Oppgave 3a

$\frac{1}{N} \cdot \Sigma_{x=0}^{N-1} \Sigma_{y=0}^{N-1} f(x,y) \cdot \mathrm{e}^{-j2\pi(ux+vy)/N}$

Oppgave 3b

2D diskret konvolusjon

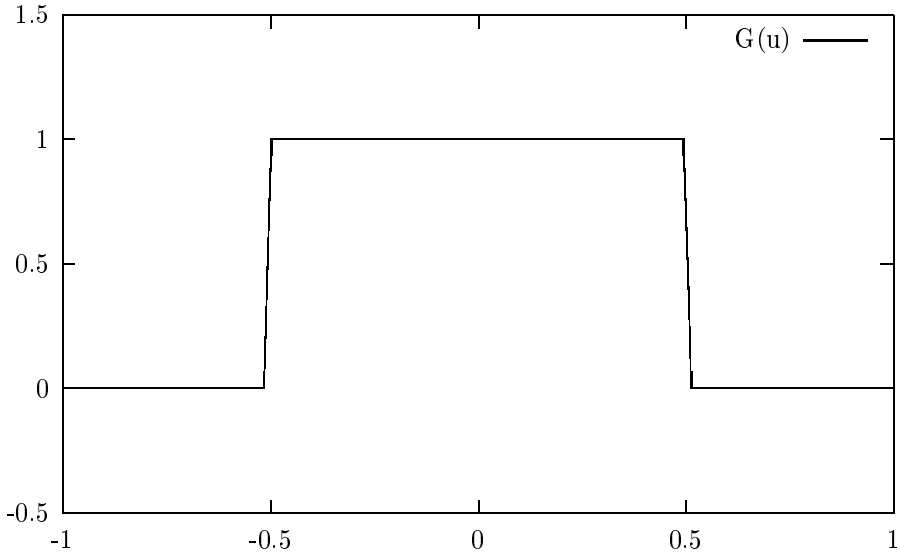
for $x = 0$ **to** M
    for $y = 0$ **to** N
        $h(x,y) = 0$
        for $m = 0$ **to** M
            for $n = 0$ **to** N
                $h(x,y) += f(m,n) * g(x-m, y-n)$
        $h(x,y) = h(x,y)/(M*N)$

2D diskret Fourier Transform

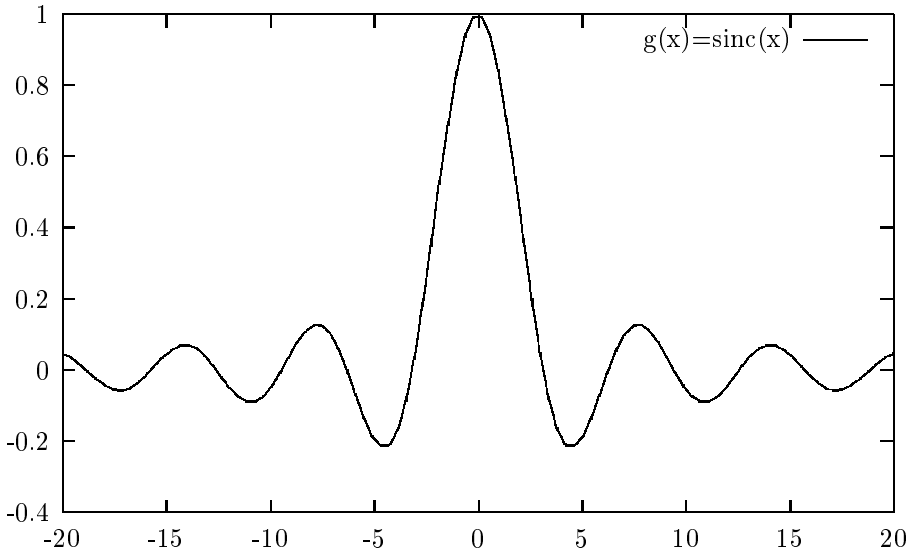
for $u = 0$ **to** $N-1$
    for $v = 0$ **to** $N-1$
        $F(u,v).re = 0$
        $F(u,v).im = 0$
        for $x = 0$ **to** $N-1$
            for $y = 0$ **to** $N-1$
                $F(u,v).re += f(x,y) * cos(2*pi*(ux+vy)/N)$
                $F(u,v).im -= f(x,y) * sin(2*pi*(ux+vy)/N)$
        $F(u,v).re /= N$
        $F(u,v).im /= N$

Oppgave 3c

Konvolusjonsteoremet: $f * g \Leftrightarrow F \cdot G$ og $f \cdot g \Leftrightarrow F * G$
Første del sier at konvolusjon i det romlige domenet $(f * g)$ er det samme som pixelvis multiplikasjon i fourierdomenet $(F \cdot G)$. Andre del gir det motsatte, og er mindre relevant ved vanlig digital bildebehandling.
Diskusjon:
Et ideelt lavpassfilter har i en dimensjon fourierrepresentasjonen u (i 2D multiplikasjon mellom fouriertransform av bildet $f(x, y)$ og $G(u, v)$).



For å forstå hva det ideelle filteret gjør med det romlige bildet $f(x, y)$, kan man bruke versjon 1 av konvolusjonsteoremet. Da må man kjenne den inverse fouriertransformen til den skisserte rektangulærfunksjonen over. Den er $g = \text{sinc}(x) = \frac{\sin(x)}{x}$.



Figurene g og G viser et eksempel på såkalte fouriertransformpar. Det ideelle lavpassfilter er altså en konvolusjon men en maske (g) med sidelober. Dette skaper er stripemønster, kalt ringing, rundt kanter i f.

Oppgave 3d

Høypassfiltre lar kun høyfrekvent bildeinformasjon slippe gjennom, dvs kanter og andre intensitetsforandringer. Høypassfiltre er derfor en vanlig preprosessering før bildeanalysemetoder som baserer seg på kantinfo, som for eksempel Hough-transformen, globale metoder for konturdeteksjon (synamiske konturer, "snakes") og lokale metoder som kantlenking.

Oppgave 4a

Ved læring ønsker man å minimalisere feilen for alle output-target vektor-par $(\vec{y}, \vec{t})$, der output er resultatet fra nettverket og target er ønsket resultat . Ved *on-line* metoden brukes sum of square error:

$$\mathsf{SSE} = \sum_i (\vec{t}_i - \vec{y}_i)^2 \tag{1}$$

der i itererer over alle neuroner i utlaget.
Alternativt kan *batch* metoden brukes. Her benyttes snittet av SSE over alle input-output par som feilmål i stedet.
Backpropagation er gradientbasert. For alle lagene, inkludert output-laget, ønsker vi å finne feil-bidraget for hver forbindelse. Derfor regner vi gradienten for alle forbindelsene mellom to lag. For forbindelsen fra neuron i i foregående lag til neuron j i etterfølgende lag (pass på index på y'er i lginingene for å skille melom y'er fra forskjellige lag):

$$\nabla_{ij} = \frac{\partial \mathsf{SSE}}{\partial w_{ij}} \tag{2}$$

En endring av w i andre forbindelser enn mellom i og j vil ikke påvirke uttrykket. For output laget gir dette:

$$\nabla_{ij} = \frac{\partial \mathsf{SSE}}{\partial w_{ij}} = \frac{\partial \mathsf{SSE}}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ij}} = -2(t_j - y_j) \cdot y_i \tag{3}$$

Dette gjelder ved lineær net-funksjon i output-laget. Dersom net-funksjon ikke er lineær vil $\frac{\partial y_j}{\partial w_{ij}}$ deles til

$$\frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}} = f'(net_j) \cdot y_i \tag{4}$$

$$\nabla_{ij} = -2(t_j - y_j) \cdot f'(net_j) \cdot y_i \tag{5}$$

Oppdatering gjøres ved å endre w_{ij} med en fraksjon av ∇_{ij} i motsatt retning av gradienten.
Ofte benyttes en sigmoid net-funksjon:

$$f(net) = \frac{1}{1 + \mathrm{e}^{-net}} \tag{6}$$

Deriverte av $f(net)$ blir da

$$f'(net) = \frac{e^{-net}}{(1 + \mathrm{e}^{-net})^2} = \frac{1}{1 + \mathrm{e}^{-net}} - (\frac{1}{1 + \mathrm{e}^{-net}})^2 = f(net) \cdot (1 - f(net)) \tag{7}$$

Med sigmoid net-funksjon blir fullstendig oppdatering for hver vekt i ut-laget:

$$\Delta w_{ij} = \eta \cdot \nabla_{ij} = \eta \cdot (t_j - y_j) \cdot y_j \cdot (1 - yj) \cdot y_i \tag{8}$$

hvor η er læringsrate. Merk at det finnes variasjoner av denne læringsregelen.

Oppgave 4b

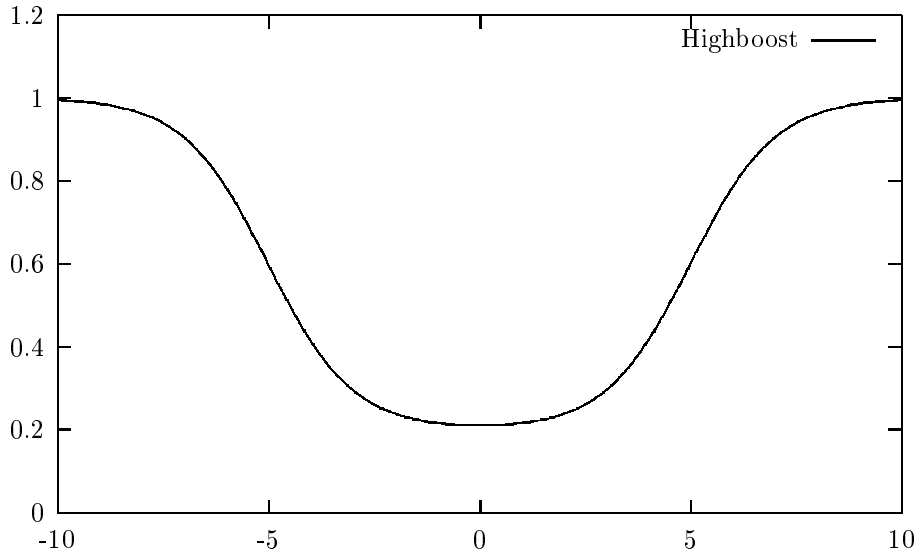
Fordeler: Metoden garanterer en lukket kontur, og det er mulig å
legge inn krav om glatthet langs kanten. Metoden er hurtig og robust for bildestøy.
Ulempe: Man må på forhånd ha bestemt et sett av linjer som alle krysser
objektkonturen.

Oppgave 4c

Algoritme: $\ln(...) \Rightarrow \text{FFT} \Rightarrow \mathsf{H}(...) \Rightarrow \text{FFT}^{-1} \Rightarrow \exp(...)$
$\mathcal{F}(f(x,y))$ vil gi konvolusjon $\mathcal{F}(i(x,y)) * \mathcal{F}(r(x,y))$ i frekvensdomenet. For å kunne skille dem i frekvensdomenet benyttes først logaritme og deretter fouriertransform:

$$\mathcal{F}(\ln(f(x,y)) = \mathcal{F}(\ln(r(x,y)) + \mathcal{F}(\ln(i(x,y)) \tag{9}$$

Nå kan man benytte en transferfunksjon med forskjellig egenskaper for lys og refleksjonskomponenten ved å utnytte at belysningen er vanligvis jevn, dvs. lavfrekvent, mens refleksjonen er høyfrekvent. Her kan en for eksempel benytte et higboostfilter for å øke kontrast samtidig som man komprimerer dynamisk område:



Etter invers transformasjon er en nødt til å benytte den inverse funksjonen til logaritmen, dvs. eksponent.

Oppgave 4d

histogramekvivalisering(I, O, N, P)
I[] : originalt bilde
O[] : ut-bilde
N : antall pixler i bildet
p : antall pixelverdier

for $i = 1$ **to** p	Initialiserer histogramtabell
$\quad T[i] = 0$	
for $x = 1$ **to** N	Histogram
$\quad T[I[x]] += 1$	
$c = 0;$	
for $i = 1$ **to** p	Akkumulert relativt histogram
$\quad c = c + T[i]$	
$\quad T[i] = (c/N) * p$	
for $x = 1$ **to** N	slår opp i tabell for hver pixel i bildet
$\quad O[x] = T[I[x]]$	

———————

Oppgave 4e

$medianfilter(I, O, N, T, M)$
I[] : originalt bilde
O[] : ut-bilde
N : antall pixler i bildet
T[] : nabotabell (hver verdi er en relativ pixelposisjon, inkluderer egen pos)
M : størrelsen på t[] (antall naboer)

```
for x = 1 to N
    m = 0
    for i = 1 to M
        if x + T[i] >= 1 ∧ x + T[i] <= N        Sjekker om nabo ligger i bildet
            V[m] = I[x + T[i]]                  Lager tabell over naboverdier
            m = m + 1                           Teller opp antall lovlige naboer så langt
    O[x] = median(V, m)
```

Prinsippet gjelder også for flere dimensjoner. Endring blir i test av gyldig nabo.

———————

$median(V, m)$
V[] : tabell med tall
m : størrelsen på tabell

```
p = 0
for i = 1 to m                                  Finder høyeste pixelverdi
    if V[i] > p
        p = V[i]
for i = 1 to p                                  Setter hele tabell til 0
    P[i] = 0
for i = 1 to m                                  histogram
    P[V[i]]+ = 1
c = 0
for i = 1 to p          finner hvor i histogramet halvparten av pixlene er telt med
    c+ = P[i]
    if c >= m/2
        return i
```

———————

Medianfilter kan benyttes til å redusere støy ved å fjerne spikes i stedet for å glatte dem slik maske- og frekvensfiltre gjør.