KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculteit
Ingenieurswetenschappen

Segmenteren van Skeletale Spiercellen in Histologische Beelden met behulp van Machinaal Leren en Beeldverwerking

Thomas Janssens

Thesis voorgedragen tot het
behalen van de graad van Master
in de ingenieurswetenschappen:
computerwetenschappen, optieAI

Promotor:
Prof. Luc De Raedt

Assessor:
(Onbekend)

Begeleiders:
ir. Laura Antanas
dr. Fabian Guiza Grandas

Academiejaar 2010 – 2011

Voorwoord

Ik wil mijn promotor en begeleiders bedanken voor hun goede raad bij het opvolgen van mijn werk en het beantwoorden van al mijn vragen. Verder ben ik mijn nalezers zeer veel dank verschuldigd voor hun goed werk. Ik wil mijn vriendin, familie en vrienden ook bedanken voor de alle steun. Tenslotte wil ik nog de onderzoeksgroep Intensive Care Medicine van het Universitair Ziekenhuis Gasthuisberg bedanken voor het gebruik van hun afbeeldingen.

Thomas Janssens

Inhoudsopgave

Samenvatting

Histologische beelden van skeletale spiercellen bevatten veel informatie over de effecten van ziektes en aandoeningen die dit weefsel aantasten. Deze afbeeldingen zijn dan ook erg interessant voor onderzoekers. Om deze informatie te bekomen dient de locatie van de individuele cellen in de afbeelding gekend te zijn (segmentatie van de afbeelding), zodat er eigenschappen van gemeten kunnen worden door software-tools. Dit opmeten gebeurt echter volledig manueel, wat een zeer tijdrovend proces is. Bestaande software-tools gericht op segmentatie zijn ontoereikend om dit soort beelden te segmenteren. In dit werk introduceren we een nieuw algoritme speciaal ontwikkeld om dit soort afbeeldingen te segmenteren. We gebruiken hiervoor een aantal technieken voor beeldverwerking en het automatisch leren van een model. We segmenteren de afbeelding eerst in een aantal segmenten. Dit kunnen individuele cellen, ander weefsel of opeenhopingen van cellen zijn. We gebruiken een getraind SVM-model om deze drie klassen te onderscheiden, waarna we de opeenhopingen met een recursieve aanpak opsplitsen in de individuele cellen. We tonen aan dat onze methode beduidend beter presteert dan de beste bestaande software-tools.

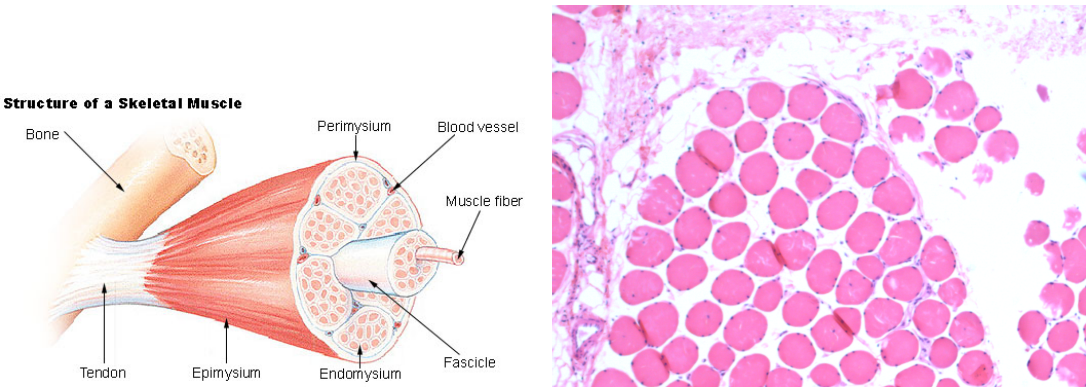
Hoofdstuk 1

Introductie

Histologische afbeeldingen van skeletale spiercellen bevatten waardevolle informatie over het effect van een ziekte op cellulair niveau, zowel bij het inschatten van de effecten van een ziekte of het herstelproces van een patiënt. Deze beelden worden bekomen door kleurenfoto's in hoge resoluties te nemen van weefselmonsters uit de spierbundel [10][15][18][23]. Een dunne dwarsdoorsnede van de spierbundel wordt onder de microscoop belicht en gefotografeerd (figuur 1.1). De toestand van de spiercellen is onderhevig aan de effecten van sommige ziektes, zoals bijvoorbeeld HIV[10], en aan de mate waarin ze gebruikt worden. Ten gevolge van een operatie kan er bijvoorbeeld spierhypertrofie of atrofie optreden. [23].

De effecten op de cellen kunnen zeer uiteenlopend zijn: de oppervlakte, omtrek en doorsnede van cellen zijn belangrijke indicatoren voor gebruik, aanwezigheid van gaten of scheuren in de cellen, het lekken van de celkernen uit de cellen, het opeenhopen van cellen, de algemene achteruitgang en uiteenvallen van cellen kunnen indicatoren zijn voor ziekten. Het verband tussen de toestand van de cellen en de ziektes is het onderwerp van veel huidig onderzoek. Vanwege de verscheidenheid van oorzaken gebeurt het bestuderen van deze cellen in een heel aantal verschillende contexten. Allen houden ze echter in dat deze histologische beelden geanalyseerd moeten worden teneinde bepaalde informatie over de cellen te winnen.

Er zijn een heel aantal metrieken die deze informatie kunnen opleveren. Zoals gezegd is de oppervlakte van de spiercellen belangrijk, alsook de omtrek, vorm, aanwezigheid van scheuren of gaten, de onderlinge nabijheid van cellen en het aantal celkernen in de cel. Om dit alles voor elke cel op te meten moeten we echter weten waar de cel zich precies bevindt en wat diens grenzen zijn. Momenteel wordt de omtrek van de cellen vaak manueel geannoteerd door een expert. Het evaluatieproces bestaat bijvoorbeeld uit het "opruimen" van de afbeelding met beeldverwerkingssoftware om daarna de cellen te scheiden met een threshold-operatie en de resultaten daarna te verfijnen. Een andere gebruikte aanpak is het gewoon manueel omcirkelen van alle cellen [18]. Er zijn methodes die semi-manueel werken [10], maar ook hier dient de gebruiker een drempelwaarde in te stellen voor elke betrokken afbeelding.



(a) Skeletale spiercellen in het lichaam. Figuur naar [20].

(b) Voorbeeld van een (gedeeltelijk weergegeven) histologische doorsnede van een spiercel.

Figuur 1.1: Illustratie van het probleemdomein

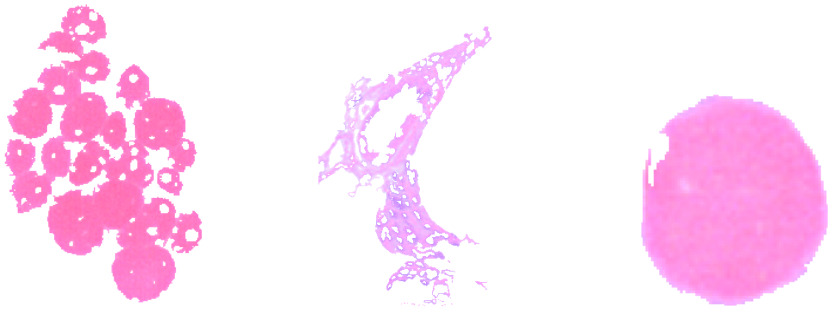
Deze manier van werken is vrij tijdrovend en inherent subjectief. Elke expert heeft een zeker niveau van kennis nodig om de afbeelding te segmenteren en geen twee experts segmenteren een afbeelding op exact dezelfde manier. Ook kan er variabiliteit zijn binnen de annotaties van dezelfde expert over langere periodes.

Deze problemen tonen duidelijk aan dat er verbetering mogelijk is. Een volledig automatische oplossing kan zowel sneller als objectiever zijn dan manuele evaluatie. Het doel van dit onderzoek is om histologische beelden van skeletale spiercellen volledig automatisch te segmenteren, zodanig dat we een volledige segmentatie bekomen waar de individuele cellen afgezonderd zijn. Een belangrijk aspect van deze oplossing is de mogelijkheid om grote hoeveelheden afbeeldingen te kunnen segmenteren zonder dat de gebruiker tussentijds manueel parameters hoeft aan te passen.

Er zijn momenteel enkele state-of-the-art-technieken en -pakketten beschikbaar die bedoeld zijn om celbeelden te segmenteren, waarvan we er later drie bespreken. We zullen zien dat hun segmentatie te wensen overlaat wat betreft dit soort beelden en er dus ruimte voor verbetering is. Kort kunnen we vertellen dat opeenhopingen van cellen zelden correct gesegmenteerd worden en het ander weefsel niet altijd correct verwijderd wordt. Ook zijn de parameters niet altijd flexibel genoeg met het oog op *batch processing*, het verwerken van grote aantallen afbeeldingen in één keer.

Om onze methode te testen vergelijken we een set afbeeldingen met zowel manuele als state-of-the-art segmentatie. Ons doel is een verbetering op de resultaten van deze laatste te boeken. We gebruiken hiervoor twee pakketen: *ImageJ* [1] en de *Weka Machine Learning Toolkit* [9][11]. Deze eerste is een pakket gericht op beeldverwerking en bevat enkele nuttige datastructuren en algoritmes hieraan gerelateerd. Het tweede bevat een heel aantal Machine Learning-algoritmes die we gebruiken om de cellen te identificeren. Meer uitleg hierover volgt in de gerelateerde hoofdstukken.

Er zijn verschillende uitdagingen bij het segmenteren van dit soort histologische afbeeldingen. Naast spiercellen is er ook ander weefsel aanwezig (*debris*). Dit is irrelevant voor deze toepassing en dient verwijderd te worden, waarbij geen verwarring met cellen mag optreden. Cellen kunnen onderling ook zeer dicht op elkaar liggen (zogenaamde *clumps* van cellen). Deze opeenhopingen moeten als dusdanig geïdentificeerd en succesvol gesplitst worden tot individuele cellen. Deze verschillende elementen in de afbeelding worden opgesomd en geïllustreerd in 1.2.



(a) Clump (b) Debris (c) Cel

FIGUUR 1.2: Elementen aanwezig in een typische afbeelding.

Om dit probleem aan te pakken bespreken we een methode die een aantal technieken uit beeldverwerking en Machine Learning combineert. Eerst voeren we een initiële segmentatie uit met Color Thresholding, een eenvoudige beeldverwerkingstechniek, om een ruwe segmentatie te bekomen. Vervolgens trainen we een model om de drie verschillende soorten segmenten te onderscheiden. Tenslotte gebruiken we een verzameling bestaande algoritmes en eigen inzichten om gevonden clumps op te splitsen en de individuele cellen te vinden. Na uitvoering van onze methode blijven enkel de gesegmenteerde cellen over. Onze bijdrage bestaat dus uit zowel het verzamelen van bestaande technieken en deze aanpassen naar onze specifieke noden als het bedenken van eigen (delen van) algoritmes om bepaalde sub-pproblemen op te lossen. We brengen deze algemene, losstaande technieken dus samen in een raamwerk dat een software-tool vormt voor het volledig automatisch segmenteren van histologische spiercelbeelden.

We bespreken eerst de bestaande state-of-the-art segmentatiepakketten en verwant onderzoek, waarna we ingaan op onze methode zelf. We schetsen eerst in grote lijnen de werking, waarna we ingaan op de specifieke onderdelen: de initiële segmentatie, de classificatie en het splitsalgoritme. Tenslotte vergelijken we experimenteel onze methode met zowel de state-of-the-art-resultaten als de manuele annotatie en bespreken deze resultaten.

Hoofdstuk 2

Verwant werk

2.1 Literatuur

Er zijn een aantal heel algemene beeldsegmentatietechnieken die op dit probleem toegepast kunnen worden, zoals besproken in [17] en [31]. Thresholding bijvoorbeeld creëert een binair masker (alle pixels 0 of 1) uit een afbeelding. De waarde van een pixel in dit masker is afhankelijk van de waarde van een eigenschap van de originele pixel en de drempelwaarde gehanteerd bij de operatie. De waarde is vaak de intensiteit, maar andere waarden kunnen ook gebruikt worden. Alle pixels met een waarde groter dan de drempelwaarde worden op 1 gezet, de andere op 0. Dit kan gebruikt worden om segmenten uit een afbeelding te halen: de aaneengesloten gebieden positieve (of negatieve) pixels vormen dan de segmenten. Thresholding met meerdere parameters (bv. kleur) is ook mogelijk. Thresholding kan gebeuren met een globale of lokale threshold-waarde . In dit laatste geval wordt er telkens gekeken naar een lokaal gebied van pixels om de threshold-waarde voor een bepaalde pixel te bepalen.

Nog een algemeen gebruikte methode is region growing: er worden een paar initiële "seed points" gekozen op een smoothed (afgevlakte) afbeelding. Deze seed points zijn punten waarvan de klasse met zeer hoge waarschijnlijkheid gekend is. Aangrenzende pixels worden iteratief toegevoegd aan de regio indien ze aan een bepaalde voorwaarde voldoen (bijvoorbeeld een intensiteit hebben die dicht genoeg ligt bij de gemiddelde waarde van de regio). Hoe meer er geïtereerd wordt, hoe moeilijker de tests worden, tot er geen pixels meer kunnen worden toegevoegd.

Wanneer er kleurgebaseerde segmentatietechnieken worden toegepast is de gebruikte kleurruimte zeer belangrijk. Het meest eenvoudige kleurmodel, RGB (Red Green Blue), komt niet goed overeen met hoe mensen kleuren waarnemen en is niet optimaal om een threshold op te baseren. Bekende alternatieven zijn HSI (hue, saturation, intensity), HSL (hue, saturation, luminance), HSD (hue, saturation, density) en Lab.

Een alternatief voor threshold-gebaseerde segmentatiemethodes wordt gepresenteerd in [2]. Hier nemen de auteurs het afgevlakt histogram van de afbeelding en interpreteren

elke piek van de afbeelding als een fuzzy klasse. Voor elke pixel wordt het lidmaatschap tot deze klasse dan bepaald als een functie van de locatie van de waarde van die pixel in het histogram en de locatie van de respectievelijke klasse-pieken. Vervolgens passen ze probabilistische relaxatie toe: de lidmaatschapswaarde van elke pixel wordt een functie van zijn eigen waarde en die van zijn buren. Zo is het waarschijnlijker dat groepen pixels tot dezelfde klasse behoren.

Nog een andere aanpak heet "Perceptual Grouping", gepresenteerd in [26]. Daar gebruiken ze een laag-niveau model van het menselijk visueel systeem om spiercellen te segmenteren, met indrukwekkende resultaten. Het nadeel aan deze aanpak is dat het (relatief tegenover de andere methodes) een beetje een "black box" is; het is moeilijk te achterhalen hoe het model precies werkt.

In [27] wordt domeinkennis over het samen voorkomen van afbeeldingskenmerken vertaald in een verzameling van symbolische regels. Het gebruik van deze regels staat een algoritme toe om bijvoorbeeld eerst afzonderlijke objecten te detecteren, bijvoorbeeld cellen en celkernen, en dan via hun spatiële relatie het aanwezig zijn van bepaalde kenmerken op hoger niveau te detecteren, bijvoorbeeld patronen van voorkomen die wijzen op tumoren.

Er wordt een Markov Random Field Bayesiaanse schattingsaanpak gebruikt in [28] om celstructuren waar kanker in aanwezig is te segmenteren. Pixels van elke klasse worden bemonsterd. Verolgens formuleren ze een MAP (Maximum a proiri) en energiefunctie voor de labeling. De schatting van de optimale labeling wordt vervolgens berekend via energieminimalisatie met relaxatiemethoden.

De methode beschreven in [32] gebruikt een "bag of" bayesiaanse classifiers om een microscopisch beeld te segmenteren. Eerst berekenen de auteurs een lokaal histogram voor een paar duizend willekeurig bemonsterde stukjes uit de afbeelding. Deze worden vervolgens in een gelijkenismatrix gestopt waarop clustering wordt uitgevoerd zodanig dat er enkele grote clusters overblijven met onderling gelijkaardige subafbeeldingen. Voor elk van deze clusters wordt een cel- en achtergrondhistogram berekend. Het onderscheid tussen de twee is hier bekend vanwege het gebruik van een geannoteerde trainingsafbeelding. De lokale bayesiaanse classifiers worden dan afgeleid van deze histogrammen. In een nieuwe afbeelding wordt elke pixel geklasseerd door een gewogen combinatie van deze classifiers.

In [7] worden morfologische operators (erosie, dilatatie,...) gecombineerd met machine learning-technieken om cellen te segmenteren en te klasseren. De eerste methoden voert de segmentatiestap uit en vervolgens wordt een Support Vector Machine getraind op een set van features om twee soorten cellen te onderscheiden.

[4] geeft een samenvatting van de voornaamste algoritmes en technieken in histologsiche afbeeldingen. Technieken betreffende onder andere kleur, textuur, architectuur en morfologie worden besproken.

In [22] wordt een grote variëteit aan cellen gesegmenteerd met een aantal technieken. Interessant voor ons is een bepaald algoritme dat een laag-energiepad tussen twee pixels

zoekt om zodoende de grens van een cel te volgen.

2.2 State-of-the-art tools

We hebben de segmentatieprestaties bekeken van enkele "state-of-the-art" tools gericht op de medische gebruiker: CellProfiler, CellTracer en CellTracker. Deze pakketten worden in de praktijk gebruikt voor taken gelijkaardig aan de onze. Geen van hen is echter specifiek gemaakt voor histologische beelden van dit soort cellen. We evalueren deze tools om twee redenen. Ten eerste willen we weten wat de beste prestaties momenteel zijn, zodat we onze methode er later mee kunnen vergelijken. Ten tweede willen we kijken of de resultaten van deze tools misschien te verbeteren valt.

2.2.1 CellProfiler

CellProfiler [5] laat de gebruiker modules selecteren om aaneen te schakelen. Deze modules kunnen bijvoorbeeld segmentatiealgoritmes zijn, laag-niveau beeldverwerkingstechnieken of gericht zijn op invoer en uitvoer. Het programma werd ontwikkeld in 2005 en wordt nog steeds onderhouden en gebruikt. Gebruikers configureren een aaneenschakeling van modules met elk hun eigen parameters en voeren deze uit op een batch van afbeeldingen. De instelbare parameters passen zich niet aan aan de specifieke afbeelding. Zo moet de gebruiker minimum en maximum grootte van de cellen instellen, alsook het percentage van de afbeelding dat cellen bevat. Voor afbeeldingen met variërende bedekking en zoom-niveaus is dat natuurlijk problematisch.

Toch presteert CellProfiler goed zolang de grenzen tussen de cellen niet vervagen. Is dit wel het geval, dan lijkt de splitsing van deze cellen vrij willekeurig. Zie bijvoorbeeld figuur 2.1 of 8.1.

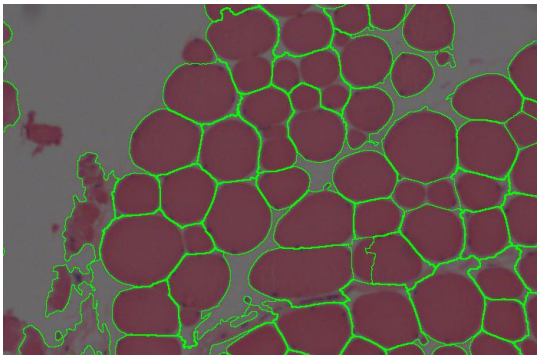
2.2.2 CellTracker

CellTracker [25] is een software-tool bedoeld om automatisch de afmetingen en locatie van cellen te berekenen met oog op de aanwezigheid van proteïnes. Het combineert geautomatiseerde cel-traceermethodes met beeldverwerkingsalgoritmes. Bij het testen op een applicatie in een bepaald domein (nucleaire transcriptiefactoren) presteerde het programma even accuraat als een menselijke expert en was het twintig keer zo snel.

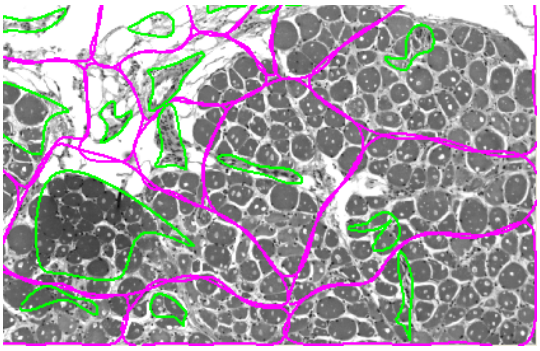
Zoals te zien is in de figuur presteert CellTracker (na verschillende pogingen) zeer slecht op de gegeven beelden. Parameters verfijnen leek hier niet te helpen. We kunnen alleen maar concluderen dat het programma beperkt is tot een toepassingsgebied waar ons probleem niet in past. Als we de resultaten en parameters bekijken lijkt het eerder dat het programma relatief grote cellen verwacht met één enkele celkern. Onze histologische afbeeldingen bevatten echter veel verschillende skeletale spiercellen, met elk meerdere kernen. We kunnen dus concluderen dat CellTracker niet geschikt is voor deze toepassing.

2.2.3 CellTracer

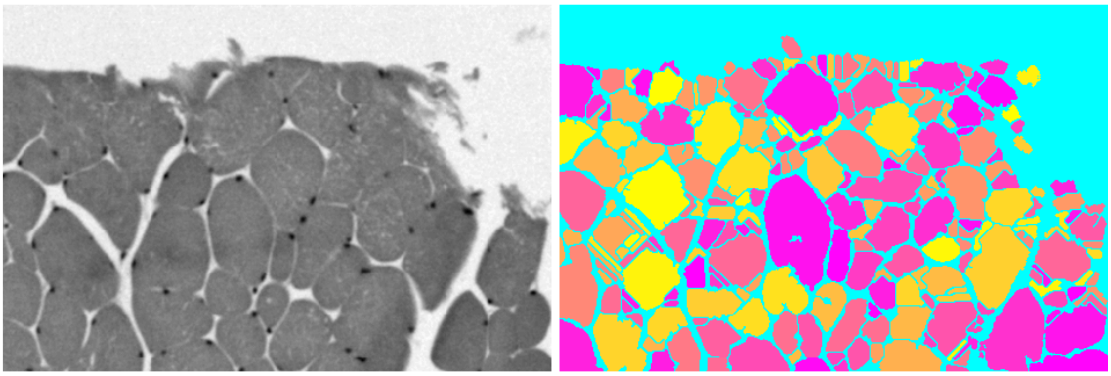
CellTracer [29] is eigenlijk vooral gericht op het segmenteren en volgen van cellen in een reeks opeenvolgende beelden. De software blijkt echter relatief goed te werken in onze context. De segmentatie van duidelijk losstaande cellen is grotendeels correct. Bij het segmenteren van clumps presteert het pakket echter minder goed, zoals op de figuur te zien is. Ook hier kan het bijstellen van parameters geen over- of ondersegmentatie voorkomen.



(a) CellProfiler



(b) CellTracker



(c) CellTracer (links origineel naar grijwaarden omgezet, rechts segmentatie)

Figuur 2.1: Segementaties van de verschillende pakketten.

2.2.4 Besluit

Na nader onderzoek blijkt dat geen enkel van de pakketten een volledig correcte segmentatie oplevert. Er is vrijwel altijd sprake van over- of ondersegmentatie. Het zou ons teveel werk kosten om de prestaties van elk pakket kwantitatief te vergelijken op dezelfde schaal, maar het is duidelijk te zien dat van alle tools CellProfiler het beste presteert. Indien we de parameters expliciet zouden instellen met het oog op ondersegmentatie kunnen we deze resultaten gebruiken om een betere segmentatie te bekomen. We hebben

dit echter niet gedaan omdat het relatief eenvoudig bleek te zijn een andere methode te maken die gegarandeerd niet oversegmenteert en flexibele parameters heeft voor het verwerken van groepen verschillende afbeeldingen (zie hoofdstuk 5).

Hoofdstuk 3

Overzicht van de methode

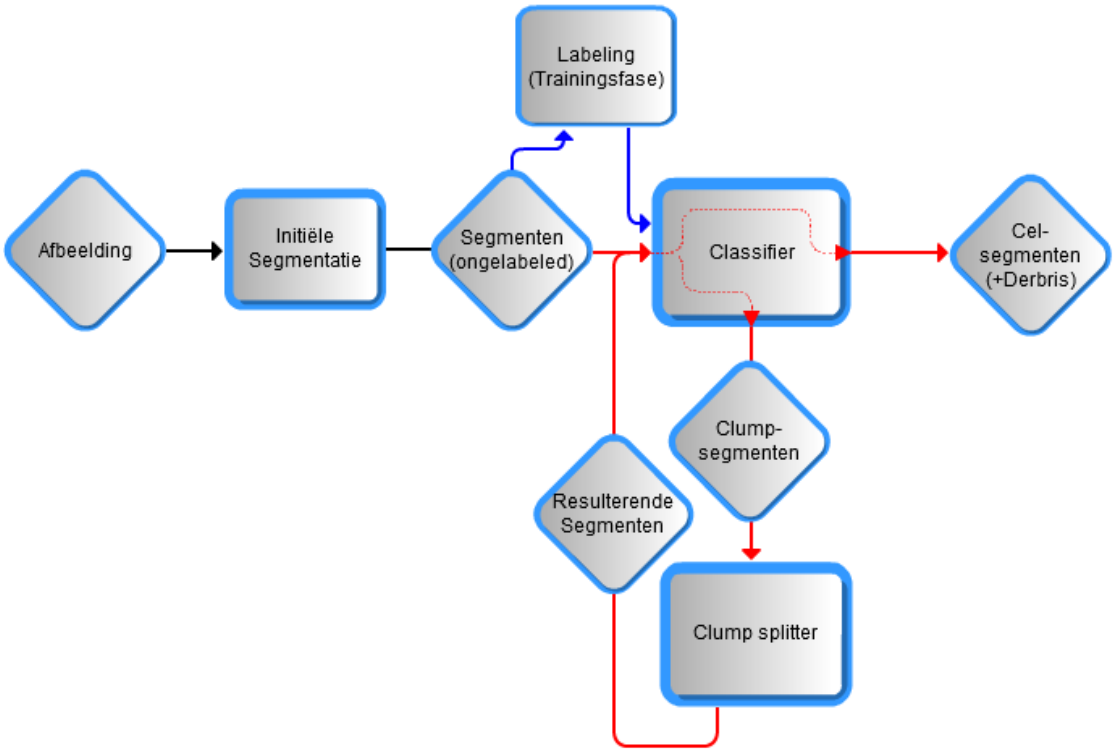
Nu de context en het verwant werk bekend zijn, schetsen we in dit hoofdstuk de elementen van onze methode. Deze valt uiteen in een aantal verschillende stappen, zoals te zien in het diagram hieronder (fig. 3.1). Voor we in de details van elke stap duiken bespreken we een overzicht van de algemene, hoog-niveau werking van onze methode.

We vertrekken van een histologische afbeelding, wat voor de huidige stap beschouwd kan worden als een raster van pixels met elk een RGB-kleurwaarde van 24 bit. Het doel van de eerste stap, de *initiële segmentatie*, is uit deze afbeelding een aantal *segmenten* te halen. We definiëren een segment als een begrensd aaneengesloten gebied van pixels. Het is de bedoeling dat deze segmenten een benadering zijn van de vorm van de cellen aanwezig in de afbeelding.

Om deze segmenten te bekomen passen we *Color Thresholding* toe , een laag-niveau beeldverwerkingstechniek, op de afbeelding. Dit houdt in dat we alle pixels met een Saturation-waarde (Saturation is een kanaal in de HSB-kleurruimte) die lager is dan een bepaalde grenswaarde verwijderen uit de afbeelding. Elk overblijvend aaneengesloten gebied van pixels beschouwen we als een enkel segment. Dit levert ons een aantal segmenten op, maar we weten nog niet of een segment bestaat uit een enkele cel, meerdere cellen (*clump*) of iets anders (*debris*).

Hierna gaan we deze gevonden segmenten klasseren volgens die drie klassen (Cel, Debris en Clump). Deze stap dient er toe restanten van ander weefsel weg te filteren en opeenhopingen van cellen te identificeren zodat deze gesplitst kunnen worden. We klasseren de segmenten aan de hand van een model dat we trainen op een reeks voorbeeldsegmenten, die we op voorhand handmatig gelabeld hebben met de juiste klasse. We identificeren een aantal *features*, eigenschappen van de segmenten die nuttig zijn bij het toewijzen van een klasse aan het segment. Deze features zijn gebaseerd op zowel eenvoudige (Oppervlakte, omtrek,...) als complexe ("Form Factor","Convexity") geometrische eigenschappen, alsook op de histogramwaarden van de pixels. Ze zijn zowel afkomstig uit de literatuur als uit eigen experimentatie.

Naast de keuze van de features is ook die van het model en diens parameters belangrijk.



Figuur 3.1: Schematisch overzicht van de methode. We beginnen met de initiële segmentatie van een afbeelding, waarna we de resulterende segmenten klasseren in drie klassen volgens een getraind model. Segmenten die het model als clump klasseert worden recursief opgesplitst tot er enkel cellen overblijven.

We hebben gekozen een Support Vector Machine, zoals geïmplementeerd in LibSVM [6] en gebruiken de Weka toolkit [9] om te experimenteren met verschillende modellen en parameters. Om deze laatste af te stellen gebruiken we het Grid Search-principe. Als trainings- en testdataset hebben we manueel gelabelde segmenten als voorbeeld nodig. We hebben hiervoor een grafische interface geschreven en hiermee enkele duizenden segmenten gelabeled. We stelden onze parameters af met kruisvalidatie op de trainingsset, waarna we de prestaties op de testset onderzochten.

Als laatste dienen we de clumps op te splitsen in de individuele cellen (en eventueel het debris) waaruit ze bestaan. Hiervoor gebruiken we een recursief algoritme: voor elke geïdentificeerde clump proberen we een pad te vinden dat deze in twee splitst volgens een gevonden celgrens. De twee resulterende segmenten laten we klasseren door ons model. Indien een segment weer een clump is splitsen we dit verder op, en zo verder. Indien we een cel uitkomen houden we deze bij, debris negeren we. Uiteindelijk houden we zo enkel nog individuele cellen over. We combineren voor het splitsen zelf verschillende

technieken tot een nieuw algoritme. We beschrijven dit hieronder kort. Het is belangrijk te onthouden dat deze stappen worden uitgevoerd voor elke interatie tijdens de recursie.

Om mogelijke begin- en eindpunten van een splitspad te vinden zoeken we voor elke concaviteit in het segment (stukken omtrek die niet overeen komen met de omtrek van de convex omhullende) het diepste punt, dit noemen we het *concaviteitspunt*. We beschouwen alle mogelijke koppels concaviteitspunten in een clump en gaan een aantal metrieken na [16]. Zo wordt gekeken naar de afstand tussen de punten in vergelijking met hun diepte (*Saliency*), in hoeverre ze naar elkaar gericht staan (*Concavity-Concavity Alignment* en *Concavity-Line Alignment*). We berekenen uit deze metrieken een score, waarna we de koppels volgens deze score rangschikken.

Voor de beste drie koppels berekenen we een splitspad tussen beide punten. Dit is concreet een aangepaste versie van het A*-algoritme [12] die rekening houdt met de intensiteit van de pixels. Er wordt gebruik gemaakt van een kost per pixel in het pad in functie van de kleur. De intuïtie is dat een wittere pixel beter is dan een rozere, aangezien het eerste duidt op achtergrond en het tweede op de binnenkant van een cel.

Na uitvoering van al deze stappen hebben we een histologische afbeelding omgezet in een verzameling van celsegementen. Hierna is het mogelijk metingen op de cellen uit te voeren om zo de gewenste informatie te achterhalen.

We geven het hoog-niveau algoritme in pseudocode.

Algorithm 1 segmenteer(Afbeelding a)

```
ruweSegmenten ← initiëleSegmentatie(a)
[cellen, debris, clumps] ← klasseerMetModel(ruweSegmenten)
cellen ← cellen ∪ splitsAlle(Clumps)
return cellen
```

Hoofdstuk 4

Initiële segmentatie

We richten we ons op de eerste stap in het segmentatieproces, de initiële segmentatie. Hier zetten we de afbeelding om in een verzameling van segmenten; deze vormen een ruwe benadering van de cellen in de afbeelding.

4.1 Invoer

De invoer van dit onderdeel bestaat uit een afbeelding in het TIFF-formaat met een resolutie van 2592 op 1944 pixels. Hierop zijn zowel spiercellen als ander weefsel te zien, zoals beschreven in de inleiding.

4.2 Color Thresholding

In een grijswaarde-afbeelding heeft elke pixel één getal dat diens helderheid uitdrukt, de *intensiteit*. In een 8-bit afbeelding kan deze waarde variëren van 0 tot 256. De *Thresholding*-operatie maakt hiervan een binaire afbeelding (een afbeelding met voor alle pixels een waarde van 0 of 1). De uitvoer wordt bepaald door de *threshold*-parameter t. Dit is een drempelwaarde: pixels met intensiteit hoger dan t worden in de uitvoer op 1 gezet, pixels met intensiteit lager of gelijk aan t worden 0. (voorbeeldfiguur maken)

Color Thresholding past deze techniek toe op een kleurafbeelding. De intensiteit van een pixel in een kleurafbeelding wordt weergegeven door een drietal van kleurwaarden, afhankelijk van het gebruikte kleurmodel. Voorbeelden van kleurmodellen zijn: RGB (Red Green Blue), HSB (Hue Saturation Brightness, ook gekend als HSV), HSL of XYZ. Om de thresholding-operatie op dit soort afbeelding uit te voeren moeten we voor elk van de drie waarden in het tuple een aparte thresholding uitvoeren, met elk ook een eigen drempelwaarde t. Dit kan nog verder uitgebreid worden door zowel te thresholden op een minimum- als maximumwaarde.

We hebben als kleurruimte gekozen voor de HSB-representatie omdat deze de kleurverschillen die we willen gebruiken het best kan onderscheiden. We zien dat het Saturation-

histogram doorgaans twee grote pieken heeft, de linkse overeenkomend met de witte achtergrond, de rechtse met de binnenkant van de cellen. Daartussen ligt soms, afhankelijk van de specifieke afbeelding, een kleinere piek veroorzaakt door het lichter gekleurd niet-spierweefsel en de randen van de cellen, die iets lichter zijn dan de binnenkant. Het Brightness-histogram bevat slechts één piek, deze is niet nuttig gebleken om de verschillende onderdelen van de afbeelding te onderscheiden. Het Hue-histogram bleek te variabel tussen de verschillende afbeeldingen om echt nuttig te zijn.

In het Saturation histogram blijkt na experimentatie het lokaal minimum tussen de twee grote pieken een goede plaats te zijn om een threshold uit te voeren. Dit zorgt ervoor dat een groot deel van het niet-spierweefsel en de achtergrond verwijderd wordt, maar de cellen (grotendeels) bewaard blijven. Zie figuur 4.1 voor een illustratie.

4.3 Heuristiek

De belichting en de verdeling van cellen, achtergrond, ander weefsel en vlekken is voor elke afbeelding anders. Dit resulteert in een variatie in de histogrammen die te groot is om vaste drempelwaarden te gebruiken voor de Color Threshold-operatie. Aangezien ons doel een volledig automatische methode is, hebben we voor het bepalen van deze waarden een heuristiek bedacht.

We maken bij de heuristieken gebruik van histogrammen van de afbeelding. We bekomen dit door alle pixels met dezelfde waarde voor een eigenschap (bv. Saturation) in een *bin* te plaatsen. In deze toepassing zijn er altijd 256 verschillende waarden mogelijk en dus ook 256 bins. Visueel wordt dit weergegeven met de bins stijgend gesorteerd van links naar rechts (zie bv. figuur 4.1). Als we het hebben over het "aflopen van het histogram" bedoelen we het itereren over deze gesorteerde lijst van bins. We inspecteren dan telkens de waarde van de bin, dit is het aantal pixels dat voor de eigenschap het nummer van de bin als waarde had. Als er in een Saturation-histogram de waarde van bin 234 45 is, betekent dit dat er 45 afbeeldingen zijn met een Saturation-waarde van 234.

Oorspronkelijk hadden we voor zowel het Hue- als Saturation-histogram heuristieken om de thresholdwaarden te vinden, maar na testen op sterk variërende afbeeldingen bleken alleen de heuristieken voor Saturation robuust genoeg te zijn. Thresholds op het Hue-histogram resulteerden daarenboven ook bij goede resultaten vaak in het verwijderen van de celkernen uit de afbeelding, wat het onderscheiden van de klassen moeilijker maakte voor het model. De enige heuristieken die we overhouden bepalen boven- en ondergrenzen op het Saturation-histogram.

Deze zijn ontworpen met de hierboven vernoemde specifieke kenmerken van de histogrammen in het achterhoofd. Ze zijn vrij ruw en kunnen waarschijnlijk nog verbeterd worden, maar voor de doeleinden van dit onderzoek volstaan ze. Voor de heuristieken toegepast worden op de histogrammen laten we deze door een *median filter* verwerken. Deze filter vervangt elke waarde van een bepaald kleurniveau in het histogram door de

mediaan van die waarde en zijn twee buren. Zo wordt de ruwheid van het histogram wat weggewerkt en verdwijnen (voor ons onbelangrijke) zeer lokale minima en maxima, die de heuristiek potentieel in de war kunnen sturen.

Dit is de heuristiek om de **o**ndergrens voor de Saturation-waarde te bepalen:

1. Beginnende van rechts, loop het histogram af en onthoud de grootst geziene waarde m tot we de rechterpiek gepasseerd zijn en daarna de vanaf dan laagst geziene waarde n tot we de linkerpiek passeren.
2. Indien voor de huidige waarde v geldt: $v < 0.8m$ en $m > 5000$ dan beschouwen we de rechterpiek als gepasseerd.
3. Indien voor de huidige waarde v geldt: $10n < m$ en $0.5v > n$ dan beschouwen we v als de linkerpiek. Dan is de huidige waarde van n het lokaal minimum en dus de drempelwaarde.

Dit is de heuristiek om de **b**ovengrens voor de Saturation-waarde te bepalen:

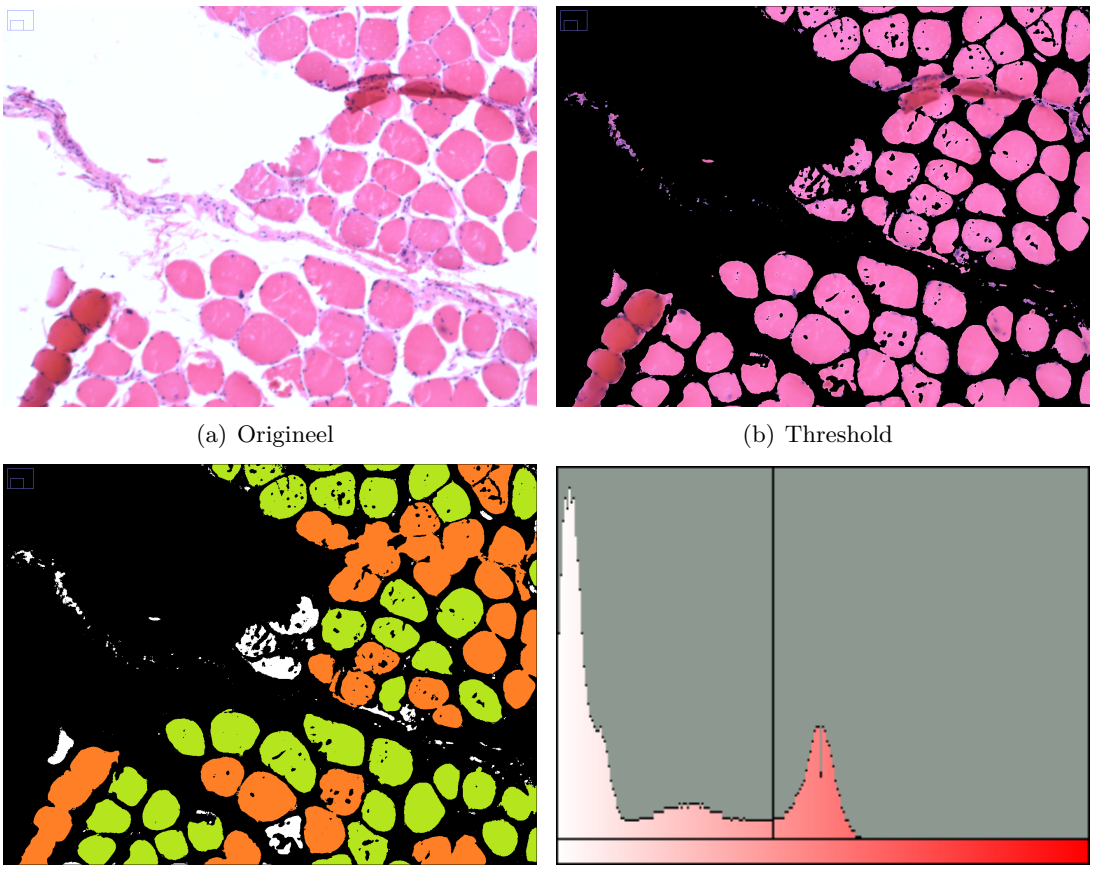
1. Beginnende van rechts, loop het histogram af en onthou de grootst geziene waarde m tot we de rechterpiek gepasseerd zijn.
2. Indien voor de huidige waarde v geldt: $v < 0.8m$ en $m > 5000$ dan beschouwen we de rechterpiek als gepasseerd.
3. Indien we de rechterpiek gepasseerd zijn beginnen we terug van rechts tot we een waarde v tegenkomen waarvoor geldt: $v > m/100$. Dan is de locatie met waarde v de drempelwaarde.

4.4 Resultaat

In figuur 4.1 is een illustratie van de operatie weergegeven. We zien dat het resultaat een degelijke schatting vormt waar verder op voortgebouwd kan worden. Deze methode op zichzelf is echter ontoereikend om een degelijke segmentatie te bekomen. Zelfs als we uitgaan van ideale heuristieken voor de drempelwaarden leert manuele experimentatie ons dat een perfecte segmentatie over de hele afbeelding onmogelijk is met een enkele thresholding-operatie. Door belichtingsverschillen doorheen de afbeelding, bijvoorbeeld vanwege variabele dikte van de “weefselplakjes”, is het onmogelijk de drempelwaarde hoog genoeg te leggen om sommige dicht opeenliggende cellen te onderscheiden zonder dat hierdoor cellen met een lichtere kleur volledig buiten de selectie vallen. Dit probleem valt mogelijk te verhelpen met een intelligentere thresholding-techniek, bijvoorbeeld één die opereert op kleine delen van de afbeelding, of een iteratief proces waarbij de drempelwaarde verhoogd wordt en er verder gewerkt wordt van het vorige resultaat.

Zelfs dit soort techniek zou echter nog moeite hebben met opeenhopingen van cellen waar de grenzen tussen de individuele cellen helemaal verdwenen zijn. Scheuren en

gaten in beschadigde cellen zouden hier namelijk nog steeds voor fouten kunnen zorgen. Tenslotte is er ook een niet-verwaarloosbare overlap in kleurwaarden tussen spierweefsel en ander weefsel. Omwille van deze fundamentele beperkingen is er nood aan een intelligentere aanpak om de afbeelding verder te segmenteren.



(a) Origineel (b) Threshold

(c) Types (d) Histogram

FIGUUR 4.1: Voorbeeld van een (manuele) color thresholding van een deel van een afbeelding. Merk in de de originele afbeelding (a) de aanwezigheid van de strook ander weefsel op en de aanwezigheid van desintegrerende en gedesintegreerde cellen. In (b) zien we wat er overblijft na thresholding van het saturation-histogram met manuele selectie van de drempelwaarde. Merk op dat er enkele vlekjes debris overblijven. In figuur (c) zijn de types segmenten gemarkeerd: clumps zijn oranje, debris is wit en cellen zijn groen. Merk op dat de overblijfselen van gedesintegreerde cellen (ongeveer centraal in de afbeelding) hier als debris gemarkeerd zijn. Dit is enigzins willekeurig aangezien onze manier van werken een harde classificatie nodig heeft en deze nuance dus niet kan modelleren. Merk ook de grote hoeveelheid clumps op, dit is door de nabijheid van de cellen onvermijdelijk. In (d) is het saturation-histogram van de afbeelding te zien met de drempelwaarde aangegeven. Dit is de meest rechtse waarde van de drempelwaarde voordat er onacceptabel veel pixels uit de cellen weggefilterd werden.

Hoofdstuk 5

Classificatie

Het resultaat van de initiële segmentatie is een verzameling segmenten. Deze kunnen overeenkomen met individuele cellen aanwezig in de afbeelding, maar ook met groeperingen van cellen (*clumps*) of restanten van ander weefsel (*debris*). Het doel van de classificatiestap is om alle in de vorige stap gevonden segmenten te identificeren als lid van één van deze drie klassen.

Er komen een aantal aspecten kijken bij het klasseren van segmenten. We belichten deze hieronder één voor één, maar eerst schetsen we het proces kort. Uit elk segment halen we een aantal zogenaamde *features*, eigenschappen van het segment numeriek voorgesteld. Voor de training-set labelen we ook elk segment met de correcte klasse. Dan slaan we voor elk segment de gevonden features samen met het klasselabel op in een *feature vector*. De verzameling van al deze vectoren dient dan als invoer voor het trainen van het model. Eens het model getraind (en getest) is kunnen we het gebruiken om ongeziene segmenten te klasseren. Uit statistieken bekomen bij het testen weten we ook ongeveer wat we prestatiegewijs kunnen verwachten van ons model.

5.1 Features

We hebben 18 features geselecteerd waarvan we denken dat ze nuttig kunnen zijn om de drie klassen van elkaar te onderscheiden. Ze vallen uiteen in een aantal klassen: eenvoudige geometrische, complexe geometrische en intensiteitgebaseerde features.

5.1.1 Eenvoudige geometrische features

Deze geometrische features zijn vrij eenvoudig en gemakkelijk af te meten aan de figuur.

Oppervlakte: Cellen zijn gewoonlijk kleiner dan clumps, aangezien deze uit meerdere cellen bestaan. Debris kan zowel veel kleiner als veel groter dan cellen zijn.

Omtrek: Clumps en debris zijn grilliger gevormd dan cellen, die relatief rond zijn en zullen dus ook een grotere omtrek hebben. Clumps zijn ook groter dan de gemiddelde cel.

Oppervlakte convex omhullende: Het verschil kan hier nog groter zijn dan in de oppervlakte, aangezien cellen zelden concaaf zijn, clumps bijna altijd wel en debris vaak ook. Van deze convex omhullende van laatste twee wordt de oppervlakte dus nog aanzienlijk groter.

Omtrek convex omhullende: Dit is wat meer bestand tegen grilligheid en geeft dus een robuustere maat voor de omtrek. Zie ook hierboven.

Grote/Kleine as van een gefitte ellips: Geeft een andere indruk van de grootte en rondheid van het segment.

5.1.2 Complexere geometrische features

Deze geometrische features zijn iets complexer en zijn voor een groot deel opgebouwd uit de eenvoudige. De meesten worden aangehaald in [7], we gebruiken hier de originele benaming. Enkel de definitie van solidity hebben we lichtjes gewijzigd.

Form Factor:

$$\frac{4 * \pi * Oppervlakte}{Omtrek^2}$$

Roundness:

$$\frac{4 * Oppervlakte}{\pi * MajorAxisLength^2}$$

Aspect Ratio:

$$\frac{LengteGroteAs}{LengteKleineAs}$$

Convexity:

$$\frac{ConvexOmhullendeOppervlakte}{ConvexOmhullendOmtrek}$$

Solidity:

$$\frac{ConvexOmhullendeOppervlakte}{Oppervlakte}$$

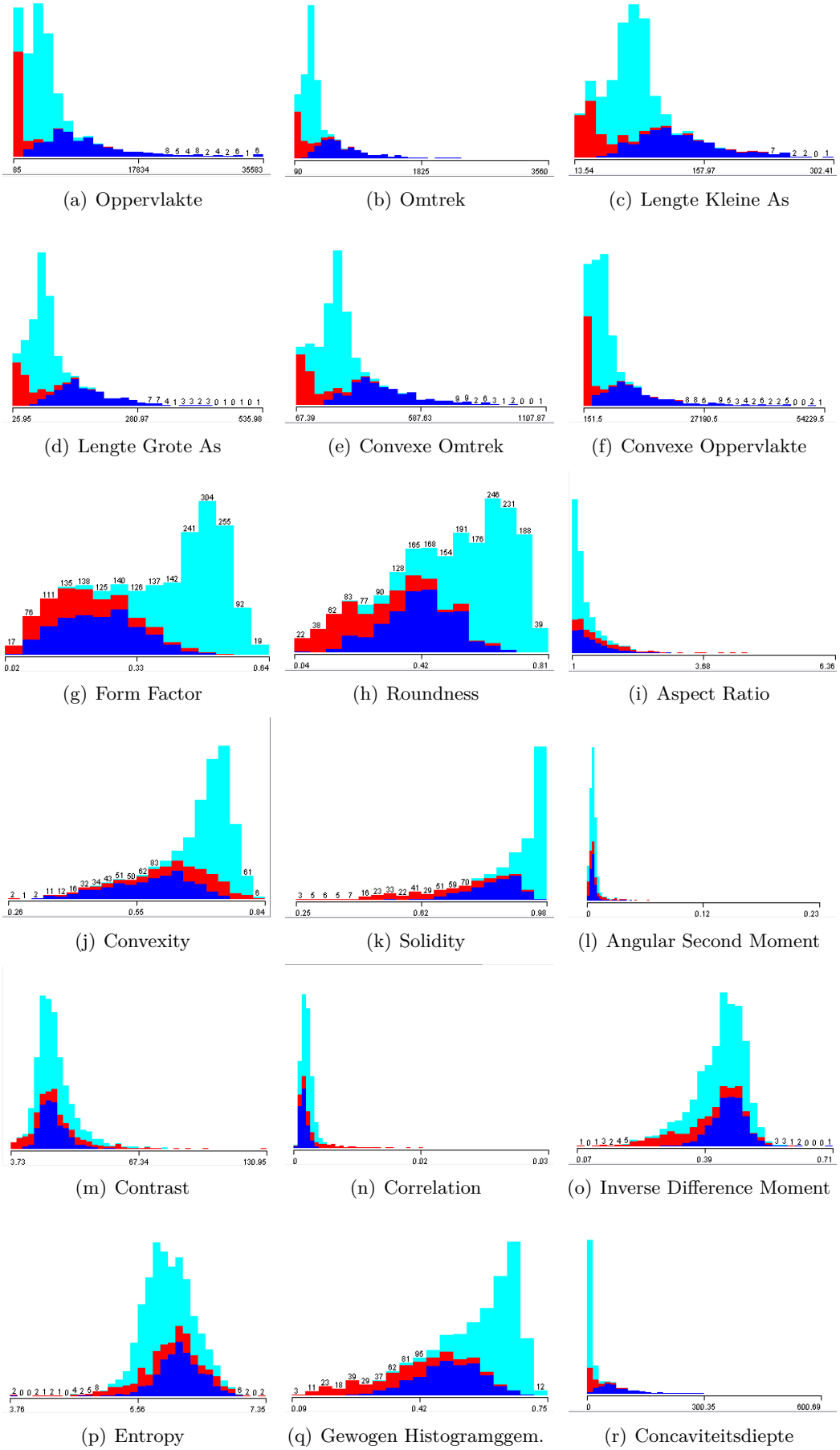
Summed Concavity Depths: Dit is de som van de dieptes van alle concaviteiten van het segment. De diepte is de grootste afstand tussen een punt op de concaviteit en het stuk van de convex omhullende dat de concaviteit overspant. Zie hoofdstuk 6 voor meer informatie.

5.1.3 Intensiteitgebaseerde features

We hebben ook een aantal features gedefiniëerd die gericht zijn op het onderscheiden van de segmenten gebaseerd op hun intensiteit. Debris leunt vaak bij lichtroze aan, terwijl cellen eerder donkerroze zijn. Clumps hebben vaak intern witte plekken waar verschillende cellen samenkomen, wat hun gemiddelde intensiteit ook iets lichter maakt. Deze eigenschappen leiden ons tot het definiëren van de "gewogen histogramgemiddelde"-feature: dit is de gemiddelde waarde van het luminositeit-histogram van het segment, gedeeld door het gemiddelde voor de hele afbeelding. Door deze deling verminderen we de rol van veranderlijke belichting op de waarde van de feature, waardoor deze robuuster wordt over een groter spectrum van afbeeldingen.

We gebruiken ook de *Grey Level Co-occurence Matrix* (GLCM) features. [3]. De GLCM wordt gebruikt om de relaties tussen koppels van pixels te bestuderen, wat ons een idee geeft over de textuur van het segment. Verder ingaan op de definitie van deze matrix en de resulterende features zou ons wat te ver leiden. We gebruikten het algoritme in [3] om de volgende features te berekenen: Angular Second Moment, Entropy, Contrast, Correlation, Inverse Difference Moment en Entropy.

Een overzicht van de distributie van de waarden uit de trainingsset voor alle features is te zien in figuur 5.1.



(a) Oppervlakte (b) Omtrek (c) Lengte Kleine As

(d) Lengte Grote As (e) Convexe Omtrek (f) Convexe Oppervlakte

(g) Form Factor (h) Roundness (i) Aspect Ratio

(j) Convexity (k) Solidity (l) Angular Second Moment

(m) Contrast (n) Correlation (o) Inverse Difference Moment

(p) Entropy (q) Gewogen Histogramggem. (r) Concaviteitsdiepte

FIGUUR 5.1: Distributies voor alle features per klasse. Clumps zijn blauw, debris rood en cellen cyaan.

5.1.4 Feature Vector

Bij het trainen en gebruiken van een model werken we met een Feature Vector. Dit is een datastructuur met de waarde van alle features van een bepaalde instantie plus het bijhorende label. Een voorbeeld voor elk type segment wordt gegeven in tabel 5.1 en figuur 5.2.

Veld	**W**aarde		
Oppervlakte	10117	740	4256
Omtrek	163	250	330
Lengte Kleine As	163	32.7	88
Lengte Grote As	182	67.7	98
Convexe Omtrek	475	145	246
Convexe Oppervlakte	13768	1188	4438
Form Factor	0.20	0.14	0.49
Roundness	0.38	0.20	0.56
Aspect Ratio	1.11	2.06	1.11
Convexity	0.60	0.57	0.74
Solidity	0.73	0.62	0.95
Angular Second Moment	0.005	0.001	0.01
Contrast	16.7	0.27	22.3
Correlation	0.001	7	0.0001
Inverse Difference Moment	0.47	0.29	0.48
Entropy	6.16	27.5	5.51
Gewogen Histogramgemiddelde	0.32	0.29	0.46
Concaviteitsdiepte	112	52	0
Klasse	Clump	Debris	Cell

TABEL 5.1: Feature Vectors voor enkele segmenten. De waardes zijn niet genormaliseerd. Zie ook figuur 5.2.

(a) Clump (b) Debris (c) Cel

FIGUUR 5.2: De segmenten horende bij de Feature Vectors in tabel 5.1

5.2 Dataset

Nu we de features hebben moeten we een database van segementen samenstellen waar we de feature-waarden samen met het label bijhouden in de feature vector. Deze database dient groot genoeg te zijn om een voldoende grote verscheidenheid van cellen, debris en clumps te demonstreren, zodat het model deze verschillen allemaal kan leren.

We hebben een dataset van 2126 gelabelde segmenten aangemaakt, afkomstig uit 11 afbeeldingen. Deze afbeeldingen zijn geselecteerd uit een reeks van 78 met het oog op zo veel mogelijk diversiteit includeren. Ze moesten ook relatief goed presteren in de initiële segmentatie, zodat er voldoende segmenten waren om te labelen. We hebben 590 segmenten gelabeld als Clump, 340 als Debris en 1196 als Cel.

Praktisch gezien moeten we een manier hebben om de dataset op te slaan. We hebben gekozen om de afbeeldingen, segmenten en labels op te slaan in een XML-file. We bepaalden hiertoe het volgende XML-schema:

```
<segmentList>
     <image name=imagex.tif>
          <segment label="clump" name="imagejsegmentname">
               <contour>
                    <point x=1 y=2>
                    ...
               </contour>
          </segment>
     </image>
     ...
</segmentList>
```

De `segmentList`-tag omvat de hele verzameling, in het niveau daaronder zijn de segmenten per afbeelding gegroepeerd. Een segment heeft een tag met zijn label (`NONE` voor ongelabelde segmenten) en zijn door ImageJ toegewezen ID (bijvoorbeeld “1547-0548”).

De data binair opslaan en oproepen door implementatie van `java.io.Serializable` is mogelijk een snellere optie, maar deze heeft als nadeel dat ze minder universeel leesbaar is. Origineel werden de labels ook handmatig ingevoerd, wat alleen mogelijk was met een door mensen leesbaar formaat.

Het labelen van de data is een tijdrovende taak. Zonder extra software is het nodig handmatig de lijst van segmenten af te gaan en de naam van elk segment op te zoeken in het XML-bestand om dan handmatig het label in te typen.

Om dit proces overbodig te maken hebben we de ImageJ API gebruikt om een geïntegreerde GUI te maken als hulpmiddel bij het labelen. In plaats van per segment de bovenstaande stappen te overlopen, wordt elk segment nu sequentieel aan de gebruiker

gepresenteerd samen met knop voor elk label. De gebruiker dient dan enkel op de knop met het juiste label dient te klikken. Verder ziet de gebruiker ook het aantal (gelabelde) segmenten en heeft hij de optie terug te gaan om fouten te corrigeren. Deze interface is ook zeer interessant voor medische professionals die relatief veel tijd spenderen aan het handmatig labelen van afbeeldingen. De GUI is te zien in figuur 5.3.

FIGUUR 5.3: De grafische gebruikersinterface om segmenten te labelen.

Het labelen van de segmenten is niet eenvoudig vanwege de vele twijfelgevallen. Zo is er de vraag of stukjes gedesintegreerde cel nog als cel tellen of debris zijn. Het verschil tussen cellen en clumps is soms evenmin duidelijk: er hangt regelmatig wat debris aan een cel, maar er is geen duidelijk punt waarop dit voldoende is om als clump geklasseerd te worden. We gaan hier later verder op in.

5.3 Model

We gebruiken de Weka-toolkit om het model aan te maken en te gebruiken. We kozen voor een Support Vecor Machine (SVM). Dit soort model is vrij populair voor het benaderen van classificatieproblemen met data afkomstig uit de echte wereld. Dit is zo omdat SVM's goed generaliseerbaar zijn (ze kunnen met hoge accuraatheid ongeziene voorbeelden klasseren) in vergelijking met veel traditionele classifiers, zoals neurale netwerken. SVM's

bieden nog enkele andere unieke voordelen. Zo vereisen ze veel minder rekenkracht (zeker in vergelijking met neurale netwerken), presteren goed in hoogdimensionale ruimtes en kunnen overweg met relatief weinig trainingsvoorbeelden. De focus van de training ligt op het minimaliseren van een schatting van de fout op de testset, niet op het minimaliseren van fouten in de trainingsset zelf. Dit voorkomt overfitting. Nog een voordeel is dat SVM's redelijk robuust zijn wat betreft ruis in de data. Er zijn in onze trainingsset vrij veel twijfelgevallen, dus dit in het bijzonder is nuttig in dit domein [8][24].

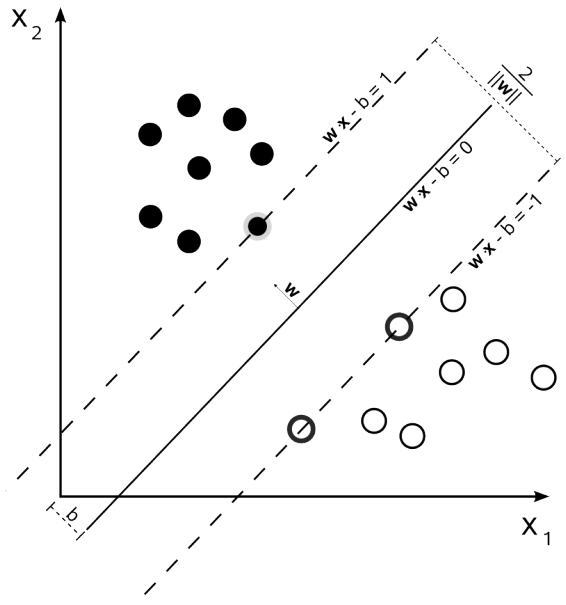
5.3.1 Support Vector Machines

Voor we verder ingaan op ons specifiek model geven we wat achtergrondinformatie over SVM's.

Een SVM die getraind wordt om instanties met n features te onderscheiden, gaat elk van de trainingsinstanties omzetten naar een punt in een n-dimensionale ruimte. Deze mapping kan lineair gebeuren. Er wordt dan een hypervlak in de ruimte bepaald om de klassen te scheiden. Dit vlak heeft als vergelijking:

$$w \cdot x - b = 0$$

met $x = \{x_1, ..., x_n\}$. We zoeken een hypervlak dat langs beide kanten een maximale marge heeft tussen het vlak en het het dichtstbijzijnde trainingssegment.



Figuur 5.4: Illustratie van een hypervlak in een feature-ruimte. We willen het vlak zo trekken dat we de breedte van de marge maximaliseren. Figuur naar [21].

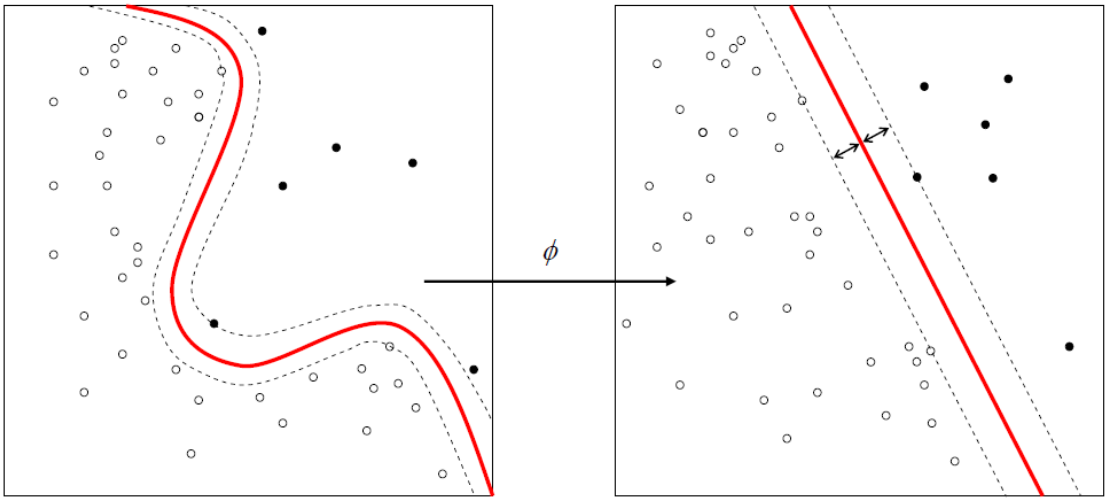
Soms zijn de trainingsinstanties echter niet goed te scheiden door een recht hypervlak. In dat geval kunnen we een niet-lineaire kernel gebruiken om de n-ruimte te transformeren.

In de praktijk gebeurt dit door het inwendig product $x_i \cdot x_j$ te vervangen door een *kernelfunctie*. Zo hebben we de polynomiale kernel:

$$(\gamma \cdot x_i \cdot x_j + a)^d$$

en de Gaussiaanse (of RBF) kernel:

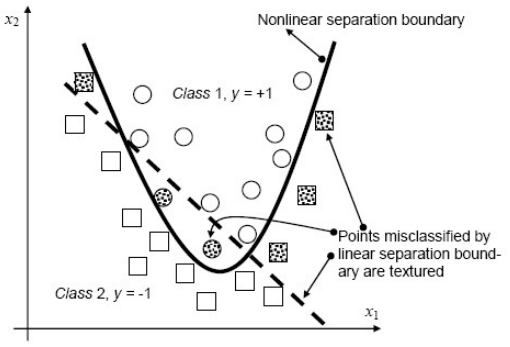
$$exp(-\gamma||x_i \cdot x_j||^2)$$



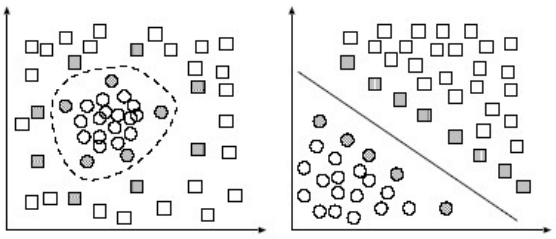
FIGUUR 5.5: Transformatie van de feature-ruimte door gebruik van een niet-lineaire kernel. Figuur naar [21].

Het is niet altijd mogelijk om zo'n hypervlak te vinden: soms zijn trainingsvoorbeelden bijvoorbeeld fout gelabeled of zijn de features zo dat dat voorbeeld tussen de voorbeelden van de andere klasse gemapt wordt. Om toch een hypervlak te vinden worden zulke fouten niet verboden, maar wel afgestraft met een kostparameter C. Hoe hoger deze parameter, hoe "duurder" het is voor de SVM om fouten te maken bij het plaatsen van het hypervlak. Een hogere kost gaat dus een vlak forceren dat correcter is voor de gegeven voorbeelden, maar dat een kleinere marge heeft en dus de kans heeft minder generaliseerbaar te zijn. Een lagere kost bevordert dan weer een hypervlak dat minder goed presteert op de trainingsset, maar dat grotere marges heeft en dus waarschijnlijk meer generaliseerbaar is. Zie figuur 5.7.

Ons probleem heeft drie klassen (Clump, Debris en Cel), dus we moeten de SVM uitbreiden om meerdere klassen te ondersteunen. Hier zijn twee mogelijke aanpakken voor: voor elke klasse een "één-tegen-allen" binair onderscheid maken, of een "één tegen één" binair onderscheid maken voor elk koppel van klassen. LibSVM gebruikt deze laatste strategie omdat de trainingstijd iets korter is [6]. Elke classifier stemt dan op zijn resultaat en de klasse met de meeste stemmen wint. Op deze manier kunnen we een



(a) Gebruik van een Polygonale kernel. Merk op dat de data beter past na de transformatie.



(b) Gebruik van een gaussiaanse kernel.

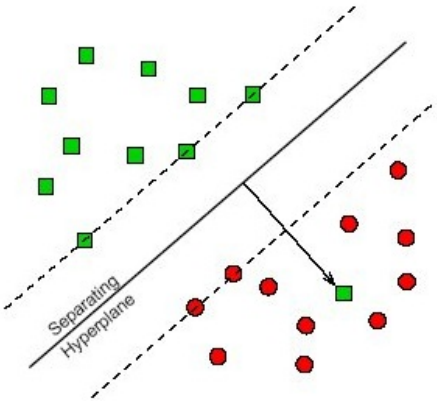
Figuur 5.6: Enkele kernels. Figuren naar [19].

inherent binair classificatiemodel toch gebruiken om een probleem met meerdere klassen op te lossen.

Indien we de hierboven beschreven features willen gebruiken om een SVM te trainen zullen we eerst *normalisatie* moeten toepassen. Dit houdt in dat we alle numerieke waarden van de features herschalen zodat alle geziene instanties binnen het $[0, 1]$-interval passen. Zo voorkomen we dat features met een per definitie hogere waarde (bijvoorbeeld oppervlakte tegenover omtrek) ook een proportioneel belangrijkere rol gaan spelen.

5.3.2 Modelkeuze en -training

Weka biedt twee opties aan om met een SVM te klasseren: er is de ingebouwde "SMO"-classifier en er is de optie om de LibSVM-classifier [6] te gebruiken via de Weka-wrapper. We kiezen voor LibSVM omdat dit sneller is dan de SMO-classifier (2.24s vs. 6.34s in een test van het bouwen van een model met een RBF-kernel) en meer parameters toelaat (de parameters van de kernels hierboven zijn volgens LibSVM). Om vanuit Weka LibSVM aan te spreken maken we gebruik van WLSVM, een wrapper voor LibSVM in Weka. Het voordeel hiervan tegenover rechtstreeks LibSVM gebruiken is het gebruiksgemak en mogelijkheid tot experimentatie van Weka.



FIGUUR 5.7: Gebruik van de kost-parameter. Merk op dat er geen volledig correct scheidend hypervlak mogelijk is. Figuur naar [19].

We hebben een aantal opties wat parameters betreft voor het trainen van ons SVM-Model. Een eerste keuze is de kernel, waar we zowel kijken naar een lineare, polynomiale en gaussische. Afhankelijk van onze keuze dienen we dan ook een aantal parameters in te stellen. Bij de onderstaande experimentaties baseerden we ons op [14] en [6].

We evalueren de parameterkeuzes voor elke kernel apart. Bij de lineaire kernel variëren we enkel de kost C. Voor de polynomiale en gaussische kernel gebruiken we een *Grid Search* om goede waarden te vinden voor C en γ. Dit houdt in dat we elk van deze parameters stapsgewijs over een interval variëren en elke combinatie evalueren. Dit is een vaak gebruikt concept voor parameteroptimalisatie bij SVM's en wordt standaard aangeboden door Weka.

We evalueren alle combinaties via interne 10-fold kruisvalidatie op de trainingsset. Dit houdt in dat de trainingsset 10 keer gesplitst wordt, met 90% van de instanties als trainingset en 10% als testset. Er wordt dan tijdens elke validatiestap een model getest op het train-gedeelte en geëvalueerd op het test-gedeelde. Na afloop wordt het gemiddelde van de individuele resultaten genomen.

Het bepalen van de ideale parameters van een model is geen gemakkelijk probleem. Stel dat we P parameters hebben waarvoor we elk N waarden willen uittesten met een interne M-fold kruisvalidatie. Dit komt overeen met $M(N^P)$ evaluaties, wat al snel onhaalbaar wordt. We moeten daarom het aantal stappen en het aantal parameters dat we tegelijk evalueren beperkt houden, wat we doen met een grid search.

Als prestatiemetriek nemen we de *accuracy*, de verhouding van het aantal juist geklasseerde segmenten op het totaal aantal en de ROC Area, de oppervlakte onder de zogenaamde *ROC-curve*.

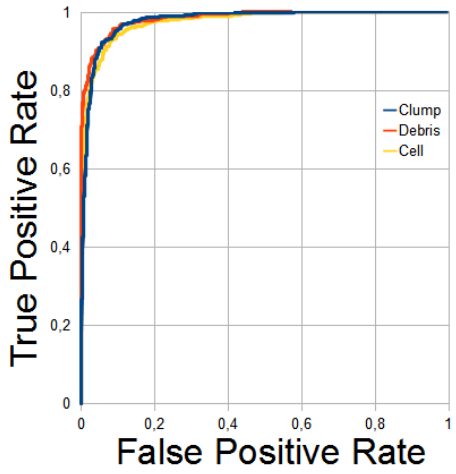
Een ROC-curve wordt gevormd door voor een classifier de drempelwaarde voor

acceptatie van de voorspelling te variëren van 0 (alles aanvaarden) naar 1 (niets aanvaarden). Voor elke drempelwaarde wordt dan de True Positive Rate en False Positive Rate berekend, dit is respectievelijk de verhouding van correct en fout aanvaarde resultaten.

$$TPR = \frac{TP}{P}$$

$$FPR = \frac{FP}{P}$$

Hier is TP het aantal correct positief geïdentificeerde resultaten, FP de foute en P het totaal aantal positieve resultaten. Door de threshold te laten variëren van 0 naar 1 en telkens de overeenkomstige FPR en TPR op respectievelijk de X- en Y-as te plotten krijgen we de ROC-curve van een classifier. In deze context bedoelen we met ROC-oppervlakte het gemiddelde van de drie ROC-curves horende bij elke klasse.



Figuur 5.8: Voorbeeld van ROC curves voor drie classifiers in ons domein

In de tabel hieronder staan enkele iteraties van een grid search met resultaten. We gebruiken dit om de beste parameterinstellingen voor de SVM te bepalen. Niet alle geprobeerde configuraties worden in de tabel vermeld, maar het geeft een idee van de procedure. We proberen de ROC-oppervlakte te maximaliseren. De precisie wordt ook gegeven.

De best bereikte classificatieprecisie is 93.17%, een derdegraads polynomiale kernel met $C = 0.1$, $\gamma = 15$. Door nog fijner te gaan werken zou het eventueel mogelijk zijn een nog hogere classificatiescore te behalen. Dit zou echter een verschil betekenen van enkele instanties op meer dan 2000, wat in de praktijk bijna geen verschil maakt, zeker gezien alle scores al zo dicht bij elkaar liggen. Experimenten met tweedegraads en vierdegraadspolynomialen en sigmoïden leverden geen verbetering op.

Kernel	**P**arameters	**A**ccuracy	**R**OC Area
Lineair	$C = 10^{-1}$	88.96	0.892
	$C = 10^{1}$	92.40	0.930
	$C = 10^{3}$	92.64	0.931
	$C = 10^{5}$	93.01	0.935
	$C = 10^{6}$	92.78	0.935
Polynomiaal (graad 3)	$C = 0.01$, $\gamma = 10$	93.11	0.936
	$C = 0.01$, $\gamma = 15$	93.25	0.938
	$C = 0.01$, $\gamma = 20$	93.25	0.938
	$C = 0.1$, $\gamma = 10$	93.39	0.939
	$C = 0.1$, $\gamma = 15$	**9**3.72	**0**.944
	$C = 0.1$, $\gamma = 20$	93.06	0.938
	$C = 1$, $\gamma = 10$	92.92	0.937
	$C = 1$, $\gamma = 15$	91.83	0.928
	$C = 1$, $\gamma = 20$	91.36	0.924
Gaussisch	$C = 0.1$, $\gamma = 0.1$	80.70	0.81
	$C = 0.1$, $\gamma = 1$	90.42	0.916
	$C = 0.1$, $\gamma = 100$	90.42	0.916
	$C = 1$, $\gamma = 0.1$	89.95	0.903
	$C = 1$, $\gamma = 1$	91.65	0.921
	$C = 1$, $\gamma = 10$	91.9	0.926
	$C = 10$, $\gamma = 0.1$	91.74	0.923
	$C = 10$, $\gamma = 1$	92.5	0.93
	$C = 10$, $\gamma = 10$	92.83	0.935

Tabel 5.2: Parameter-experimentatie op de SVM.

5.3.3 Modelevaluatie

Het is interessant om ons SVM-model op een onafhankelijke manier te evalueren, zodat we de prestaties ervan tijdens het uitvoeren van onze methode beter inschatten. Het geeft ons mogelijk ook meer inzicht in wat er fout kan gaan.

Interne kruisvalidatie is hier niet geschikt voor aangezien de trainingsdata werd gebruikt om de modelparameters te bepalen. We hebben een aparte testset nodig die enkel ongeziene segmenten bevat. Hiertoe labelden we een gemakkelijke (overwegend cellen, weinig debris of clumps), normale en moeilijke (veel grote clumps) afbeelding, zodat we genoeg verscheidenheid tussen de segmenten hadden. Dit leverde ons een set van 586 segmenten op. De classificatie werd slechts met de hierboven genoemde set parameters uitgevoerd om onafhankelijkheid te garanderen. We geven in tabel 5.3 de matrix met classificaties.

We zien dat de veruit meest voorkomende misclassificatie is dat cellen aanzien worden als clumps. De accuraatheid van de evaluatie bedraagt 89.08%, de ROC Area is 0.904.

Geklasseerd als →	**C**lump	**D**ebris	**C**el
Clump	104	4	8
Debris	4	55	10
Cel	31	5	347

Tabel 5.3: Verwarringsmatix van de testset.

Deze resultaten zijn iets minder goed dan die van de kruisvalidatie, zoals verwacht kan worden, maar ze zijn nog steeds acceptabel. Het enige voorzienbare probleem is dat de kans op misclassificatie snel gaat toenemen bij herhaaldelijk opsplitsen van clumps. Zie sectie 8.1.3 voor een bespreking van dit probleem en mogelijke oplossingen.

5.3.4 Andere Modellen

SVM's zijn niet de enige optie als model om te klasseren. We hebben een aantal andere modellen getest met hun standaardparameters in Weka, hieronder geven we de resultaten van een interne 10-fold kruisvalidatie.

Classifier	**A**ccuracy	**R**OC Area
SMO (Polynomiaal)	91.50	0.932
SMO (Gaussisch)	80.75	0.83
Logistic	92.65	0.987
SimpleLogistic	92.97	0.987
RBFNetwork	91.46	0.96
MultilayerPerceptron	92.35	0.98
NaiveBayes	89.38	0.974
J48	90.94	0.935

Tabel 5.4: Resultaten van andere modellen.

We zien dat de modellen over het algemeen gelijkaardig scoren. Dit doet ons vermoeden dat onze features zeer geschikt zijn om $\sim 94\%$ van de segmenten correct te klasseren. De overige $\sim 6\%$ is waarschijnlijk inherent ambigu. In de volgende sectie bekijken we enkele geklasseerde segmenten om een concreet idee te krijgen van de resultaten.

Hoofdstuk 6

Splitsalgoritme

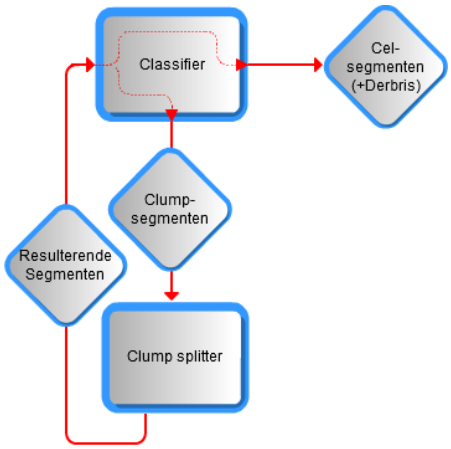
Via het model beschreven in het vorige hoofdstuk kunnen we alle segmenten klasseren als clump, debris of cel. Het gewenste eindresultaat is een segmentatie van alle individuele cellen. Debris dient dus genegeerd te worden en clumps dienen opgesplitst te worden in hun afzonderlijke cellen. Dat laatste is de focus van dit hoofdstuk: we beschrijven een algoritme dat vertrekkende van een clump alle individuele cellen in deze clump als segmenten teruggeeft. Hiervoor gebruiken we een combinatie van bestaande technieken en eigen inzichten.

6.1 Overzicht algoritme

Er zijn twee mogelijke aanpakken te bedenken om het splitsprobleem op te lossen. Ofwel kunnen we de hele clump in één keer opsplitsen in individuele cellen, ofwel kunnen we een iteratieve, recursieve aanpak volgen. Bij deze laatste methode wordt de clump recursief in twee gesplitst worden tot het bodemgeval, een individuele cel of debris, bereikt wordt. Wij hebben voor deze laatste aanpak gekozen. Dit staat ons toe om per iteratie telkens enkel de splitsing waarin we het meeste vertrouwen hebben uit te voeren. We kunnen dan ook elk door het opsplitsen bekomen nieuw segment klasseren en enkel indien het een clump is verder splitsen.

Door ons SVM-model uit het vorige hoofdstuk te hergebruiken kunnen we de verkregen gesplitste segmenten klasseren. Het zou namelijk moeilijk en enigzins zinloos zijn de experimenteel bekomen kennis van ons model te proberen evenaren met een eenvoudige heuristiek die beslist of er verder gesplitst wordt of niet. Dit zou niet alleen betekenen dat we hetzelfde werk opnieuw doen, maar ook op een minder krachtige en flexibele manier. Door slechts één splitsing per verificatie door het model uit te voeren beperken we de beslissingsverantwoordelijkheid van het splitsalgoritme zo veel mogelijk; we laten zowel het splitsalgoritme als het model doen waar ze het beste in zijn.

Vanwege deze redenering hebben we een splitsproces ontworpen dat recursief werkt en alternerend klasseert en splitst. Hieronder is een hoog-niveau splitsalgoritme in pseudocode te zien.



Figuur 6.1: Het recursief proces alterneert tussen klasseren en splitsen van segmenten bekomen door het splitsen.

Algorithm 2 splitsAlle(Segment[] alleSegmenten)

```
cellen ← ∅
for all Segment s in alleSegmenten do
  cellen ← cellen ∪ splitRecursief(s)
end for
return cellen
```

Algorithm 3 splitsRecursief(Segment s)

```
gesplitsteSegmenten ← splits(s) //Splitst clump, geeft twee segmenten terug
resultaat ← ∅
for all Segment t in gesplitsteSegmenten do
  label ← klasseer(t)
  if label = CLUMP then
    resultaat ← resultaat ∪ splitsRecursief(t)
  else if label = DEBRIS then
    //negeer
  else if label = CEL then
    resultaat ← resultaat ∪ t
  end if
end for
return return resultaat
```

6.2 Concaviteitspunten

6.2.1 Hoog-niveaubeschrijving

Een clump heeft een aantal kenmerken die kunnen dienen als aanwijzingen bij de splitsing. Zo heeft een cel die aan de buitenkant van de clump komt minstens één buur die ook aan

Algorithm 4 splits(Segment s)

```
//Zoek beste koppel van concaviteitspunten
cp ← zoekConcaviteitsPunten(s)
koppels ← ∅
//Bouw lijst van mogelijke koppels
for all Punt p1 in cp do
  for all Punt p2 in cp do
    if p1 ! ≠ p2 then
      koppels ← koppels ∪ {p1,p2}
    end if
  end for
end for
for all koppel {p1, p2} in koppels do
  s ← berekenScore(p1,p2)
  if s > derde_beste_s then
    Indien betere score gevonden, update lijst met beste scores
    beste_drie_s ← beste_drie_s ∪ s \laagste(beste_drie_s)
  end if
end for
for all koppel {p1, p2} in besteDrie do
  s ← berekenPadScore(p1,p2)
  if s < bestS then
    bestS ← s
    bestKoppel ← {p1,p2}
  end if
end for
return s.splitsVolgens(besteKoppel)
```

de buitenkant komt. De grens tussen deze twee cellen neemt dan de vorm aan van een "inham" langs de contour van het clump-segment. Het diepste punt op deze inham, het *concaviteitspunt*, maakt dan doorgaans deel uit van de grens tussen beide cellen. Deze aanpak werd met succes gevolgd in [16]. We beschrijven hieronder onze implementatie van dit algoritme wat meer in detail.

We halen de lijst van de punten in de contour van de clump en zijn convex omhullende op en zoeken een punt waar deze twee overeen komen. Vervolgens itereren we synchroon punt per punt over de countouren van beiden. We slaan elke serie countourpunten van de clump die niet in de convex omhullende voorkomt (en dus een concaviteit is) op. Hierna berekenen we voor elk van deze concaviteiten het punt dat het verst verwijderd is van de convex omhullende (met andere woorden, het verst verwijderd van de rechte die door het start- en eindpunt van de concaviteit loopt).

Vervolgens berekenen we voor elk mogelijk koppel concaviteitspunten in het segment een score. Hoe hoger deze score is, hoe beter de splitsing die het koppel oplevert geschat

wordt. We volgen [16] vrij strikt in het berekenen van deze score. Een belangrijk concept is de zogenaamde "Concavity Depth" (concaviteitsdiepte, CD), de afstand tussen de concaviteitspixel en de convex omhullende. Eens de waarde van een koppel op de volgende metrieken dient aan een drempelwaarde te voldoen om geen score van 0 te krijgen:

Saliency: Drukt uit dat grote concaviteitsdiepten en korte afstanden tussen de concaviteitspunten wenselijk zijn. De formule is:

$$Saliency_{i,j} = \frac{min(CD_i, CD_j)}{min(CD_i, CD_j) + afst(C_i, C_j)}$$

De threshold voor het minimum ligt op: 0.01. Dit is veel kleiner dan in de originele paper. Dit komt door de veel grotere clumps. Een zeer grote clump kan een onvoorspelbaar grote minimale afstand hebben tussen de twee concaviteitspunten. Hier een strenge harde threshold opleggen heeft dan weinig zin en kan clumps zelfs onsplitsbaar maken.

Concavity-Concavity Alignment: Drukt uit hoe goed de concaviteiten naar elkaar gericht staan. In het ideaal geval, waarbij de concaviteitspunten i en j aan weerszijden van twee dezelfde rakende cellen liggen, is de hoek die gevormd wordt door de vectoren v_i, v_j vertrekkende uit het midden van het overspannend stuk van de convex omhullende en gaande door het respectievelijk concaviteitspunt precies π rad. Zie ook figuur 6.3. De formule is:

$$CC_{i,j} = \pi - \arccos(v_i \cdot v_j)$$

De threshold is $CC < 1.83$, naar [16].

Concavity-Line Alignment: Drukt uit hoe goed beide concaviteiten gealigneerd zijn met de rechte u_{ij} die door beide concaviteitspunten i en j loopt. Zie ook figuur 6.3. De formule is

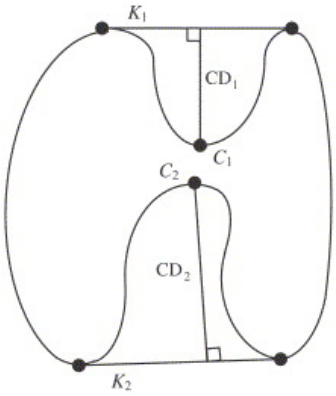
$$CL_{i,j} = max(\arccos(v_i \cdot u_{ij}), \arccos(v_j \cdot (-u_{ij}))),$$

De threshold is $CL < 1.22$, naar [16].

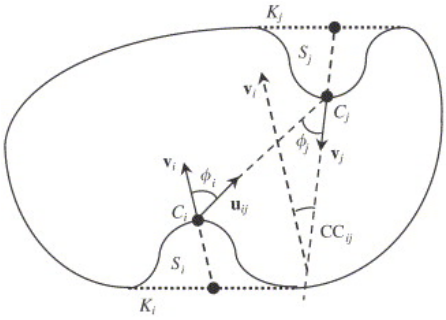
Indien aan deze voorwaarden voldaan is, is de score:

$$Score_{i,j} = \frac{c_1 CD_i + c_1 CD_j + c_2}{afst(C_i, C_j) + c_1 CD_i + c_1 CD_j + c_2}$$

Hier zijn c_1 en c_2 gewichten, vastgelegd op 1.73 en -4.5. Net als bij de saliency-metriek wordt hier de nadruk gelegd op de diepte van de concaviteiten en de afstand tussen de punten. Dit zijn criteria die ook zinvol blijven voor grotere clumps, waar de twee concaviteitspunten niet noodzakelijk op de rand van dezelfde twee cellen liggen.



FIGUUR 6.2: Voorbeeld van een clump met twee concaviteiten. C_1 en C_2, de punten het verst verwijderd van hun respectievelijk overspannend stuk convex omhullende, zijn de concaviteitspunten met diepte CD_1 en CD_2. Figuur naar [16].



FIGUUR 6.3: Illustratie van Concavity-Concavity Alignment en Concavity-Line alignment. De eerste is aangeduid als CC_{ij}, de tweede is het maximum van de hoeken $\phi_i = \arccos(v_i \cdot u_{ij}), \phi_j = \arccos(v_j \cdot (-u_{ij}))), CL_{ij} = max(\phi_i, \phi_j)$. Figuur naar [16].

6.2.2 Enkele opmerkingen

Hoewel dit niet te zien is in de hoog-niveau-werking van het algoritme is het beheren van de datastructuren verantwoordelijk voor de contouren vrij complex. Zowel de contour van de clump als die van de convex omhullende zijn een ImageJ ROI (region of interest) en kunnen omgezet worden naar een `java.awt.Polygon`. We itereren over de contour met diens `PathIterator`. De convex omhullende van een (clump)segment bekomen we met roi.getConvexHull() (Even werd de methode in [30] overwogen, maar de API-methode bleek betrouwbaar genoeg). De PathIterator van deze begint niet op hetzelfde punt als die van de originele clump. We weten echter wel dat er een gemeenschappelijk punt moet zijn, per definitie van de convex omhullende. We itereren over alle punten van het origineel segment tot we een gemeenschappelijk punt ontdekken tussen de twee contouren.

Vertrekkende van dit punt zetten we de PathIterators van beide contouren telkens één stap verder. Indien de coördinaten van het punt niet meer overeen komen hebben we een concaaf segment gevonden. We itereren dan enkel het origineel segment verder en slaan alle tegengekomen punten op als deel van een nieuw concaaf segment. Dit gaat door tot het volgend contourpunt van het gevonden segment overeen komt met hetgeen volgend op het laatst geziene punt van de originele contour. We beschouwen het concaaf segment als compleet en itereren beide contouren verder. Dit alles gaat door tot alle punten van de contouren bekeken zijn. Aangezien een contour circulair is eindigen we nooit met een niet-afgesloten concaaf segment.

Een tekortkoming van dit algoritme laat zich zien bij grotere clumps. De intuïtie achter het originele algoritme is dat elk (voldoende diep) concaaf segment overeen komt met precies twee naast elkaar liggende cellen. Bij de grotere clumps geldt deze veronderstelling niet steeds. Een concaviteit kan vele van deze grenspunten bevatten. Met de huidige methode selecteren we daar maar één uit, wat het aantal mogelijke splitspaden beperkt en mogelijk een correcte splitsing verhindert.

Indien we geen paar tegenkomen dat aan de threshold-condities voldoet, beschouwen we de clump als onsplitsbaar en breken we de splitsing af. Er wordt dan een leeg resultaat teruggegeven.

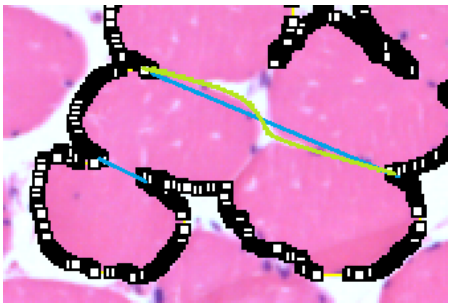
De methode in [16] trekt als splitspad een lijn tussen de twee concaviteitspunten van het beste koppel. Indien de clump op die locatie klein genoeg is zodat een gelijkaardig punt aan de overkant zichtbaar is, is dit een goede benadering. Soms is de samenstelling van een clump echter zo dat er voor geen enkele cel meer dan één inham in de contour geïdentificeerd kan worden. In dat geval bestaat een splitsing van de clump uit meer dan één celgrens. In dit geval moet er extra aandacht gegeven worden aan het pad dat de splitsing volgt. Een rechte lijn gaat dan meer dan waarschijnlijk doorheen interne cellen snijden. Zie ook figuur 6.4.

6.3 Splitspad

Om de beperkingen van het splitsen met rechte lijnen aan te pakken moeten we een techniek implementeren die domeinkennis toepast. Eerst enkele observaties.

We merken op dat de grens tussen twee cellen vaak dezelfde kleur heeft of iets lichter is. Voorts merken we ook op dat in grote clumps er vaak interne “lege plekken” zijn tussen verschillende cellen, met een witte achtergrondkleur. Deze twee soorten regio’s zijn plaatsen waar het splitspad doorheen getrokken moet worden.

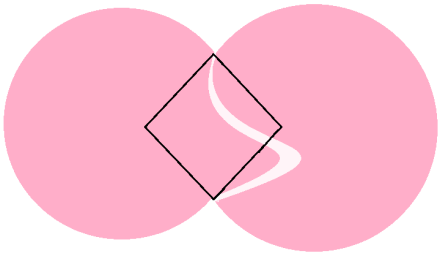
De grootte van de clumps in dit probleemdomein vormen een belangrijke complicatie bij het splitsen niet aanwezig in [16]. Daar werden de gevonden concaviteitspuntenkoppels door middel van een rechte verbonden als splitsing. Dit is slechts een benadering van het werkelijke pad. Voor eenvoudige clumps waar een splitsing slechts twee cellen moet scheiden werkt dit goed, maar voor de langere splitspaden levert dit onacceptabele resultaten.



FIGUUR 6.4: Illustratie van een situatie waar een rechte splitslijn goed (linksonder) en slecht (midden) is. De blauwe lijn toont een rechte splitsing, de groene een flexibele. Het is duidelijk dat dit in het tweede geval natuurlijkere segmenten oplevert.

Voor de oplossing van dit probleem vinden we inspiratie in [22], waar een "laag-energiepad" gevonden wordt tussen twee punten. In deze aanpak werd het pad gevonden volgens de principes van dynamisch programmeren: er werd de veronderstelling gemaakt dat het pad altijd vooruit ging (dus naar de volgende rij pixels). Dan kan verondersteld worden dat het kortste pad tussen het startpunt en het eindpunt dat pixel X bevat ook het kortste pad tussen het startpunt en X bevat. Door hiervan gebruik te maken kan het pad gevonden worden in lineaire tijd. De afstand om een beeldpunt te overbruggen is een functie van diens intensiteit.

Bij het overbrengen van het hierboven beschreven algoritme naar ons probleemdomein komen een aantal bezwaren naar boven. De beperking op de richting van het pad (altijd naar het einde toe) is bijvoorbeeld vrij streng, zeker gezien het feit dat in grote clumps het splitspad tussen een groot aantal cellen moet kronkelen en niet-verwaarloosbare bochten kan maken. Hierbij aansluitend is de "diamantvorm" van de zoekruimte, opgelegd door het feit dat het pad geen rechte hoek kan maken, ook wat beperkend voor dit probleem.



FIGUUR 6.5: Schets van een situatie waar de standaard diamantvormige zoekruimte niet in staat is het ideale splitspad te bevatten.

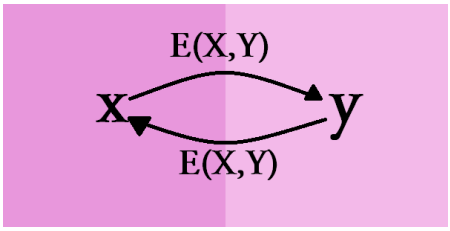
Vanwege deze overwegingen werd geopteerd voor een implementatie van A*[12] om het pad te vinden. A* is een zoekalgoritme om het kortste pad te vinden tussen twee

punten in een grafe. Het houdt in elke iteratie een lijst van gedeeltelijke paden bij van het beginpunt naar het eindpunt. Per iteratie wordt één pad met één segment verlengd, tot dat een pad het einde bereikt. Om te beslissen welk gedeeltelijk pad in een iteratie genomen wordt kijken we naar de verwachte kost van het pad. Dit is de som van de werkelijke kost van het gedeeltelijk pad en een *onderschattende* heuristiek voor het resterend deel.

We modelleren het beeldrooster als grafe: elk beeldpunt is een vertex, met een (gerichte) boog naar elke buur (dus acht bogen indien de pixel zich niet aan de rand bevindt). De gewogen afstand van beeldpunt X naar buur-beeldpunt Y overeenkomend met het gewicht van de boog $E(X,Y)$, modelleren we als een functie van zowel euclidische afstand tussen beide punten als de kleurwaarden van X:

$$W(E(X,Y)) = afst(X,Y) * kost(Kleur_X)$$

Deze laatste component is hetzelfde voor alle bogen vertrekkende uit X. De eerste component (met *afst* de euclidische afstand) is 1 of $\sqrt{2}$, afhankelijk van de positie van de buurpixel.



Figuur 6.6: Voorbeeld van twee pixels als vertices in een grafe. Niet alle bogen zijn weergegeven, maar zowel X als Y hebben een boog van en naar al hun buren. In deze figuur is het gewicht van $E(X,Y)$ de kleur van X vermenigvuldigd met 1. Was Y een diagonale buur van X, dan was de factor $\sqrt{2}$.

Als onderschattende heuristiek voor de overgebleven afstand tussen een tussenliggend punt en het eindpunt hebben we de euclidische afstand genomen, met als gewicht per pixels de kost van een volledig witte pixel (dit is de laagst mogelijk kost).

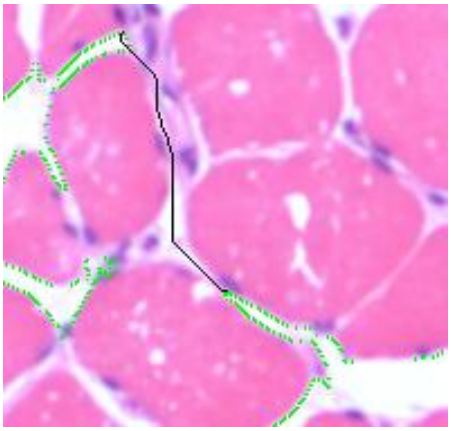
$$H(X,Y) = afst(X,Y) * Kost(wit)$$

Om de kost van een bepaalde kleur te bepalen gebruiken we de volgende formule:

$$Kost(Kleur_X) = [(Groen_{max} - Groen_X) + (Blauw_{max} - Blauw_X)]^2$$

De intuïtie achter deze formule is als volgt: een wit beeldpunt heeft als waarde (Max,Max,Max). Een volledig rood beeldpunt (nemen we hier als model voor een cel)

heeft (Max,0,0). We beschouwen het eerste als het meest wenselijk en het laatste als het minst wenselijk. Experimenteel bleek dat een lineair verband niet sterk genoeg was om paden die recht door cellen gingen te vermijden. Een kwadratische kost lost dit probleem op.

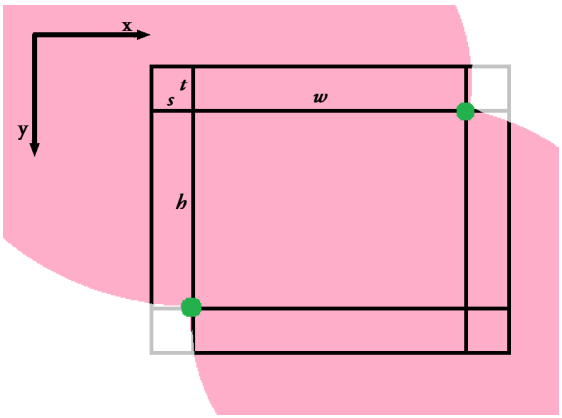


FIGUUR 6.7: Een voorbeeld van een pad berekend met de gegeven kostheuristiek. Merk op dat debris de voorkeur krijgt boven de binnenkant van de cel, zoals het hoort.

We hebben een heuristiek ontworpen om de zoekruimte voor het splitspad te bepalen gegeven een clump en twee concaviteitspunten. We beginnen met de as-gealigneerde rechthoek R te definiëren als de kleinst mogelijke as-gealigneerde rechthoek die de twee cocaviteitspunten bevat. Dit als zoekruimte zou echter niet voldoende zijn: als de punten bijna boven of naast elkaar liggen benadert de oppervlakte nul. Daarom breiden we de rechthoek langs elke zijde uit: de hoogte h en de breedte w langs elke kant met respectievelijk $s = 1/f * h + 1$ en $t = 1/f * w + 1$. De parameter f controleert hoeveel de rechthoek uitgebreid wordt. Ze dient om een evenwicht te vinden zodat de uitbreiding groot genoeg is om nagenoeg elk juist pad te vinden en klein genoeg om het zoeken relatief performant te houden. We zetten $f = 10$. Omdat we onze zoekruimte binnen de clump willen houden zetten we ze op de intersectie van deze uitgebreide rechthoek en de clump. Zie figuur 6.8 voor een schematische voorstelling.

Deze heuristiek presteert over het algemeen goed, maar er zijn enkele gevallen waarin ze tekort schiet. Zo zijn er soms donkere strepen in de afbeelding zichtbaar tussen de cellen. Dit zorgt ervoor dat de cellen als clump geklasseerd worden. Tijdens het splitsen gaat de donkere vlek echter een zeer hoge kost krijgen en dus niet gekozen worden. Een mogelijke verbetering zou eruit bestaan een lage kost toe te kennen aan het stuk van het kleurenspectrum waarin dit soort vlekken voorkomen (zonder daardoor de kost van een pad door de cel zelf te verlagen).

Een ander probleem is het feit dat de rand van de cellen vrij licht gekleurd is, waardoor sommige paden “aangetrokken” worden tot de rand van de clump. Dit zorgt er regelmatig



Figuur 6.8: Schets van de opbouw van onze zoekruimte. De groene stippen stellen de concaviteitspunten voor. Het deel van de uitgebreide rechthoek dat de zoekruimte voorstelt is zwart, het deel dat niet intersecteert met de clump en dus genegeerd wordt is grijs.

voor dat een clump 'gesplitst wordt door een splitspad dat bijna volledig langs de rand van de clump loopt en dus nutteloos is. Een mogelijke oplossing kan eruit bestaan de kosten zo te herbalanceren dat de lichtere kleur van de pixels aan de rand niet meer opweegt tegen de extra lengte van het pad. Er zouden ook restricties kunnen opgelegd worden, bijvoorbeeld het bewaren van een minimale afstand tot de rand van de clump (met hulp van een erosie-operatie op het gebied, bijvoorbeeld). Dit zou er echter toe kunnen leiden dat een pad niet met het start- of eindpunt verbonden kan raken omdat ze voorbij dit minimum liggen. Het is ook denkbaar dat bepaalde uitzonderingen bestaan waar een splitspad dicht langs de rand correct is. Een ander idee is de kost van een pad te linken aan de metrieken die we gebruikten om het paar concaviteitspunten te vinden waartussen we splitsen. Zo kan de grootst mogelijke hoek gevormd tussen de richting van de concaviteit en de rechte gevormd door een pixel van het pad en het concaviteitspunt een maat zijn voor de "afwijking" van het pad en een extra kost meebrengen.

Hoofdstuk 7

Experimenten

We vergelijken onze aanpak met zowel de door experts vastgelegde waarheid, de zogenaamde "Ground Truth", als de resultaten behaald door CellProfiler, het segementatiepakket dat we als state-of-the-art beschouwen.

7.1 Opzet

We maakten een selectie van 50 afbeeldingen uit een dataset van 349. Hierop voerden we zowel onze methode uit als een speciaal hiervoor opgezette segmentatiepipeline in CellProfiler. Beide resultaten worden vergeleken met door experts geannoteerde waarheid (*Ground Truth*). Op deze manier kunnen we inschatten hoe goed onze methode in absolute en relatieve termen presteert.

Als evaluatiecriteria gebruiken we de methode voorgesteld in [13]. Daar worden vijf mogelijke mappings tussen gedetecteerde segmenten (D) en ground truth (GT) voorgesteld, in functie van een threshold T, met $0.5 < T < 1.0$:

Correct: Een mapping tussen een D- en een GT-segment is geklasseerd als "correct" indien voor beide ten minste $T\%$ van de pixels van hun oppervlakte in de intersectie tussen beide zitten.

Oversegmentatie: Een mapping tussen enkele D-segmenten en een GT-segment is geklasseerd als "oversegmentatie" indien voor zowel de unie van de D-segmenten als het GT-segment ten minste $T\%$ van de pixels van hun oppervlakte in de intersectie tussen de beide zitten.

Ondersegmentatie: Een mapping tussen een D-segment en enkele GT-segmenten is geklasseerd als "ondersegmentatie" indien voor zowel de unie van de GT-segmenten als het D-segment beide ten minste $T\%$ van de pixels van hun oppervlakte in de interesectie tussen beide zitten.

Ruis: Een D-segment is geklasseerd als "ruis" indien er geen correcte, over-, of ondersegmentatie-mapping bestaat waarin het deelneemt.

Gemist: Een GT-segment is geklasseerd als "gemist" indien er geen correcte, over-, of ondersegmentatie-mapping bestaat waarin het deelneemt.

Het is mogelijk dat een segment in meerdere mappings deelneemt. In dat geval telt de mapping met de grootste relatieve intersectie. Bij gelijke verhoudingen krijgt "correct de voorkeur, vervolgens "oversegmentatie" en tenslotte "ondersegmentatie".

Vanwege deze complicatie was het implementeren van deze metrieken niet erg voor de hand liggend. Zeker in het geval van over- en ondersegmentatie zijn er erg veel verschillende mappings mogelijk, aangezien enkel de oppervlakte van de unie telt, niet van de individuele segmenten. Dit betekent dat van zodra een koppel D en GT segmenten ook maar één pixel delen het nodig kan zijn ze te beschouwen als deel van een over- en ondersegmentatie (indien ze geen pixels delen zal er altijd een betere segmentatie zijn). Dit maakt de algoritmes om ze te berekenen potentieel erg duur. Hoewel we ze geïmplementeerd hadden bleek onze code echter veel te rekenintensief, waardoor we ervoor opteerden het klassificatieschema wat te vereenvoudigen. Hierbij laten we de twee problematische classificiaties vallen, waardoor we "Correct","Ruis" en "Gemist" overhouden. Deze kunnen we respectievelijk interpreteren als *True Positive* (TP, echt positief), *False Positive* (FP, vals positief) en *False Negative* (FN, vals negatief). Deze metrieken gebruiken we om de resultaten te kwantificeren.

Naast deze drie metrieken gebruiken we ook de *Precision*- en *Recall*-metrieken. Ze worden gegeven door de volgende formules:

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

$$Recall = \frac{TP}{TP + FN}$$

Precision staat hier voor de verhouding van het aantal geïdentificeerde cellen die werkelijk cellen zijn tegenover alle gevonden cellen, dit kan geïnterpreteerd worden als de mate van *exactheid* van het model. Recall staat voor de verhouding van het aantal geïdentificeerde cellen op het totaal aantal cellen en kan geïnterpreteerd worden als de mate van *volledigheid* van het model. In de context van *Information Retrieval* staat Precision voor het verhouding van het aantal gevonden relevante documenten tegenover alle gevonden documenten, Recall staat voor de verhouding van alle gevonden relevante documenten tegenover alle relevante documenten.

We kunnen ook de *F1-score*, een metriek die Precision en Recall combineert, berekenen als volgt:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Deze score heeft als voordeel dat ze in beide metrieken in één cijfer giet.

We variëren in onze experimenten de thresholdwaarde T. Het is belangrijk te weten wat deze parameter concreet betekent. Hoe lager de waarde T, hoe lager de vereiste die we stellen aan een gedetecteerde cel om als correct bevonden te worden. Een waarde van $T = 0.5$ betekent bijvoorbeeld dat een gedetecteerde cel die slechts voor de helft met zijn oppervlakte overeen komt met een echte cel en vice versa, zal gedetecteerd worden als juist. Indien we eigenchappen opmeten van de gevonden cel zullen deze ook niet overeen komen met die van de werkelijke cel. Met $T = 1$ eisen we dan weer dat de cellen 100% overeen komen in oppervlakte. In de praktijk is het niet te garanderen dat elke pixel overeen komt, dus dit is geen realistische waarde. Concreet is het voor experts wenselijk T zo hoog mogelijk te houden, zodat de kwaliteit van de gevonden segmenten gegarandeerd is, zonder het zo hoog te nemen dat geen enkel gevonden segment de test doorstaat. Welke keuze goed is hangt af van de concrete toepassing en de middelen voorhanden om de segmenentatie uit te voeren.

7.2 Resultaten

In tabel 7.1 zijn de gesommeerde resultaten van onze methode en CellProfiler, de huidige state-of-the-art voor 50 afbeeldingen en 5 waarden voor T. In figuur 7.1 zetten we de Precision, Recall en F1-score in functie van de threshold-waarde uit in een grafiek. In tabel 7.2 presenteren we de met thresholdwaarde $T = 0.75$ de behaalde resultaten per afbeelding voor dezelfde 50 afbeeldingen. Deze waarde voor T is volgens medische experts de meest bruikbare.

Het is duidelijk te zien dat onze methode een grote vooruitgang betekent tegenover de state-of-the-art. Ze presteert wat alle metrieken betreft beter op elk getest threshold-niveau. We kunnen ook zien, zeker in de grafiek, dat onze segmentatieresultaten bij een hogere thresholdwaarde beter overeind blijven dan die van CellProfiler: onze T1-score varieert van 0.70 tot 0.55 ($T = 1$ uitgezonderd), voor CellProfiler is dit 0.53 tot 0.11. Onze methode segmenteert 290 segmenten perfect overeenkomstig met de ground truth ($T = 1$), CellProfiler slechts 7. Dit is natuurlijk bij beiden slechts een klein deel van het totaal en geen haalbaar doel, maar toch is het een mooie indicatie van de kracht van onze methode.

In deze toepassing kan gezegd worden dat Precision belangrijker is dan Recall. Het is namelijk belangrijk dat alle segmenten die we identificeren als cellen ook daadwerkelijk cellen zijn. Hoe meer echte cellen we vinden hoe beter, maar dit is niet cruciaal. Het is daarom extra interessant dat de Precison met onze methode relatief hoog ligt: bij $T = 0.75$ zijn 76% van onze geïdentificeerde cellen correct.

Het hoge aantal valse positieven van beide methodes kan deels verklaard worden door het feit dat enkele van de afbeeldingen slechts deels geannoteerd waren, waardoor sommige correcte detecties als fout beschouwd kunnen worden. Dat het aantal valse positieven van CellProfiler aanzienlijk hoger ligt dan dat van onze methode kan verklaard worden

door het feit dat CellProfiler geen aangepaste algoritmes heeft om debris te verwijderen. In meerdere afbeeldingen werden grote oppervlaktes debris als een verzameling van cellen beschouwd.

Verder kan het hoge aantal valse positieven en negatieven van CellProfiler beschouwd worden als een resultaat van de zwakke segmentatie van clumps: indien een splitsingslijn verkeerd getrokken wordt is het goed mogelijk dat geen enkel geidentificeerde cel overeen komt met een werkelijke cel, waardoor er zowel een aantal nieuw valse positieven (de geidentificeerde cellen in de clump) als een aantal valse negatieven (de werkelijke cellen in de clump) bijkomt.

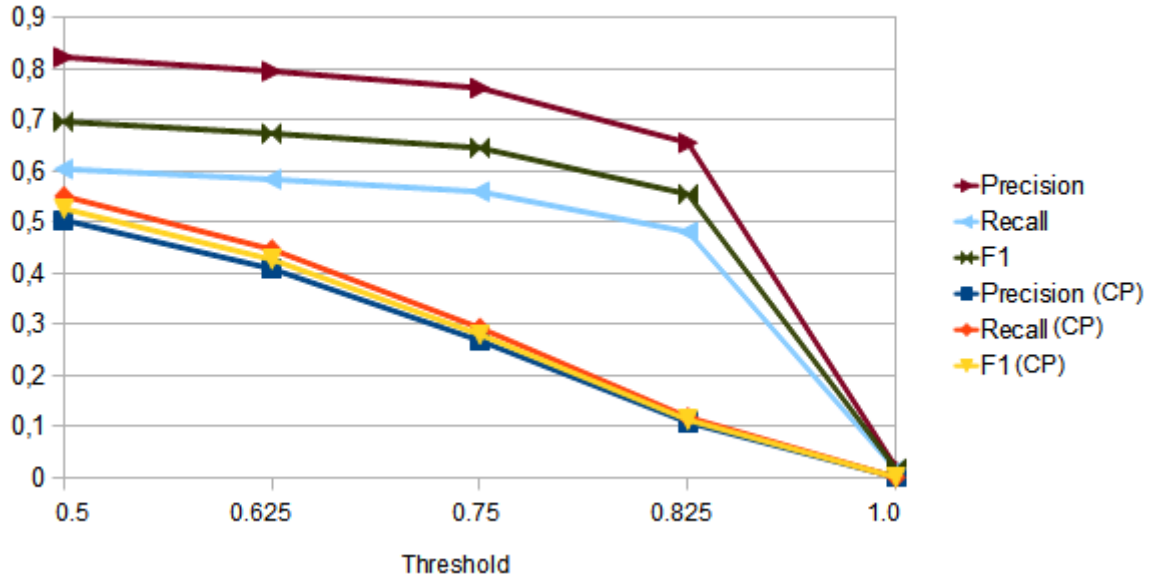
Het valt op dat er een groot verschil is in de tussen de prestaties van beiden op de beste en slechtste afbeelding. Ook is er een zekere correlatie tussen de prestaties van beide methodes in functie van de afbeelding. Dit bevestigt dat afbeeldingen een sterk variërende moeilijkheidsgraad hebben. Gemakkelijke afbeeldingen bevatten bijvoorbeeld veel losstaande cellen en weinig grote clumps of debris. Moeilijke afbeeldingen bevatten dan weer clumps die gemakkelijk de helft van de afbeelding in beslag nemen, een mix van herkenbare en gedesintegreerde cellen, of vervormde cellen. De 50 geselecteerde afbeeldingen zijn gekozen met het oog op segmenteerbaarheid door zowel onze methode als CellProfiler, waardoor we de ergste gevallen van dit soort afbeeldingen niet behandeld hebben.

Het segmentatieproces duurde ongeveer 10u, of gemiddeld 12 minuten per afbeelding. Het grootste deel van deze tijd werd ingenomen door het splitsen van grote clumps. Voor sommige grote clumps kon een individuele splitsing oplopen tot 30 minuten. Afbeeldingen met relatief weinig grote clumps kunnen binnen enkele minuten volledig gesplitst zijn.

In afbeelding 7.2 analyseren we enkele segmentaties.

	Onze Methode						CellProfiler					
T	**T**P	**F**P	**F**N	**P**r	**R**e	**F**1	**T**P	**F**P	**F**N	**P**r	**R**e	**F**1
0.5	11740	2525	7697	0,82	0,60	0,70	10708	8729	10541	0.50	0.55	0.53
0.625	1139	2916	8088	0.80	0.58	0.67	8686	10755	12567	0.41	0.45	0.43
0.75	10877	3388	8560	0.76	0.56	0.65	5695	13742	15554	0.27	0.29	0.28
0.875	9349	4916	10088	0.66	0.48	0.55	2293	17144	18956	0.11	0.12	0.11
1.0	290	13975	19147	0.02	0.01	0.02	7	19430	21242	0.00	0.00	0.00

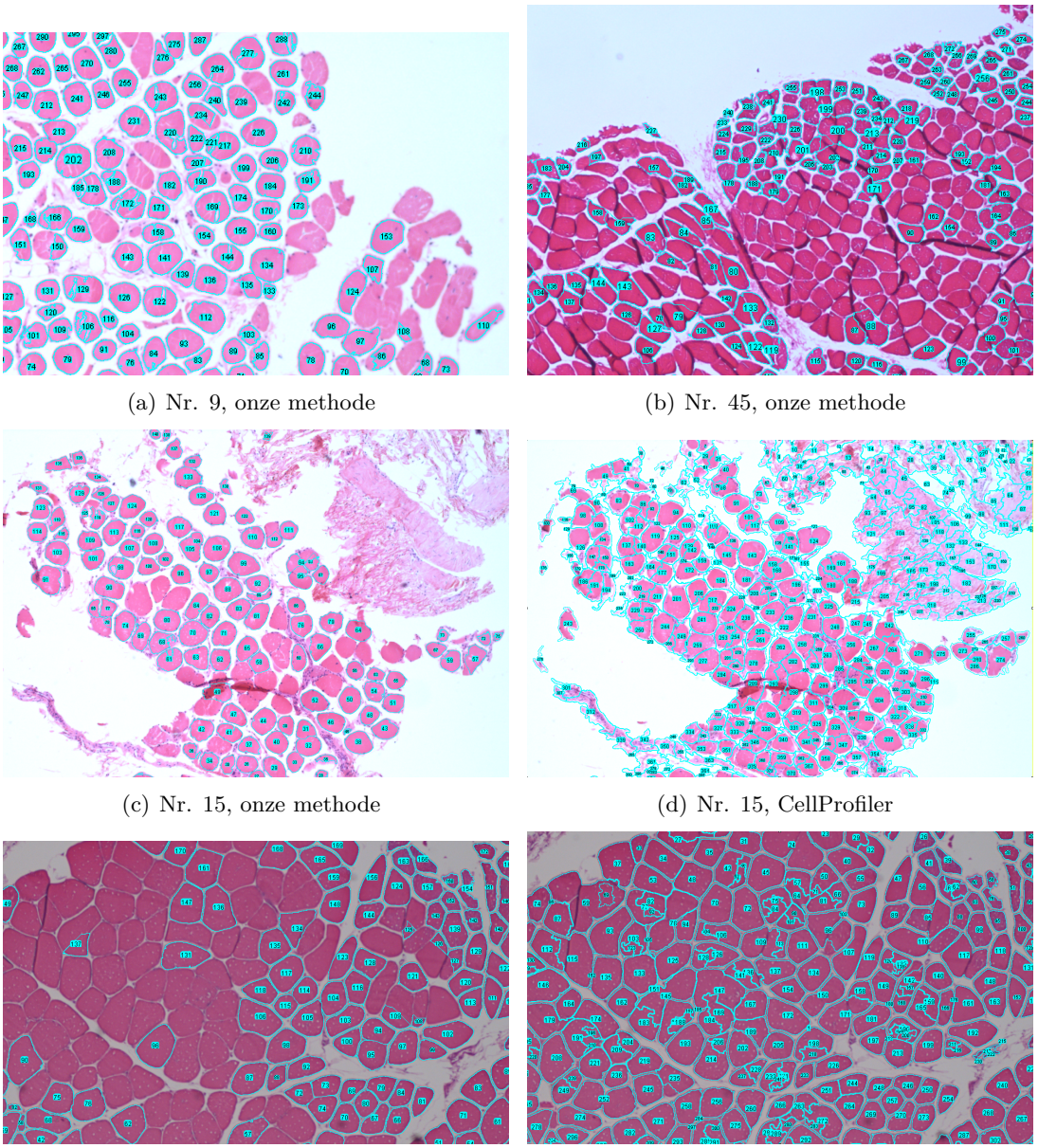
Tabel 7.1: Gesommeerde resultaten van segmentatie van 50 afbeeldingen. De gebruikte thresholdwaarde bij de evaluatie varieerde in stappen van 0.125 van $T = 0.5$ tot $T = 1$.



FIGUUR 7.1: De score van de metrieken voor beide methodes geplot in functie van de threshold-waarde op de minimale overlap.

	Onze Methode						CellProfiler					
#	**T**P	**F**P	**F**N	**P**r	**R**e	**F**1	**T**P	**F**P	**F**N	**R**e	**P**r	**F**1
1	376	57	61	0.87	0.86	0.86	314	155	123	0.67	0.72	0.69
2	227	35	44	0.87	0.84	0.85	187	94	84	0.67	0.69	0.68
3	364	40	100	0.90	0.78	0.84	288	110	176	0.72	0.62	0.67
4	370	71	76	0.84	0.83	0.83	23	467	423	0.05	0.05	0.05
5	331	49	93	0.87	0.78	0.82	185	361	239	0.34	0.44	0.38
6	76	14	19	0.84	0.80	0.82	28	105	67	0.21	0.29	0.25
7	588	97	166	0.86	0.78	0.82	50	674	704	0.07	0.07	0.07
8	330	59	93	0.85	0.78	0.81	152	442	271	0.26	0.36	0.30
9	398	65	153	0.86	0.72	0.79	376	145	175	0.72	0.68	0.70
10	411	68	168	0.86	0.71	0.78	180	449	399	0.29	0.31	0.30
11	383	57	167	0.87	0.70	0.77	163	485	387	0.25	0.30	0.27
12	211	67	61	0.76	0.78	0.77	32	289	240	0.10	0.12	0.11
13	194	48	85	0.80	0.70	0.74	169	157	110	0.52	0.61	0.56
14	366	70	185	0.84	0.66	0.74	379	172	172	0.69	0.69	0.69
15	111	29	50	0.79	0.69	0.74	13	366	148	0.03	0.08	0.05
16	219	44	120	0.83	0.65	0.73	118	366	221	0.24	0.35	0.29
17	116	49	52	0.70	0.69	0.70	29	182	139	0.14	0.17	0.15
18	232	73	130	0.76	0.64	0.70	192	277	170	0.41	0.53	0.46
19	124	39	74	0.76	0.63	0.69	82	113	116	0.42	0.41	0.42
20	230	71	141	0.76	0.62	0.68	177	265	194	0.40	0.48	0.44
21	179	58	116	0.76	0.61	0.67	123	240	172	0.34	0.42	0.37
22	271	85	185	0.76	0.59	0.67	129	451	327	0.22	0.28	0.25
23	152	36	130	0.81	0.54	0.65	177	209	105	0.46	0.63	0.53
24	177	99	110	0.64	0.62	0.63	74	461	213	0.14	0.26	0.18
25	347	135	275	0.72	0.56	0.63	75	556	547	0.12	0.12	0.12
26	498	118	481	0.81	0.51	0.62	61	641	918	0.09	0.06	0.07
27	402	100	401	0.80	0.50	0.62	277	475	526	0.37	0.34	0.36
28	152	59	134	0.72	0.53	0.61	110	158	176	0.41	0.38	0.40
29	230	78	219	0.75	0.51	0.61	126	412	323	0.23	0.28	0.26
30	144	68	120	0.68	0.55	0.61	107	309	157	0.26	0.41	0.31
31	67	20	69	0.77	0.49	0.60	70	91	66	0.43	0.51	0.47
32	286	100	318	0.74	0.47	0.58	222	205	382	0.52	0.37	0.43
33	91	35	101	0.72	0.47	0.57	67	176	125	0.28	0.35	0.31
34	121	29	154	0.81	0.44	0.57	42	245	233	0.15	0.15	0.15
35	400	215	408	0.65	0.50	0.56	123	645	685	0.16	0.15	0.16
36	90	62	89	0.59	0.50	0.54	43	353	136	0.11	0.24	0.15
37	173	87	207	0.67	0.46	0.54	104	313	276	0.25	0.27	0.26
38	52	31	66	0.63	0.44	0.52	30	151	88	0.17	0.25	0.20
39	406	249	563	0.62	0.42	0.50	80	673	889	0.11	0.08	0.09
40	230	68	441	0.77	0.34	0.47	25	644	646	0.04	0.04	0.04
41	32	36	40	0.47	0.44	0.46	24	89	48	0.21	0.33	0.26
42	291	187	569	0.61	0.34	0.43	230	538	630	0.30	0.27	0.28
43	22	17	45	0.56	0.33	0.42	2	133	65	0.01	0.03	0.02
44	51	37	111	0.58	0.31	0.41	17	362	145	0.04	0.10	0.06
45	191	92	512	0.67	0.27	0.39	64	412	639	0.13	0.09	0.11
46	100	88	327	0.53	0.23	0.33	49	449	378	0.10	0.11	0.11
47	6	8	26	0.43	0.19	0.26	18	34	14	0.35	0.56	0.43
48	51	71	233	0.42	0.18	0.25	70	326	214	0.18	0.25	0.21
49	4	14	30	0.22	0.12	0.15	13	58	21	0.18	0.38	0.25
50	4	4	42	0.50	0.09	0.15	6	71	40	0.08	0.13	0.10
Totaal:	10877	3388	8560	0.76	0.56	0.65	5695	15554	13742	0.27	0.29	0.28

Tabel 7.2: Resultaten van segmentatie van 50 afbeeldingen, gerangschikt volgens F1-score van onze methode. De gebruikte thresholdwaarde bij de evaluatie was $T = 0.75$.



(a) Nr. 9, onze methode

(b) Nr. 45, onze methode

(c) Nr. 15, onze methode

(d) Nr. 15, CellProfiler

(e) Nr. 23, onze methode

(f) Nr. 23, CellProfiler

FIGUUR 7.2: Enkele voorbeelden van segmentaties in de afbeeldingen. Gedetecteerde cellen zijn in het blauw weergegeven. a) toont een goed resultaat van onze methode, b) een slecht. Merk op dat het groot leeg gebied in het midden van b) een clump was die niet succesvol gesplitst is kunnen worden. In c) en d) zien we respectievelijk de segmentatie van onze methode en CellProfiler op een afbeelding met veel debris. Het valt op dat deze laatste er niet in slaagt het debris te scheiden van het relevant materiaal en zo veel ruis introduceert. Hoewel de segmentatie van CellProfiler op het eerste zicht volledige lijkt, is ze veel minder precies dan die van onze methode. Indien de $T = 0.75$ gesteld wordt gelden er maar 13 segmenten uit de CellProfiler-segmentatie van de afbeelding waaruit dit stuk afkomstig is als correct, tegenover 111 met onze methode (van de 161 werkelijke cellen). e) en f) tenslotte demonstreren één van de zeldzame afbeeldingen waar CellProfiler (f) het beter bij doet. Dit is waarschijnlijk omdat onze intitiële segmentatie hier wat tekort schiet.

Hoofdstuk 8

Besluit

We hebben in dit werk een nieuwe methode geïntroduceerd om histologische beelden van skeletale spiercellen te segmenteren. Voor dit specifieke probleemdomein bestond er nog geen volledig automatische segmentatietechniek. Door nuttige technieken uit de literatuur te combineren met eigen inzichten hebben we een methode ontworpen die deze taak op een behoorlijke manier volbrengt.

Met dit onderzoek tonen we aan dat de resultaten van eenvoudige algoritmes, zoals color thresholding, sterk verbeterd kunnen worden door toevoeging van complexere lerende onderdelen, zoals SVM-classificatie. We demonstreren ook hoe deze laatste recursief in tandem kan werken met een complex beeldverwerkingsalgoritme, zodat ze elkaars tekortkomingen compenseren.

We zien dat de resultaten beter zijn dan de huidige state-of-the-art, gerepresenteerd door CellProfiler. We hebben bij een test van 50 afbeeldingen zowel een significant hogere Precision als Recall kunnen bereiken. Het merendeel van deze afbeeldingen werd door onze methode acceptabel gesegmenteerd. Er zijn echter nog types van afbeeldingen die niet behoorlijk gesegmenteerd kunnen worden, noch door onze methode, noch door de state-of-the-art. Onze methode kan echter een belangrijke basis vormen voor algoritmes die ook deze types aankunnen. We geven in de sectie hieronder enkele suggesties.

De resultaten van deze methode kunnen een basis vormen voor verder onderzoek dat de relatie tussen de toestand van cellen en die van de patiënt verder bestudeert. Hiervoor is namelijk een performante segmentatie van de cellen nodig, zodat de metrieken berekend uit de gevonden cellen overeen komen met de werkelijkheid. Met onze methode is dit soort onderzoek haalbaarder dan met de huidige state-of-the art.

Dit onderzoek werd verricht in samenwerking met de Research Group of Intensive Care Medicine van het Universitair Ziekenhuis Gasthuisberg (http://www.kuleuven.be/licm/).

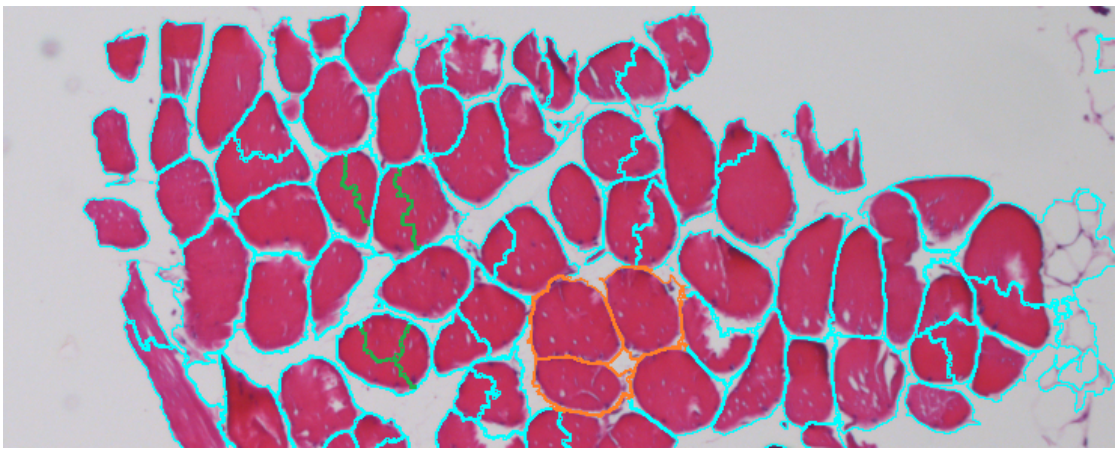
8.1 Mogelijke verbeteringen en uitbreidingen

Er zijn een aantal vlakken waarop onze methode verbeterd kan worden. Door het groot aantal gecombineerde technieken zijn niet alle aspecten van het algoritme optimaal uitgewerkt kunnen worden en niet alle ideeën geïmplementeerd. We geven een overzicht van veelbelovende mogelijke verbeteringen of uitbreidingen.

8.1.1 Initiële Segmentatie

De heuristiek gebruikt voor het bepalen van de drempelwaarde is robuust genoeg om die voor het merendeel van de afbeeldingen even goed te kiezen als een menselijke gebruiker. Toch kan hier grondiger mee geëxperimenteerd worden. Het opstellen van een baseline van menselijk bekomen resultaten om de prestaties te kwantificeren zou interessant kunnen zijn. Dit kan eventueel nog verder gebracht worden door een model te gebruiken dat leert van de menselijke voorbeelden.

Het valt op dat CellProfiler een vrij degelijke ruwe segmentatie lijkt te doen van de clumps, maar dat ze vervolgens vaak slecht verder gesegmenteerd worden. Er is een merkbaar verschil tussen het uitzicht van deze twee veronderstelde stappen: de eerste heeft zeer gladde randen, terwijl de tweede vrij grillige grenzen oplevert. Debris wordt door CellProfilerregelmatig gemist. Debris herkennen en clumps goed verder segmenteren zijn de sterktes van onze methode, initieel clumps segmenteren doet CellProfiler beter. Als er een manier is om die functionaliteit te isoleren (of dat deel van het algoritme te implementeren) en te combineren met onze methode zouden de clumps waarmee het splitsalgoritme moet werken heel wat kleiner kunnen zijn, wat de snelheid en precisie ten goede komt. De “slechte grenzen” zouden misschien gebruikt kunnen worden om de zoektocht van ons splitsalgoritme wat te richten.



Figuur 8.1: Detailbeeld van CellProfiler-segmentatie. Enkele goede splitsingen aangeduid in oranje, enkele slechte in groen.

8.1.2 Classificatie en model

Momenteel wordt ons model enkel getraind op een 2000-tal handgelabelde segmenten. Naar het einde van ons onderzoek zijn we echter in het bezit gekomen van 300 door experts geannoteerde afbeeldingen (onze trainingsdata was afkomstig uit 11). Deze annotaties bevatten de locatie van elke cel in de afbeelding. We zouden deze extra informatie mogelijk kunnen gebruiken om het model verder te trainen.

We kunnen de annotaties rechtstreeks gebruiken om voorbeelden van cel-segmenten te vinden, maar we kunnen ze ook vergelijken met de initiële segmentatie om voorbeelden te vinden van debris (door te zien welke segmenten niet aanwezig zijn in de annotatie) of clumps (de segmenten die de annotatiedata ondersegmenteren).

De verkregen prestaties van het model lijken er echter op te wijzen dat een deel van de data gemakkelijk te onderscheiden is en een deel van nature zeer moeilijk. Het is niet erg waarschijnlijk dat meer trainingsdata daar veel aan gaat veranderen. Zo haalden we bijvoorbeeld bijna even goede resultaten (92% Accuracy) met een trainingsset van slechts 586 segmenten, wat op het bereiken van een afvlakking van de prestaties in functie van een toenemend aantal voorbeelden wijst.

Een bijna fundamentele beperking van ons model zit in het feit dat het een discrete grens trekt tussen de drie klassen. Dit is wenselijk bij clump, dat per definitie een samenstelling van de andere twee klassen is. Het verschil tussen debris en cel is in sommige situaties echter wat minder duidelijk. In sommige van de beelden (vaak van erg zieke patiënten of postmortem genomen) is een aanzienlijk deel van de cellen deels gedesintegreerd. Dit kan deels verklaren waarom extra features en meer data ons niet veel dichter bij een volledig juiste classificatie brengen. Er zou dus plaats kunnen zijn voor een uitbreiding van het algoritme die de discrete classificatie vervangt door een continue. Een segmentatie zou dan bijvoorbeeld een vertrouwen in elk segment kunnen uitdrukken.

Tenslotte valt het op dat er vrij veel misclassificaties lijken voor te komen in het recursief splitsen, hoewel we hier geen data voor hebben. Hoe groter de clump, hoe gevoeliger ze ook is aan misclassificatie: stel dat de kans op misclassifiatie van een clump gelijk is aan 0.1. Als we een clump n maal moeten splitsen om aan een bepaalde cel te komen is de kans dat we ten minste één van de omvattende clumps fout klasseren gelijk aan $1 - 0.9^n$. Voor $n = 3$ is dit meer dan een vierde van alle cellen, voor $n = 4$ een derde, voor $n = 7$ de helft. Een clump die uit c^n cellen bestaat kan in het beste geval gespitst worden in n stappen, in het slechtste in c^n. Een clump bestaande uit 8 cellen heeft volgens deze redenering gemiddeld tussen de $0.27 * 8 = 2.16$ en de $0.56 * 8 = 4.55$ cellen die niet als dusdanig herkend worden. Dit is een zeer ruwe schatting, maar ze illustreert wel het belang van een goed model. Als alternatief bespreken we later ook een methode die grote clumps opbreekt om dit soort kettingen klein te houden.

We hebben er dus belang bij dat ons model zo goed mogelijk opgesplitste clumps kan onderbrengen bij de juiste klassen. Deze segmenten zien er niet helemaal uit als gewone

segmenten vanwege het ander proces dat is gebruikt om ze te verkrijgen (splitsen vs. thresholding). We hebben ook geen trainings- of testdata van dit soort segmenten. Het zou dus interessant zijn te kwantificeren hoe ons model voor specifiek dit soort segmenten scoort en te kijken of het helpt deze in de trainingset op te nemen.

8.1.3 Splitsen van clumps

Eén van de grootste nadelen aan onze methode is dat het zeer lang duurt om clumps te splitsen. Dit komt vooral door het pad-zoekende algoritme. Indien we voor A* optimaliteit willen garanderen moet onze heuristiek onderschattend zijn. Dit forceert ons om met het (onwaarschijnlijke) beste geval rekening te houden: dat het huidige punt zich bevindt op een recht wit pad naar het doel. Het A*-algoritme werkt zodanig dat het van alle gedeeltelijk gevonden paden hetgeen met de laagste schatting uitbreidt. Vanwege onze heuristiek gaat dat vrij vaak het pad zijn dat het verst verwijderd is van het einde. Dit resulteert in een verwachte complexiteit van $O(min(n^2, o))$ voor het splitsen van een clump, met n de afstand tussen de eindpunten van de splitsing en o de oppervlakte van de clump. Reken hierbij het feit dat in het ergste geval de splitsing resulteert in twee nieuwe clumps die op hun beurt gesplitst moeten worden en het is duidelijk dat dit een vrij rekenintensief algoritme is. Een afbeelding met grote clumps kan zo zeker enkele tientallen minuten duren om te splitsen, wat onacceptabel is.

Een mogelijke versnelling is de heuristiek wat realistischer te maken, door een langer dan recht of minder wit pad te veronderstellen. Hierdoor verliezen we de garantie van optimaliteit, maar het is mogelijk dat we in de meeste gevallen er nog altijd in slagen een (bijna even goed) pad te vinden. Precisie inruilen voor snelheid is echter nooit ideaal.

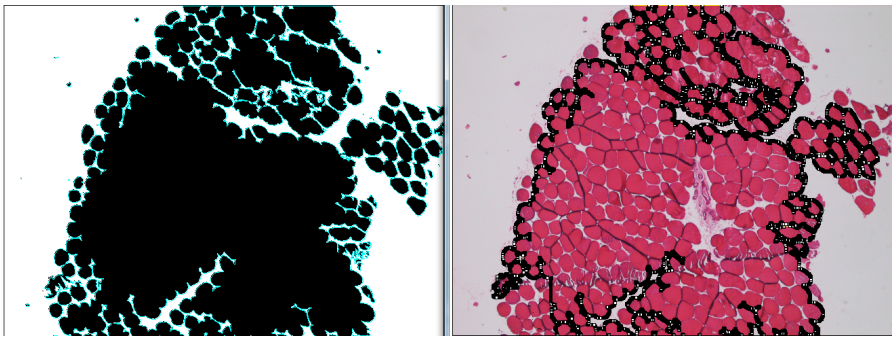
Een andere gemiste opportuniteit van het splitsalgoritme is het gebruiken van informatie over de interne holtes van de clump. Vaak zijn er grote plekken intern in de clump leeg, maar worden ze opgevuld met de *fill holes*-operatie tijdens de initiële segmentatiestap. Deze operatie is nodig om de contouren van de clumps te vinden, maar ze vernietigt wel deze informatie. Bij diepe interne inhammen in de clump wordt er momenteel ook maar één (het diepste) concaviteitspunt gekozen, uit vele kandidaten die aansluiten bij het intuïtief idee erachter.

Een mogelijke oplossing hiervoor is een concaviteit van een segment punt voor punt afgaan en de diepte d van punt p_i te beschouwen als een functie

$$d = f(i) = afstand(p_i, rechte(p_0, p_n))$$

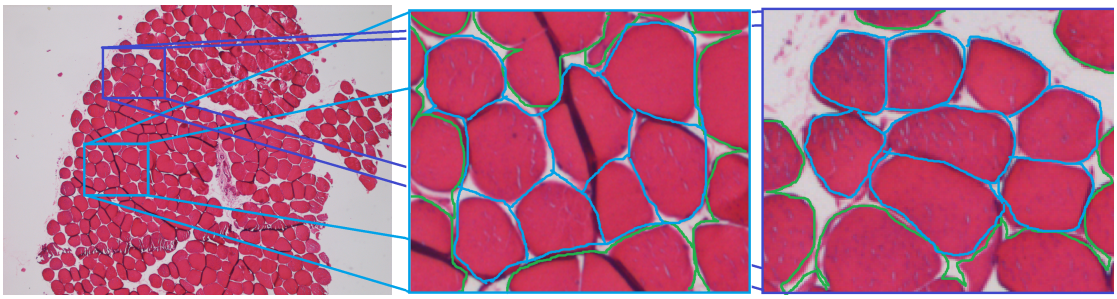
We kunnen verwachten dat de lokale maxima van deze functie overeen komen met plaatselijke concativiteitspunten. Door al deze punten te selecteren, in plaats van enkel het diepste, vergroten we het aantal kandidaat-splitspadeindpunten, waardoor we de kans een optimaler splitspad te vinden vergroten.

Een waarschijnlijk betere (en mogelijk complementaire) oplossing voor beide bovenvernoemde probemen kan eruit bestaan de clump in verschillende stukken te delen (door



FIGUUR 8.2: Voorbeeld van een afbeelding waar de initiële segmentatie niet kan voorkomen dat er een zeer grote clump gevormd wordt. Merk op hoe alle interne holtes verloren gaan.

er bijvoorbeeld een rooster over te plaatsen) en het splitsalgoritme op deze delen toe te passen. Concreet voeren we eerst de thresholding-stap terug uit om de interne holtes terug zichtbaar te maken. Vervolgens beschouwen we het resulterend segment als een clump op zich die we met het gewone algoritme splitsen. Cellen die volledig aanwezig zijn in dit segment zullen dan correct herkend worden. Cellen die slechts gedeeltelijk inbegrepen zijn zullen, na afsplitsing van de volledige cellen, niet verder gesplitst kunnen worden. Na afloop van het splitsen kunnen we dan de volledige cellen als afgehandeld beschouwen. De onvolledige cellen, herkend aan het feit dat ze de grens van het segment raken, kunnen samengevoegd worden met de onvolledige segmenten aan de overkant van de buur-deelclumps. Deze hele operatie kan dan bijvoorbeeld herhaald worden door het rooster een halve rechthoeklengte naar rechts en naar beneden te verplaatsen.

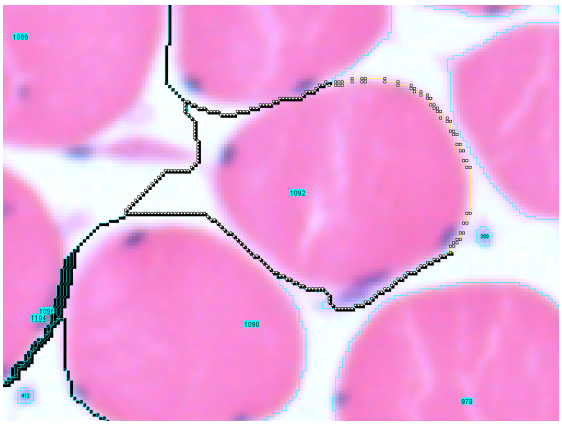


FIGUUR 8.3: Voorbeeld van een rechthoek uit een rooster. Links is de context van de rechthoeken te zien. Midden en rechts is te zien welke cellen wel (blauw) en niet (groen) gesegmenteerd kunnen worden met enkel deze rechthoek. Merk op dat de segmenten in het midden ingesloten zitten, die rechts niet.

De afmetingen van de rechthoeken in het rooster kunnen gekozen worden zo dat ze

groot genoeg zijn om enkele volledige individuele cellen per rechthoek te bevatten en klein genoeg om de lengte van het met A* te berekenen splitspad genoeg naar beneden te halen. Als bijkomend voordeel heeft deze methode ook nog dat de steeds groeiende probabiliteit van misclassificatie bij een lange reeks splitsingen hier wordt ingeperkt.

Het opgevuld zijn van de clumps heeft nog een onaangenaam neveneffect: holtes in de clump zelf worden nooit weggefilterd en blijven dus deel uitmaken van de door de splitsing gevonden cellen. De oplossing hiervoor is vrij eenvoudig: we herhalen de initiële segmentatiestap voor elke gesplitste clump met de gevonden parameters. Dit voorkomt dat de misleidende features (andere oppervlakte enz.) voor een verkeerde classificatie zorgen.



Figuur 8.4: Voorbeeld van de creatie van vertekende segmenten bij het splitsen van een clump met interne holtes.

8.1.4 Overig

Het oorspronkelijk doel van deze thesis omvatte ook het meten van enkele eigenschappen van de bekomen cellen, om zo meer te weten te komen over de link tussen deze eigenschappen en de toestand van de patiënt. Zo zou gemeten kunnen worden wat voor effecten een bepaalde therapie of operatie heeft op de conditie van de spiercellen. Dit gaat van eenvoudige dingen als de oppervlakte, omtrek, etc. tot iets minder eenvoudige dingen als het aantal gaten of scheuren in de cel, diens grootte, het aantal celkernen in de cel, de afstand tot de buren van de cel, etc. Dit implementeren zou vrij eenvoudig moeten zijn: er moet een set metrieken gemeten worden voor elke gevonden celsegment.

Wat we met dit onderzoek hebben bereikt is veruit de moeilijkste stap van het proces dat begint met een histologische afbeelding en eindigt met deze analyse van individuele cellen. Door een methode te ontwerpen die veel beter presteert dan de huidige state-of-the-art hebben we het volledig automatisch uitvoeren van het totale segmentatie- en analyseproces voor verschillende afbeeldingen praktisch haalbaar gemaakt.

Bijlagen

Bijlage A

Documentatie bij de code

De code is volledig geschreven in Java, als plugin voor ImageJ. Ze gebruikt LibSVM en Weka als externe bibliotheken. Zowel mijn project als ImageJ zelf dienen beiden gecompileerd te worden, aangezien de gebruikte bibliotheken ook bij ImageJ moeten geïncludeerd worden (weka.jar etc.). De padnamen van de afbeeldingen dienen in de code zelf te worden ingesteld.

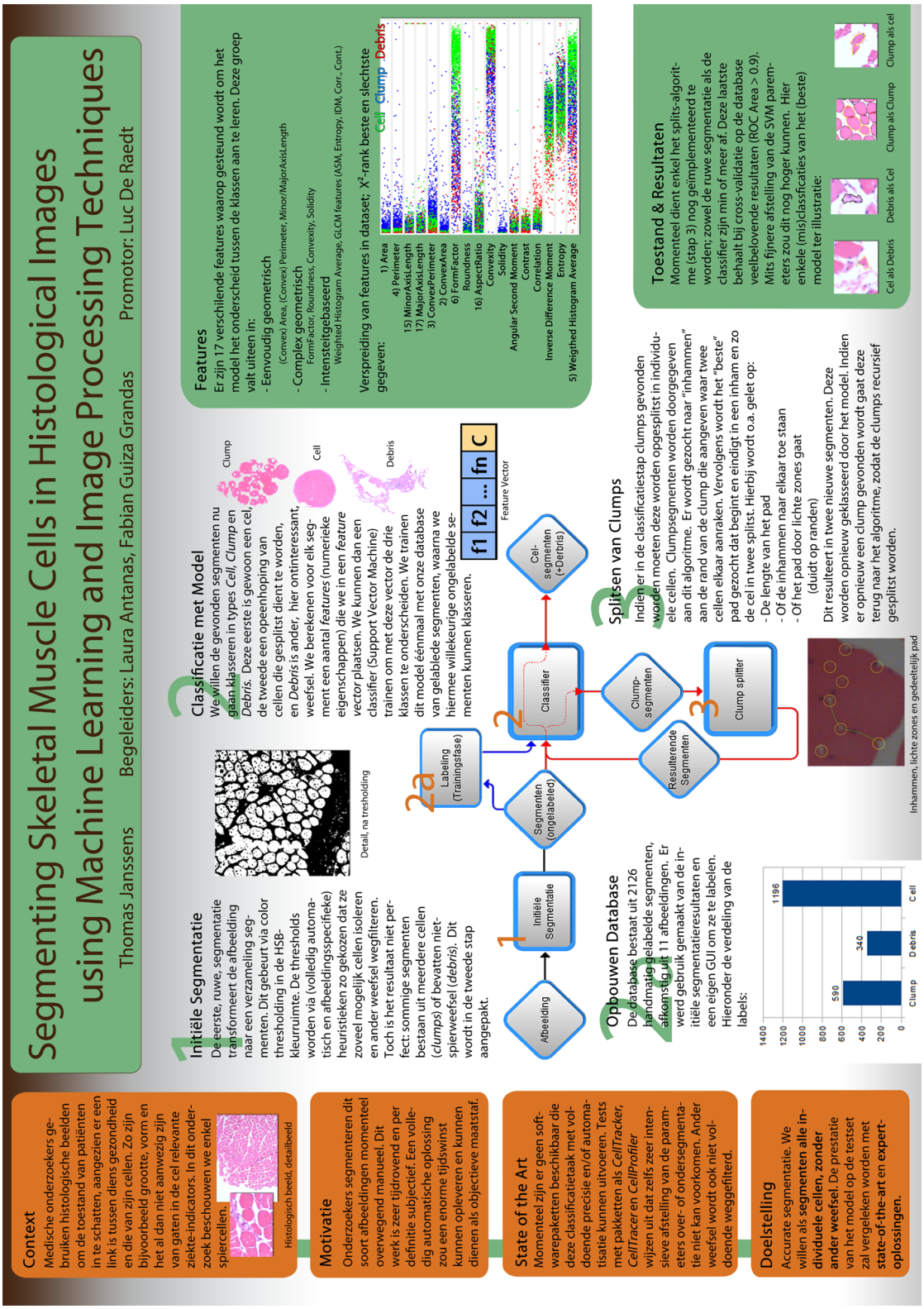
Segmentation_.java bevat het grootste deel van de code. FeatureCollect.java bevat code aangaande het berekenen van de features en aanmaken feature-vectoren. ConcavityPoint.java wat code in verband met concaviteitspunten. GLCM.java bevat code van derden om de GLCM features te berekenen. Label.java, Point.java en Segment.java zijn kleiner en houden wat informatie aangaande hun klasse bij.

Na het opstarten kan de gebruiker kiezen een afbeelding te labelen, een classifier te trainen, een enkele afbeelding te segmenteren of een batch te segmenteren. De console print een aantal statistieken uit.

Alle uitvoer komt in de map van ImageJ terecht.

Bijlage B

Poster



Figuur B.1: Poster

Bijlage C

Wetenschappelijk artikel

Bibliografie

[1] M. Abramoff, P. Magelhaes, and S. Ram. Image processing with ImageJ. *Biophotonics international*, 11(7):36–42, 2004.

[2] N. Bonnet, J. Cutrona, and M. Herbin. A 'no-threshold' histogram-based image segmentation method. *Pattern Recognition*, 35(10):2319–2322, 2002.

[3] J. E. Cabrera. Gray level correlation matrix texture analyzer, 2005.

[4] J. C. Caicedo. Features for Histology Images.

[5] A. Carpenter, T. Jones, M. Lamprecht, C. Clarke, I. Kang, O. Friman, D. Guertin, J. Chang, R. Lindquist, J. Moffat, et al. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology*, 7(10):R100, 2006.

[6] C. Chang and C. Lin. Libsvm: a library for support vector machines. 2001.

[7] F. de Assis Zampirolli, B. Stransky, A. Lorena, and F. de Melo Paulon. Segmentation and classification of histological images-application of graph analysis and machine learning methods.

[8] K. Fatima and H. Majeed. Meningioma subtype classification: A survey. In *Emerging Technologies (ICET), 2010 6th International Conference on*, pages 55–60. IEEE.

[9] E. Frank, M. Hall, L. Trigg, G. Holmes, and I. Witten. Data mining in bioinformatics using Weka. *Bioinformatics*, 20(15):2479, 2004.

[10] F. Garton, J. Seto, K. North, and N. Yang. Validation of an automated computational method for skeletal muscle fibre morphometry analysis. *Neuromuscular Disorders*, 20(8):540–547, 2010.

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[12] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[13] A. Hoover, G. Jean-Baptiste, X. Jiang, P. Flynn, H. Bunke, D. Goldgof, K. Bowyer, D. Eggert, A. Fitzgibbon, and R. Fisher. An experimental comparison of range image segmentation algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(7):673–689, 1996.

[14] C. Hsu, C. Chang, C. Lin, et al. A practical guide to support vector classification, 2003.

[15] N. Kawai, R. Sano, J. Korfage, S. Nakamura, N. Kinouchi, E. Kawakami, K. Tanne, G. Langenbach, and E. Tanaka. Adaptation of rat jaw muscle fibers in postnatal development with a different food consistency: an immunohistochemical and electromyographic study. *Journal of anatomy*, 216(6):717–723, 2010.

[16] S. Kumar, S. Ong, S. Ranganath, T. Ong, and F. Chew. A rule-based approach for robust clump splitting. *Pattern Recognition*, 39(6):1088–1098, 2006.

[17] C. Loukas and A. Linney. A survey on histological image analysis-based assessment of three major biological factors influencing radiotherapy: proliferation, hypoxia and vasculature. *Computer methods and programs in biomedicine*, 74(3):183–199, 2004.

[18] A. Matsakas, K. Foster, A. Otto, R. Macharia, M. Elashry, S. Feist, I. Graham, H. Foster, P. Yaworsky, F. Walsh, et al. Molecular, cellular and physiological investigation of myostatin propeptide-mediated muscle growth in adult mice. *Neuromuscular Disorders*, 19(7):489–499, 2009.

[19] Onbekende Auteur. Introduction to support vector machine (svm) models. *http://www.dtreg.com/svm.htm*.

[20] Onbekende Auteur. Structure of skeletal muscle. *http://commons.wikimedia.org/wiki/File:Illu_muscle_structure.jpg*.

[21] Onbekende Auteur. Support vector machine. *http://en.wikipedia.org/wiki/Support_vector_machine*.

[22] J. Pan, T. Kanade, and M. Chen. Learning to detect different types of cells under phase contrast microscopy. *Microscopic Image Analysis with Applications in Biology (MIAAB)*, 2009.

[23] S. Park, Y. Kim, S. Kim, and S. Oh. Changes in muscle fiber size and in the composition of myosin heavy chain isoforms of rabbit extraocular rectus muscle following recession surgery. *Japanese journal of ophthalmology*, 52(5):386–392, 2008.

[24] K. Rajpoot and N. Rajpoot. Wavelets and support vector machines for texture classification. In *Multitopic Conference, 2004. Proceedings of INMIC 2004. 8th International*, pages 328–333. IEEE, 2004.

[25] H. Shen, G. Nelson, D. Nelson, S. Kennedy, D. Spiller, T. Griffiths, N. Paton, S. Oliver, M. White, and D. Kell. Automated tracking of gene expression in individual cells and cell compartments. *Journal of the Royal Society Interface*, 3(11):787, 2006.

[26] A. Todman, E. Claridge, and A. Todman. Cell Segmentation in Histological Images of Striated Muscle Tissue-A Perceptual Grouping Approach. 2008.

[27] A. Tutac, D. Racoceanu, T. Putti, W. Xiong, W. Leow, and V. Cretu. Knowledge-guided semantic indexing of breast cancer histopathology images. In *BioMedical Engineering and Informatics, 2008. BMEI 2008. International Conference on*, volume 2, pages 107–112. IEEE, 2008.

[28] C. Wang. A Bayesian Learning Application to Automated Tumour Segmentation for Tissue Microarray Analysis. *Machine Learning in Medical Imaging*, pages 100–107, 2010.

[29] Q. Wang, J. Niemi, C. Tan, L. You, and M. West. Image segmentation and dynamic lineage analysis in single-cell fluorescence microscopy. *Cytometry Part A*, 77(1):101–110, 2010.

[30] G. Wilson and B. Batchelor. Convex hull of chain-coded blob. In *Computers and Digital Techniques, IEE Proceedings-*, volume 136, pages 530–534. IET, 1989.

[31] H. Wu, L. Deligdisch, and J. Gil. Segmentation of microscopic nuclear images—a review. *Recent Res. Devel. Electronics, edited by SG Pandalai (Transworld Research Network, Kerala, India, 2004) Vol*, 2:1–17, 2004.

[32] Z. Yin, R. Bise, M. Chen, and T. Kanade. Cell segmentation in microscopy imagery using a bag of local Bayesian classifiers. In *Biomedical Imaging: From Nano to Macro, 2010 IEEE International Symposium on*, pages 125–128. IEEE, 2010.

Fiche masterproef

Student: Thomas Janssens

Titel: Segmenteren van Skeletale Spiercellen in Histologische Beelden met behulp van Machinaal Leren en Beeldverwerking

Engelse titel: Segmenting Skeletal Muscle Cells in Histological Images using Machine Learning and Image Processing Techniques

UDC: 681.3

Korte inhoud:

Histologische beelden van skeletale spiercellen bevatten veel informatie over de effecten van ziektes en aandoeningen die dit weefsel aantasten. Deze afbeeldingen zijn dan ook erg interessant voor onderzoekers. Om deze informatie te bekomen dient de locatie van de individuele cellen in de afbeelding gekend te zijn (segmentatie van de afbeelding), zodat er eigenschappen van gemeten kunnen worden door software-tools. Dit opmeten gebeurt echter volledig manueel, wat een zeer tijdrovend proces is. Bestaande software-tools gericht op segmentatie zijn ontoereikend om dit soort beelden te segmenteren. In dit werk introduceren we een nieuw algoritme speciaal ontwikkeld om dit soort afbeeldingen te segmenteren. We gebruiken hiervoor een aantal technieken voor beeldverwerking en het automatisch leren van een model. We segmenteren de afbeelding eerst in een aantal segmenten. Dit kunnen individuele cellen, ander weefsel of opeenhopingen van cellen zijn. We gebruiken een getraind SVM-model om deze drie klassen te onderscheiden, waarna we de opeenhopingen met een recursieve aanpak opsplitsen in de individuele cellen. We tonen aan dat onze methode beduidend beter presteert dan de beste bestaande software-tools.

Thesis voorgedragen tot het behalen van de graad van Master in de ingenieurswetenschappen: computerwetenschappen, optieAI

Promotor: Prof. Luc De Raedt

Assessor: (Onbekend)

Begeleiders: ir. Laura Antanas
dr. Fabian Guiza Grandas