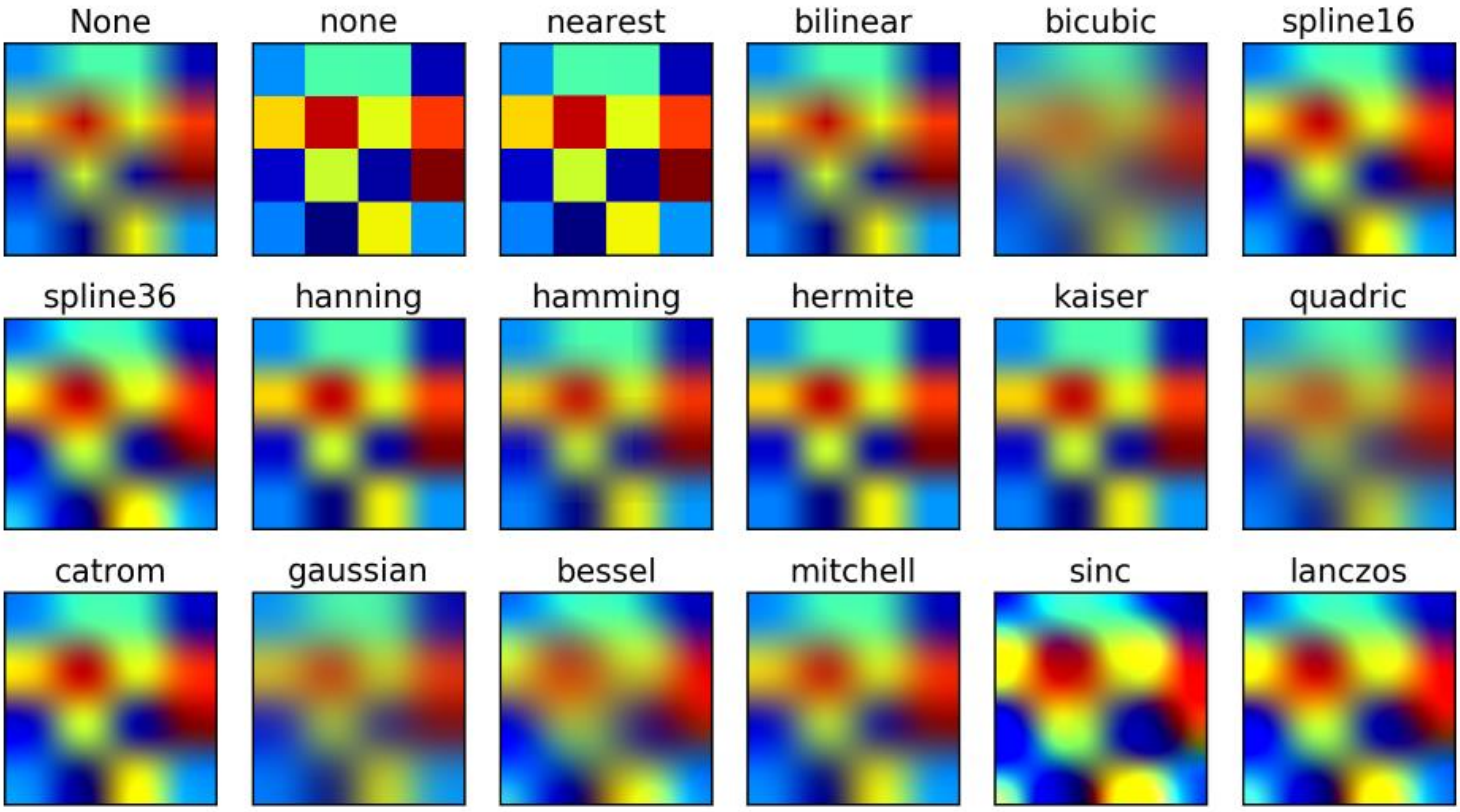
IMAGE SCALING IMPLEMENTATIEPLAN



FIGUUR 1: VERSCHILLENDE SCALING ALGORITMES
UIT: HTTPS://MATPLOTLIB.ORG/MPL_EXAMPLES/IMAGES_CONTOURS_AND_FIELDS/INTERPOLATION_METHODS.HIRES.PNG

Gemaakt door: Patrick van der Bend

Datum: 9 February 2019

Versie: 1.1

Versiebeheer

Versie	Wijzigingen	Datum
0.1	Eerste versie: - Alle headings gemaakt - Doel geschreven - Begonnen met het schrijven van Methoden: o Inleiding geschreven en tabel gemaakt	6 February 2019
0.2	Methoden afgemaakt: - Uitleg voor alle methoden geschreven - Voordelen en nadelen geschreven. Keuze geschreven Deel Implementatie geschreven Evaluatie geschreven	7 February 2019
1.0	Document is klaar: - Keuze verder onderbouwt - Implementatie afgemaakt	8 February 2019
1.1	Wat problemen bij Implementatie en Evaluatie weggehaald.	9 February 2019

Doel

In dit document wordt een plan beschreven om een implementatie van een image scaling-algoritme te bouwen. Het doel van deze implementatie is het formaat van afbeeldingen naar een bepaalde pixelhoeveelheid te brengen als deel van een groter framework dat bedoeld is voor gezichtsherkenning. Het is hierom belangrijk om deze implementatie te optimaliseren voor gezichtsherkenning en later in dit document wordt dan ook een overweging beschreven om te bepalen welk algoritme het best is voor deze implementatie.

Methoden

Er zijn veel scaling-algoritmes met allemaal hun eigen eigenschappen, in dit document worden er vier besproken: Nearest-neighbour interpolation, Bilinear interpolation, Bicubic interpolation en Lanczos interpolation. Om te bepalen welk algoritme het best is voor het doel, worden de algoritmes in een tabel hieronder vergeleken. Daarna wordt een korte uitleg gegeven over elke algoritme en tot slot wordt er in de kop Keuze een besluit gegeven en gemotiveerd.

Tabel

De onderstaande tabel laat een vergelijking zien tussen verschillende algoritmes bij het vergroten en verkleinen van afbeeldingen:

Name	Smoothness (up-scaling)	Smoothness (down-scaling)	Speed
Nearest-neighbour interpolation	Lowest	Highest	Highest
Bilinear interpolation	high	Average	high
Bicubic interpolation	Average	Average	Average
Lanczos interpolation	low	Average	lowest

Hier is gebruik gemaakt van een vergelijking door Anthony Tanbakuchi in 2016: http://tanbakuchi.com/posts/comparison-of-openv-interpolation-algorithms/

Nearest-neighbour interpolation

Bij interpolatie (interpolation) wordt de waarde van een pixel bepaald door te kijken naar de waardes van pixels in de originele afbeelding. Bij Nearest-neighbour interpolation wordt alleen gekeken naar de pixel in de originele afbeelding wiens positie het meest overeen komt (de dichtstbijzijnde) met de onbekende pixel. De pixels in de buurt van de positie van de pixel met de onbekende waarde (noem ik vanaf nu de onbekende pixel) worden compleet genegeerd. Dit betekend dat de waarde van de onbekende pixel compleet overgenomen wordt van de dichtstbijzijnde pixel op de originele afbeelding.

Voordelen van dit algoritme zijn: simpliciteit, snelheid en de “scherpte” van het resultaat: dit algoritme behoudt de randen van de originele afbeelding erg goed. De scherpe randen kunnen, en worden meestal, gezien als een nadeel, maar bij gezichtsherkenning kan dit een voordeel zijn omdat de details van het gezicht niet zo smooth worden dat ze erg moeilijk te herkennen zijn. De scherpte kan echter ook

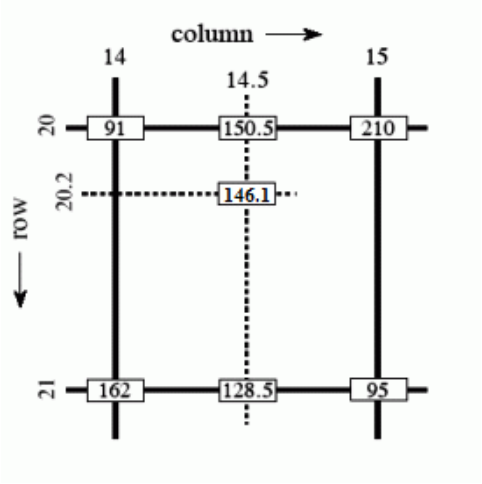
een nadeel zijn omdat de waardes van pixels zo ver van elkaar kunnen zijn dat er te veel randen ontstaan, die het herkennen van bijvoorbeeld gezichtskenmerken juist kunnen vermoeilijken.

Bilinear interpolation

Bij Bilinear interpolation wordt er voor een onbekende pixel gekeken naar de vier dichtstbijzijnde pixels op de originele afbeelding. Eerst wordt twee keer het gewogen gemiddelde bepaald van een bekend pixelpaar, en daarna het gewogen gemiddelde tussen de twee paren. Dit gemiddelde wordt dan de waarde voor de onbekende pixel.

Een voordeel van dit algoritme is dat het resultaat smoother is dan bij Nearest-neighbour interpolation; dit zorgt voor een visueel mooier resultaat, terwijl de snelheid van Bilinear interpolation nog steeds relatief groot is.

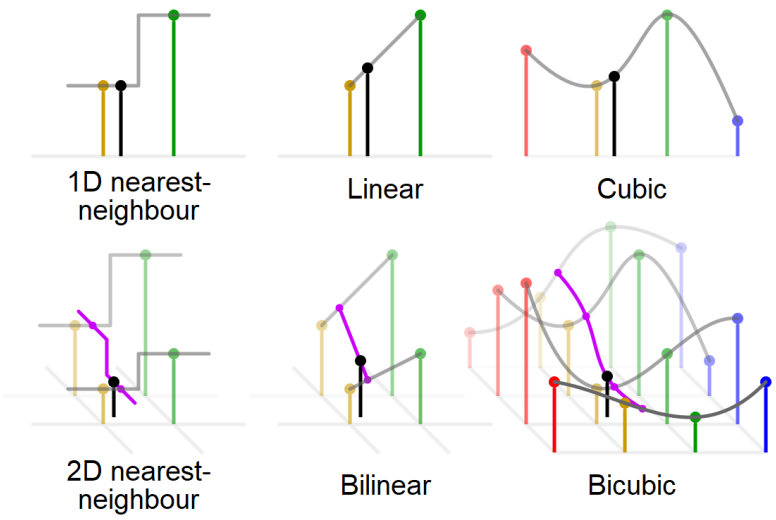
Een nadeel van Bilinear interpolation is dat het resultaat te smooth kan worden waardoor er weinig contrast bij de randen overblijft. Dit kan het herkennen van gezichtskenmerken vermoeilijken.



FIGUUR 2: VISUALISATIE VAN BILINEAR INTERPOLATION, HET COÖRDINAAT VAN DE ONBEKENDE PIXEL IS (14.5;20.2). DE WAARDE VAN EEN PIXEL STAAT IN HET VIERKANT DIE DE PIXEL VOORSTELT.

Bicubic interpolation

Bij Bicubic interpolation wordt niet alleen de vier dichtstbijzijnde bekende pixels maar ook hun buren, dus in totaal zestien pixels. Met deze extra pixels wordt een gradiënt berekend tussen de vier dichtstbijzijnde bekende pixels. Dit gradiënt is dan, in contrast tot Bilinear interpolation, niet lineair; de gradiënt houdt rekening met vier pixels tegelijk om zo een derde graad polynoom op te stellen die dan de gradiënt vormt. Dit gebeurt dan vier keer in dezelfde richting (bijvoorbeeld horizontaal) zodat daarna een polynoom opgesteld kan worden in de andere richting (dan dus verticaal) op de verticale positie van de onbekende pixel. Met de gradiënt die hier uitkomt is dan de waarde van de onbekende pixel te achterhalen. [TODO: formules?]



FIGUUR 3: MEERDERE INTERPOLATIE ALGORITMES, WAARONDER CUBIC, MET HUN 1D EN 2D VERSIES. BIJ DE 1D VERSIE VAN CUBIC IS DE GRADIENT (POLYNOOM) GOED TE ZIEN. IN DE 2D VERSIE WORDEN ER VIER HORIZONTALE GRADIENTEN BEREKEND, EN DAAROVER NOG EEN VERTICALE GRADIENT ZODAT DE WAARDE VAN HET ONBEKENDE PUNT, HET ZWARTE PUNT IN DE FIGUUR, BEREKEND KAN WORDEN.
DOOR CMGLEE - OWN WORK, CC BY-SA 4.0, HTTPS://COMMONS.WIKIMEDIA.ORG/W/INDEX.PHP?CURID=53064904

Een voordeel van dit algoritme is dat de afbeelding smoother is dan Nearest-neighbour interpolation, maar de randen weet te behouden.

Een nadeel is clipping: in de polynoom kunnen waarden op een top of dal buiten de minimale of maximale waarde vallen. Hierdoor moet de waarde afgekapt worden. Dit geeft ook rare artefacten als het contrast bij een rand op de originele afbeelding erg groot is. Een ander nadeel is dat de snelheid van Bicubic interpolation is langzamer dan de snelheid van Bilinear interpolation en Nearest-neighbour interpolation.

Lanczos interpolation

Lanczos interpolation doet ongeveer hetzelfde als Bicubic interpolation, maar in plaats van een derde-graad polynoom te berekenen, wordt een coëfficiënt gebruikt. Dit coëfficiënt is berekend met een gevensterde sinc-functie genaamd de Lanczos kernel.

Bij de uitleg hieronder is vooral informatie gehaald uit:
https://software.intel.com/en-us/ipp-dev-reference-lanczos-interpolation

De Lanczos kernel is gedefinieerd als:

$$L(x) = \operatorname{sinc}(x) \cdot Lanczos2(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} \cdot \frac{\sin((\pi x)/2)}{(\pi x)/2}, & 0 \le |x| < 2 \\ 0, & 2 \le |x| \end{cases}$$

FIGUUR 4: DE LANCZOS KERNEL, SINC(X) IS DE GENORMALISEERDE SINC

Om de intensiteit van een onbekende pixel op coördinaat (X_D ; Y_D), wordt er gekeken naar zestien pixels op de originele afbeelding met de coördinaten: (X_{SI} ; Y_{SI}) waar geldt:
$x_{S0} = \text{int}(x_D) - 1$; $x_{S1} = x_{S0} + 1$; $x_{S2} = x_{S0} + 2$; $x_{S3} = x_{S0} + 3$; en
$y_{S0} = \text{int}(y_D) - 1$; $y_{S1} = y_{S0} + 1$; $y_{S2} = y_{S0} + 2$; $y_{S3} = y_{S0} + 3$;

De interpolatie op de x-as gebeurt volgens:

$$I_k = \sum_{i=0}^{3} a_i \cdot s(x_{Si}, y_{Sk}), 0 \le k \le 3$$

FIGUUR 5: LANCZOS INTERPOLATIE OP DE X-AS, HIER IS:
I_K: DE INTENSITEIT OP DE X-AS.
A_I: HET COËFFICIËNT BEREKEND MET: $L(x_S - x_{SI})$
S(X,Y): DE INTENSITEIT OP DE SOURCE AFBEELDING OP COÖRDINAAT (X;Y)

Daarna kan de intensiteit op de onbekende afbeelding berekend worden door interpolatie op de y-as volgens:

$$D(x_D, y_D) = \sum_{k=0}^{3} b_k \cdot I_k$$

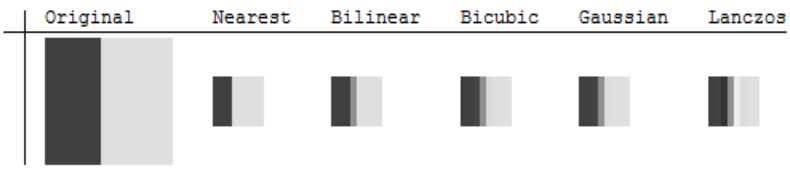
FIGUUR 6: LANCZOS INTERPOLATIE OP DE Y-AS, HIER IS:
D(X,Y): DE INTENSITEIT OP DE DESTINATION AFBEELDING COÖRDINAAT (X;Y)
B_K: HET COËFFICIËNT BEREKEND MET $L(y_S - y_{SI})$
I_K: DE INTENSITEIT OP DE X-AS

Kenmerkend van Lanczos interpolation is dat het contrast en randen in de afbeelding behoud en soms zelfs vergroot. Terwijl gebieden zonder randen nog steeds smooth worden. Dit kan een groot voordeel zijn omdat dit het herkennen van gezichtskenmerken makkelijker maakt.

Een groot nadeel van dit algoritme is de lage snelheid; van alle algoritme die hier besproken zijn, is Lanczos interpolation het langzaamst (zie de tabel). Een ander nadeel van Lanczos interpolation is overshoot en undershoot van waarde dicht bij zware contrasten. Dit betekent dat de waardes van de

pixels waar overshoot of undershoot plaats vindt niet meer representatief zijn van de originele afbeelding.

Keuze



FIGUUR 7: VERGELIJKING TUSSEN VIJF INTERPOLATIE TECHNIEKEN BIJ HET VERKLEINEN VAN EEN AFBEELDING. HIER IS GOED TE ZIEN DAT LANCZOS HET CONTRAST VERGROOT. UIT: HTTPS://GIS.STACKEXCHANGE.COM/QUESTIONS/10931/WHAT-IS-LANCZOS-RESAMPLING-USEFUL-FOR-IN-A-SPATIAL-CONTEXT/14361#14361

Bij de keuze tussen de algoritmes moet er rekening gehouden worden met het doel van het resultaat van de algoritmes. In dit geval is het doel het herkennen van gezichtskenmerken in een groter framework. Voor dit framework moeten de afbeeldingen naar een bepaalde pixelhoeveelheid gebracht worden. Deze hoeveelheid is relatief laag en dus moet er vooral gekeken worden naar de resultaten en prestaties van de algoritmes tijdens het verkleinen van afbeeldingen. Verder kan snelheid van belang zijn, hier moet dus ook naar gekeken worden.

In dit document gaat de voorkeur uit naar Lanczos interpolation, hieronder de overweging:

Als snelheid de grootste bezorgdheid zou zijn, had Nearest-neighbour interpolation de voorkeur gekregen. De snelheid van dit algoritme is het grootst van alle algoritmes die in dit document beschreven zijn en de randen worden goed behouden. Maar omdat er helemaal geen smoothing plaats vindt kan het contrast tussen pixels zo groot zijn dat dit verwarrend kan werken tijdens het zoeken naar gezichtskenmerken.

Bij Bilinear interpolation is het resultaat erg smooth wat ervoor kan zorgen dat de gezichtskenmerken niet meer te onderscheiden zijn van de rest van de afbeelding.

De overweging was vooral tussen Bicubic interpolation en Lanczos interpolation. Lanczos interpolation behoudt de randen het best, maar Bicubic interpolation is het snelst. Uiteindelijk kwam het dus neer op wat het belangrijkst is, snelheid of het behouden van randen, en dat lijkt het behouden van de randen te zijn. Dit kan echter nog niet met volledige zekerheid gezegd worden omdat er eerst getest zou moeten worden of de randen goed genoeg worden behouden bij Bicubic interpolation voor gezichtsherkenning. Als dat zo is, is de snelheid van Bicubic interpolation een duidelijk voordeel over Lanczos interpolation. Maar zonder deze test, lijkt Lanczos interpolation de beste keuze.

Implementatie

De uiteindelijke implementatie wordt geschreven in de functie:
IntensityImage * stepScaleImage(const IntensityImage &image)

Bij Lanczos interpolation zijn er twee stappen te onderscheiden:

1. Interpolatie op de x-as
2. Interpolatie op de y-as

Hiervoor kunnen twee functies voor gemaakt worden:

1. Intensity lanczosInterpolate(Intensity* source, float x):
   voert de interpolatie uit met vier verschillende punten.
2. Intensity biLanczosInterpolate(IntensityImage source, float x, float y):
   voert vijf keer lanczosInterpolate uit, eerst vier keer op de x-as, daarna één keer op de y-as, met in totaal 16 verschillende punten.

Als deze functies gemaakt zijn kan er door een array van pixels van de nieuwe afbeelding geloopd worden en hun waarde bepaald worden door eerst biLanczosInterpolate uit te voeren met de zestien dichtstbijzijnde pixels op de originele afbeelding. Met de array van pixels wordt nu een afbeelding gemaakt worden die teruggegeven wordt door de stepScaleImage functie.

Evaluatie

Om deze implementatie te evalueren moeten twee tests gedaan worden:

De eerste test om te zien of de implementatie net zo goed werkt als de default functie die al in het framework zit. Hiervoor moet er in ieder geval gekeken worden naar:

- Kwaliteit: als de default functie gebruikt wordt en de gezichtsherkenning is succesvol, geldt dat dan ook voor deze implementatie?
- Snelheid: is deze implementatie sneller, net zo snel of langzamer dan de default functie? Hoe groot is het verschil?

De tweede test is om te bepalen of de juiste beslissing gemaakt bij het kiezen van algoritmes, hiervoor moet deze implementatie vergeleken worden met een nieuwe implementatie die gebruik maakt van Bicubic interpolation. Er moet in ieder geval gekeken worden naar:

- Kwaliteit: werkt de gezichtsherkenning net zo goed als er Bicubic interpolation gebruikt wordt in plaats van Lanczos interpolation?
- Snelheid: hoe groot is het verschil in snelheid als er Bicubic interpolation gebruikt wordt in plaats van Lanczos interpolation?

Deze twee tests zullen moeten worden uitgevoerd en de uitvoering moet goed gedocumenteerd worden in twee aparte testverslagen. De implementatie functioneert goed genoeg als het verschil in succesvolle gezichtsherkenning tussen de default implementatie en deze implementatie niet onder de 80% valt en de snelheid van deze implementatie niet onder de 50% van de snelheid van de default implementatie valt.

Als het blijkt dat Lanczos interpolation niet goed genoeg functioneert, zal er een ander algoritme gekozen moeten worden. Dit kan Bicubic interpolation worden, als de tweede test heeft aangetoond dat Bicubic interpolation beter functioneert dan Lanczos interpolation. Wat deze nieuwe implementatie ook wordt, het zal dan net zo goed getest moeten worden om te zien of het nieuwe algoritme wel goed genoeg functioneert.

References

Breeuwsma, P. (n.d.). *Cubic interpolation*. Retrieved from Paulinternet: http://www.paulinternet.nl/?page=bicubic

Computerphile. (2016, November 23). *Bicubic Interpolation - Computerphile*. Retrieved from YouTube: https://www.youtube.com/watch?reload=9&v=poY_nGzEEWM

Intel Software. (2018, November 7). *Appendix B: Interpolation in Image Geometric Transform Functions*. Retrieved from Intel Software Developer Zone: https://software.intel.com/en-us/ipp-dev-reference-appendix-b-interpolation-in-image-geometric-transform-functions

Tanbakuchi, A. (2016, February 21). *Comparison of OpenCV Interpolation Algorithms*. Retrieved from Tanbakuchi: http://tanbakuchi.com/posts/comparison-of-openv-interpolation-algorithms/