Implementatieplan Scaling

Chris Smeele (1681546), Philippe Zwietering (1685431)

22-02-2017

Contents

1 Doel

Voor het vak Vision van de opleiding HBO-ICT hebben we de opdracht gekregen om een stap uit een gezichtsherkenningsproces te herschrijven met onze eigen code. Wij hebben hier gekozen voor het schalen van afbeeldingen bij het pre-processen. Wij denken dat wij dit minstens zo goed kunnen doen als de oplossing die er al is. Het is voor de rest van het proces belangrijk dat alle afbeeldingen ongeveer evenveel pixels bevatten.

Het probleem dat bij scaling moet worden opgelost is dat afbeeldingen tot het juiste formaat worden geschaald, waarbij er rekening mee wordt gehouden dat er zo min mogelijk informatie verloren gaat bij het omlaag schalen van afbeeldingen.

Bij het opschalen van afbeeldingen moet er juist worden nagedacht hoe ervoor wordt gezorgd dat de juiste invulling wordt gegeven aan de extra pixels, oftewel interpolatie.

Aan de andere kant is het prettig om het proces zo snel mogelijk te laten verlopen. Het verwachte eindresultaat is een afbeelding in totaal ongeveer 40.000 pixels, die nog goed herkenbaar is als gezicht.

2 Methoden

Voor het schalen van afbeeldingen kun je backwards-mapping goed gebruiken, waarbij je vervolgens moet interpoleren om de beste pixelwaarden terug te krijgen. Een aantal bekende interpolatiemethoden zijn:

- Nearest-neighbor interpolation
- Bilinear interpolation
- Bicubic interpolation
- Box sampling
- Edge-directed interpolation

Dit lijstje staat al op volgorde van moeilijkheidsgraad en benodigde rekenkracht. Nearest-neighbor interpolation is duidelijk het simpelst, terwijl er in edge-directed interpolation recent nog nieuwe vooruitgangen zijn geboekt. [1, 2, 3]

Nearest-neighbor tot box-sampling staan ook op omgekeerde volgorde van nauwkeurigheid.

Bicubic interpolation heeft alleen wel het nadeel ten opzichte van bilinear interpolation dat het overbelichting kan veroorzaken, maar heeft wel een groter detailbehoud dan bilinear interpolation.[3]

Box-sampling heeft dit probleem niet en is nauwkeuriger dan bicubic interpolation, maar is aanzienlijk moeilijker te implementeren.[4] Box-sampling en bicubic interpolation hebben wel dezelfde benodigde rekenkracht.

Edge-directed interpolation is niet per se een nauwkeurigere interpolatie methode, dit is terug te zien in [5], want het focust vooral op het behouden van edges in het schalingsproces.

3 Keuze

Omdat we graag verschillende, veelgebruikte methoden willen vergelijken hebben en de hogere orden methoden waarschijnlijk te moeilijk zijn om door ons geïmplementeerd te worden, hebben wij er voor gekozen om nearest-neighbor, bilinear en bicubic interpolation te gebruiken om te schalen. We denken dat deze drie methoden zeer haalbaar zijn, omdat ze wiskundig niet heel ingewikkeld zijn. Daarnaast zijn er genoeg dingen die we kunnen meten om een goed onderzoek uit te kunnen voeren.

4 Implementatie

Wij implementeren de drie bovengenoemde schalingsmethoden binnen de bestaande HU-Vision repository.

Het schalen is onderdeel van de pre-processing stap van gezichtsherkenning. Om deze reden zullen wij de code verwerken in de methode `stepScaleImage()` van de `StudentPreProcessing` klasse.

Deze functie zal een grijswaardenafbeelding moeten schalen zodat deze, zonder de verhoudingen te veranderen, het dichtst bij 40.000 pixels in de buurt komt. Bronafbeeldingen die kleiner zijn dan 40.000 pixels worden onveranderd geretourneerd.

Gezien de berekening van elke nieuwe pixel onafhankelijk van andere nieuwe pixels is, kunnen we het schalen parallelliseren.

Hieronder volgt per methode een korte beschrijving van de implementatie[6]:

4.1 Nearest-neighbor

1. Bereken de nieuwe afmetingen:
   - `scale = sqrt(40000 / width*height)`
   - `(newWidth, newHeight) = (width*scale, height*scale)`
2. Maak een afbeelding van de zojuist berekende afmetingen.
3. Voor elke (x,y) in de nieuwe afbeelding:
   - Vind de dichtsbijzijnde pixel in het origineel: (`round(x*schaal)` , `round(y*schaal)`)
   - Stel (x,y) in op de waarde van deze pixel in het origineel

[1] https://www.academia.edu/3636528/Comparative_Analysis_of_Different_Interpolation_Schemes_in_Image_Processing

[2] http://s3.amazonaws.com/academia.edu.documents/38411794/image_scaling_comp_using_quality_index_int_conf.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1487856758&Signature=0a0LAmigralkaS29EuBjeJY5f%2FQ%3D&response-content-disposition=inline%3B%20filename%3DIMAGE_SCALING_COMPARISON_USING_UNIVERSAL.pdf

[3] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.298.358&rep=rep1&type=pdf

[4] https://www.ldv.ei.tum.de/fileadmin/w00bfa/www/content_uploads/Vorlesung_3.4_Resampling.pdf

[5] https://en.wikipedia.org/wiki/Comparison_gallery_of_image_scaling_algorithms

[6] http://pippin.gimp.org/image_processing/chap_resampling.html

4.2 Bilinear

1. Bereken de nieuwe afmetingen:
   - `scale = sqrt(40000 / width*height)`
   - `(newWidth, newHeight) = (width*scale, height*scale)`
2. Maak een afbeelding van de zojuist berekende afmetingen.
3. Voor elke (x,y) in de nieuwe afbeelding:
   - Neem `(X,Y) = (x/scale, y/scale)` (coördinaten binnen het origineel, tussen pixels in).
   - Neem `(X1,X2,Y1,Y2) = (floor(X), ceil(X), floor(Y), ceil(Y))`, de coördinaten van de dichtsbijzijnde 4 pixels in het origineel.
   - Interpoleer lineair horizontaal punt (X,Y1) tussen (X1,Y1) en (X2,Y1)
   - Interpoleer lineair horizontaal punt (X,Y2) tussen (X1,Y2) en (X2,Y2)
   - Interpoleer lineair verticaal punt (X,Y) tussen (X,Y1) en (X,Y2)
   - `(x,y) = (X,Y)`

4.3 Bicubic

1. Bereken de nieuwe afmetingen:
   - `scale = sqrt(40000 / width*height)`
   - `(newWidth, newHeight) = (width*scale, height*scale)`
2. Maak een afbeelding van de zojuist berekende afmetingen.
3. Voor elke (x,y) in de nieuwe afbeelding:
   - Neem `(X,Y) = (x/scale, y/scale)` (coördinaten binnen het origineel, tussen pixels in).
4. Voor elke (x,y) in de nieuwe afbeelding:
   - Neem `(X,Y) = (x/scale, y/scale)` (coördinaten binnen het origineel, tussen pixels in).
   - Neem `(X1,X2,Y1,Y2) = (floor(X), ceil(X), floor(Y), ceil(Y))`, de coördinaten van de dichtsbijzijnde 4 pixels in het origineel.
   - Interpoleer kubisch horizontaal punt (X,Y1) tussen (X1,Y1) en (X2,Y1)[7]
   - Interpoleer kubisch horizontaal punt (X,Y2) tussen (X1,Y2) en (X2,Y2)
   - Interpoleer kubisch verticaal punt (X,Y) tussen (X,Y1) en (X,Y2)
   - `(x,y) = (X,Y)`

5 Evaluatie

We willen graag aantonen dat onze methode sneller of beter is dan de methode die nu wordt gebruikt voor het schalen.

Daarom denken wij dat het nuttig is om de snelheid en de kwaliteit van de verschillende interpolatie methoden met elkaar te vergelijken. Het vergelijken van snelheid spreekt voor zich, maar voor het vergelijken van kwaliteit is er nog een tussenstap nodig. Het makkelijkst is om verschillende methoden te vergelijken door de plaatjes van elkaar af te trekken, hierdoor kun je direct zien wat

[7] Gebruik hiervoor de formule $f(p_{0,p1,p2,p3,x}) = (-\frac{1}{2}p_0 + \frac{3}{2}p_1 - \frac{3}{2}p_2 + \frac{1}{2}p_3)x^3 + (p_0 - \frac{5}{2}p_1 + 2p_2 - \frac{1}{2}p_3)x^2 + (-\frac{1}{2}p_0 + \frac{1}{2}p_2)x + p_1$, waar `p[0-3]` de 4 waarden rondom x zijn.

het verschil tussen plaatjes is. Op basis van het resultaat kun je vervolgens conclusies trekken door te letten op bijvoorbeeld scherpte en het behoud van edges. De kwaliteitsverschillen kunnen ook bepaald worden door te kijken naar de verschillen in uitkomsten van het gehele gezichtsherkenningsproces.

De huidige implementatie van interpolatie is bilinear en singlethreaded. Wij verwachten dat onze implementatie van bilineare interpolatie sneller zal zijn omdat wij deze parallel gaan uitvoeren.

We denken dat de bicubic de langzaamste methode is, maar waarschijnlijk wel de beste, al zal er niet heel veel verschil zitten in kwaliteit met bilinear interpolation. Bilinear is sneller en gebruikt minder geheugen, dus in de praktijk is bilinear even goed voor leken, maar het is wel sneller en gebruikt minder resources. Nearest-neigbor is relatief heel snel en goedkoop, alleen iedereen kan vaak meteen zien dat er slecht is geïnterpoleerd.