Beeldcompressie

VWO Masterclass '08

21 oktober 2008

1 Voorbereiding

In dit practicum doen we hetzelfde als in het hoorcollege (Fourier-transformatie op geluid), maar dan voor plaatjes. Jullie werken in teams van twee en gebruiken MATLAB voor de experimenten. Om te beginnen heb je ook wat start-bestanden nodig. Start hiervoor een browser (firefox-icoon bovenin het beeld) en navigeer naar:
`http://www.math.uu.nl/people/yzelman/software/masterclass08.tar.gz`

Sla deze in een handige plek op. Open de map waarin je de `.tar.gz` hebt opgeslagen, klik erop mbv de rechter-muisknop en klik op `Extract here`.

Start nu MATLAB door linksboven op `Applications` te klikken, dan op `Math packages` en vervolgens op `MATLAB 2007a`. Bijna bovenaan, ergens in het horizontale midden, zie je de huidige map waarin MATLAB nu werkt. Wijzig dit naar de map waar je de bestanden hebt uitgepakt. Nu kunnen we beginnen met het echte werk.

2 Lena weergeven

We gebruiken als plaatje een zwart/wit versie van Lena. Deze staat in het bestand `lena.mat` en kan als volgt worden geopend door MATLAB. In het *command window* kun je commando's typen als MATLAB daar klaar voor is (te zien aan het `>>`-prompt). Typ daar:
```
>>load 'lena.mat'
```
Dit zorgt ervoor dat er nu binnen MATLAB er een matrix genaamd `im` beschikbaar is die de grijswaarden van Lena bevat. Laten we eerst kijken hoe groot de matrix is:
```
>>size(im)
```
Dit geeft het aantal pixels in de verticale en horizontale richting weer. Om het plaatje te laten zien, gebruiken we het commando `imagesc`:
```
>>imagesc(im)
```
Ook moeten we aangeven dat het om een grey-scaled image gaat:
```
>>colormap gray;
```
Dit kunnen we in de toekomst makkelijker in één commando doen:
```
>>imagesc(im); colormap gray;
```

We kunnen door middel van 2×2 matrices transformaties uitvoeren op Lena. Zo kunnen we bijvoorbeeld spiegelen rond de verticale as door de volgende matrix te gebruiken:

$$C = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{1}$$

In MATLAB:

```
>>C=[-1 0;0 1]
>>gedraaid=rotate(c,im);
```

Nu bevat de matrix `gedraaid`, het gespiegelde plaatje.

Opdracht Probeer het volgende te doen:

1. Bekijk het gespiegelde plaatje door middel van het `imagesc` commando.
2. Probeer Lena ook langs de horizontale as te spiegelen.
3. Probeer Lena 45 graden rechtsom te draaien door een slimme C te kiezen.
4. Draai Lena ook 60 graden linksom.

3 Fourier-transformatie

Zowel de rijen als de kolommen van een plaatje kunnen we beschouwen als een signaal opgebouwd uit harmonische (sinus/cosinus) functies. De Fourier-transformatie kunnen we uitbreiden tot een 2-dimensionale methode, zodat we de frequencies in zowel de rij- als kolom-richting tegelijkertijd kunnen bepalen. Zodra de rijen en kollomen beide machten van 2 zijn, kunnen we de zogenaamde 'Fast Fourier Transform' (FFT) toepassen in matlab. Controleer dat het plaatje van Lena inderdaad de juiste afmetingen heeft, en pas dan de transformatie toe:

```
>>imfft=fft2(im);
```

Hier komt een matrix uit van precies dezelfde grootte. Als we geen getallen uit deze matrix weggooien, is de transformatie precies; laten we dit controleren door de inverse 2D Fourier-transformatie (`ifft2`) toe te passen:

```
>>imagesc(ifft2(imfft));colormap gray;
```

Opdracht Vergelijk dit met het originele plaatje. Komen ze inderdaad overeen?

Laten we de eerste 4 bij 4 getallen van de Fourier-getransformeerde matrix bekijken:

```
>>imfft(1:4,1:4)
```

Nu vallen er gelijk 2 dingen op:

- De getallen zitten niet tussen 0 en 1 (of: niet in $[0, 1]$).
- De getallen zijn *complex*; naast een reëel deel, heeft elk getal ook een deel met i; dit laaste noemen we het complexe deel. i is gedefiniëerd als het getal waarvoor $i^2 = -1$; hierover morgen meer.

Van complexe getallen kunnen we de *absolute waarde* nemen. Dit is als volgt gedefiniëerd: $|a+bi| = \sqrt{a^2 + b^2}$ (waarvoor wederom morgen de motivatie), met $a, b \in \mathbb{R}$. Merk op dat de absolute waarde zelf in $\mathbb{R}$ zit, en dus niet meer een complex getal is. Elk getal in `imfft` correspondeert met de gevonden frequencies; hoe groter de absolute waarde van zo'n getal, hoe groter de amplitude van de bijbehordende frequentie. De maximale absolute waarde van `imfft` kunnen we als volgt bepalen:

```
>>m=max(max(abs(imfft)))
```

Let op: de dubbele 'max' is geen typfout; 'max' losgelaten op een matrix zorgt ervoor dat MATLAB de maximale waarde in elke kolom opslaat in een rij-vector. Hiervan nogmaals het maximum vragen geeft het maximum van die rij-vector, en dus het maximum van de hele matrix.

Stel dat we alle getallen kleiner dan 10 procent van m als onbelangrijk beschouwen, en deze uit `imfft` willen gooien. Dit kunnen we bewerkstelligen door over alle waarden in de matrix te lopen, dan te controleren of deze kleiner is dan $0.1 * m$, en zo ja, dit getal op 0 zetten:

```
>>imfft_origineel=imfft;
>>for i=1:256
for j=1:256
if abs(imfft(i,j))<0.1*m
imfft(i,j)=0;
end
```

```
end
end
```

Met `find` vinden we een vector met de indices van alle getallen ongelijk nul, en met `length` bepalen we de lengte van vectoren, dus
`>>length(find(imfft))`
geeft ons de hoeveelheid overgebleven getallen in `imfft`.

Vraag Vind je dat dat er veel zijn? De originele hoeveelheid getallen is nog op te vragen met
`>>length(find(imfft_origineel))`
(Als de matrix vol was, dan is het aantal getallen $256 * 256$.)

Het is natuurlijk interessant om te zien wat deze `imfft` voor een plaatje oplevert. Pas de functies `imagesc` en `ifft2` toe om een gereconstrueerd plaatje te krijgen.

Vraag Gezien het aantal niet-nullen in `imfft`, was deze reconstructie te verwachten?

De voorgeprogrammeerde functie `basic_compression` doet precies wat we hier gedaan hebben, voor willekeurige percentages van m. Controleer dat je hetzelfde plaatje krijgt als daarnet met
`>>basic_compression('lena.mat',0.1);`

Opdracht Probeer 0.1 te vervangen met een percentage, zodat we een redelijke compressie krijgen met een nog goede reconstructie.

4 Middelen rond 0

Er is echter nog een cruciaal verschil met harmonische (sin/cos) functies en de grijswaarden zoals we ze nu gebruiken. We kunnen de grijswaarden gebruikt in de hele foto achterelkaar plotten door eerst de matrix te transformeren in een vector dmv het `reshape` functie. Deze vector kunnen we vervolgens plotten:
`>>plot(reshape(im,1,256*256));`
We zien een lastig te voorspellen kleurverloop (natuurlijk is nu ook elke correspondentie die we in 2 dimensies nog hadden, nu ook verdwenen). Het erge punt is echter, dat de grijswaarden blijkbaar tussen 0 en 1 liggen, terwijl harmonische functies tussen -1 en 1 liggen. Als we nu eerst alle kleurwaarden verminderen met 0.5, en vervolgens maal 2 doen, dwingen we af dat de waarden zich in $[-1, 1]$ bevinden. Bedenk wel als we nu transformeren en daarna weer de inverse toepassen, de waardes nog steeds in $[-1, 1]$ liggen! Na de `ifft2`-bewerking moeten we dus weer terugtransformeren (delen door 2, dan 0.5 optellen).

Opdracht Vind en dubbelklik links van het scherm op het bestand `basic_compression.m` en probeer dit in de functie in te bouwen. Vergeet overigens niet dat je altijd een van de begeleiders om hulp kan vragen :).

Vraag Beantwoord daarna het volgende:

1. Krijgen we nu beter resultaat als we alle waarden $< 0.1 * m$ weggooien?
2. Kun je nu je percentage hoger kiezen dan bij het vorige onderdeel, en nog steeds een goed resultaat krijgen?

5 .JPG

Foto's verschillen typisch niet veel op kleine stukjes; kleine kleurveranderingen komen dan vaker voor dan een groot contrastverschil. Dit zagen we ook bij MP3: volgens de officiële specificatie, worden er telkens 576

samples getransformeerd. Bij de JPEG-standaard, wordt voorgeschreven de foto op te hakken in blokken van 8×8 pixels. Van elk van deze blokken worden de kleurwaarden rond 0 geschaald (zie vorige onderdeel). Vervolgens gaat men wat slimmer verder dan alleen componenten met lage amplitude weg te gooien; men heeft onderzoek gedaan naar welke componenten mensen meer opvallen, en welke componenten ons klaarblijkelijk minder of helemaal niet opvalt. Hieruit heeft men een 8×8 matrix kunnen afleiden, die aangeeft hoe belangrijk een bepaalde matrix-waarde is.

Stel dat we onze (nu 8×8) *getransformeerde* foto-matrix even $C(=\texttt{imfft})$ noemen, en de matrixwaarden individueel c_{ij} noemen. Dan voeren we de volgende bewerking uit:

$$\tilde{c}_{ij} = \text{round}(\frac{c_{ij}}{q_{ij}}),$$

waar round() de breuken afrondt.

Deze $\tilde{c}_{ij}$ vormen dan een matrix $\tilde{C}$ welk een benadering is van de originele C. Als q_{ij} vrij groot is, dan komt bovenstaande vrij snel op 0 uit; aan de andere kant, als q_{ij} dichtbij 1 is, wordt de originele c_{ij} waarde met rust gelaten. Belangrijke kanalen hebben dus een lage q_{ij} en onbelangrijke een hoge. Typische waarden voor q_{ij} gegeven in de standaard worden gegeven door de volgende marix:

$$Q = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \tag{2}$$

Dit is in feite de enige plek in het JPG-coderen waar verlies van informatie kan optreden. Het verlies kan nog worden beperkt door ervoor te kiezen de waarden in Q gelijkmatig te verkleinen; daarom werkt men ook vaak met $\frac{1}{s}Q$, waar s een kwaliteitsfactor is.

Vraag Als we zo min mogelijk informatie weg willen gooien, moeten we s dan zo groot of zo klein mogelijk kiezen?

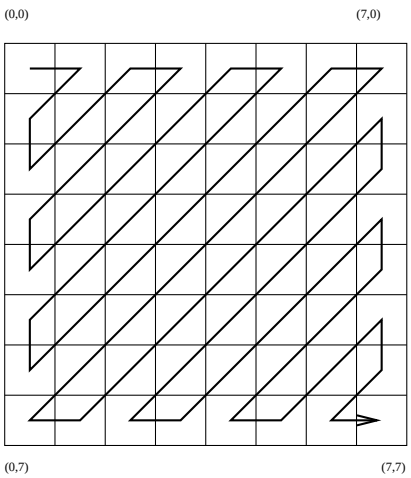
Na de bovenstaande stap hebben we dus een $\tilde{C}$ met relatief veel nullen. Dit gegeven moeten we nog uitbuiten willen we optimale compressie halen. Het uitbuiten gebeurt op een manier die geen informatie verliest en gebeurt in principe in twee stappen:

1. Zig-zag opslaan met 'Zero-RLE', met daarna
2. Huffman encoding.

Het laatste is een standaard-compressie om lange strings van bits zonder verlies op te slaan; hoe dit werkt is mooi te zien op de volgende website:
`http://www.cs.auckland.ac.nz/software/AlgAnim/huffman.html`

Opdracht Bekijk de animatie op deze site en probeer Huffman-codering te begrijpen.

Let wel dat we hier, in plaats van letters, de Fourier-getransformeerde waarden opslaan. Zig-zag met Zero-RLE is ook een lossless compressie-methode, welk makkelijker is uit te leggen. Stel dat we de getallen in onze 8×8 $\tilde{C}$ matrix opslaan in de volgende volgorde:



dan komen we naarmate we het eind van de getallen naderen, steeds meer nullen tegen. Vergelijk de zig-zag volgorde met de getallen in Q. Zero-RLE slaat van de rij getallen in principe telkens het volgende op: hoeveel nullen je tegenkomt voordat je een bepaalde niet-nul tegenkomt. Dit gebeurt in plaats van de getallen gewoon direct opslaan. Voorbeeld: we hebben de rij `12,0,0,1,0,0,-4,14,0,0,0,2`. Dit zijn 12 integers. Passen we Zero-RLE toe, dan zien we dat we direct met 12 beginnen; er staan geen nullen voor, dus we slaan op: $(0, 12)$. Daarna volgen 2 nullen en een 1: $(2, 1)$. Daarna wederom 2 nullen gevolgd door -4: $(2, -4)$. We gaan op dezelfde manier verder en noteren: $(0, 14)$, $(3, 2)$. In plaats van 12 getallen slaan we dus 5 paren op; 10 getallen. De compressie hier door codering is dus $100 - \frac{10}{12} * 100 = 16\frac{2}{3}\%$.

Vraag Vind zelf de Zero-RLE codering van de volgende vectoren:

1. (12,0,0,2,0,3,1,5,0,0,0,10)
2. (0,0,12,0,5,0,0,0,0,3,0,4)
3. (0,1,2,3,2,1,0)
4. (1,0,0,1,1,1,0,0,0,1)

Vraag Bij hoeveel niet-nullen in verhouding tot het totaal aantal getallen, verliest Zero-RLE het ten opzichte van gewoon de hele vector opslaan?

In MATLAB hebben we JPG-compressie geprogrammeerd in de `jpg`-functie. Probeer:
`>>jpg('lena.mat',1);`
Dit voert jpg-compressie uit op lena met $s = 1$. Vergelijk de kwaliteit en compressie met onze eerdere methode door met s te schuiven. Als je nog tijd over hebt kun je proberen onze eigen methode toe te passen op de 8×8 blokken in plaats van op de hele foto; dit kun je zelf programmeren, of proberen in te passen door `jpg.m` te bewerken. Zie je daadwerkelijk verschil tussen het gebruik van de matrix Q en het weggooien van lage frequenties? Een hint om voorgaande simpeler te implementeren: de methode van alles onder een bepaalde waarde weggooien, kan worden bevat in een speciaal daarvoor ontworpen matrix Q.

Dankwoord

Dit document en bijbehorende software is deels gebaseerd op een soortgelijk practicum door Arno Swart & Gerard Sleijpen; bedankt voor het beschikbaar stellen van het materiaal daarvan.