1. Implementatieplan Image Shell

1.1. Namen en datum

Jeroen Steendam (1607288)
Timon van den Brink (1664554)
27/4/2016

1.2. Doel

Het doel van deze implementatie is het maken van een image shell class. Deze class heeft als doel om de afbeeldingen op te slaan en de gebruiker gemakkelijk toegang te geven tot de data uit de afbeelding.
Image shells zijn belangrijk voor de rest van het systeem. Als de ImageShell traag is, zal de rest van het face recognition systeem ook traag zijn.

Er wordt een implementatie geschreven voor zowel kleur afbeeldingen als de intensiteit plaatjes. De implementatie zal pixels uit een afbeelding kunnen opslaan, hetzij een 2D grid van pixels of een enkele rij van pixels. Belangrijk is dat de implementatie makkelijk te gebruiken is en veilig en efficient werkt.

1.3. Methoden

Er zijn twee methodes voor het implementeren van een image shell.

Datatype
Onze lijsttype, uitgelegt hieronder, worden gebruikt om RGB en Intesity Datatypen op te slaan. Deze worden ook gebruikt in de Default implementaties.

Lijsttype
Er zijn verschillende methoden om een lijst op te kunnen slaan. Voor onze implementatie zal een pixel lijst moeten worden opgeslagen. Er kan bijvoorbeeld gebruikt gemaakt worden van een std::array, een std::vector of een raw pointer array, met ieder zijn voor- en nadelen.

std::array
Een std::array kan worden gebruikt om de pixels opslaan, een std::array is niet resizeable. De classe std::array brengt een hoop functionaliteit met zich mee zoals std::sort en std::reverse_copy. Deze functionaliteit is voor onze implementatie overbodig. Deze functionaliteit is een wrapper om de C-style array (raw pointer array). Een voordeel van

std::array is dat deze eenvoudig te gebruiken is en geen gevaar ontstaat van segmentation faults, ook is deze sneller dan de std::vector.

std::vector
Bij het gebruik van een std:vector, wordt een std::vector<pixel> aangemaakt. Een van de functionaliteten van std::vector is dat deze resizeable is. Hierdoor is de width en de height van de afbeelding in deze std::vector gemakkelijk aan te passen. Nadelig is dat het lezen en schrijven in een std::vector minder snel is dan het lezen en schrijven in een array.

Raw pointer array
Een raw pointer array is de snelste oplossing. Maar aan een raw pointer array zitten vele risico's verbonden. Een raw pointer array kan bijvoorbeeld niet worden geresized en is erg fout gevoelig. Hierdoor kunnen segmentation faults ontstaan waardoor het programma crashed en het gewenste resultaat niet kan worden bereikt.

1.4. Keuze

Er is gekozen om de raw pointer te implementeren. Er moet wel goed opgelet worden tijdens de implementatie om memory leaks te voorkomen. Daarmee kost het wel meer tijd om te implementeren maar is het ook sneller dan de alternatieven.

1.5. Implementatie

De ImageShell klassen RGBImageStudent en IntensityImageStudent zullen worden geimplementeerd met gebruik van een `RGB* pixels;` en een `Intensity* pixels;`.
Er is hiervoor gekozen omdat de raw pointer oplossing de meest snelle oplossing is. Maar omdat een raw pointer niet kan resizen moet er voor gezorgd worden dat er geen memory leaks kunnen ontstaan. Als er nog geen array is dan zal pixels de waarde nullptr hebben.
Als de copy constructor of de width height constructor wordt aangeroepen dan wordt er een array gemaakt. En als de set methode aangeroepen wordt wordt de array eerst verwijderd voordat er een nieuwe wordt gemaakt. Mocht de set methode aangeroepen worden met de zelfde width en height dan wordt er niets gedaan.

1.6. Evaluatie

Omdat de snelheid en het geheugen gebruik essentieel is voor de ImageShell klassen wordt er in het meetrapport twee implementaties getest, de raw pointer en de std:vector. Beide zullen getest op geheugen gebruikt, processor gebruik en snelheid.