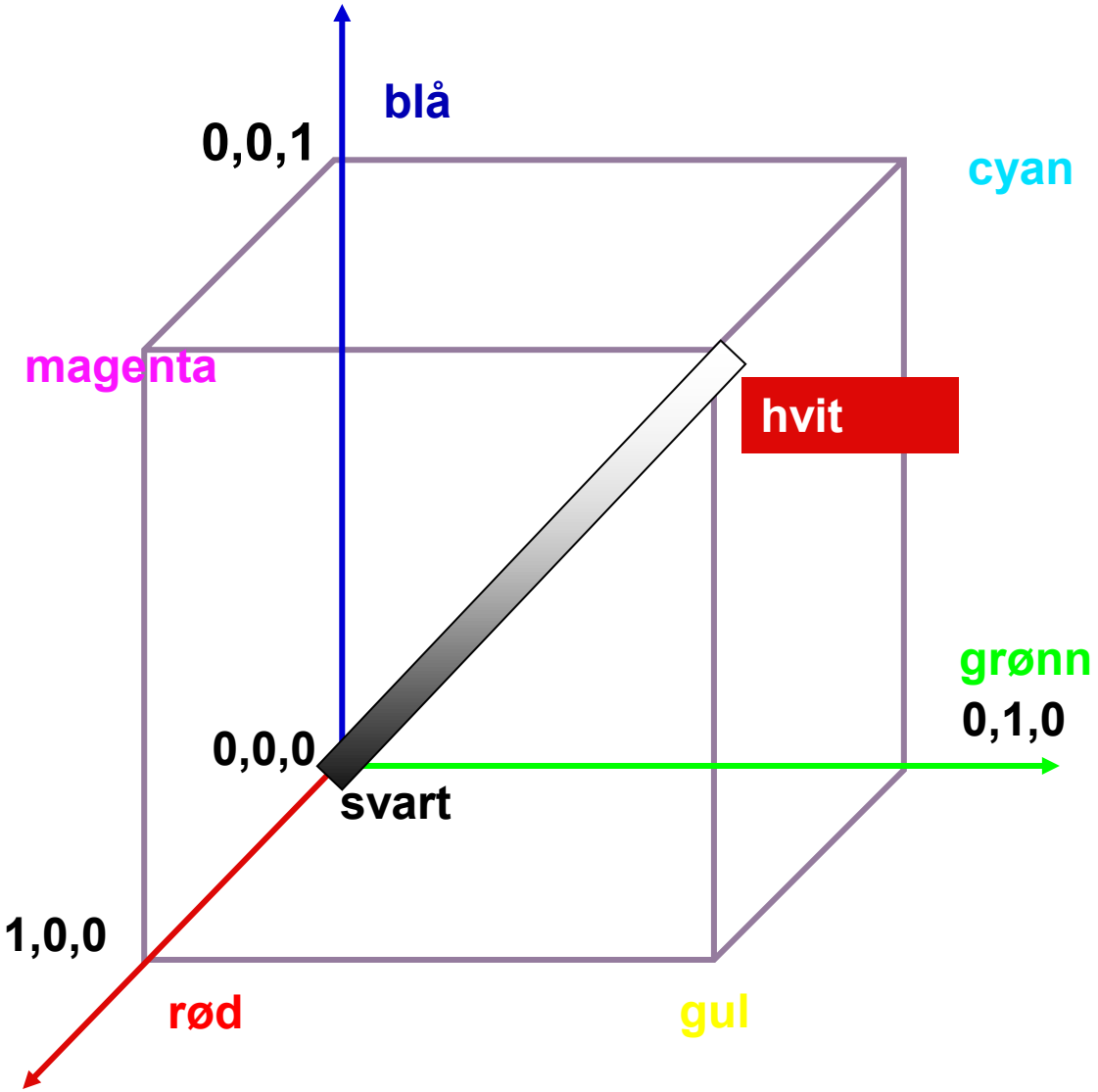
$$B = \int E(\lambda)\ S(\lambda)\ q_B(\lambda)\, d\lambda$$

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

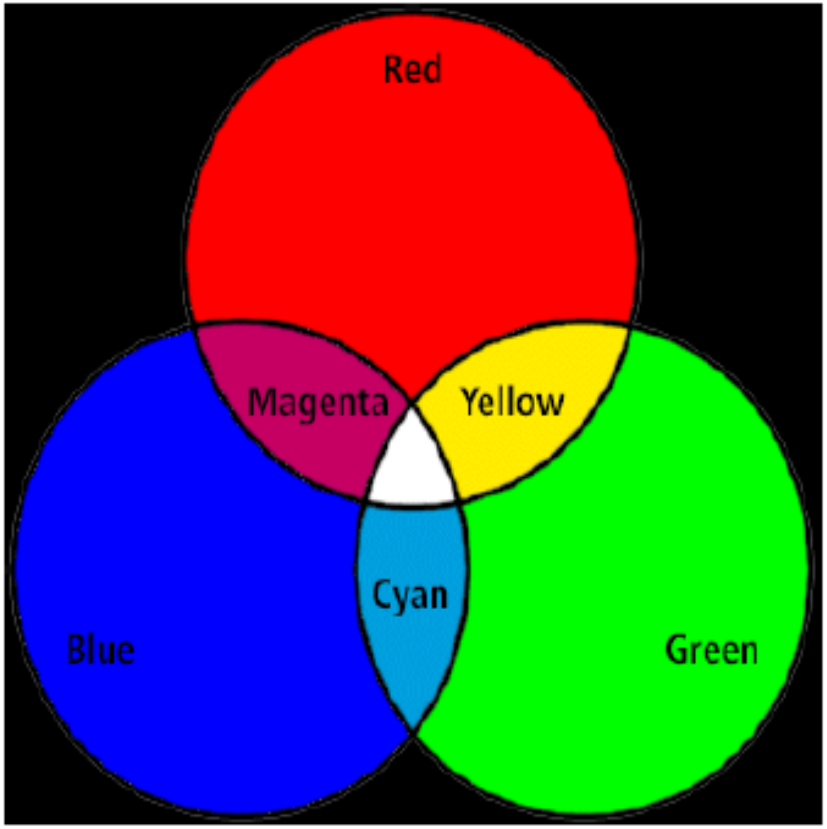
RGB-kuben

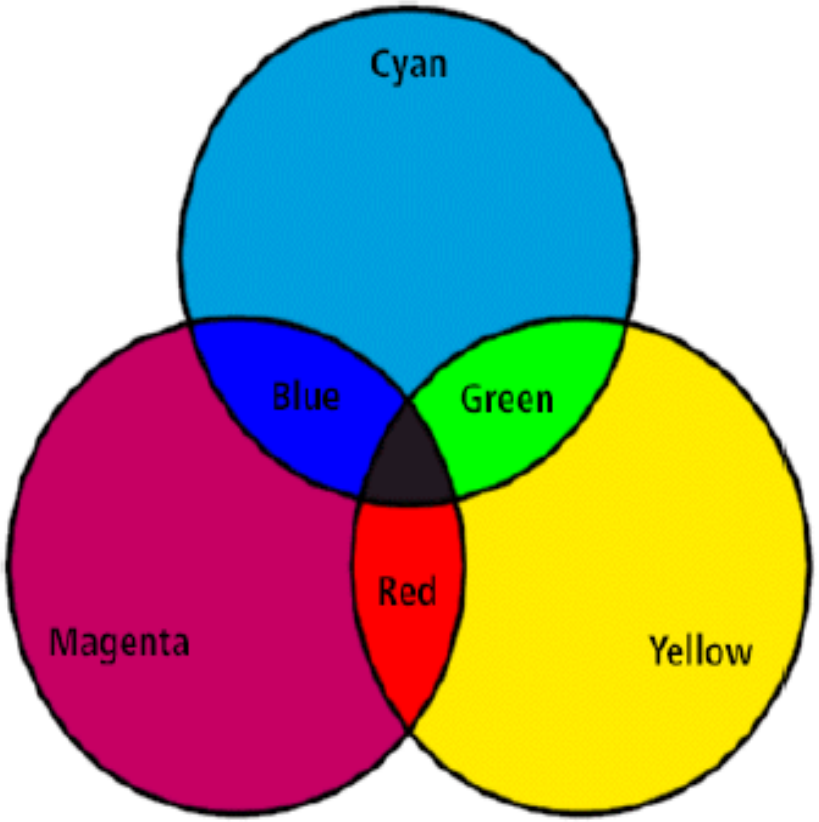
Gråtonebilder:
r=g=b



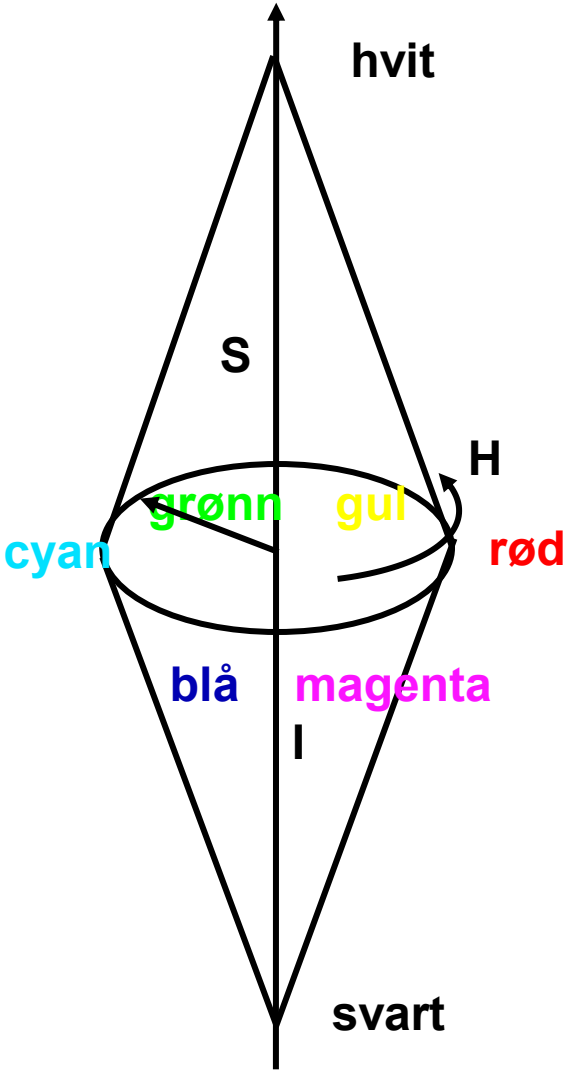
RGB og CMY

- 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= 2π/3, blå= 4π/3,
  gul: H=π/3, cyan= π, magenta= 5π/3

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