3. obligatoriske innlevering, høsten 2014

{Jonathan Feinberg, Joakim Sundnes}
{jonathf,sundnes}@simula.no

October 28, 2014

Innleveringskrav

Innlevering skal skje ved opplasting til github, som for den første obligatoriske oppgaven. Generelle krav til innleveringen er oppgitt på emnesidene. I tillegg kommer eventuelle krav til hver enkelt oppgave, som er spesifisert i oppgaveteksten.

Merk at denne obligatoriske oppgaven er mindre omfattende enn Oblig 2, men deler av oppgaven er relativt krevende. Det anbefales sterkt å starte arbeidet så raskt som mulig. Ved å starte tidlig øker man også muligheten for å få hjelp av gruppelærere underveis. Riktig strategi er derfor; start tidlig, les hele oppgaven, få oversikt over vanskelige spørsmål, og spør om disse enten på gruppetime eller i mail til forelesere.

Frist for innlevering: Torsdag 13. november kl 16.00

Oversikt

disasterbefore.jpg

disasterafter.jpg

Figure 1: Et bilde før og etter støy-reduksjon

I denne oppgaven skal vil lage et program som fjerner støy fra et bilde. Et eksempel kan ses i Figur 1. For å få til dette bruker vi en støy-reduksjons-algoritme. Algoritmen er som følger:
Hvert punkt i det nye bildet blir laget som et vektet gjennomsnitt av alle nabo-punktene i det gamle bildet. Mer spesifikt:

```
data_new[i][j] = data[i][j] +kappa*(data[i-1][j]
                +data[i][j-1] -4*data[i][j] +data[i][j+1]
                +data[i+1][j]) ;
```

Alle punktene langs kantene skal kopieres over urørt fra gammelt til nytt bilde. Graden av filtreringen bestemmes av parameteren `kappa` som velges i intervallet $[0, 1]$. Prosessen for å laget et nytt bildet skal deretter bli repetert flere ganger. Antallet repetisjoner kan blir bestemt av parameteren `iter`.

Et program har allerede blitt skrevet i C: `denoise.c` (Et medfølgende bibliotek kan bli funnet i mappen `jpeg-simple`, men den er det ikke nødvendig å røre). Les filen `SETUP` for å bruke denne implementasjonen selv. Kjernen av algoritmen kan bli funnet i funksjonen `iso_diffusion_denoising`.

I denne obligen skal dere lage egne implementeringer av denoise i Python med diverse utvidelser. Forskjellige implementasjonene skal times ved hjelp av Timeit og Profile.

Til å importere og eksportere bildefiler kan dere bruke Python Imaging Library (PIL). Følgende kode-snutt illustrerer hvordan det brukes:

```
from PIL import Image
import numpy as np

# fra bildefil til Numpy:
data = np.array(Image.open("disaster_before.jpg"))
n, m = data.shape[:2]

# Fra Numpy til bildefil:
Image.fromarray(data).save("disaster_after.jpg")

# fra bildefil til liste:
im = Image.open("disaster_before.jpg")
data = list(im.getdata())
n, m = im.size()

# Fra list til bildefil:
im = Image.new("L", (n, m))
im.putdata(data)
im.save("disaster_after.jpg")
```

Her `"L"` står for Luninence og refererer til at bildet er svart/hvit med gråtoner. Hvis det er farger skal `"L"` byttes med `"RGB"`. Merk at i Numpy import blir bildedata representert som en todimensjonal array hvis bildet er sort/hvitt, og en tredimensjonal array hvis det er et fargebilde. hvis det er farger. I tilfellet med liste-import er dataen en 1-dimensjonal liste av enten verdier, eller tupler med fargene.

Oppgave 1: Implementasjon i Python, Numpy og Weave

Lag to implementasjoner av programmet:

- Løsning ved bruk av kun Python (altså ingen Numpy og Weave)
- Løsning som inkluderer både Numpy og Weave.

De to løsningene skal skrives i to forskjellige filer. Kopier gjerne kode fra `denoise.c` for bruk i Weave-implementasjonen.

Bruk `timeit` til å sammenlikne resultatene fra dine to implementasjoner mot C-implementasjonen. (Skru gjerne opp verdien på `iter` for å få mer målbare tider.) Hvis det er gjort riktig skal Numpy/Weave-implementasjonen være vesentlig raskere enn en ren Python-implementasjon. For å få oppgaven godkjent bør dine programmer også vise dette.

Merk at et sort/hvitt bildet lagres som en todimensjonal array, mens Weave-arrays er endimensjonale. Det er nødvendig å konvertere mellom ulike array-dimensjoner.

Oppgave 2: Bruk av profilering

Lag en profilering av dine to implementasjoner. I rapporten skal de tre linjene med høyest kumulativ tid skrives ut for begge programmene.

Kommenter forskjellene i hastighet, og gi en kort forklaring av hvorfor den er forskjellig. (Bruk gjerne profilering også som et verktøy til å løse oppgave 1, for eksempel hvis kjøretiden er overraskende høy.)

Oppgave 3: Utvidelse til farger

Programmet så langt er laget for svart hvitt bilder. I denne oppgaven skal dere utvide programmet deres til bruk av farger. Farger på datamaskinen er splittet i tre farge-komponenter: rød, grønn og blå (RGB). Hver verdi R,G, og B er et tall mellom 0 og 255, som i C-koden er lagret som en `int8`. Algoritmen for denoise i Oppgave 1 er basert på lys og mørke, som ikke helt fungerer med farge. For å kunne bruke algoritmen må man derfor konvertere RGB til fargeformatet HSI, eller «hue», «saturation» og «intensity». Utvidelsen dere skal lage skal kunne fungere på hver av kanalene i HSI.

Som hjelp med konverteringen, her er formlene for konvertering fra RGB til HSI:

$$I = \frac{R+G+B}{3}$$

$$S = \begin{cases} 1 - \frac{\min(R,G,B)}{I} & \text{if } I > 0 \\ 0 & \text{if } I = 0 \end{cases}$$

$$H = \begin{cases} \cos^{-1}\left(\frac{R-G/2-B/2}{\sqrt{R^2+G^2+B^2-RG-RB-GB}}\right) & \text{if } G \geq B \\ 360 - \cos^{-1}\left(\frac{R-G/2-B/2}{\sqrt{R^2+G^2+B^2-RG-RB-GB}}\right) & \text{if } G < B \end{cases}$$

Her $\cos^{-1}$ er den inverse av cosinus-funksjonen, målt i grader. I C kan man bruke denne funksjonen funksjonen:

```
acos(value)*180/3.14159256
```

Merk konverteringen til grader, siden returverdien fra funksjonen er radianer. Som det går fram av formlene er hver verdi H,S og I et tall mellom 0 og 360. Om det brukes `float` eller `int` for denne spiller ingen rolle. Funksjonen `acos` kan du finne i C-biblioteket `math.h`. Det samme gjelder kvadratrot-funksjonen `sqrt`. (Se forelesningsfoiler om Weave for hvordan C-biblioteker skal importeres.)

Tilsvarende etter at man har brukt denoising på et eller flere av båndene i HSI, kan man bruke følgende formel for å konvertere tilbake til RGB:

$$
\begin{array}{llll}
R = I + 2IS & G = I - IS & B = I - IS & \text{if } H = 0 \\
R = I + IS\frac{\cos(H)}{\cos(60-H)} & G = I + IS\left(1 - \frac{\cos(H)}{\cos(60-H)}\right) & B = I - IS & \text{if } H \in (0, 120) \\
R = I - IS & G = I + 2IS & B = I - IS & \text{if } H = 120 \\
R = I - IS & G = I + IS\frac{\cos(H-120)}{\cos(180-H)} & B = I + IS\left(1 - \frac{\cos(H-120)}{\cos(180-H)}\right) & \text{if } H \in (120, 240) \\
R = I - IS & G = I - IS & B = I + 2IS & \text{if } H = 240 \\
R = I + IS\left(1 - \frac{\cos(H-240)}{\cos(300-H)}\right) & G = I - IS & B = I + IS\frac{\cos(H-240)}{\cos(300-H)} & \text{if } H \in (240, 360)
\end{array}
$$

Igjen skal cos regnes ut i grader. Fra `math.h` can vi regne det ut som:

```
cos(value*3.14159256/180)
```

Merk at her konverteres argumentet fra grader til radianer før funksjonen kalles. Denne oppgaven behøver man kun gjøre i Numpy/Weave. Det forventes at utregningen blir utført på innsiden av Weave.

Oppgave 4: Lineær manipulering

Utvid funksjonaliteten slik at hvert bånd av R, G, B, H, S og I kan justeres opp eller ned. Justering opp og ned gjøres ved å legge til eller trekke fra et heltall. Det er viktig at endringene ikke bringer verdiene utenfor yttergrensene, dvs $[0, 255]$ for RGB, og $[0, 360]$ for HSI.

Oppgave 5: Frontend

Implementer et felles brukergrensesnitt for dine to løsninger samt C-løsningen ved hjelp av `argparse`-modulen. Følgende funksjoner forventes å være inkludert:

- Spesifiser én input- og én output-fil.
- Switch for å si at man skal utføre denoising.
- Spesifiser parameterene `iter`, `kappa` og `eps`, og gi dem default-verdier `10`, `0.1`, `2` respektivt.

- En switch for å bytte mellom de tre backendene
- Verbose-mode skriver hva programmet foretar seg.
- En switch for å skru på en Timit modulen.
- Muligheten til å inviduelt justere på de 6 båndene opp og ned.

Programmet skal kunne gjøre flere manipuleringer i samme kall. Dvs. at man skal for eksempel kunne både skru ned på intensiteten (I) samtidig som man øker grønnfargen (G) og kjører filtrering.

Hvis backend ikke støtter en fuksjonalitet (som farge-bilde med C-backend), skal programmet gi en passende feilmelding og avslutte.

Oppgave 6: Testing og dokumentasjon

Som alltid skal programmet inneholde god dokumentasjon, doc-tester hvor det passer og en implementasjon av en test-suite.

Følgende test er forventet:

- Generer output for `kappa` lik 0.1 og 0.2, `iter` lik 5, 10 og 20 for alle tre implementasjonene. Gi hver fil passende navn.
- For hver `kappa` og `iter` konstant, åpne alle bildene i de tre implementasjonene.
- Sjekk at verdiene i bildet er tilnærmet lik hverandre. Bruk en feiltolleranse `eps` for hvor stor feilen maksimalt for være for hver piksel i bildet.

I tillegg skal dere teste at oppgaven 4 for én farge, én av båndene i HSI, samt én kombinasjon hvor én farge og én HSI-kanal er testet samtidig.

Oppgave 7: Rapport

En rapport av hva du har gjort skal inn i en LaTeX-rapport. Bruk modulen du laget i oblig 2 til å skrive denne.