KTH Datavetenskap
och kommunikation

Spektrala transformer
Laboration: bildbehandling

Introduktion

Den här laborationen kommer du att få experimentera med enkel linjär bildbehanding i Python. Du kommer att få bygga upp ett filter från grunden och sedan testa olika filter med avseende på prestanda och resultat.

Filter i 2D kan användas för många ändamål - lågpassfilter kan användas för att göra bilder oskarpa, reducera högfrekvent brus eller när man vill skala om en bild (omsampling). Olika typer av högpassfilter kan användas för att detektera kanter i bilden eller förhöja skärpan, mm.

Linjär bildbehandling och faltning i 2D

Inom bildbehandling är faltning (eng. *convolution*) en av de absolut vanligaste operationerna. En faltning kan ses som en summa av förskjutna versioner av den ena signalen/bilden där varje term är multiplicerad med en koeffcient - dessa koefficienter utgör *impulssvaret*, eller *filterkärnan* som det ofta benämns i bildsammanhang. Notera att vid faltning i 2D är oftast impulssvaret/kärnan centrerat kring origo, emedan endimensionella filter för ljudsignaler ofta har sin *början* i origo.

Den generella formen för faltning i två dimensioner ges av uttrycket

$$Y(u,v) = X(u,v) * H(u,v) = \sum_{s=-\infty}^{\infty} \sum_{t=-\infty}^{\infty} H(s,t)X(u-s, v-t) \tag{1}$$

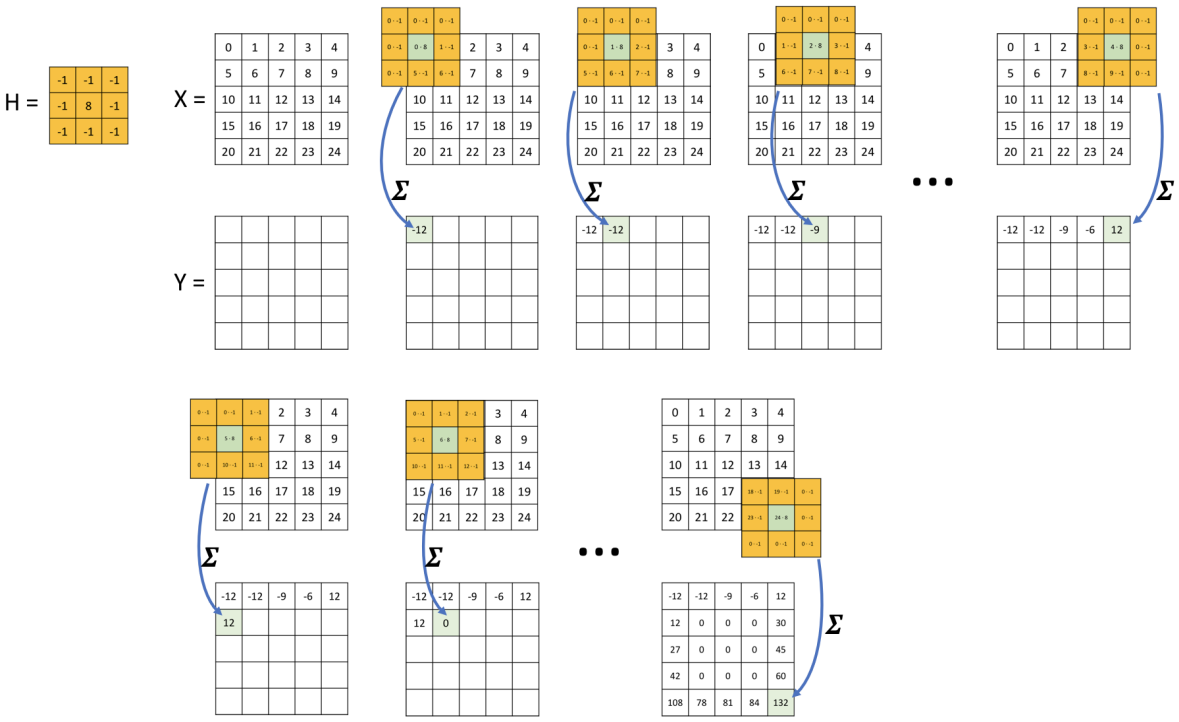
Figur 1 visar hur faltning i 2D går till i praktiken genom att låta kärnan H rulla över bilden X.

Utförande

Du ska utföra ett antal uppgifter med hjälp av Matlab eller Python, och svara på de frågor som ställs i peket. Laborationen utföres självständigt, antingen enskilt eller i grupp om två. Börja med att hämta `lab-improc.zip` från kurshemsidan.

I pythonkoden i peket används genomgående konventionen att skriva all kod med `skrivmaskinsstil`. Den kod som efterfrågas (och ska redovisas) ska ofta vara i form av pythonfunktioner. Ofta beskrivs även hur du kan testa att funktionen gör vad den ska. Du gör klokt i att skriva dessa tester i py-filer (behöver *ej* vara funktioner) för att göra själva testandet rationellt och konsekvent.

Var noga med att dokumentera vad du gör och spara kod, plottar, bilder, och vad det kan vara till redovisningstillfället, då du ska redogöra för hur du kommit fram till dina resultat.



Figur 1. Faltning i två dimensioner, steg för steg. Kärnan H faltas med bilden X vilket resulterar i Y. Elementen i Y beräknas ett i taget: kärnan centreras över det första elementet i X och elementen i H och X multipliceras och summeras till det första elementet i Y. Proceduren upprepas tills alla element i Y har beräknats.

Python-tips

För att läsa in en bild kan man använda funktionen `scipy.misc.imread()`. Den läser de flesta format och returnerar en matris av pixelvärden mellan 0 och 255. Färgbilder returneras som en $M \times N \times 3$-matris. Dessa kan konverteras till en vanlig 2D-gråskalematris genom att ta medelvärdet av de tre färgplanen.

`np.mean(scipy.misc.imread('minblid.gif'), axis=2)` tar hand om detta.

För att visa gråskalebilden kan man använda matplotlibs `plt.imshow(I, cmap='gray')`

```
import matplotlib.pyplot as plt
import scipy.misc
import numpy as np
img = np.mean(scipy.misc.imread('minblid.gif'), axis=2)
plt.imshow(img, cmap='gray')
plt.show()
```

Uppgift 1: läs in en bild och gör den lite oskarp

Välj ut en bildfil som du vill jobba med. Gärna en som har gott om skarpa kanter. Läs in blden till matrisen I.

Implementera faltning (med hjälp av for-loopar) enligt principen i figur 1 (eller direkt med ekv. 1) för att falta I med en $3x3$ rullande-medelvärdeskärna:

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

Plotta originalbild och filtrerad bild sida vid sida
(använd `plt.subplots()` och `plt.imshow(I, cmap='gray')`)

Ett enklare sätt att falta med python är att använda `scipy.signal.convolve` och `scipy.signal.convolve2d`. Läs på i scipys dokumentation om dessa funktioner, och jämför resultatet av ditt faltningsuttryck med de inbyggda funktionerna. Vad skiljer?

Att redovisa: python-uttryck

Uppgift 2: mera oskärpa med Gauss-filter!

Rullande medelvärde är enkelt att implementera och applicera, men inte alltid optimalt. Ett problem är t.ex. att filtret inte påverkar bilden lika i alla riktningar (varför inte?). Ofta vill man också ge större inflytande åt pixlarna i mitten av kärnan än de längs kanterna.

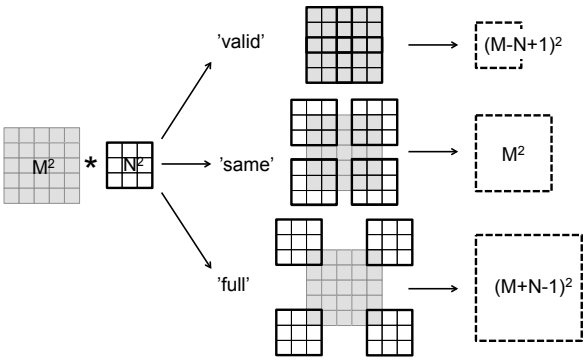
Ett filter som uppfyller dessa krav och ofta används för oskärpa/lågpass är gauss-filtret där kärnan ges av den klassiska *gaussklockan* i två dimensioner:

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

Där k är en konstant skalfaktor, x och y motsvarar horisontella och vertikala dimensionerna, samt σ bestämmer gausklockans bredd. Förutom att detta filter behandlar bilden lika i alla

Angående gränser för faltning

I exemplet i figur 1 har X och Y samma storlek, vilket ofta är praktiskt. Det finns dock andra sätt att hantera gränserna för faltningen. I `convolve2d` kan man ange argumentet `shape` som kan vara antingen `valid`, `same` eller `full`. *Valid* innebär att man aldrig låter kärnan gå utanför bildens kant. Detta resulterar i en utbild som är mindre än inbilden. *Same* motsvarar exemplet i figur 1 där utbilden får samma storlek som inbilden. *Full* innebär att kärnan går över alla positioner där det finns ett överlapp mellan inbild och kärna. De tre varianterna illustreras i figuren nedan.



riktningar, så har det den stora fördelen att vara linjärt separerbart i x- och y-led. Det innebär att faltningen kan delas upp i två oberoende steg.

För att inte påverka bildens absoluta ljusstyrka nivå är det även viktigt att filterkärnans element summerar till ett (precis som moving-average-filtret i uppgift 1). Enklaste sättet att åstadkomma detta är att dela alla filterkärnans element med totalsumman.

Välj $\sigma = \frac{N}{6}$. (Ett praktiskt problem är ju att gauss-funktionen aldrig riktigt blir noll, dvs man måste trunkera impulssvaret någonstans. I praktiken brukar man i bildbehandlingstillämpningar anse att funktionen är ”tillräckligt liten” vid avståndet 3σ från mitten på kärnan, vilket ger $\sigma = \frac{N}{6}$.)

Läs in bilden `lynn-eyes-halftone.png` som 2d-matrisen I för denna uppgift

a) Skriv en funktion för gaussisk oskärpa enligt funktionsprototypen given i `gaussblur.py`. Funktionen ska alltså skapa ett gauss-filter av önskad storlek (N) - centrerat kring $\frac{N}{2}$och sedan filtrera bilden I med detta. Det är ok att använda t.ex. `scipy.signal.convolve` eller `scipy.signal.convolve2d` inifrån funktionen.

b) Visa att gaussfiltret är linjärt separerbart! Läs i S.W.Smith, kap. 24 om separerbara filterkärnor (Convolution by separability) om du känner dig osäker. Skriv sedan en funktion för gaussisk oskärpa enligt funktionsprototypen i `gaussblur.py` där funktionen utnyttjar separerbarheten och då gör två faltingar med en endimensionell kärna (i x- reps. y-led).

c) Testa filtret på bilden I med olika storlek på kärnan, och jämför resultaten. Jämför även tidsåtgången mellan att falta i 2d samt att använda separerbarheten. Detta kan göras med:

```
import time
t = time.time() # Ger tiden just nu
faltning_i_2d()
tid_det_tog = time.time() - t # Ger tiden det tog att köra faltning_i_2d()
```

Kantdetektion och högpass

En vanlig tillämpning av bildfilter inom bl.a. datorseende är att hitta kanter i bilden. Det kan man göra genom att beräkna bildens *gradient* i x- resp. y-led. Den s.k. $Sobel - operatorn$ gör detta:

$$H_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_Y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Om man filtrerar en bild med dessa två kärnor får man alltså gradientens $x-$ repsektive y-komponenter, vi kan kalla dem G_X och G_Y. Dessa kan kombineras till för att få gradientens belopp:

$$g_{i,j} = \sqrt{g_{x,i,j}^2 + g_{y,i,j}^2}$$

där $g_{x,i,j}$ motsvarar pixeln på rad i, kolumn j av G_X.

Uppgift 3: sobeloperatorn

Beräkna gradienterna G_X, G_Y samt beloppet G enligt ovan för bilden `terracotta-wall.jpg`, och plotta sida vid sida. Studera och kommentera.

att redovisa: kod, bilder och kommentarer.

Uppgift 4: högpassfilter

Ett sätt att se på ett högpassfilter är som motsatsen till ett lågpassfilter. Dvs det släpper igenom allt det lågpassfiltret stoppar och tvärt om. Detta faktum kan vi utnyttja för att ”bygga om” ett lågpassfilter till ett högpass. Om vi tar skillnaden mellan en bild och den lågpassfiltrerade versionen av samma bild så borde det som blir kvar vara ”högpass-delen”. För att det ska fungera krävs att lågpasskärnans alla element summerar exakt till ett. Skriv några python-rader som högpassfiltrerar en bild på detta sätt baserat på *gauss*-oskärpefiltret du gjorde i uppgift 2!

Notera: Subtraktionen mellan originalbild kan även implementeras direkt i filterkärnan. Då kommer filterkärnan summeras till noll. Alltså: för lågpass summerar filterkärnan till *ett* och för högpass till *noll*. Hur kan detta tolkas/förklaras i *frekvensdomänen*?

att redovisa: kod, bilder och kommentarer.

Uppgift 5 (frivillig): Filtrering i frekvensdomänen

Nu ska du pröva att göra samma sak i frekvensdomänen. Kom ihåg faltningsteoremet: en faltning i spatial-domänen motsvarar en multiplikation i frekvensdomänen. Det handlar alltså om att transformera bilden till frekvensdomänen med hjälp av FFT (Fast fourier transform) - för bilder kan man använda numpy-kommandot `np.fft.fft2()` - multiplicera med filtrets frekvensdomänsrepresentation och sedan transformera tillbaka.

Det finns ett användbart numpykommando som heter `np.fft.fftshift()`. Det möblerar om lite i resultatet från FFT:n så att frekvensen noll hamnar i mitten och höga frekvenser utåt kanterna, istället för noll vid kanterna och höga frekvenser i mitten. Det blir mer intuitivt så. Börja med att transformera en bild med `np.fft.fftshift(np.fft.fft2(I))` och studera resultatet med `plt.imshow`. Det är en komplex matris så du behöver ta beloppet eller realdelen för att kunna visa den som en bild. Använd därefter `np.log` på resultatet.

För att få filterkärnans frekvensrepresentation kan man förstås också använda FFT, men här ska vi istället direkt specificera hur filtret ska se ut i frekvensdomänen, och tillämpa maskning, dvs sätta vissa frekvensområden till noll. Bestäm en radie (ett lämpligt utgånsvärde kan vara t.ex. 20) - det är filtrets brytfrekvens. Frekvensskalan utgår från bildens mitt, så ett lågpassfilter fås genom att sätta alla punkter som ligger utanför radien till noll. För ett högpassfilter gäller det omvända.

Resultatet av maskningen transformeras sedan tillbaka till spatialdomänen med `np.fft.ifft2(np.fft.fftshift())`. Resultatet kommer vara en komplex matris, så du behöver ta realdelen för att kunna visa den som bild.

Plotta originalbild och filtrerad bild sida vid sida. Testa både högpass och lågpass för olika brytfrekvenser.