KTH Datavetenskap
och kommunikation

Spektrala transformer
Laboration: JPEG-kodning

1 Introduktion

I denna laboration kommer du att få experimentera med transfom-baserad bildkompression enligt JPEG-metoden. Du kommer att implementera en förenklad form av JPEG-kodare/avkodare för gråskalebilder, och pröva olika strategier för att ta bort redundant information i bilden, uppskatta kompressionsgrad och ställa denna i relation till upplevd bildkvalitet.

2 JPEG-komprimering

JPEG-metoden för bildkomprimering är en transformbaserad kompressionsmetod. Det innebär att bilden transformeras med en spektral transform för att informationen ska kunna packas effektivare. Det är en approximativ metod, där information går förlorad i processen (s.k. *lossy compression*). Transformen som används i JPEG heter DCT, Discrete Cosine Transform[1], och är en nära släkting till DFT, den diskreta fouriertransformen. DCT'n skiljer sig från DFT'n genom att endast använda cosinus-funktioner som basvektorer. DCT'n är en helt reell transform, dvs basfunktionerna är reella, och därmed också invektor och utvektor.

Det första som görs när en bild ska JPEG-kodas är att dela upp bilden i block av 8x8 pixlars storlek. Detta görs huvudsakligen av effektivitetsskäl. Att göra DCT på hela bilden skulle ta alldeles för lång tid, även om större block principiellt leder till högre möjlig kompressionsgrad.

Därefter transformeras varje block med DCT i två dimensioner. Det transformerade blocket, en matris med 8x8 koefficinter, beskriver blockets frekvensinnehåll. Första koefficienten (1,1) motsvarar blockets genomsnittliga intensitet. Det visar sig att den mesta informationen finns i övre vänstra hörnet, som motsvarar låga spatiala frekvenser. Ju längre mot högra nedre hörnet man kommer, desto lägre belopp har koefficienterna och många är i praktiken noll. Därför kan dessa koefficienter elimineras eller kodas med färre bitar (kvantiseras) utan att bilden försämras lika mycket.

Efter att koefficienterna har kvantiserats kommer en stor del av dessa att vara noll, och kan därför packas ihop mycket effektivt med "traditionella" datakompressionsmetoder, bl.a. Huffmankodning. (Detta steg hoppar vi över i denna laboration, vi nöjer oss med att räkna andelen koefficenter skiljda från noll och tar detta som ett grovt mått på kompressionsgraden.)

Avkodningen tillbaks till en bild sker därefter på omvänt sätt: koefficientmatriserna transformeras tillbaks till 8x8-bilder, vilka tillsammans bildar den avkodade bilden.

För mer information om JPEG-komprimering, läs kapitel 27 i gratisboken av Stephen W. Smith (se länk från kurshemsidan).

[1]Detta gäller den ursprungliga JPEG-metoden. I en uppdaterad standard kallad JPEG2000 används istället en s.k. Wavelet-transform

3 Utförande

Du ska utföra ett antal uppgifter med hjälp av Python/Numpy, och svara på de frågor som ställs i peket. Laborationen utföres självständigt, antingen enskilt eller i grupp om två. Börja med att hämta `lab-jpeg.zip` från kurshemsidan. Denna fil innehåller några hjälpfunktioner och andra filer som du kommer att behöva. Packa upp arkivet i din hemkatalog och använd detta som din projektmapp. Under `prototypes/` finner du även platshållare för de funktioner du själv ska skriva. Det är filer som endast innehåller funktionshuvuden och kanske någon ledtråd, där du själv ska fylla på med kod.

I matlabkoden i peket betecknas alltid arrayer och matriser med versaler, medan skalärer betecknas med gemener. Vidare används genomgående konventionen att skriva all kod med `skrivmaskinsstil`. Den kod som efterfrågas (och ska redovisas) ska oftast vara i form av python-funktioner. 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, ljudfiler och vad det kan vara till redovisningstillfället, då du ska redogöra för hur du kommit fram till dina resultat.

4 DCT

DCT är kärnan i JPEG-algoritmen. DCT-transformen kan beräknas effektivt med en metod som liknar FFT, men för korta sekvenslängder, t.ex. 8 som är aktuellt här, är ändå det snabbaste sättet att beräkna den direkt genom en matrismultiplikation med en basvektormatris T. Denna $M \times M$-matris, där kolumnerna motsvarar DCT'ns basvektorer, beskrivs av

$$t_{pq} = \begin{cases} \frac{1}{\sqrt{M}} & 0 \leq p \leq M-1, q = 0 \\ \sqrt{\frac{2}{M}} \cos \frac{\pi(2p+1)q}{2M} & 0 \leq p \leq M-1, 1 \leq q \leq M-1 \end{cases}$$

där t_{pq} beteknar elementet på rad p, kolumn q.

Givet en radvektor X av längd M ges DCT'n av matrisprodukten $Y = XT$. Om X är en matris så transformeras varje rad i matrisen.

För att transformera en bild I i två dimensioner, så ska först raderna transformeras, sedan kolumnerna (eller vice-versa). Givet att I är en MxM-matris (dvs samma storlek som basvektormatrisen) så motsvarar detta $C = (((IT)^T)T)^T$ vilket kan förenklas till följande kompakta uttryck för 2D DCT'n av matrisen I:

$$C = T^T I T$$

där T^T betecknar matrisens T transponat.

Den inversa transformen, dvs att återgå till X från DCT-koefficienter, innebär helt enkelt att multiplicera med basvektormatrisens T invers. Nu är dock T så beskaffad att inversen T^{-1} är lika med transponatet T^T [2], vilket innebär att inversen av transformen $Y = XT$ ges av $X = YT^T$. I det tvådimensionella fallet, där det gäller att återfå "bilden" I från koefficientmatrisen C ges av

$$I = TCT^T$$

Uppgift 1: Beräkna basvektormatrisen

Skriv funktionen `dct_basis()` som beräknar DCT'ns basvektormatris av ordningen m, enligt prototypen i templates.py

[2] Detta förhållande gäller för alla ortogonala matriser, dvs matriser där kolumnernas ibördes skalärprodukter alla är noll.

Testa din DCT-transform!

För att testa att basvektormatrisen är korrekt kan du transformera några speciella 8x8-matriser:

- en matris med bara ettor - ska resultera i en koefficientmatris där endast elementet i övre vänstra hörnet är skiljt från noll (detta element motsvarar konstant-nivån)
- en matris där varannan kolumn är 1 och varannan kolumn -1 - ska resultera i en koefficientmatris där störta elementet finns i övre högra hörnet.
- en enhetsmatris `eye(8)` - ska resultera i en enhetsmatris!

Du ska även testa invers-transformen. Gör detta genom att skriva en loop som går igenom alla 64 koefficienter, och sätter en i taget till ett (övriga noll) och invers-transformerar detta tillbaka till bild-domänen. Varje pixelblock plottas med hjälp av imagesc. Du ska använda subplot för att kunna se alla på en gång - resultatet ska alltså bli en bild bestående av åtta rader och åtta kolumner av 8x8-pixelblock, där det första blocket är inverstransformen av en matris av nollor och en etta i övre vänstra hörnet, det andra motsvarar en etta i andra positionen osv. Detta kan ses som en visualisering av DCT-transfrmens basvektorer i två dimensioner.
Att redovisa: .py fil(er)

Uppgift 2: JPEG-kodaren

Skriv nu funktionen `jpeg_encode` som tar in en bildmatris I, traverserar denna i block om 8x8 pixlar, beräknar 2D DCT på varje block, och sparar koefficienterna på motsvarande positioner i koefficientmatrisen C. Dvs, funktionen ska returnera en matris C av samma storlek som I, inneållande 8x8-block av DCT-koefficienter.

För att kodningen ska fungera optimalt ska pixelvärdena centreras kring nollnivån. Du kan anta att I är en gråskalebild där varje pixel har ett värde mellan 0 och 255, vilket innebär att 128 ska subtraheras från varje pixel.

Det är tillåtet att endast räkna hela block, dvs i praktiken krympa bildens dimensioner till närmsta jämna multipel av 8. Om du vill, kan du få hjälp med blockuppdelningskoden ifrån funktions-templaten i templates.py.

Skriv sedan ett pythonscript för att testa din funktion: Läs in en liten bild t.ex. `uggla2.tif` (se tipsruta) och anropa kodar-funktionen. Plotta bilden I och koefficientmatrisen C sida vid sida (kolla dokumentationen eller exempel online för matplotlib, subplot osv för tips). Kofficientmatrisen kommer ha ljusa prickar i övre vänstra hörnet av varje block, och annars vara ganska mörk. I områden med skapra kanter bör du kunna se andra koefficienter aktiveras.
Att redovisa: python-script, funktion och plot

Python-tips
För att läsa in en bild i python kan man t.ex. använda paketet imageio och funktionen `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. Testa t.ex. numpy.mean.

Uppgift 3: JPEG-avkodaren

Skriv nu den kompletterande JPEG-avkodarfunktionen `jpeg_decode` som tar in koefficientmatrisen C och returnerar en bildmatris I, vilken altså fås fram ur den inversa DCT'n på 8x8-blocken i koefficientmatrisen. Glöm inte att lägga tillbaka de 128 som du drog från varje pixel i kodaren!

Testa funktionen genom att koda en bild som du sedan kodar av och visar sida vid sida som i förra uppgiften (återanvänd testprogrammet). De två bilderna ska vara identiska (här har vi ju ännu inte slängt bort någon information).
Att redovisa: python-kod och plot

5 Maskering och kvantisering

Som tidigare nämnts så bygger JPEG-kodning på att man kan göra sig av med stora mängder av koefficienterna, vilket inte känns helt orimligt när man tittar på koefficintmatrisen så som du gjorde i uppgift 2. Nedan beskrivs flera sätt att göra detta på, inklusive det sätt som används i JPEG-standarden.

Ett rättframt sätt att slänga information är att helt sonika bestämma vilka koefficienter som ska behållas. Vi har sagt att de viktigaste finns högt upp till vänster i sub-matriserna. Man kunde ju alltså tänka sig att multiplicera varje 8x8 koefficientmatris med en *mask* av typen

```
1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Med ett sådant grepp slänger vi alltså alla utom 10 av 64 koefficinter, vilket motsvarar en kompressionsgrad av 6.4:1.

Ett annat sätt effektivt sätt kan vara att sätta en tröskel för koefficientvärdenas belopp, och sätta alla koefficienter som faller under tröskeln till noll.

Tröskeln kan vara konstant över hela bilden, men effektivare är att låta den variera block-till-block, och istället ha ett fixt antal "toppkoefficienter" från varje block.

I JPEG-standarden har man valt en något mindre drastisk metod än att rått slänga bort värdena: istället *viktar* man koefficinterna genom att dividera med ett förbestämt nummer och runda av till närmasta heltal. De vikiga lågfrekventa koefficinterna delas med mindre värden, och de högfrekvnta delas med större värden. Detta leder till att de höga koefficienterna oftast blir noll vid avrundningen, och kan då packas ihop effektivt av efterföljande datakodningsalgoritmer.

Python-tips
För att slänga bort värden nära noll titta på funktionen trunkeringsfunktionen np.trunc().

De vikter man använder i standard-JPEG ges i matrisen nedan (matrisen återfinns även i filen weights.txt). Dessa värden är norannt utprovade genom perceptuella försök. Genom att applicera en skalfaktor på matrisen, kan man styra hur hårt bilden ska komprimeras.

```
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
```

Notera: eftersom man delar koefficienterna med dessa värden, måste samma matris användas för multiplikation vid avkodning av bilden!

Uppgift 4: Kvantisering

Nu är det dags att börja göra sig av med koefficienter. Här får du experimentera fritt, med inspiration från texten ovan. Testa gränser. Hur blir det med bara en koefficient per block? osv. Målet är alltså att få så många nollor som möjligt, och erhålla accptabel bildkvalitet. Skriv din kvantisering i form av en matlab-funktion som tar in koefficientmatrisen, och returnerar den i kvantiserad form. Du får ingen prototyp för denna funktion, eftersom den kanske ska ta ytterligare parametrar, t.ex. en kompressionsfaktor e.dyl. Gör även ett överslag på kompressionsgraden genom att räkna antalet koefficienter som inte är noll och relatera till totala antalet koefficienter.

Python-tips
För att kolla antal element som inte är 0 kolla upp numpy-funktionen count_nonzero()

Återanvänd testprogrammet från föregående uppgift för att plotta original och avkodad bild sida vid sida.
Att redovisa: python-kod för din "bästa" kvantisering, och en beskrivning av vad du provat (även "misslyckade" varianter), inkl uppskattad kompressionsgrad och visuellt resultat!

Uppgift 5 (Frivillig): Färg!

Utveckla din kodare så den kan hantera färgbilder. Enligt JPEG-standarden kodas färgbilder i termer av luminans-krominans. Dvs innan man kan applicera DCT-algoritmen måste RGB-pixelväden omvandlas till Y'UV. Det sker med följande formler:

$$Y' = 0.299R + 0.587G + 0.114B$$

$$U = -0.147R - 0.289G + 0.436B$$

$$V = 0.615R - 0.515G - 0.100B$$

och omvänt:

$$R = Y' + 1.13983V$$

$$G = Y' - 0.39465U - 0.5806V$$

$$B = Y' + 2.03211U$$

Varje komponent kodas separat. Luminansen (Y') är perceptuellt viktigare än krominansen (U, V) vilket betyder att man kan använda färre bitar/hårdare kvantisering för U och V. Gör så att du enkelt kan kontrollera hur mycket varje komponent ska komprimeras, och experimentera med detta.