

Finding edges in (labelled faces in) the wild

Tony van der Krogt
Jasper de Winther

23-03-2020

Docent: Diederik Yamamoto-Roijers

Finding edges in (labelled faces in) the wild

Doel	2
Hypothese	3
Werkwijze	4
Criteria	5
Validatie data	5
Resultaten	6
Verwerking	10
Conclusie	10
Robuustheid	10
Afwijking	10
Snelheid	10
Evaluatie	11
Hypothese/conclusie	11
Meetonzekerheden	11

Doel

In dit verslag wordt gekeken of het huidige face recognition programma, door middel van een andere edge detection manier, meer gezichten kan herkennen en wat de performance impact hiervan is. Uiteindelijk willen we een implementatie ontwikkelen die robuuster werkt dan de default implementatie, ook kijken we naar de performance impact van onze aanpassingen.

Hypothese

In de implementatie van Arno Kamphuis (de default implementatie) wordt gebruik gemaakt van een 9x9 laplacian kernel zonder de diagonalen. Alle 3x3 kernels zijn sneller, dan de 9x9 kernels, maar leveren slechtere herkenning percentages (zo goed als geen). De 9x9 laplacian kernel met diagonalen geeft een beter herkenningpercentage dan de default implementatie met vergelijkbare performance. De prewitt en sobel operators werken allebei niet als deze maar in één richting worden gebruikt, maar even goed als de default implementatie wanneer allebei de richtingen worden gebruikt en dan samen worden gevoegd.

Hieronder een tabel met de verwachte uitkomsten:

	detecties	snelheid	error	valse detecties
default	x	x	x	x
SobelX (9x9)	---	-	+	x
SobelX (3x3)	---	++	+	x
SobelY (9x9)	---	-	+	x
SobelY (3x3)	---	++	+	x
SobelX+SobelY (9x9)	-	---	+	x
SobelX+SobelY (3x3)	+	+	+	x
Laplace (9x9)	x	-	+	x
Laplace (3x3)	-	++	+	x
Laplace + diagonaal (9x9)	+	-	+	x
Laplace + diagonaal (3x3)	-	++	+	x
PrewittX (9x9)	---	-	+	x
PrewittX (3x3)	---	++	+	x
PrewittY (9x9)	---	-	+	x
PrewittY (3x3)	---	++	+	x
PrewittX+PrewittY (9x9)	-	---	+	x
PrewittX+PrewittY (3x3)	+	+	+	x

+ betekent een positief verschil, - betekent een negatief verschil en x betekent geen verschil (allemaal met de default implementatie). De kolom 'error' is de hoeveelheid crashes in het programma. Het verschil tussen de kolommen met een 9x9 en een 3x3 is de kernel grootte.

Werkwijze

Wij hebben een kernel klasse geïmplementeerd die alle kernels kan maken die wij nodig hebben voor dit onderzoek, en in alle grootten die nodig zijn. Om de implementatie te testen hoeft alleen de grootte van de kernel en het type kernel aangepast te worden (dit is snel gedaan want het zijn maar 2 waarden, niet de hele kernel moet worden uitgetyped). Als deze aanpassingen zijn aangebracht moet de exe worden gecompileerd, en kunnen er tests worden uitgevoerd van de command line. Om al onze afbeeldingen te testen hebben we een python script gemaakt die automatisch door een hele folder gaat en alle gevonden afbeeldingen analyseert. Deze analyse duurt ongeveer 5 tot 10 minuten. Hierna wordt een totale executietijd berekend, het percentage herkende gezichten, de standaarddeviatie van het aantal herkende gezichten, het percentage crashes en de standaarddeviatie van het aantal crashes.

Criteria

Om te besluiten welke edge detection het beste werkt, hebben we besloten vooral te kijken naar de robuustheid (of meer afbeeldingen worden herkend) en de snelheid (executietijd), en hopen deze te verbeteren. De robuustheid wordt getest door (zoals onder het kopje validatie data beschreven staat) meerdere afbeeldingen door de hele pipeline te halen en te kijken hoeveel gezichten worden herkend (als percentage). De snelheid wordt berekend door de applicatie meerdere keren met verschillende afbeeldingen volledig door te lopen en te kijken wat het verschil is tussen de verschillende edge detection methoden. Ook is het belangrijk dat alle features die al werkend zijn blijven werken, en de geheugen efficiëntie hetzelfde blijft.

Ook berekenen we de standaarddeviatie van elke implementatie, dit doen we door onze dataset op te splitsen in batches van 100 afbeeldingen, hier het percentage detecties van berekenen en met deze uitkomsten berekenen we de standaarddeviatie.





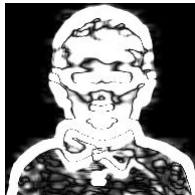

Validatie data

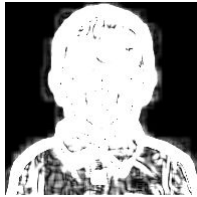






Om onze implementaties te testen hebben we een dataset van voldoende grootte nodig. Wij gaan er van uit dat gezichten moeten worden herkend in natuurlijke situaties. Hieronder vallen mensen die niet in de camera kijken, hun hoofd lichtelijk gedraaid hebben en schaduwen aanwezig zijn. Uiteindelijk hebben we besloten de “LFW Face Database” met meer dan 13000 afbeeldingen te gebruiken, aangezien die aan alle hierboven genoemde criteria voldoet en in meer dan 3900 papers is geciteerd (dus hoogstwaarschijnlijk goede kwaliteit). Door gebruik van deze “natuurlijke” database, kunnen we het verschil in robustness vergelijken met de default implementatie. Ook kunnen we door het gebruik van 13000 afbeeldingen berekenen wat de gemiddelde performance impact is. Een van de nadelen van de gebruikte dataset is wel dat alle afbeeldingen 250 bij 250 pixels zijn. Hierdoor zijn de resultaten pas toepasbaar na het schalen van nieuwe afbeeldingen naar 250 bij 250 pixels.






Ook hebben we een dataset met 3000 afbeeldingen om te valideren dat er geen valse detecties zijn. Dit zijn afbeeldingen van de coco 2017 validatieset, waarbij wij zelf alle afbeeldingen waar minimaal de helft van een gezicht te zien is hebben verwijderd. Ook hebben we deze afbeeldingen geschaald naar 250 bij 250 pixels zodat ze beter vergelijkbaar zijn met de LFW Face Database.

Resultaten

In onderstaande tabel staat de oorspronkelijke afbeelding bovenaan, en daaronder de resultaten van de verschillende kernels. Wel is het belangrijk om te weten dat de kernels na de greyscaling worden toegepast. En de hieronder vertoonde afbeeldingen voor de thresholding zijn gemaakt.

Kernel	Result
None	
Default	
SobelX 9x9	
SobelX 3x3	
SobelY 9x9	
SobelY 3x3	

SobelX + SobelY 9x9	
SobelX + SobelY 3x3	
Laplace 9x9	
Laplace 3x3	
Laplace + diagonaal 9x9	
Laplace + diagonaal 3x3	
PrewittX 9x9	

PrewittX 3x3	
PrewittY 9x9	
PrewittY 3x3	
PrewittX + PrewittY 9x9	
PrewittX + PrewittY 3x3	

Hieronder staat een tabel met onze resultaten, groen is beter. De kolom "Precision" geeft de precision weer (meer over precision staat onder het kopje verwerking), TP de true positive, FP de false positive, "Total errors" het aantal errors dat zodanig erg was dat het programma niet meer verder kon, "Total runtime" de totale tijd die het programma er over deed om de volledige dataset (16466 afbeeldingen) door te lopen, en de "Standard deviation runtime" die de standaarddeviatie van de runtime van elke afbeelding weergeeft.

Kernel	Precision	TP %	FP %	Total errors	Total runtime	Standard deviation runtime
Default	0.950	0.58%	0.03%	78	374	0.046
SobelX (9x9)	Nan	0.00%	0.00%	1	522	0.005
SobelX (3x3)	0.271	0.02%	0.06%	4	336	0.016
SobelY (9x9)	1.000	0.01%	0.00%	3	561	0.016
SobelY (3x3)	0.798	0.12%	0.03%	33	414	0.033
SobelX + SobelY (9x9)	Nan	0.00%	0.00%	4	778	0.018
SobelX + SobelY (3x3)	0.942	1.00%	0.06%	87	451	0.054
Laplace (9x9)	0.825	0.58%	0.12%	64	580	0.045
Laplace (3x3)	Nan	0.00%	0.00%	0	316	0.002
Laplace + Diagonal (9x9)	1.000	0.13%	0.00%	40	570	0.035
Laplace + Diagonal (3x3)	Nan	0.00%	0.00%	0	327	0.003
PrewittX (9x9)	1.000	0.01%	0.00%	1	587	0.011
PrewittX (3x3)	Nan	0.00%	0.00%	2	337	0.007
PrewittY (9x9)	0.634	0.05%	0.03%	16	530	0.063
PrewittY (3x3)	0.426	0.02%	0.03%	8	330	0.013
PrewittX + PrewittY (9x9)	Nan	0.00%	0.03%	4	783	0.016
PrewittX + PrewittY (3x3)	0.908	0.30%	0.03%	32	397	0.032

Verwerking

Om de standaarddeviatie van de executietijd te berekenen hebben we onderstaande functie gebruikt.

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

Waarbij N het aantal afbeeldingen is, x_i de executietijd van die specifieke afbeelding en μ de mean van alle executie tijden is.

Om de “Precision” te berekenen hebben we de true en false positive samen gebruikt om een beeld te krijgen van het ratio tussen deze twee waarden. Hieronder staat de berekening die wij hebben gebruikt:

$$Precision = \frac{TP}{TP + FP}$$

$TP = \text{True positive}$
 $FP = \text{False positive}$

Conclusie

Robuustheid

Zoals bij de resultaten te zien is, heeft sobelX + sobelY 3x3 het hoogste true positive percentage. Echter heeft het wel een hoger false positive percentage waardoor de precisie lager is dan de default implementatie.

Afwijking

Als we kijken naar de total errors- en standard deviation kolom zien we des te hoger het TP percentage des te hoger de total errors- en standard deviation. Dit is omdat de code, bij een hoge precision, meer stappen van het gezichtsherkenning proces doorloopt, en hierdoor meer mogelijkheden voor fouten optreden.

Snelheid

Voor de kernels zijn er geen standaard kernel libraries gebruikt (bijvoorbeeld OpenCV)⁵. We hebben een eigen implementatie gemaakt voor de kernels waardoor de total runtime niet genoeg is geoptimaliseerd, hierdoor krijgen we langzamere runtimes dan mogelijk met dezelfde kernel. We kunnen wel opmerken dat de 3x3 kernels sneller zijn dan de 9x9 kernels vanwege het aantal berekeningen dat ze moeten doen. Een 3x3 kernel moet in totaal $562.500 = (250 \times 2)(3 \times 2)$ berekeningen doen in tegenstelling tot de 9x9 kernel $5.062.500 = (250 \times 2)(9 \times 2)$. Ook als we dubbele kernels gebruiken wordt het aantal berekeningen hoger. Voor 3x3 wordt het $1.187.000 = (250 \times 2)(3 \times 2) \times 2 + 250 \times 2$ berekeningen en 9x9 wordt het $10.187.500 = (250 \times 2)(9 \times 2) \times 2 + 250 \times 2$ berekeningen.

Evaluatie

Hypothese/conclusie

We hadden verwacht dat de combinatie kernels 3x3 het hoogste slagingspercentage hadden wat uiteindelijk dus ook bleek bij de sobelX + sobelY 3x3 kernel. Echter zijn de andere combinatie kernels slechter in het herkennen dan de default implementatie. Ook zijn alle combinatie kernels langzamer dan de default implementatie. Dit was niet volgens verwachtingen, want we verwachtte dat alle 3x3 kernels sneller zouden zijn dan 9x9 kernels.

Meetonzekerheden

Aangezien de bestaande code in een visual studio project staat betekent dit dat de code alleen op windows werkt (zonder het project flink aan te passen) hierdoor moesten alle testen op windows computers worden gedaan. Door de huidige staat van windows 10 (veel achtergrondprocessen, updates die in de achtergrond worden gedownload en windows defender die elke paar minuten automatisch aanspringt(en enorm veel cpu gebruikt)) is het lastig om gelijke tests uit te voeren.

Verder zitten in het geobfusceerde gedeelte van de code een aantal bugs waardoor, ten eerste, het programma soms in een oneindige loop vast komt te zitten (dit heeft redelijke impact op de standaarddeviatie van de runtime). Dit hebben we opgelost door een timeout op het programma te zetten zodat er nooit meer dan één seconde aan zo een bug wordt besteed. En ten tweede wordt ergens een ROI (region of interest) geselecteerd buiten de afbeelding. Hierdoor crasht het programma. Dit gebeurt echter alleen als er al een aantal lokalisatie stappen zijn voltooid (en er dus een goede kans is dat er een gezicht wordt gevonden), en kan dus mogelijke true positives laten crashen.