IN2070 - Repetisjon del 2

18. mai 2022

- Naboskapsoperasjoner del 1 og 2
- Morfologi
- Kompresjon og koding del 1 og 2
- Farger og fargerom

Regne ut en konvolusjon

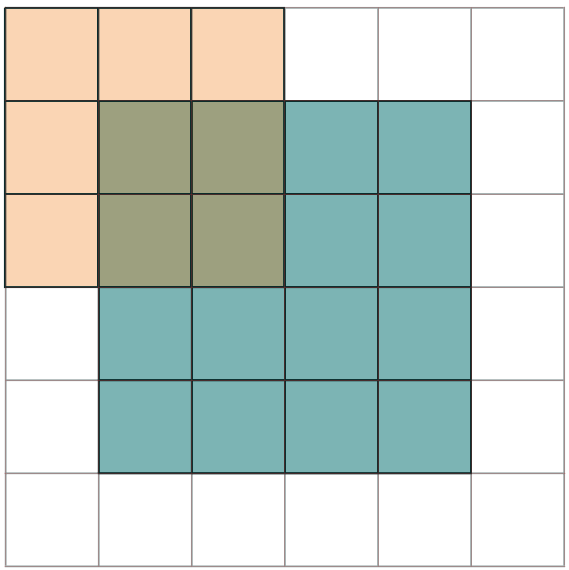
Konvolusjon mellom et filter h og bilde f:

$$g(x,y) = \sum_{s=x-a}^{x+a} \sum_{t=y-b}^{y+b} h(x-s, y-t) f(s,t)$$

- Regne ut responsen i en posisjon (x, y):
  1. Roter konvolusjonsfilteret 180 grader
  2. Legg det roterte filteret over bildet slik at filterets origo overlapper posisjonen (x, y) i bildet
  3. Multipliser hver vekt i det roterte konvolusjonsfilteret med underliggende pikselverdi
  4. Summen av produktene gir responsen $g(x, y)$
- Regne ut responsen i alle posisjoner: Flytt filteret over bildet og beregn responsen for hver posisjon med overlapp

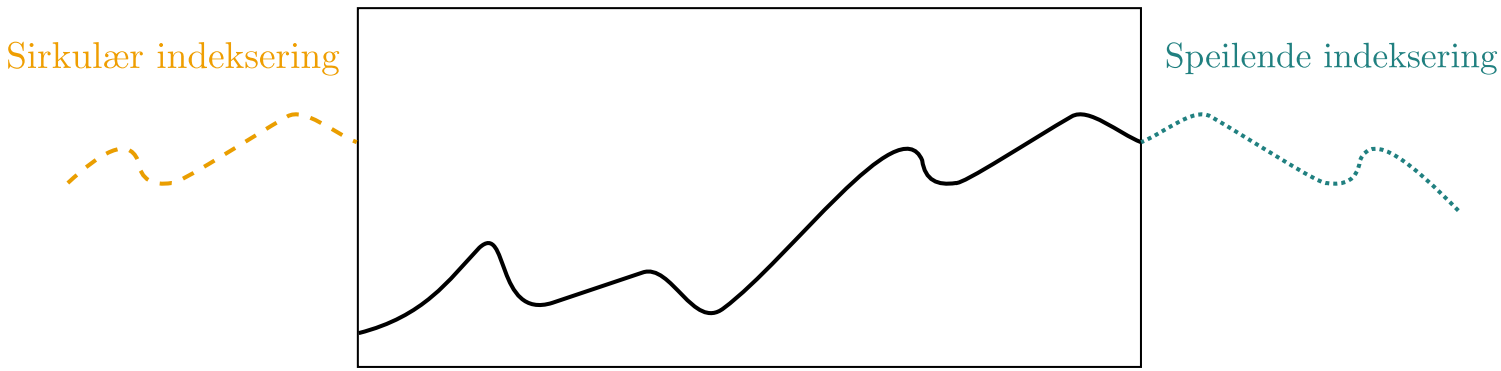
Hvor stort blir utbildet?

- **Alt. 1:** Beregn pikselverdier bare der hele filteret er innenfor innbildet.
  Hvis bildet er $M \times N$ og filteret er $m \times n$ (m, n odde), vil utbildet få størrelse: $(M - m + 1) \times (N - n + 1)$.
- **Alt. 2:** Behold innbildets størrelse:
  - Filterorigo innenfor innbildet
  - Langs bilderanden må det gjøres et valg:
    - **Utvide** bildet ved å legge til flere verdier rundt det
    - Endre filterets form og størrelse



Hva gjør vi langs bilderanden?

- Utvid innbildet:
  - Med 0-ere (nullutvidelse, zero padding)
  - Med en annen, konstant verdi som f.eks bildets gjennomsnitt
  - Ved nærmeste pikselverdi
  - Ved bruk av speilende indeksering
  - Ved bruk av sirkulær indeksering



- Sett pikselet til utbildet til en fast verdi, f.eks $g(x, y) = 0$ eller $g(x, y) = f(x, y)$
- Ignorer posisjonene uten fult overlapp (men mindre utbilde)

Korrelasjon: definisjon

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} h(s, t) f(x + s, y + t)$$

- Ulikt fra konvolusjon at det er pluss istedenfor minus $\rightarrow$ betyr vi ikke skal rotere filteret!
- Tilsvarende som konvolusjon: legger korrelasjonsfilterets origo over (x, y) og multipliserer hver vekt med underliggende pikselverdi. Responsen i (x, y) er summen av disse produktene.

Lavpassfiltre

- Filtre som slipper gjennom *lave frekvenser* og demper/fjerne høye frekvenser
  - **Lave frekvenser:** Trege variasjoner, store trender, myke overganger
  - **Høye frekvenser:** Skarpe kanter, støy detaljer
  - ... dette kommer vi tilbake til i Fourier-forelesningene!
- Effekt fra lavpassfiltre: Glatting/utsmøring/"blurring" av bildet
- Typiske mål: Fjerne støy, finne større objekter
- Utfordring: Vi ønsker å bevare kanter!

Lavpassfilter: Gauss-filter

- For heltallsverdier av x og y, er Gauss-filteret definert ved

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

  der A settes slik at summen av vektene blir 1
- Ikke-uniformt lavpassfilter

Middelverdi eller median?

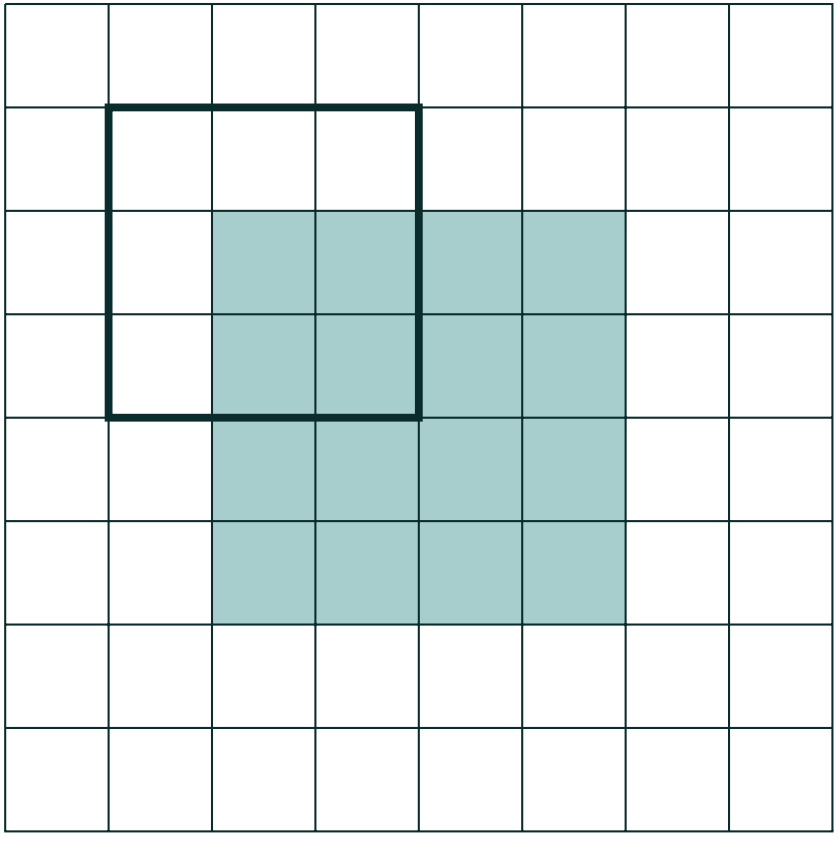
- **Middelverdifilter:**
  - Glatter lokale variasjoner og støy, men også kanter
  - Spesielt god på lokale variasjoner, som kan være mild støy i mange pikselverdier
- **Medianfilter:**
  - Bedre på visse typer støy og til å bevare kanter, men dårligere på lokal variasjon og annen type støy
  - Fungerer spesielt godt på salt-pepper-støy

Lavpassfilter: Rang-filtrering

- Lager en 1D-liste av alle pikslene i naboskapet om (x, y)
- Listen sorteres i stigende rekkefølge
- Responsen i (x, y) er åikselverdien i en bestemt posisjon i den sorterte listen, eller en veiet sum av en del av listen
- Dette er et **ikke-lineært** filter

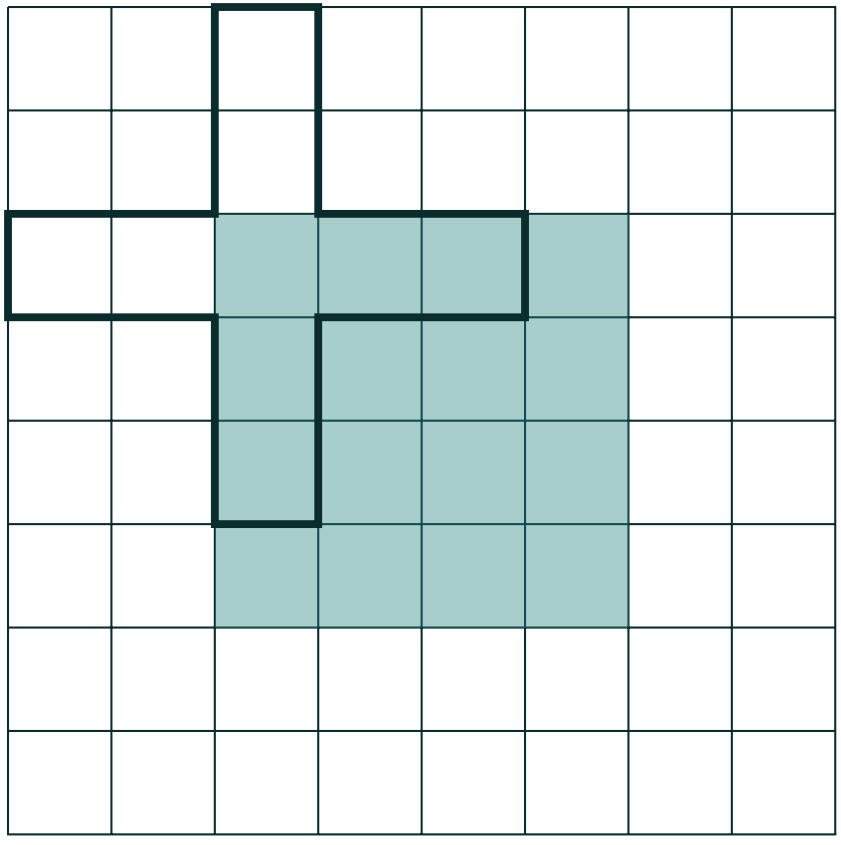
Medianfiltrering og hjørner

Kvadratisk naboskap:



Hjørnene avrundes

Pluss-formet naboskap:



Hjørnene bevares

Høypassfiltre

- Slipper gjennom høye frekvenser og demper/fjerner lave frekvenser
- Effekt:
  - Demper langsomme variasjoner
  - Fremhever skarpe kanter, linjer og detaljer
- **Typiske mål:** Forbedre skarpheten, detektere kanter
- Utfordring: Kan risikere å fremheve støy!

Høypassfiltrering med konvolusjon

- Summen av vektene i konvolusjonsfilteret er typisk 0
- Får positive og negative pikselverdier i utbildet
- Kan være uheldig å alltid bruke resultatet etter konvolusjonen med et høypassfilter, $g(x, y)$, sin absoluttverdi, $|g(x, y)|$
- For framvisning: Gjør $g(x, y)$ positiv med å addere med en konstant og skalér resultatet til et ønsket intervall
- Antar her nullutvidelse og bruker alle posisjoner med overlapp!

Gradient-operatorer

- Prewitt-operatoren:

$$h_x = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \quad h_y = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

- Sobel-operatoren:

$$h_x = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad h_y = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

- Frei-Chen-operatoren:

$$h_x = \begin{pmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{pmatrix} \quad h_y = \begin{pmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{pmatrix}$$

Filtrene kan avvike med 180 grader fra boka siden vi angir dem til bruk ved konvolusjon. Boka antar de skal brukes til korrelasjon.

Gradient-beregning

- Horisontale kanter: $g_x = h_x * f$
- Vertikale kanter: $g_y = h_y * f$
- Gradient-magnitude og -retning:

$$M(i,j) = \sqrt{g_x^2(x,y) + g_y^2(x,y)}$$

$$\theta(i,j) = \tan^{-1}\left(\frac{g_y(i,j)}{g_x(i,j)}\right)$$

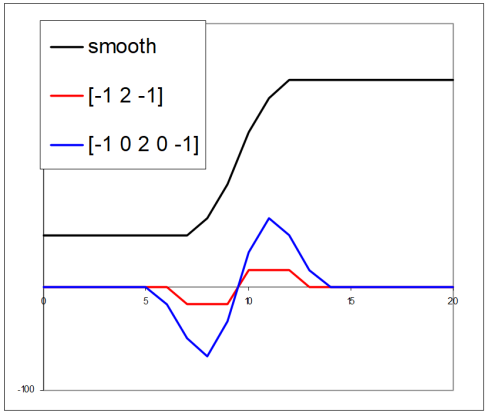
Laplace - operatoren

- Laplace-operatoren er gitt ved:

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

  - Den endrer fortegn der f er et vendepunkt
  - $\nabla^2 f = 0$ markerer kant-posisjon
  - $\left|\nabla^2 f\right|$ har to ekstremverdier pr kant; på starten og på slutten av kanten $\rightarrow$ var dette vi brukte tidligere til å forbedre bildeskarphet!

- **Nullgjennomgang:** Kantens eksakte posisjon
- Finner kant-posisjoner, men ikke retninger



2D-Laplace-operator

- Anvend 1D-Laplace operatoren i begge retninger og summér:

$$\begin{aligned} -\nabla^2 f &= -\frac{\partial^2 f}{\partial x^2} - \frac{\partial^2 f}{\partial y^2} \\ &\approx -f(i-1,j) + 2f(i,j) - f(i+1,j) - \\ &\quad f(i,j-1) + 2f(i,j) - f(i,j+1) \end{aligned}$$

- Dette kan beregnes ved å konvolvere f med:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Full 3×3-Laplace - operator

- Hvis vi i tillegg anvender 1D-Laplace tilnærmingen langs begge diagonaler, får vi det som kan beregnes ved:

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

- Dette er filteret vi brukte til:
  - Punkt-deteksjon
  - Øke bildeskarpheten

Sobel vs. Laplace

- **Sobel-filtrering:** bred kant

- **Laplace-filtrering:** dobbel kant

Fra Laplace til LoG

- Vi gjorde gradient-operatorene støy-robuste ved å bygge inn lavpassfiltrering

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

- Kan gjøre det samme med Laplace-operator!
- Vanlig å bygge inn et Gauss-filter G med gitt σ:

$$h_{\mathrm{LoG}} = -\nabla^2 * G,$$

som gir oss **Laplacian-of-Gaussian**-operatoren (LoG)!

Cannys algoritme

1. Lavpassfiltrér med 2D Gauss-filter med en bruker-bestemt σ
2. Finn gradient-magnituden og gradient-retningen
3. Tynning av gradient-magnitude ortogonalt på kant
   - Hvis en piksel i gradient-magnitude bildet har en 8-nabo i eller mot gradient-retningen med høyrere verdi, settes pikselverdien til 0
4. Hysterese-terskling med to bruker-bestemte terskler T_h og T_l:
   - 4.1 Merk alle piksler der $g(x, y) \geq T_h$
   - 4.2 For alle piksler der $g(x, y) \in [T_l, T_h)$:
     Hvis pikselen er (4 eller 8)-nabo til en merket piksel, så merkes denne pikselen også
   - 4.3 Gjenta trinnet over frem til det ikke merkes noen piksler

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

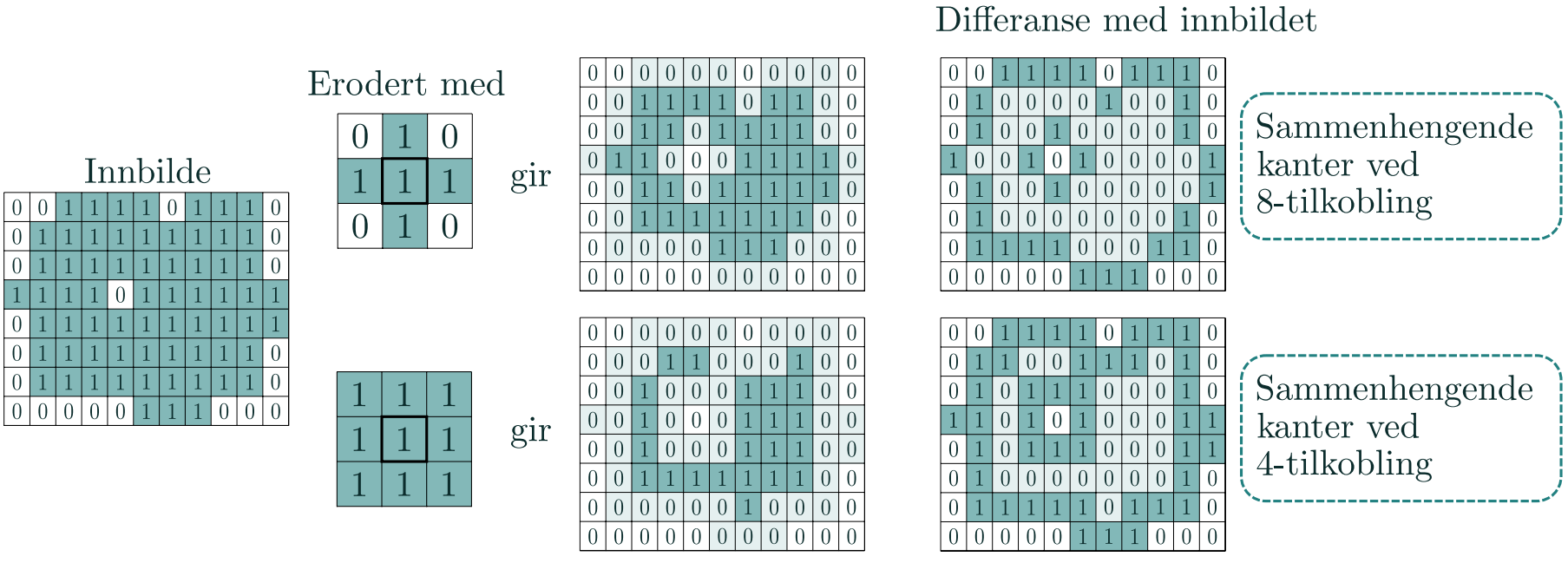
gir

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

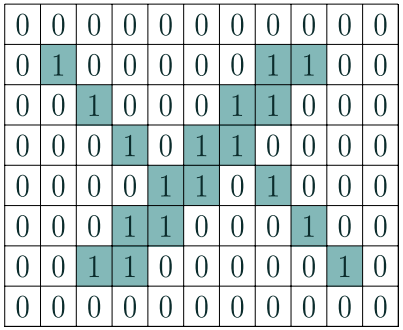
- Erosjon er ikke kommutativ: $f \ominus S \neq S \ominus f$
- Erosjon er ikke assosiativ, men har heller $(f \ominus S_1) \ominus S_2 = f \ominus (S_1 \oplus S_2)$

Dilasjon (dilatasjon)

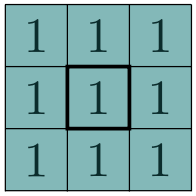
- Rotér S 180 grader for å få $\hat{S}$ og plasser det slik at origo overlapper posisjon (x, y) i innbildet f og beregn utbildet g ved

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

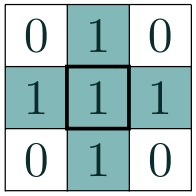
- Dilasjon av et bilde f med strukturelement S skrives som: $f \oplus S$



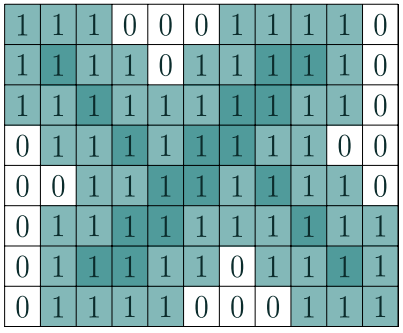
Dilatert med

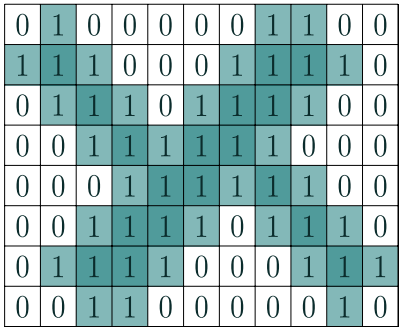


gir



gir

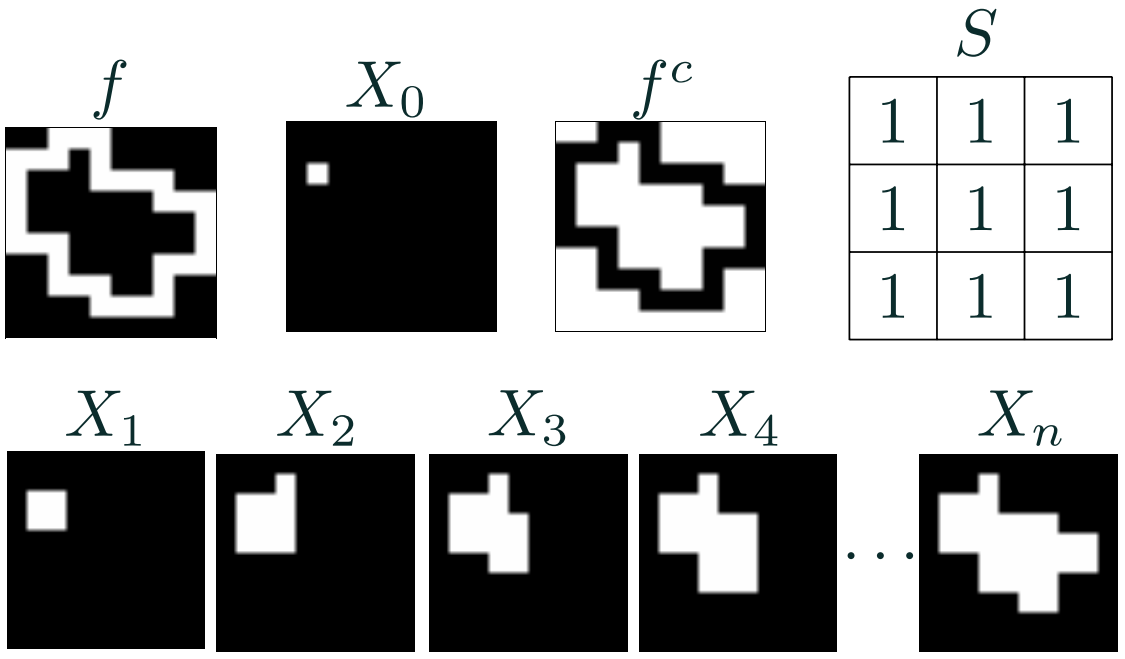




Anvendelse av dilasjon: region-fylling

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

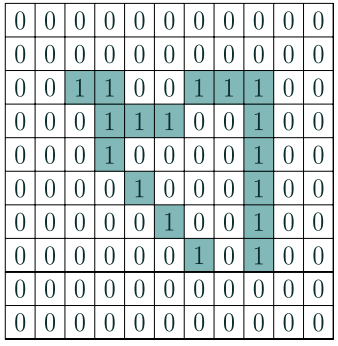
- Dilasjon og erosjon er **duale**, dvs at vi kan uttrykke dem ved hjelp av hverandre:

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

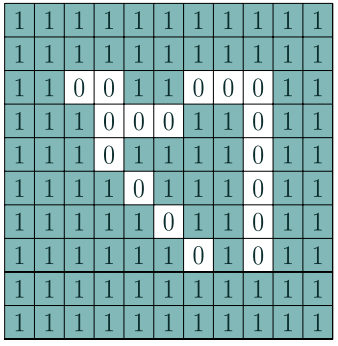
$$f \ominus S = \left(f^c \oplus \hat{S}\right)^c$$

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

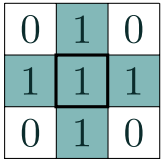
Bildet f



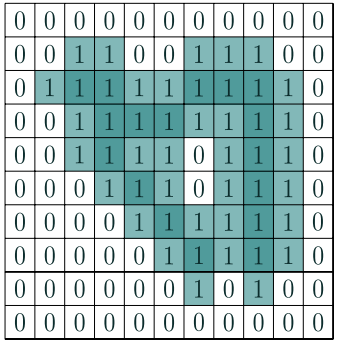
Komplementet f^c



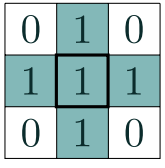
Dilatert med



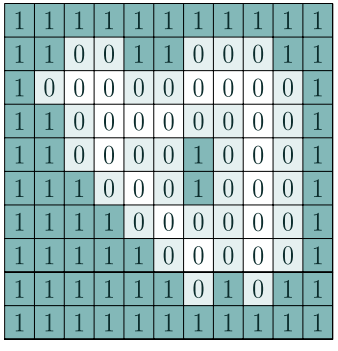
gir



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

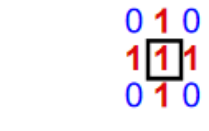
- Har bilde f og strukturelement S
- Lar S være definert ved $\{S_1, S_2\}$, som er to strukturelementer som har ikke noe til felles
- **“Hit-or-miss”-transformasjonen:**

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

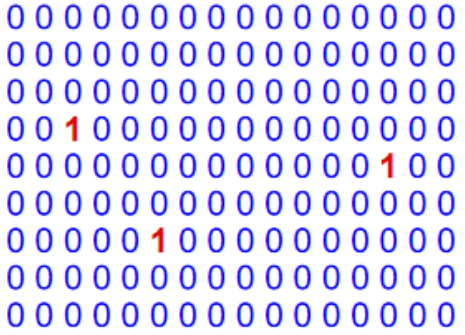
- Vi får en forgrunnspiksel i utbildet bare hvis
  - S_1 passer forgrunnen rundt pikselen **og**
  - S_2 passer bakgrunnen rundt pikselen
- Kan brukes til finne/behandle visse mønstre i et bilde, f.eks ved
  - Finne bestemte strukturer
  - Fjerne enkeltpiksler
  - Brukt til tynning

Eksempel: “Hit-or-miss”

Et bilde A

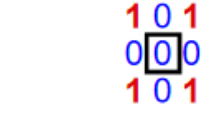


Strukturelement S_1



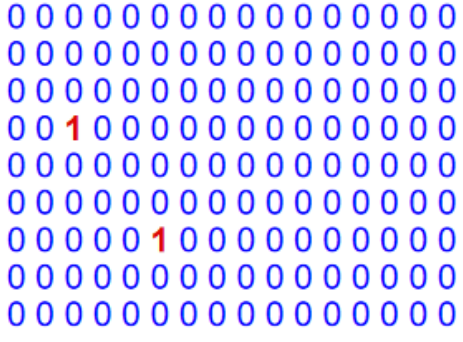
Resultat etter erosjon med S_1

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



Strukturelement S_2

A^c erodert med S_2

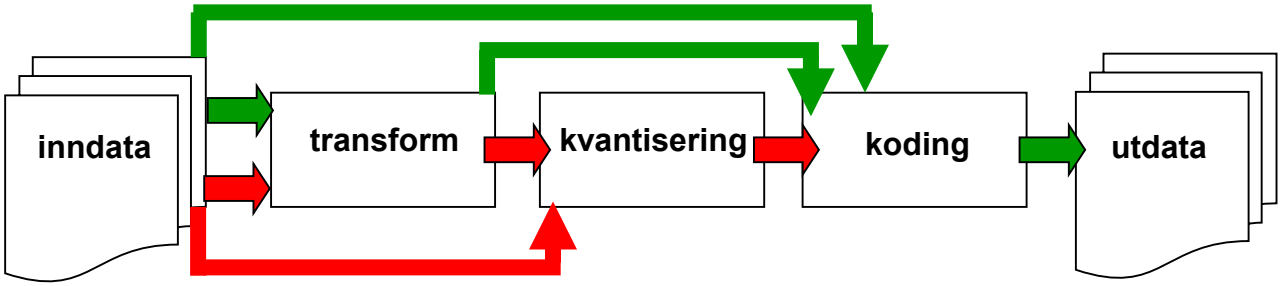


«Hit-or-miss»-resultatet

Logisk AND av de
to delresultatene

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

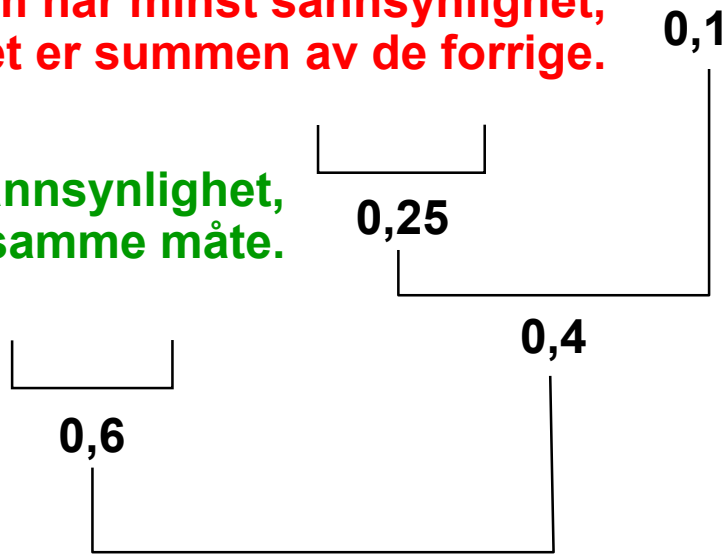
- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

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.

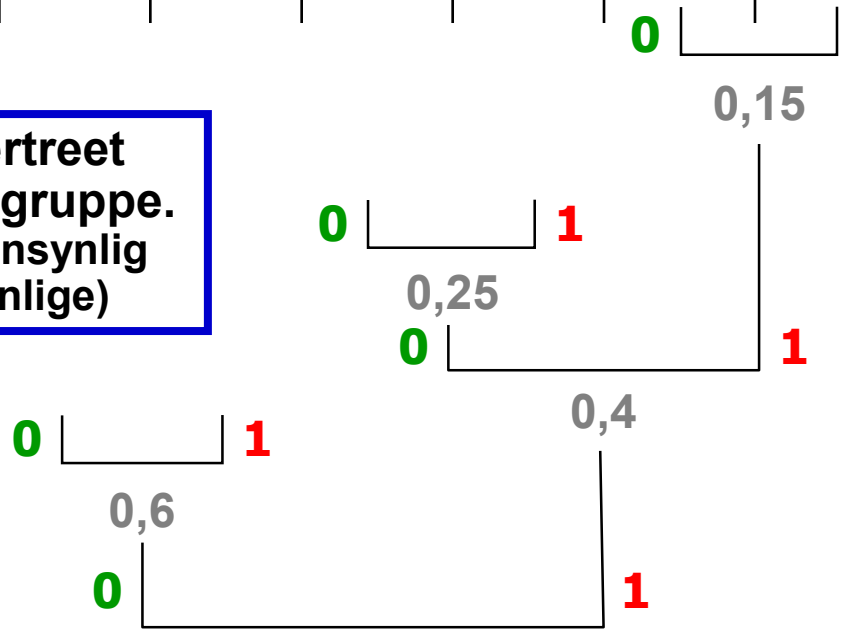


Eksempel: Huffman-koding

- 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

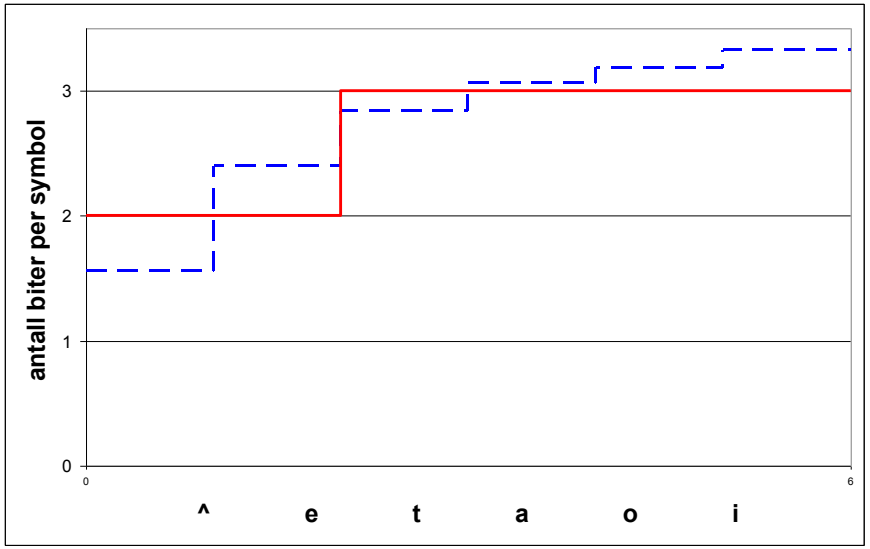
- Hvis gjennomsnittlig antall biter per symbol, c, skal være lik entropien, H, så må:

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

- 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

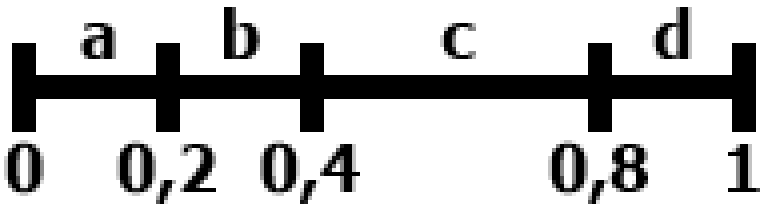
er gjennomsnittlig bitforbruk per symbol etter Huffman-koding:

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

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.

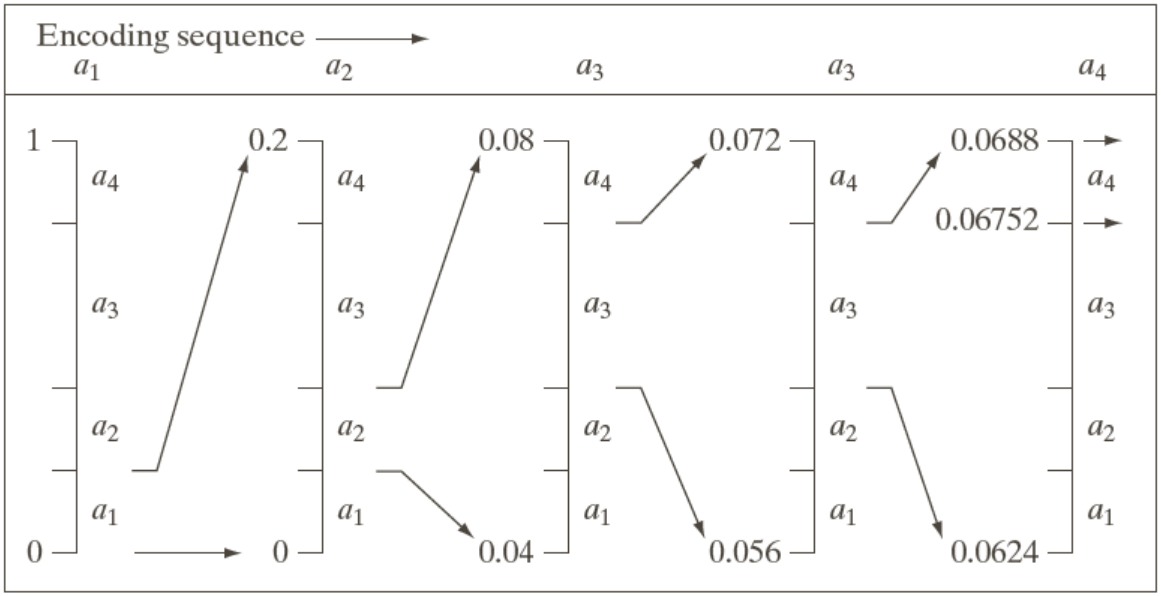


- Har vi to symboler etter hverandre, kan vi **oppdele intervallet som representerer det første symbolet**.
  - Hvert delintervall representerer symbolparet; det første symbolet etterfulgt av ett symbol.

- 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

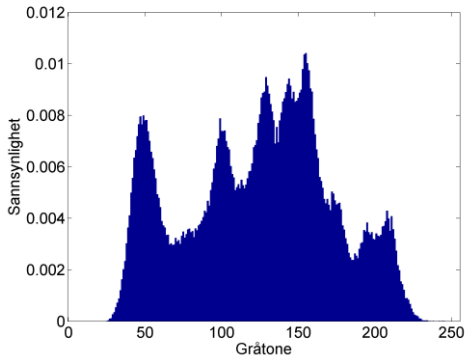
- Modell: Alfabet [a, b, c] med sannsynligheter [0,6, 0,2, 0,2].
- Hvilket delintervall av [0, 1) vil entydig representere meldingen **acaba** ?

- **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).

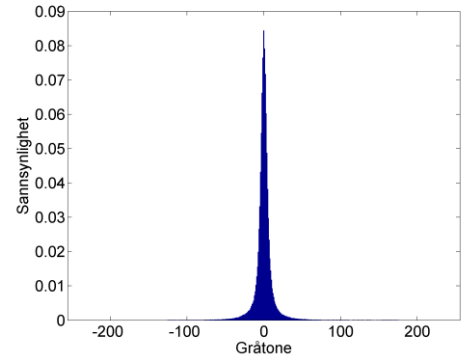
- Et tall i dette intervallet, f.eks. 0,53125, vil entydig representere **acaba**, forutsatt at mottakeren har den samme modellen og vet når å stoppe.

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

=

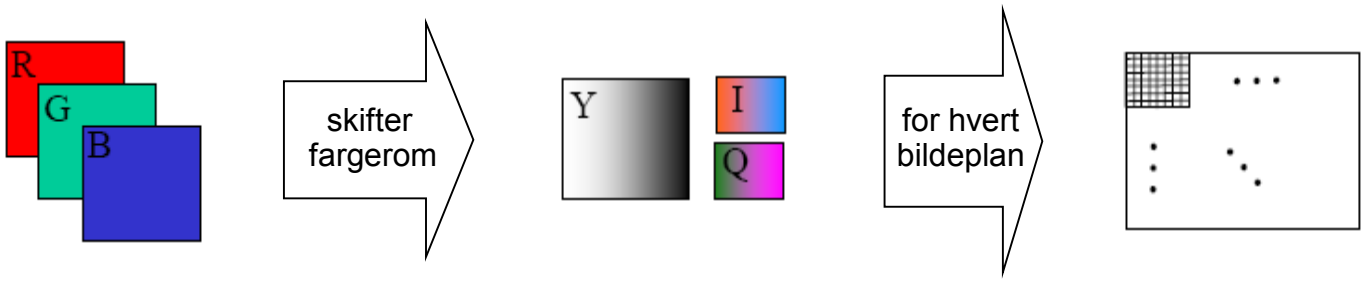
1	3	0	-1	-2	-1	-2	-1
0	0	-1	-1	0	2	2	0
5	3	3	3	3	3	3	3
0	0	2	3	4	4	4	4
-3	-3	-1	0	2	0	-1	-2
2	1	3	2	1	0	1	2
2	-4	1	1	2	3	1	3
-3	0	0	-2	-1	2	2	1

- 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

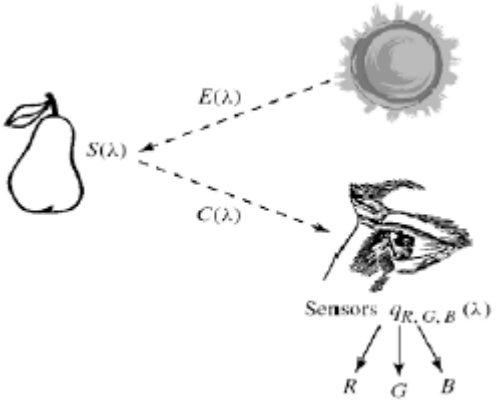
- Skifter fargerom for å separere lysintensitet fra kromasi.
  - Stemmer bedre med hvordan vi oppfatter et fargebilde.
    - Lysintensiteten er viktigere enn kromasi for oss.
  - Kan også gi lavere kompleksitet i hver kanal.
- Nedsampler (normalt) kromasitet-kanalene.
  - Typisk med en faktor 2 i begge 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 $Y'C_BC_R$



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:

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

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

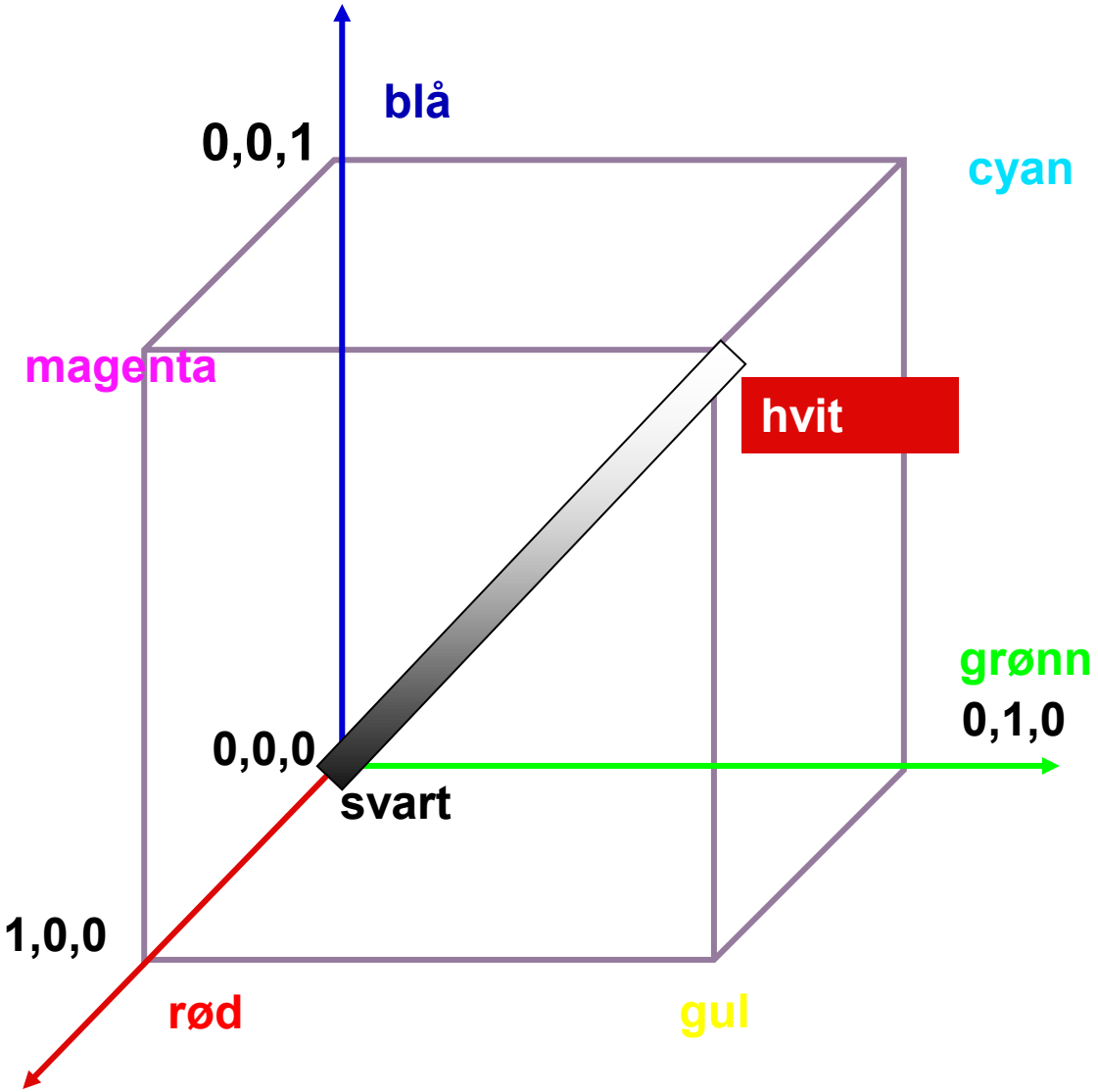
$$B = \int E(\lambda)\ S(\lambda)\ q_B(\lambda)\, d\lambda$$

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

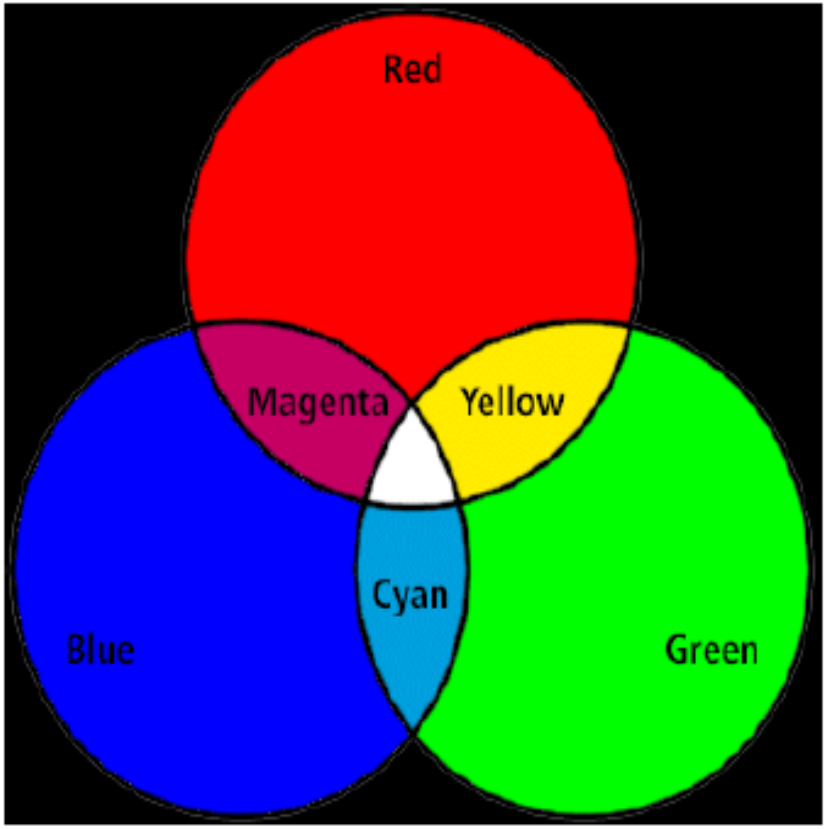
RGB-kuben

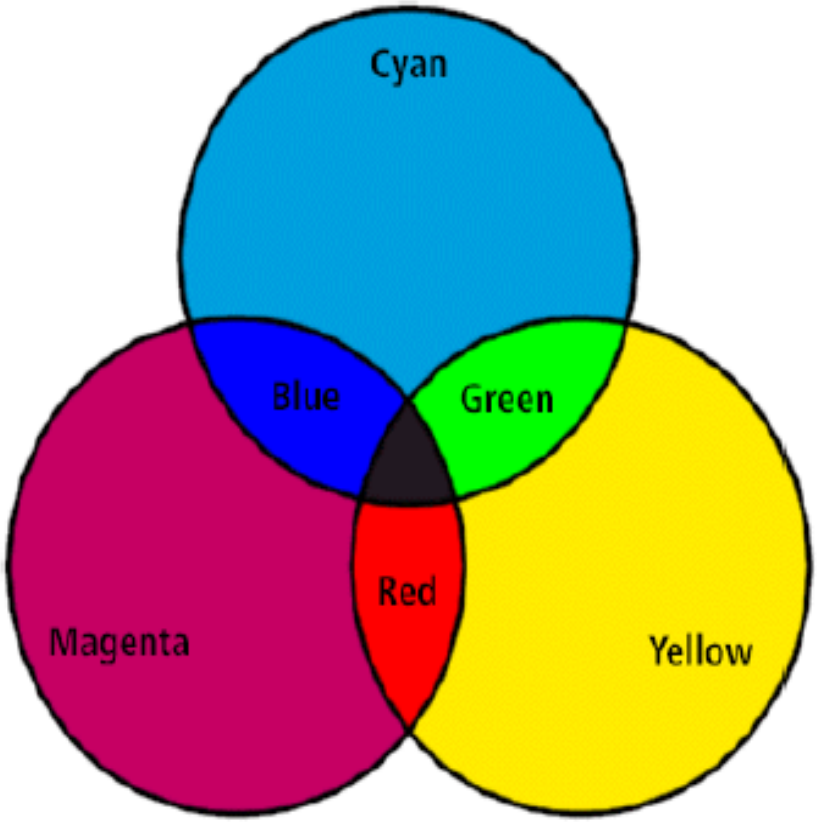
Gråtonebilder:
r=g=b



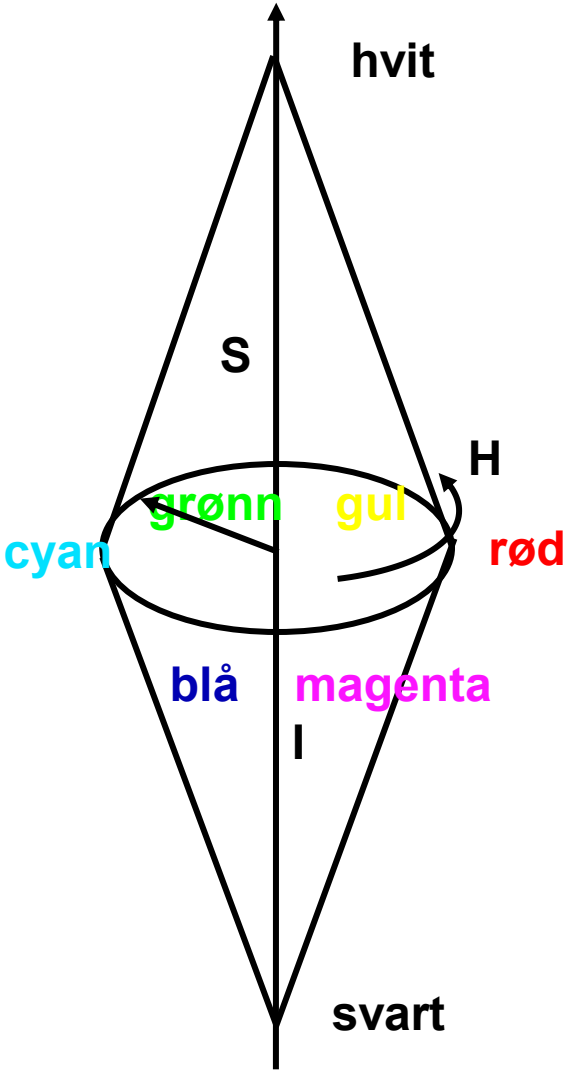
RGB og CMY

- 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.