Implementatieplan edge detection

Gemaakt door:
Jasper de Winther
Tony van der Krogt

Doel

Op dit moment wordt, in de gezichtsherkennings-applicatie, de Laplacian operator gebruikt om edges te berekenen. Wij gaan onderzoeken of het gebruik van een Sobel of Prewitt operator de gezichtsherkenning kan verbeteren. Ook gaan we onderzoeken of de Laplacian operator beter werkt als ook de diagonale mee worden genomen in de berekening. Het laatste wat wij gaan onderzoeken is, de impact van de grootte van de kernel op het uiteindelijke slagingspercentage van de gezichtsherkenning.

Methode:

Bestaande pipeline

De bestaande applicatie heeft een pipeline waar afbeeldingen doorheen gaan. Het is belangrijk om te weten dat een verandering in een van de stappen het resultaat van alle daarop volgende stappen verandert. Hierdoor is het belangrijk dat elke stap zo goed mogelijk werkt. Tijdens dit onderzoek focussen wij op de edge detection stap. Dit gebeurt tijdens de pre-processingen het resultaat van deze stap wordt gebruikt om alle gezichts features te berekenen. De volledige pipeline bestaat uit meerdere stappen en substappen die hieronder beschreven staan:

Pre-processing

Tijdens deze stap wordt de afbeelding eerst van kleur naar een grijze afbeelding verandert. Daarna wordt de afbeelding geschaald tot hij ongeveer 40.000 pixels bevat. Dan worden door middel van kernels (zie kopje Kernels) edges gedetecteerd, en als laatste wordt door middel van thresholding een binaire afbeelding (2D array met true/false waarden) gemaakt die door de volgende stappen wordt gebruikt.

Localization

Tijdens deze stap worden een aantal features een voor een herkend. Eerst de boven en zijkanten van het hoofd, dan de neus, mond en kin. Daarna de contouren van de kin, de neusvleugels en zijkanten van het hoofd rond de hoogte van de neus. En als laatste de ogen.

Extraction

Tijdens deze stap worden worden de locaties die tijdens de lokalisatie stap zijn gemaakt vooral verfijnd. De exacte locatie van de ogen, neusgaten en mondhoeken worden bepaald.

Kernels

In de wereld van computer vision is een kernel een 2D matrix die een oneven aantal pixels hoog en breed is. Deze kernel wordt dan op elke pixel van de afbeelding gelegd, de pixelwaarden van de afbeelding worden dan bij elkaar opgeteld zoals in de kernel beschreven, en een nieuwe pixelwaarde wordt berekend. Hierdoor ontstaat een nieuwe afbeelding. Er bestaan veel verschillende kernels, bijvoorbeeld om een afbeelding te vervagen, visual scherper te maken en om edges te berekenen.

Sobel edge detection

De kernel die voor Sobel edge detection wordt gebruikt is een blur en een filter tegelijk die maar voor een richting werkt, horizontaal of verticaal. En ziet er als volgt uit:

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

Laplace

De kernel die voor Laplacian edge detection wordt gebruikt bestaat in twee versies. Een versie die kijkt naar de horizontale en verticale omliggende pixels, en een versie die neemt ook de diagonale pixels mee. De Laplace kernel ziet er als volgt uit:

0	-1	0
-1	4	-1
0	-1	0

The laplacian operator

-1	-1	-1
-1	8	-1
-1	-1	-1

The laplacian operator (include diagonals)

Prewitt

De kernel die voor Prewitt edge detection wordt gebruikt is hetzelfde als Sobel kernel, maar dan zonder de blur. De Prewitt kernel ziet er als volgt uit:

-1	0	1
-1	0	1
-1	0	1

h_x

-1	-1	-1
0	0	0
1	1	1

h_y

Edge detection

Tijdens het berekenen van edges wordt een kernel gebruikt waarbij een groot verschil tussen naast elkaar liggende pixels de nieuwe pixel lichter maakt. En op het moment dat

naast elkaar liggende pixels exact hetzelfde zijn de nieuwe pixel zwart wordt. Hierdoor worden vlakken van dezelfde kleur zwart en op randen van objecten de kleur wit.

Edge detectie van zwart-wit afbeelding d.m.v. de Sobel operator

Keuze

We hebben gekozen om Sobel en Prewitt te testen omdat dit kernels zijn die of horizontaal werken, of vertikaal, in tegenstelling tot de Laplacian operator. Hierdoor kunnen we vergelijken of er een verschil is tussen de horizontale en verticale implementatie, en of de Laplacian operator (door het gebruik van beide assen) sneller of preciezer is. Ook gaan we testen hoeveel impact de grootte van de kernel heeft op de snelheid en robuustheid van het programma.

Implementatie

De algoritmen worden in c++ geschreven door middel van de al bestaande code (https://github.com/arnokamphuis/HU-Vision-1819-Base) die door Arno Kamphuis is geschreven. Wij vervangen dan het stuk code dat verantwoordelijk is voor de edge detection en maken hier verschillende implementaties voor (Sobel, Prewitt, Laplacian met diagonalen, en verschillende kernel grootten). Deze implementaties worden dan getest op snelheid door middel van de chrono library (steady clock) van de c++ standard library. Ook worden de implementaties getest op robuustheid door middel van het testen van verschillende inputs en dan te kijken naar het percentage waarbij gezichten worden herkend. Om al deze implementaties meerdere keren automatisch te testen wordt een python script geschreven die het gezichtsherkennings programma aan roept en alle data opslaat, om aan het einde de resultaten te berekenen.

Evaluatie

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 herkent (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.

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 qualiteit). 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 verwijdert. Ook hebben we deze afbeeldingen geschaald naar 250 bij 250 pixels zodat ze beter vergelijkbaar zijn met de LFW Face Database.