3. obligatoriske innlevering, høsten 2014

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

November 3, 2014

Innleveringskrav

Denne skal følge malen gitt på emnesidene (Legges ut 2. september). Innlevering skjer ved opplasting til github. Mer informasjon om klasseromsløsningen vi bruker på github vil bli gitt i god tid før innleveringsfristen.

Oversikt

disasterbefore.jpg

disasterafter.jpg

Figure 1: Et bilde før og etter støy-reduksjon

I denne oppgaven skal vil lage en 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 bilde blir laget som en vektet gjennomsnitt av alle nabo-punktene. 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. Graden av filtreringen bestemmes av parameteren `kappa` som ligger på 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 hjelp med å importere og eksportere bildefiler kan dere bruke av følgende kode-snutt:

```
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 så er dataen formatert som en multidimensjonal array med 2 akser hvis bildet er svart/hvitt og 3 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 `timit` til å sammenlikne resultatene fra dine to implementasjoner mot C-implementasjonen. (Skru gjerne opp verdien på `iter` for å få mer målbare tider.) For å få godkjent på denne oppgaven må implementasjonen i Numpy/Weave skal være merkbart raskere enn løsningen i «bare» Python.

Merk at koden der er rettet mot 2-dimensjonale arrays og at weave arrays er 1-dimensjonale.

Oppgave 2: Bruk av profilering

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

Bruk noen ord på å forklare hvordan de forskjellige programmene sammenlikner i hastighet og hvorfor. 1 Bruk gjerne profilering når du løser oppgave 1, men ikke klarer å få kjøretiden ned.

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). Algoritmen for denoise er bassert på lys og mørke, som ikke helt fungerer med farge, må man vekk fra RGB og over i 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:

$$
\begin{aligned}
I &= \tfrac{R+G+B}{3} \\
S &= \begin{cases} 1 - \frac{\min(R,G,B)}{I} & \text{if } I > 0 \\ 0 & \text{if } I = 0 \end{cases} \\
H &= \begin{cases} \cos^{-1}\left(\frac{R-G/2-B/2}{\sqrt{R^2+G^2+B^2-RG-RB-GB}}\right) & \text{if } G \geq B \\ 360 - \cos^{-1}\left(\frac{R-G/2-B/2}{\sqrt{R^2+G^2+B^2-RG-RB-GB}}\right) & \text{if } G < B \end{cases}
\end{aligned}
$$

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

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

Funksjonen kan du finne i C-biblioteket `math.h` Det samme gjelder kvadratrot-funksjonen `sqrt`.

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)
```

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 funksjonalitet slik at vært bånd av R, G, B, H, S og I kan justeres opp eller med.

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. (`eps` brukes hvis man ønsker å implementere testing i frontend.)
- 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.