Obligatorisk oppgave 2
INF2310 Våren 2018

Dette oppgavesettet er på 7 sider, og består av 2 bildebehandlingsoppgaver.

Besvarelsen av denne og neste obligatoriske oppgave må være godkjent for at du skal få anledning til å gå opp til endelig skriftlig eksamen i kurset. Besvarelsene kan utarbeides i smågrupper på opptil to studenter, men det er ikke noe i veien for å arbeide alene. Studenter i samme smågruppe kan levere identisk besvarelse, men samarbeidet må framgå av navnene på forsiden av besvarelsen.

Av forsiden skal det fremgå hvem som har utarbeidet besvarelsen.

Det forventes at arbeidet er et resultat av egen innsats. Å utgi andres arbeid for sitt eget er uetisk og kan medføre sterke reaksjoner fra IFIs side. Se
http://www.uio.no/studier/admin/obligatoriske-aktiviteter/mn-ifi-oblig.html
Den skriftlige rapporten leveres primært som en PDF-fil som inneholder hele besvarelsen, med figurer og bilder. Kode skal leveres i tillegg til PDF-en.
Besvarelsen skal leveres via https://devilry.ifi.uio.no

Legg merke til følgende:

- Alle filene må lastes opp hver gang man skal levere.
- PDF-en skal ha følgende navn: inf2310-oblig1-brukernavn.pdf, der brukernavn byttes ut med ditt eget sådan.
- Oppgaven skal kunne kjøres fra Matlab- eller Python- skript med navn: oppgave1.m eller oppgave1.py (evt. oppgave1a.m osv.).
- Spørsmål angående innlevering: krisbhei@student.matnat.uio.no

.
Bildene det refereres til vil være å finne under:
http://www.uio.no/studier/emner/matnat/ifi/INF2310/v18/undervisningsmateriale/bilder/

Oppgaven utleveres fredag 27. april 2018
Innleveringsfrist er fredag 11. mai 2018

Lykke til!

Oppgave 1. Implementasjon av konvolusjonsfiltre i frekvensdomenet

I implementeringsdelen av denne oppgaven kan du benytte deg av eksisterende funksjoner som er implementert i det språket du velger. Se *Appendiks A* for hvordan noen funksjoner brukes i hhv. Matlab og Python. Når funksjonene blir referert under benyttes navngivingen fraMatlab.

Oppgave 1.1

Last inn bildet cow.png (vist i fig. 1). Gjør en 15x15 middelverdifiltrering, implementert både som en romlig konvolusjon (benytt conv2-funksjonen) og ved å gå til frekvensdomenet og gjøre filtreringen der (benytt fft2-funksjonen).

Figur 1: Bilde cow.png

Oppgave 1.2

En mulighet her er å benytte same som tredjeparameter i conv2-funksjonskallet, samt implisitt å nullutvide filterkjernen til samme størrelse som bildet ved å benytte antall rader og kolonner i bildet som andre og tredje parameter i fft2-kallet. Ved å benytte en slik, ovennevnt variant får vi to tydelige forskjeller på resultatbildene:

1. en forflytning (translasjon) av det frekvensdomenefiltrerte bildet,
2. samt forskjeller i kantene (randen) i de to filtrerte bildene.

Forklar hvorfor disse to fenomenene oppstår.

Oppgave 1.3

Vi skal nå utforske hva som er raskest; å implementere middelverdifilteret som en konvolusjon (ikke tenk på å separere filteret eller bruken av overlapp, kun ren 2D-konvolusjon) eller ved å gå til frekvensdomenet, filtrere, for så å gå tilbake.

Lag et program som filtrerer bildet med et sett av middelverdifiltre med forskjellig størrelse, både i det romlige domenet (konvolusjon) og ved å gå til frekvensdomenet.

Lag én kurve over kjøretiden ved å konvolvere direkte, og én kurve over kjøretiden ved å bruke frekvensdomenet, begge i samme figur. Diskuter de to kurvene, samt forklar for hvilke filterstørrelser det vil lønne seg å utføre filtreringen i frekvensdomenet (med akkurat denne spesifikke implementasjonen på akkurat din datamaskin).

Oppgave 2. Ikke-tapsfri JPEG-kompresjon

I denne oppgaven skal du implementere de viktigste delene av ikke-tapsfri JPEG-kompresjon. Du skal lage en funksjon (eller et program) som har to parametre:

1. et filnavn som spesifiserer plasseringen til bildefilen som skal benyttes, og
2. ett tall, *q*, som indirekte vil bestemme kompresjonsraten.

Funksjonen skal beregne omtrentlig hvor stor lagringsplass det angitte bildet vil bruke etter JPEG-komprimering, og finne hvilken kompresjonsrate dette tilsvarer. Vi vil anta at inputbildet er et gråtonebilde med heltallsintensiteter i intervallet [0,255] og har både en bredde og en høyde som er multipler av 8. Bruk gjerne bildet uio.png (vist i fig. 2) for å teste implementasjonen din underveis.

Fremgangsmåten du skal følge for å lage programmet følger i stegene under.

Figur 2: Bilde uio.png

Steg 1

Last inn bildet som den første parameteren spesifiserer.

Steg 2

Trekk 128 fra alle pikselintensiteter (dette gjør at den forventede gjennomsnittlige intensitets-verdien er omtrent 0). Pass på at bildet ditt er på et format med fortegn før du trekker fra 128 (mange innlesningsfunksjoner leser bilder til uint8, som kun tar verdier i [0,255]).

Steg 3

Begynn i øverste, venstre hjørnet og del opp bildet i 8 x 8-blokker. Transformér hver blokk med den todimensjonale diskrete cosinus-transformen (2D DCT).

Hint 1: Siden hver blokk har størrelse 8*x*8, kan vi forenkle den generelle formelen for 2D DCT i forelesningsnotatene til:

$$F(u,v) = \frac{1}{4} c(u)\, c(v) \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right), \qquad (1)$$

der

$$c(a) = \begin{cases} \frac{1}{\sqrt{2}} & hvis\ a = 0 \\ 1 & ellers \end{cases} \qquad (2)$$

Hint 2: De transformerte 8*x*8-blokkene kan lagres som en matrise der hver 8*x*8 transformblokk er plassert på samme sted som den 8*x*8-bildeblokken den er beregnet fra.

Steg 4

Rekonstruér det opprinnelige bildet ved å invertere transformen du utførte i forrige steg og addere 128 til alle pikselintensiteter. Programmatisk verifiser at det rekonstruerte bildet er identisk som originalen.

Hint: Den forenklede formelen for den inverse todimensjonale diskrete cosinus-transformen (2D IDCT) er:

$$f(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} c(u)\,c(v)\,F(u,v)\,\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right), \tag{3}$$

der funksjonen c er definert som over i ligning (2). Pga. upresis flyttallsaritmetikk vil det generelt være behov for å avrunde de resulterende verdiene til nærmeste heltall.

Steg 5

La Q være følgende kvantifiseringsmatrise:

$$Q = \begin{vmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{vmatrix}$$

Punktvis divider hver av de transformerte 8*x*8-blokkene fra steg 3 med qQ , dvs. produktet av tallparameteren q og kvantifiseringsmatrisen over. Avrund de resulterende verdiene til nærmeste heltall.

Steg 6

Dersom vi skulle fulgt (den sekvensielle modusen i) JPEG-algoritmen videre så skulle vi nå separert det øverste, venstre elementet av hver transformert og kvantifisert 8x8-blokk, kalt et kvantifisert DC-element, fra de resterende 63 elementene i hver blokk. DC-elementene skulle blitt differansetransformert (på tvers av blokkene), og de 63 elementene skulle blokk for blokk blitt sikk-sakk-skannet og deretter (0-basert) løpelengdetransformert. Til slutt skulle (antall biter i) differansene og løpelengdeparene blitt entropikodet. Dette skal du ikke gjøre her!

I stedet for å følge denne prosedyren vil vi her anta at den grovt sett vil resultere i et dataforbruk som tilsvarer entropien til datasettet som består av alle elementene i alle de transformerte og kvantifiserte 8*x*8-blokkene. Beregn denne entropien og bruk den til å estimere hvor stor lagringsplass det spesifiserte bildet vil bruke etter JPEG-komprimeringen og hvilken kompresjonsrate dette tilsvarer.

Steg 7

Bruk de transformerte og kvantifiserte 8*x*8-blokkene fra steg 5 til å rekonstruere en tilnærming av det opprinnelige bildet. Skriv dette bildet til fil.

Du skal teste funksjonen din ved å anvende den på bildet uio.png når du benytter hver av følgende verdier av tallparameteren

$$q = \{0.1, 0.5, 2, 8, 32\}.$$

Studer rekonstruksjonene, og bemerk når og hvor i bildet du først oppdager rekonstruksjonsfeilene blokk-artefakter, glatting og ringing. Vurder også for hvilke(n) verdi(er) av tallparameteren *q* som du synes rekonstruksjonen er god nok for fremvisning av hele bildet på en vanlig dataskjerm.

NB!
Dere kan benytte ferdige Matlab/Python-funksjoner til å lese/skrive fra/til fil. Blokkoppdelingen, transformasjonene, og entropiberegningen *må* dere implementere selv.

Hva skal leveres

Oppgave 1

1. Besvarelse (inkludert figur med kjøretidskurver).
2. Programkode fra deloppgave 1.1 og 1.3.

Oppgave 2

1. De rekonstruerte bildene av uio.png med de ni forskjellige verdiene av tallparameteren *q*.
2. Drøfting av komprimeringen av uio.png, inkludert
   a) dine oppdagelser om rekonstruksjonsfeilene,
   b) din vurdering av når rekonstruksjonsfeilen er "god nok", og
   c) forklaring av hvorfor den estimerte kompresjonsraten øker med verdien av tallparameteren *q*.
3. Kommentert programkode.

Lykke til!

Appendiks A

Eksempler i Matlab og Python som utfører samme oppgave med noen nyttige funksjoner. Altså vil variablene a, b, c, d, og e inneholder de samme verdiene i begge implementasjonene.

```
% Matlab

tic; % Starter klokken

a = [1, 2, 3; 4, 5, 6; 7, 8, 9;];
b = [1, 1, 1; 2, 2, 2; 3, 4, 5];

% 2D konvolusjon
% Dokumentasjon: https://se.mathworks.com/help/matlab/ref/conv2.
    html

c = conv2(a, b);
%c =
%     1     3     6     5     3
%     6    15    27    21    12
%    18    43    76    61    36
%    26    61   106    83    48
%    21    52    94    76    45

d = conv2(a, b, 'same');
%d =
%    15    27    21
%    43    76    61
%    61   106    83

% 2D Fourier-transform
% Dokumentasjon: https://se.mathworks.com/help/matlab/ref/fft2.html

e = fft2(a);
%d =
%   45.00000 +  0.000001    -4.50000 +  2.598081    -4.50000 -
    2.598081
%  -13.50000 +  7.794231     0.00000 +  0.000001     0.00000 +
    0.000001
%  -13.50000 -  7.794231     0.00000 -  0.000001     0.00000 -
    0.000001

tid = toc; % tid er nå antall sekunder siden tic
```

```python
# Python

import time
from scipy import signal
import numpy as np

start_tid = time.time()

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
b = np.array([[1, 1, 1], [2, 2, 2], [3, 4, 5]])

# 2D konvolusjon
# Dokumentasjon: https://docs.scipy.org/doc/scipy/reference/
    generated/scipy.signal.convolve2d.html

c = signal.convolve2d(a, b)
d = signal.convolve2d(a, b, 'same')

# 2D Fourier-transform
# Dokumentasjon: https://docs.scipy.org/doc/numpy/reference/
    generated/numpy.fft.fft2.html
e = np.fft.fft2(a)

stopp_tid = start_tid - time.time()
```