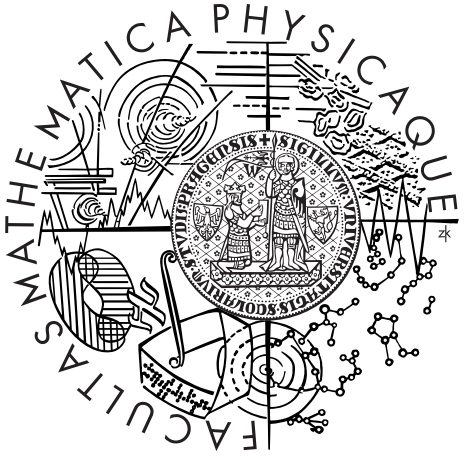
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Michal Lašan

Segmentation of cells from microscopic images

Department of Software and Computer Science Education

Supervisor of the bachelor thesis: Mgr. Jindřich Soukup
Study programme: Informatics
Specialization: Programming

Prague 2014

Hereby, I would like to thank my supervisor Jindřich Soukup and my classmate Adam Blažek for their valuable advice.

I would also like to thank the staff of Working place of tissue culture - certified laboratory at Nové Hrady, namely Monika Homolková and Šárka Beranová, for performing the manual segmentation of the cells.

I am grateful to my parents for supporting me during the studies.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, July 25, 2014 Michal Lašan

Název práce: Segmentace buněk z mikroskopických snímků

Autor: Michal Lašan

Katedra: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Jindřich Soukup

Abstrakt: V této práci prezentujeme novou metodu na automatickou segmentaci savčích rakovinových buněk z časozběrných snímků pořízených mikroskopem na bázi fázového kontrastu. Tato metoda je sledem kroků založených na základních technikách z oblasti zpracování obrazu, matematické morfologie a teorie grafů. Její hlavní myšlenkou je využít přítomnosti halo artefaktů kolem buněk, díky nimž jsou hranice mezi buňkama světlejší než zbytek obrázku. Navazuje na metodu navrženou Jindřichem Soukupem, která umí oddělovat buňky od pozadí. Srovnáme tuto metodu s watershed - věřejně dostupným algoritmem z odvětví matematické morfologie. Jako referenci použijeme segmentaci lidským expertem. Prezentována metoda je implementována v MATLABu a Javě s jednoduchým a intuitivním rozhraním. Také připojujeme přímočarý editor segmentace, pomocí něhož užívatel může napravit nepřesnosti segmentace, nebo dokonce vytvořit svou vlastní manuální segmentaci.

Klíčová slova: Segmentace buněk, Mikroskopie na bázi fázového kontrastu, Analýza medicínských snímků

Title: Segmentation of cells from microscopic images

Author: Michal Lašan

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Jindřich Soukup

Abstract: In this thesis, we present a new method for the automatic segmentation of mammalian cancer cells from time-lapse images obtained by a microscope based on phase contrast. This method is a pipeline composed of basic techniques from the field of image processing, mathematical morphology and the theory of graphs. Its main idea is to utilize the presence of halo artifacts around the cells, which cause the boundaries between the cells to be lighter than the rest of the image. It follows up to the method proposed by Jindřich Soukup which is able to separate the cells from the background. We compare this method to the watershed - a publicly available algorithm from the field of mathematical morphology. We use segmentation by a human expert as the ground truth. The presented method is implemented in MATLAB and Java with a simple and intuitive interface. We also attach a straightforward GUI editor of segmentation written in Java, with the help of which a user can correct imprecisions in the segmentation, or even create their own manual segmentation.

Keywords: Cells segmentation, Phase contrast microscopy, Medical image analysis

Contents

Introduction

For research purposes, biologists create time-lapse microscopic image series of living cells (see Fig. 1). Such records allow them to analyze the behavior and characteristics of the observed cells over time. They are interested in the count of the cells, their shape, movement, reproduction, etc. An analysis of these aspects can be useful. For example, it can help evaluate the efficiency of a chemical substance, the purpose of which is to kill the cells or to reduce their reproduction rate [1].

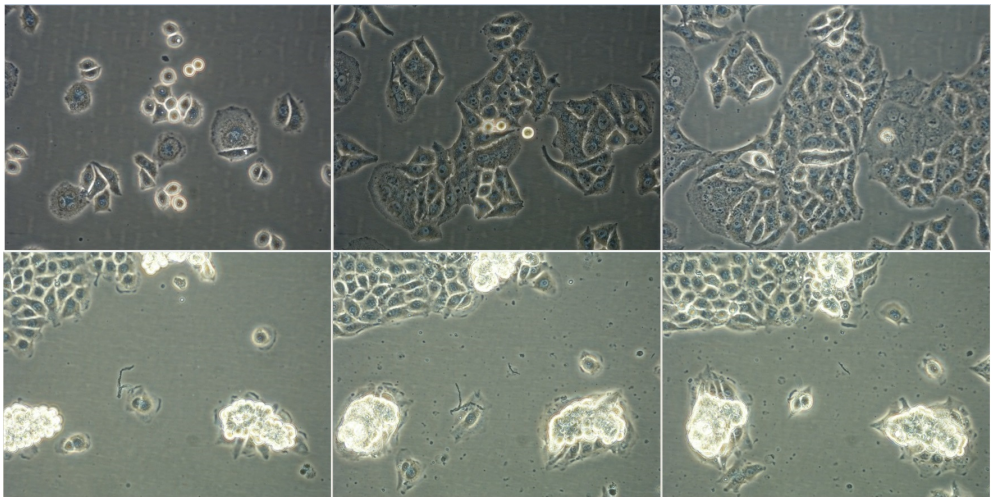


Figure 1 Snapshots from some series

To obtain this information from the images, *segmentation* of individual cells is unarguably needed. *Segmentation* is a process, the aim of which is to label individual cells in the image and for each cell accurately specify the area in the image occupied by it. The area of a cell can be specified by an enumeration of the pixels which belong to it[2]. Refer to Fig. 2 for a visualization of a result of *segmentation*.

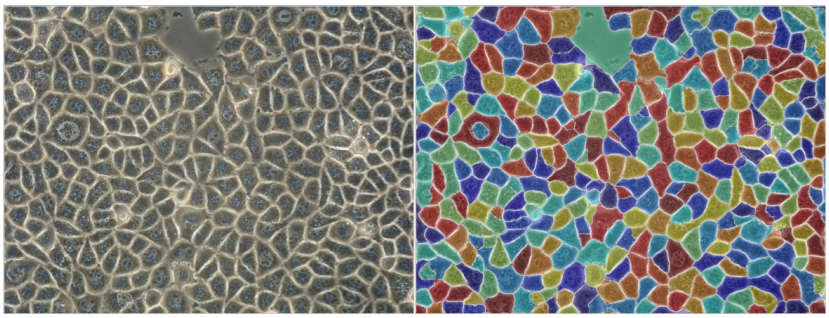


Figure 2 An example of segmentation

Segmentation of an image can be obtained in basically two ways:

[1] When dealing with cancer cells, such a substance could be then used to cure cancer, if it is proven effective. At the same time, it should damage regular cells as little as possible.

[2] We suppose the images are digital. Analogue images cannot be processed this way - they need to be digitalized first.

- **Manual labeling performed by a human expert.**

  It is the most accurate way. However, it is quite a tedious and time-consuming work.

- **Automatic labeling performed by a computer.**

  Generally, such segmentation is less accurate. The results may vary depending on the chosen algorithm and on the type of the cells in the image. An automatic method tends to be more prone to imperfections of the image and many other factors. By far, to our best knowledge, no universally successful segmentation algorithm[3] has been proposed.

Despite the numerous disadvantages, computer processing of microscopic images is becoming more and more common. Even though less precise and not universal, automatic segmentation is much faster than manual. After all, one can always use an automatic method only for the images, where it works sufficiently.

In this thesis, we describe a new segmentation method which is focused on processing images of live mammalian cancer cells obtained with a phase contrast microscope [25] (see Fig. 1). These images are characteristic by the frequent presence of so-called *halo artifacts* near the borders of cells. This causes the borders to be typically lighter than cells and the background. The image series for this work were provided by Working place of tissue culture - certified laboratory at Nové Hrady. The experts from this place also performed manual segmentation of a few images for us.

Our method takes advantage of the *halo artifacts* and its success depends on their presence. The method is a fusion of subtle modifications of known basic methods. It utilizes them in a non-traditional way. We are aware of the imperfections of this method. This is why we include a straightforward GUI[4] tool, with the help of which a user can easily correct the inaccuracies.

Structure

At first, we describe the basic concepts of image processing related to this thesis. In Chap. 1, we discuss the current related methods. We mention their main concepts and compare them to our method. We also explain how this method follows up to the method proposed by Jindřich Soukup [23], the supervisor of this thesis.

In Chap. 2, we explain why we decided to develop a new method. We give reasons for the choice of our approach. In Chap. 3, we explain the main idea of the method. In Chap. 4, we complete the description of the method by explaining its modifications and additional steps. In Chap. 5, we describe how we evaluated the resulting segmentation. In Chap. 6, we evaluate our method together with the watershed method, present the results and compare them. In Chap. 7, we sum up the advantages and disadvantages of our method. We present our future perspective.

Attachment 7.1, contains a CD with a GUI program enabling a user to segment images of cells by the proposed method. It includes a subprogram for manual

[3] A method which would work with any type of cells, under any viewing conditions.

[4] Graphical user interface.

editing of the segmentation, alternatively for performing manual segmentation. The CD contains a user manual for each one of the components.

Key contributions

The key contributions of this thesis are:

- We propose and describe in detail a new method based on simple concepts for segmentation of individual cells from images of live mammalian cancer cells obtained with a phase contrast microscope.
- We compare the new method to the segmentation performed by a human expert. For this purpose, we develop an evaluation metrics. Then we evaluate the publicly available watershed algorithm [2] the same way and compare its results with the results of our method.
- We introduce a tool with an easy-to-use GUI for manual correction of segmentation.

Preliminaries

In this chapter, we summarize the theory from the field of image processing which we will need in this thesis. Some of the later mentioned terms might also have a bit different meanings in another contexts. We only focus on the definitions related to this thesis.

Firstly, we introduce the basic terminology and conventions used in this thesis. Then, we describe *Convolution* and explain how it can be used to blur an image. We explain the term *Thresholding* and mention a widely used thresholding method - *Otsu's method* [20]. We describe the principle of *Skeletonization*. Then we briefly describe the *watershed algorithm* [2] - a simple, publicly available method for digital image segmentation.

We describe very briefly the main idea of *machine learning* and mention some of the state-of-the-art algorithms from this field. We also mention the key principles of some more image processing methods and concepts. We provide links to external sources for detailed explanations of more sophisticated concepts. Lastly, a detailed explanation of Dijkstra's algorithm [8] is provided.

Terminology and conventions

A *gray-scale digital image* is a rectangular mesh of *pixels*, every one of which has a single value. In image processing, it can be viewed as a two-dimensional real function - $f : \mathbb{Z}^2 \longrightarrow \mathbb{R}$ with a finite domain. A color image is then perceived as three such functions; one for each color component (red, green, blue). In this thesis, we denote the set of gray-scale images as $F_{\mathbb{Z}^2 \to \mathbb{R}}$. For an image f from this set, $f(x,\ y)$ denotes the value of its pixel at the position (x, y), where x, y are from the domain of f. We display these images, so that the higher the value of a pixel, the lighter it appears (see Fig. 5).

A digital image, every pixel of which has the value of either 1 or 0, is called a *binary digital image*. It can be viewed as a special case of the function mentioned above - $f : \mathbb{Z}^2 \longrightarrow \{0, 1\}$. In this thesis, we denote the set of binary digital images as $F_{\mathbb{Z}^2 \to \{0,1\}}$. We display such images similarly to the previous case: the pixels with the value of 0 are black and the pixels with the value of 1 are white (see Fig. 6b). However, the pixels with the value of 1 may have a different color, when two binary images with the same domain are displayed together (see Fig. 7). In such cases, we specify the color of the pixels with the value of 1 for every displayed binary image.

Let b_1, b_2 be binary images and let g be a gray-scale image (it can be binary, too). Let all these images have the same domain. All the semi-products of the method presented in this thesis have the domains equal to each other. We shall define some terminology which will help us express ourselves more efficiently while describing the method:

- We say, that a certain pixel p from the domain of g is *in* b_1, if the position of p at g is (x, y) and $b_1(x,\ y) = 1$.
- We define the operations $\cup, \cap : F_{\mathbb{Z}^2 \to \{0,1\}} \times F_{\mathbb{Z}^2 \to \{0,1\}} \longrightarrow F_{\mathbb{Z}^2 \to \{0,1\}}$. Let $union = b_1 \cup b_2$ and $inter = b_1 \cap b_2$. Then, the images *union* and

inter have the same domain as b_1 and b_2 and for (x, y) from this domain, the following applies: $union(x,\ y) = sup(b_1(x,\ y),\ b_2(x,\ y))$ and $inter(x,\ y) = inf(b_1(x,\ y),\ b_2(x,\ y))$.

Additionally, for the purpose of visualization of the mutual coherence of b_1 and g, we may *superimpose* b_1 *on* g. The superimposing is performed as a suitably weighted summation of the intensities in b_1 and g, so that the pattern in b_1 is clearly visible atop the underlying g (see Fig. 6c). Moreover, b_1 and b_2 may be displayed together (as described before) and superimposed on g.

Let us introduce some definitions related to the neighborhoods of pixels. They simplify the description of our method:

- We say, that two pixels at $(x_1,\ y_1)$ and $(x_2,\ y_2)$ are *8-connected*, if they are neighbors (can also be diagonal) - more formally, if $|x_1 - x_2| \leq 1$ & $|y_1 - y_2| \leq 1$.
- We say, that two pixels at $(x_1,\ y_1)$ and $(x_2,\ y_2)$ are *4-connected*, if they are neighbors, but not diagonal - more formally, if they are *8-connected* and $|x_1 - x_2| + |y_1 - y_2| \leq 1$.

The term *8-connected* comes from the fact, that a non-border pixel has eight *8-connected* neighbors. Analogically, such a pixel has four *4-connected* neighbors. The following definitions are all related to binary images:

- We call a series of pixels in a binary image a *k-connected path*, if their values are equal to each other and every consecutive pair of them is *k-connected*, where $k \in \{4, 8\}$.
- We call a set of pixels in a binary image a *k-continuous area*, if their values are equal to each other and for each pair of them, there exists a *k-connected path* starting at the first pixel of the pair and finishing at the second one, where $k \in \{4, 8\}$.
- A *k-continuous area* in a binary image is called *maximal*, if there does not exist any pixel which does not belong to the area and can be added there, so that the area remains *k-continuous*.
- A *maximal 4-connected area*, the pixels of which have the value of 1, is called an *object*. If its pixels have the value of 0, it is called a *hole*.
- The set of all pixels with the value of 1 is called a *pattern*.

Let us mentioned some exceptions to this terminology:

Our method creates 8-connected paths in the role of the boundaries between the cells. We say, that the holes between them are objects segmented by them. For a binary image b, we often say "b" instead of "the *pattern* in b".

Convolution

One-dimensional *convolution* is an operation which is given two real functions and returns a real function. Let $*$ be the symbol representing this operation. Let $F_{\mathbb{R}\to\mathbb{R}}$ be the symbol representing all one-dimensional real integrable functions. Then we write:

$$* : F_{\mathbb{R}\to\mathbb{R}} \times F_{\mathbb{R}\to\mathbb{R}} \longrightarrow F_{\mathbb{R}\to\mathbb{R}} \tag{1}$$

Let f, g be one-dimensional real functions: $f,\ g : \mathbb{R} \longrightarrow \mathbb{R}$. This is the formula for the value of *convolution* of f and g at the point x:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)\ \mathrm{d}t \quad for\ x \in \mathbb{R} \tag{2}$$

We can imagine this as follows: To compute $(f * g)(x)$, we perform these steps:

- We "flip" g around the y *axis*; formally, we create a new function $g\prime$, such as:

$$g\prime(t) = g(-t) \tag{3}$$

- We "shift" $g\prime$ to x; formally, we create a function $g\prime\prime$ (see Fig. 3a), such as:

$$g\prime\prime(t) = g\prime(t-x) \tag{4}$$

- We create a new function: a point-wise multiplication of f and $g\prime\prime$ and name it m (see Fig. 3b):

$$m(t) = f(t)g\prime\prime(t) \tag{5}$$

- We integrate m along $\mathbb{R}$ to get the value of the resulting function at x:

$$(f * g)(x) = \int_{-\infty}^{\infty} m(t)\ \mathrm{d}t \tag{6}$$

Thus, $(f * g)(x)$ is a weighted average of the neighborhood of x.

Blurring a gray-scale image

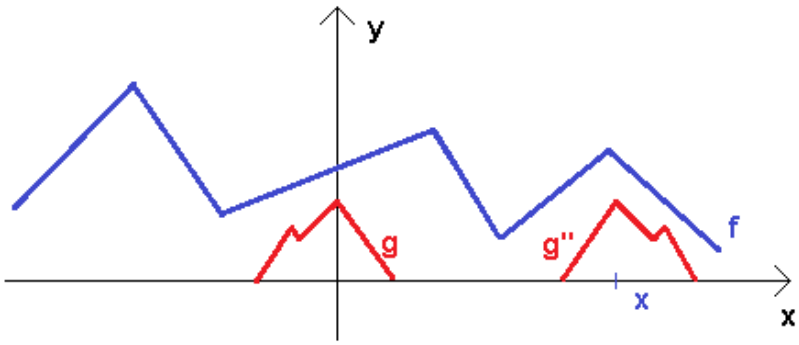
In Fig. 3b, $f * g$ is "a smoothened f". This reflects one possible use of *convolution* - to smoothen a function. *Convolution* can also be used for two-dimensional real functions with discrete domains:

$$* : F_{\mathbb{Z}^2\to\{0,1\}} \times F_{\mathbb{Z}^2\to\{0,1\}} \longrightarrow F_{\mathbb{Z}^2\to\{0,1\}} \tag{7}$$

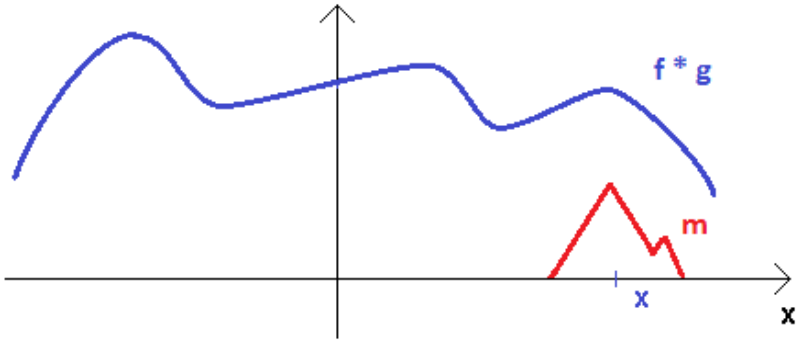
The value of *convolution* of f and g at the position (x, y) is now determined by the following formula:

$$(f * g)(x,\ y) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f(u,v)g(x-u, y-v) \quad for\ x, y \in \mathbb{Z} \tag{8}$$

If we consider f as a digital image, g is then called a *convolution core*. *Convolution* does the following thing: it sets the value of each pixel in f to a weighted



(a) Flipping g around the y axis followed by shifting it to the point x. The result of this operation is g''.



(b) The product of the point-wise multiplication of f and g'' (m) and the final $f * g$.

Figure 3 An illustration of computing $(f * g)(x)$. The value of the functions are 0 outside the lines denoting their values.

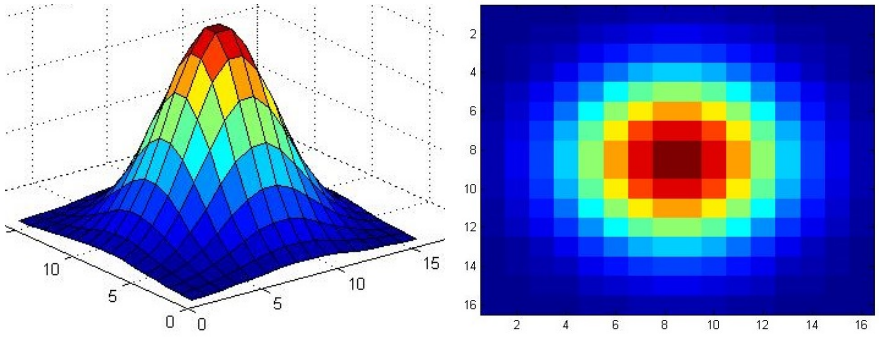
sum of the values of its neighbors. The weights and the size of the neighborhood are determined by g. We define the *size* of the *convolution core* as the maximum distance from the origin: (0, 0) to its pixel with a non-zero value. In practice, the *convolution core* is often a lot smaller than the image.

A digital image has a finite domain, which introduces the problem, how to treat the area out of its domain during the *convolution*. One of the appropriate solutions is to symmetrically extend the image. The amount of the extension in pixels must be greater than or equal to the *size* of the *convolution core*. In Fig. 5c, you can see a greatly symmetrically extended image from Fig. 5a.

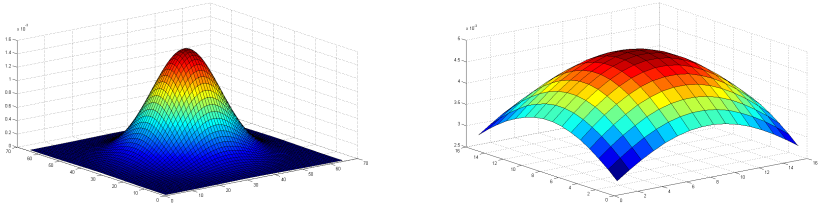
The shape of the *convolution core* influences the result of the *convolution*. If we choose an appropriate shape, we get a blurred image. One of such shapes is the well-known *Gaussian function*. This function is symmetric around the origin, so it produces an omni-directional blur. As you can see in Fig. 5b, such blurring rids the image of a certain amount of noise and the inner structure of cells. In Fig. 4, you can see an example of the *Gaussian core*. In these visualizations, the function is not centered at the origin. However, to make the *convolution* work properly, the *Gaussian core* must be centered at the origin, otherwise it not only blurs, but also shifts the image.

The Gaussian core has one parameter - *variance*. The higher the variance, the *larger* the core is and the more substantial blurring is performed (Fig. 4c).

Alternatively, this core can have another parameter - *diameter*. Here is the purpose of it: $g(x,\ y)$ is set to 0 in the places where $x > diameter/2$ or $y > diameter/2$. In other words, a smaller symmetric segment of the Gaussian function is cut out (Fig. 4d). We utilize a core similar to this in our method.



(a) A spatial view. **(b)** A plane view (from above).



(c) A core with a higher *variance*. **(d)** A core with the same *variance* as 4c, but a smaller *diameter*.

Figure 4 Examples of the *Gaussian convolution core*.

Thresholding

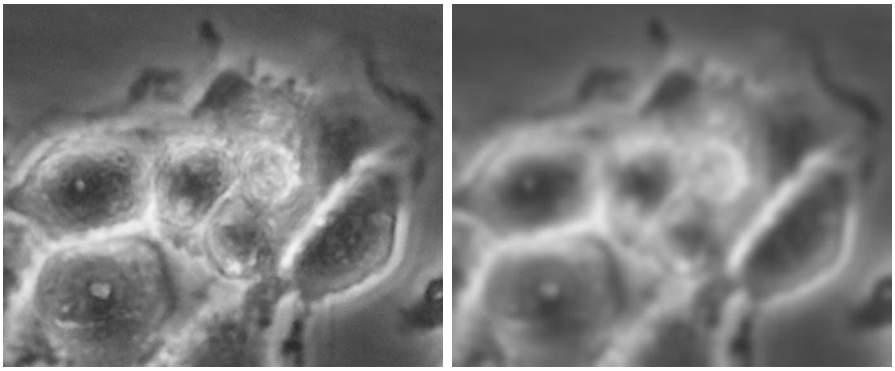
Thresholding is a transformation of a gray-scale image to a binary image. Formally:

$$T:\ F_{\mathbb{Z}^2\to\mathbb{R}} \times \mathbb{R} \ \longrightarrow\ F_{\mathbb{Z}^2\to\{0,1\}}, \tag{9}$$

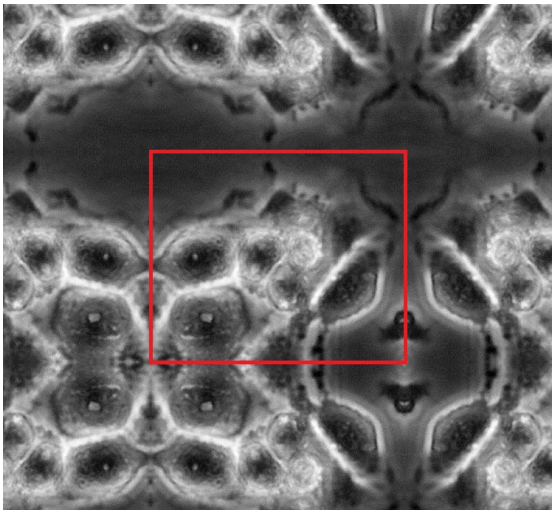
where T is the symbol for *thresholding*. It has two input variables. The first one of them is a gray-scale image and the second one of them is a *threshold*. Let f be the input image, let t be the *threshold*. The output binary image b is constructed as follows:

$$\begin{aligned} b(x,\ y) = 0 \quad &\Leftrightarrow \quad f(x,\ y) \leq t \\ b(x,\ y) = 1 \quad &\Leftrightarrow \quad f(x,\ y) > t \\ &\quad\ for\ x, y \in \mathbb{Z} \end{aligned} \tag{10}$$

There are several strategies how to choose the value of the *threshold*. It can be chosen either manually or automatically. The most common automatic solution is *Otsu's method* [20]. As we can see in 10, *thresholding* separates the pixels into two classes. The aim of *Otsu's method* is to find the value of the *threshold* which minimizes the sum of the variances of the original values of the pixels inside these two classes. Formally:



(a) The original image. **(b)** The blurred image.



(c) The symetrically extended original image; the red frame delimits the original part.

Figure 5 Blurring a gray-scale image with the *Gaussian core*.

$$\min_{t\in\mathbb{R}} \; var(\{f(x,y) \mid f(x,y) \leq t\}) + var(\{f(x,y) \mid f(x,y) > t\}) \tag{11}$$

Thresholding can be used for isolating the halos from the image. See Fig. 6 for a demonstration of the result of *Otsu's method*.

Mathematical morphology

Mathematical morphology is a field studying shapes. Its methods can also be applied to patterns in binary images.

The basic operations of *mathematical morphology* are:

- **Erosion** - setting the value of every pixel to the minimum value among its *neighborhood*. This operation "deflates" *patterns*; it decreases their size.
- **Dilation** - setting the value of every pixel to the maximum value among its *neighborhood*. This operation "inflates" *patterns*; it increases their size.

(a) The gray-scale image. **(b)** The thresholded image. **(c)** The thresholded image superimposed on the original.

Figure 6 A part of an image before and after *thresholding* using *Otsu's method*.

The mentioned *neighborhood* does not necessarily have to consist only of the closest neighbors of a pixel. It can be larger or smaller than that. The larger the neighborhood, the more significant these operations are. In the following sections, we describe some more concepts and algorithms related to *mathematical morphology*.

The basic measures of shape

A *measure of shape* is a function f:

$$f : F_{\mathbb{Z}^2 \to \{0,1\}} \longrightarrow \mathbb{R} \tag{12}$$

It computes the value of a certain characteristics of the pattern in the given binary image. In this thesis, we refer to the following measures of shape:

- **Area** - the number of pixels in the pattern.
- **Circularity** - let S_C be the area of the part of the pattern which lies in the circle centered at the pattern's centroid with the *area* equal to the *area* of the pattern. Then, *circularity* is the ratio of S_C to the area of the pattern. Thus, it is a number in the range $[0, 1]$.

Skeletonization

Skeletonization is a transformation of a binary image. Formally:

$$T : F_{\mathbb{Z}^2 \to \{0,1\}} \longrightarrow F_{\mathbb{Z}^2 \to \{0,1\}}, \tag{13}$$

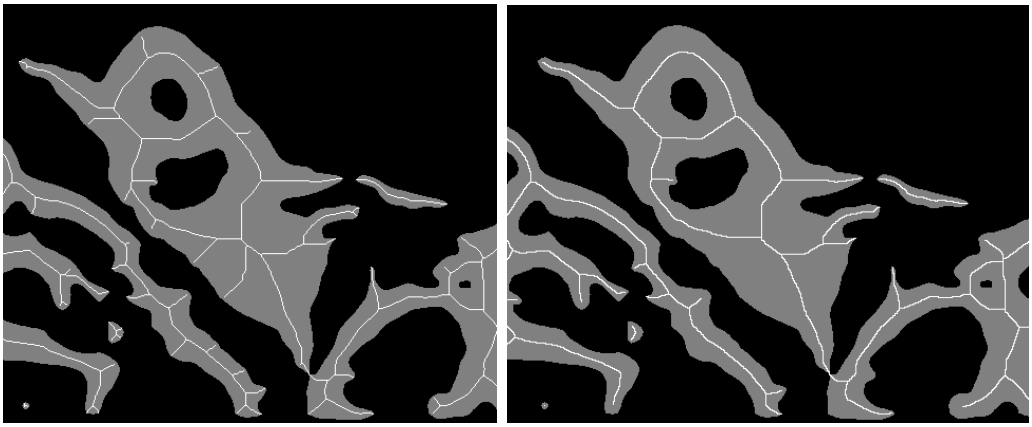
where T is the symbol for *skeletonization*. As the name suggests, its aim is to create a *skeleton* of the pattern in the given image. There are many definitions of *skeleton*.

We shall mention an informal, intuitive definition inspired by Harry Blum's paper [3]:

Let us imagine that we set the border of the pattern to fire, all parts of it at once. Let us suppose, that all parts of the pattern are flammable and nothing else

is flammable. Let us suppose that the speed, with which the fire progresses through the pattern is the same in all its places at every moment and it is more than zero. Then, the skeleton consists of the points, where at least two distinct waves of the fire meet.

One of reasonably appropriate and computationally fast approaches to *skeletonization* is to iteratively peel pixels off the pattern until there is only a thin line of it left [14]. The algorithms based on this approach may differ in the way they peel off the pixels. Accordingly, the skeletons they produce may differ too. In Fig. 7 we can see an example of such a difference. The skeleton produced by Hilditch's algorithm [12] seems to be more compliant with the mentioned intuitive definition of *skeleton* than the skeleton created by the method available in MATLAB [13]. The output of Hilditch's algorithm runs along the main course of the pattern more accurately. Both methods create 8-connected skeletons.



(a) MATLAB 2013a - bwmorph('skel'). **(b)** Hilditch's algorithm.

Figure 7 An example of the results of two different *skeletonization* algorithms. The pattern is gray, its *skeleton* is white.

The watershed algorithm

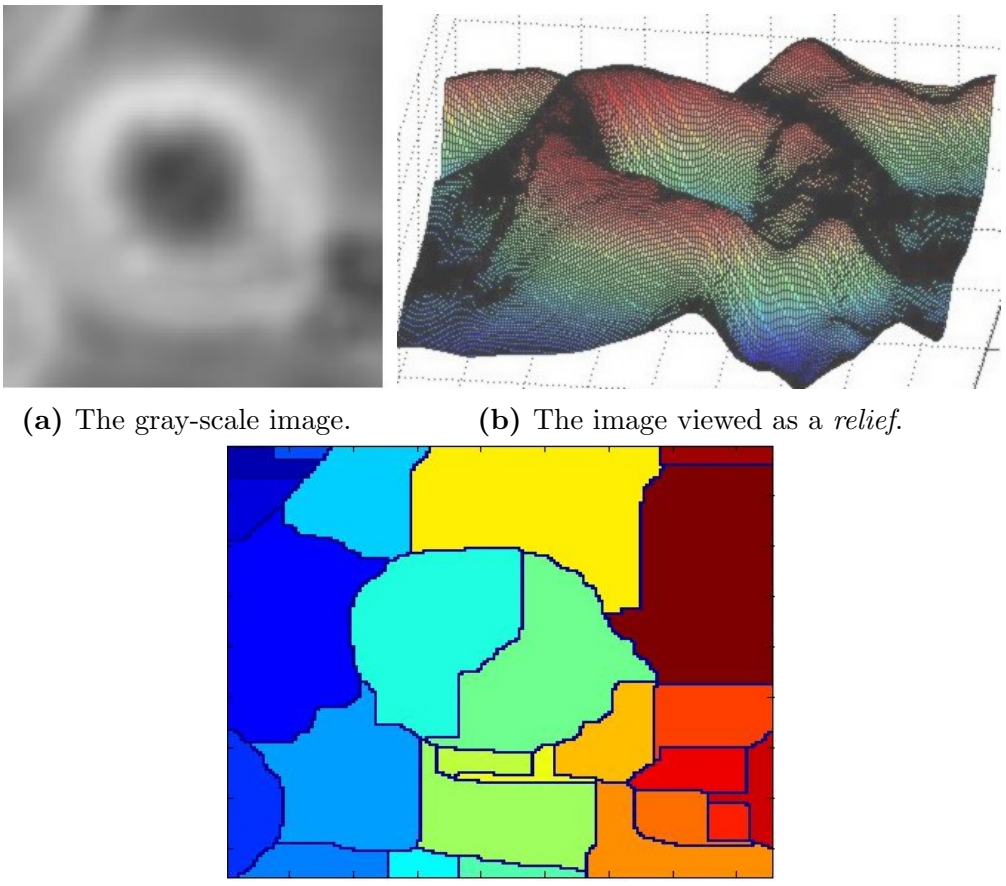
The *watershed algorithm* [2] is a basic, publicly available method for segmentation of gray-scale digital images. It views an image as a *relief* (Fig. 8b), which is then flooded by gradually rising *water*.

This method firstly sets so-called *seeds* to the local minima of the *relief*. The *water level* is set to the height of the lowest *seed* (equivalently the lowest point of the whole relief). From then on, the *water level* increases step after step - in every step, it is set to the height of the lowest point which has not been flooded yet. When *lakes* from two distinct *seeds* meet, a *boundary* is build at the point where it has happened. The resulting segmented areas are delimited by these *boundaries*.

This algorithm is very sensitive to noise and the inner structure of cells. As we can see in Fig. 8c, the demonstrative image is vastly *oversegmented*[5] by this method, even though it is blurred, and thus rid of some amount of noise and

[5]Cells are split into several segmented areas (see Chap. 5).

inner structure of the cell. A more substantial blurring might solve this problem. However, it might cause *undersegmentation*[6], as some information related to boundaries between cells might be lost as well.



(a) The gray-scale image. **(b)** The image viewed as a *relief*.

(c) Its watershed segmentation.

Figure 8 A demonstration of the watershed algorithm.

Machine learning

The main idea of *machine learning* can be described as follows:

Let us consider a function f:

$$f : \mathbb{R}^n \longrightarrow \mathbb{R} \tag{14}$$

This function is given a vector of *features* as the input. The *features* can be for example some data from a certain sector of an image - pixel intensities, gradients, etc. The object represented by the *features* is called a *sample* (the sector of an image in this case).

This function outputs a single real number: a *verdict*. The *verdict* can be for example an indicator, whether there is a cell center at the given sector of an image, in which case the value of the *verdict* might be 0 or 1. More generally, if we want to divide the *samples* into a finite set of classes, we are facing a *classification* problem and thus f is called a *classifier*.

[6]The segmented areas contain several cells (see Chap. 5).

The problem is, that we do not know the function. However, we have some pairs (*features, verdict*) available - these form the *training set*. The goal of *machine learning* is to use the *training set* to construct the function f, so that when f is given a previously unknown vector of *features*, it outputs a *verdict* which is as close to the truth as possible.[7] The process of constructing f is called *training the classifier*.

A *machine learning* algorithm describes, how the function f is constructed from the *test set*. Some of the main state-of-the-art algorithms are:

- **Neural networks** [19] - a concept inspired by the structure of the human brain; f is constructed with the help of an intricate propagation of the input information (*features*) through mutually interconnected *neurons*
- **Support vector machines** [7], [4] - a concept using geometrical methods.
- **Conditional random fields** [17] - a concept which takes into account the neighboring *samples* when evaluating the verdict for a *sample*.
- **AdaBoost** [22] - a method based on combining the *verdicts* of several *machine learning* algorithms into a single, more precise *verdict*.

More image processing methods and concepts

In this section, we list the rest of the image processing methods and concepts which are mentioned later in this thesis. The products of the first two of them are often used as inputs to machine learning algorithms.

- **Local binary patterns** [10] - ways how to store the information about the neighborhood of a pixel. They are usually used for pattern recognition.
- **Wavelet transform** [5] - a method which decomposes an image into several images, from which the original image can be reconstructed. Based on the way this decomposition is performed, the products of it can contain various information - for example, information about edges or pattern.
- **Deconvolution** - let $F = f * g$, where f is an image function and g is a convolution core. Let us assume that we know F and g, but we do not know f. Then, *deconvolution* is the operation which is able to restore f. For example, we can restore the original image from a blurred one, but only if we know the shape of the convolution core which blurred it, which is rarely the case in practice.
- **Active contour model** [16] - a method which physically simulates movement of thin elastic material, mostly in order to determine the boundary of an object. The desired result is that the material wraps itself around the object.

[7] More exactly, this is called *supervised learning*. However, we will not go into more detail on this topic.

Dijkstra's algorithm

In this section, we explain, how this algorithm [8] works and enumerate its main procedural components and the data structures it uses. The main purpose of this description is to adopt a consistent terminology regarding the algorithm. This will allow us to comprehensibly introduce some modifications to it further in this thesis.

This is an algorithm for finding the cheapest paths in an oriented graph with priced edges from a certain node to all other nodes. It accomplishes this goal by a smart successive visiting of the nodes, beginning at the given starting node. It maintains a *list* of the nodes to visit. Each node is assigned a so-called *path record* when inserted into this *list*. The *path record* consists of:

- The *price* of the path leading to the node.
- A link to the *previous node* on the path leading to the node.

From the backward links, the path leading to the node can be reconstructed.

The next node to visit is the node from the *list* with the lowest *price* in its *path record*[8]. It is a proven fact, that as long as the prices of the edges in the graph are non-negative, the path to the next node to visit is the cheapest one from the starting node to this node. Thus, at the moment when the path to the current next node to visit is all we want to know, we can finish. Such a statement can be encapsulated in the *end condition* of the algorithm.

During the visit of a node, its neighbors are updated, which might include *assigning new path records* to them (see Algorithm. 1) or adding them to the *list*. This is called *updating neighbors* (see Algorithm 2). However, no *updating of neighbors* takes place, if the *continuing condition* is not satisfied. This condition is always satisfied in the basic version of the algorithm, but we include it in this description, because we will need it later on. Algorithm. 3 manifests Dijkstra's algorithm, using the formerly introduced terminology.

Algorithm 1 Procedure *assignNewPathRecord*(current_node, previous_node, new_price)

```
current_node.createNewPathRecord()
current_node.path_record.price ← new_price
current_node.path_record.previous_node ← previous_node
```

[8]For this reason, it is effective to implement this *list* as a heap.

Algorithm 2 Procedure *updateNeighbors*(visited_node, *list*)

```
for each neighbor of visited_node do
    if neighbor ∉ list then
        list.insert(neighbor)
    end if
    price_from_visited ←
        visited_node.path_record.price +
        priceOfEdgeBetween(visited_node, neighbor)
    if price_from_visited < neighbor.path_record.price then
        assignNewPathRecord(neighbor, visited_node, price_from_visited)
    end if
end for
```

Algorithm 3 Dijkstra's algorithm

```
for each node do
    node.path_record ← null
end for
assignNewPathRecord(starting_node, null, 0)
list.add(starting_node)
while list.nonEmpty() do
    visited_node ← list.pollNodeWithMinimumPrice()
    if endCondition(visited_node) then
        return;
    end if
    if continuingCondition(visited_node) then
        updateNeighbors(visited_node, list)
    end if
end while
```

1. Related work

In the latest related work, several segmentation strategies are employed.

Wang et al. [11] firstly use a support vector machine classifier fed by pixel intensities, gradients and local binary patterns to separate the cells from the background. Then they use the watershed method to separate individual cells. To reduce the oversegmentation caused by this method, they detect the centers of the cells with the help of an AdaBoost algorithm fed by features based on the wavelet transform.

Pan et al. [21] have developed an algorithm based on a conditional random field, so that it is capable of separating individual cells.

The two mentioned methods have been tested on images containing cells attached to each other. To the contrary, the following two methods have only been tested on images, where there is at least a small gap between every pair of cells. Hence, it remains unclear, how they would succeed on more intricate data.

Li et al. [18] use a combination of mathematical morphology and an active contour model to segment individual cells.

Yin et al. [24] use advanced deconvolution methods to get rid of halos and thus separate individual cells. To compute the necessary convolution core, they exploit the characteristics of the phase-contrast microscope. The cells are subsequently segmented by thresholding, using Otsu's method.

The existing related methods still have some disadvantages and thus proposing alternatives might be helpful. It can also help others by clarifying, how well such an approach works.

Our method is conceptually much simpler than all the mentioned methods. Unlike the last two methods, it has also been tested on images, where the cells are attached to each other. In contrast to the fourth method, it does not require any precise knowledge of the used microscope and takes advantage of the halos. Differently from the first two methods, it is not based on machine learning. It does not require any training set and it is less computationally demanding.

On the other hand, it means that it is much less adaptable to impurities or uneven distribution of illumination thorough the image. It is dependent on the presence of halos. It relies on the assumption that the interiors of the cells are darker than the halos, so light points inside the cells might cause its failure. It does not take advantage of colors, nor the inner structure of the cells, nor the shape of the cells. The independence of the color, structure and shape can be advantageous - the method is theoretically more universal. However, a lot of information is ignored this way. It is also dependent on a good setting of its parameters.

However, due to its simplicity, it could be used in connection with a more sophisticated method, just like the watershed method is used in Wang's approach [11]] together with a machine learning method.

The following two methods only focus on the separation of groups of cells from the background; equivalently the detection of cell clusters. Their approaches differ a lot.

Soukup et al. [23] take advantage of the time-lapse character of the image series to separate the cells from the background. They observe that the cells almost

always move in between subsequent images. Thus, they compute the difference of such images to identify the areas, where the cells are located. This method also relies on parameters - the sensitivity of response (resistance to noise) and the span between the images, the difference of which is computed (depends on how quickly the cells move).

Ersoy et al. [9] use geometric and active contour methods to separate the cells from the background.

Our method only focuses on segmentation of individual cells. It does not deal with separating the cells from the background. Quite the opposite, it presumes, that such a separation has already been performed and expects the result of it at the input. It uses the results of Soukup's [23] method.

2. Analysis and methodology

In this chapter, we explain the reasons which led us to our approach. We describe the process of creating the proposed method.

2.1 Analysis

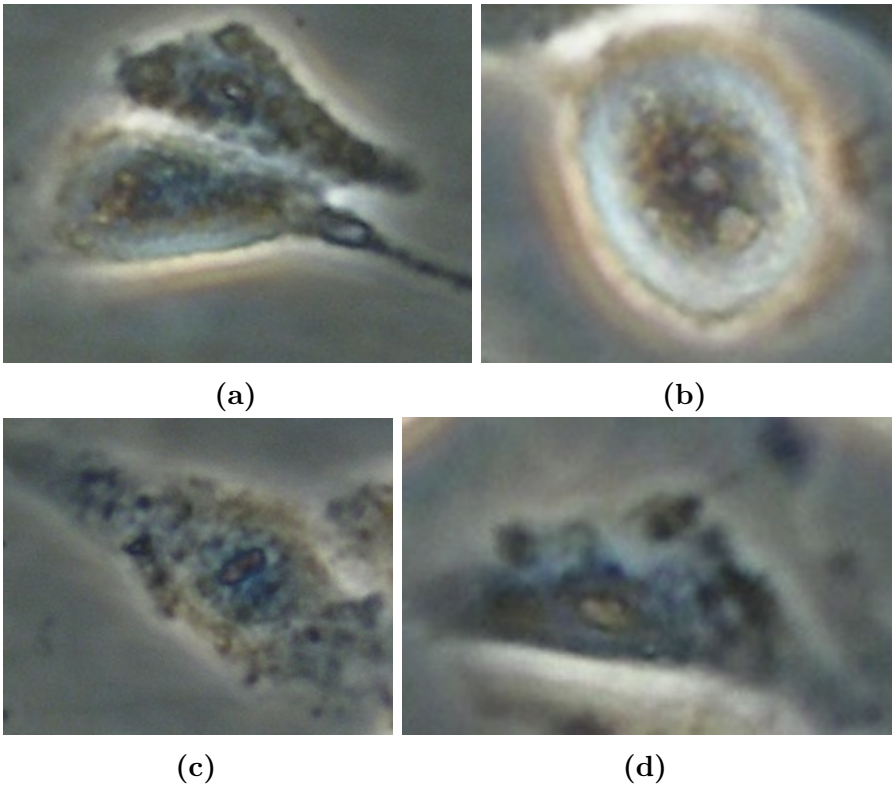


Figure 2.1 A demonstration of various appearance of the cells.

As we can see in Fig. 2.1, the cells appear in various shapes, colors and have diverse inner structure. The quality of the images is not optimal - the cells tend to be a bit blurred. Probably the only aspect which they have in common is the frequent presence of halos near their borders.[1] This observation led us to the conviction, that we were going to take advantage of it. We decided to focus on detecting boundaries between the cells with the help of halos and not to take advantage of color, shape and structure of the cells, since these characteristics vary a lot in the images and the quality of the images is far from optimal. Moreover, we wanted to avoid the need of a training set and keep the principles of this method simple. It is highly probable, that a highly advanced method would be needed if we decided to take advantage of these properties.

[1] In other words, their borders tend to be lighter than the rest.

2.2 Methodology

The method which we have finally created is quite straight-forward and has the character of a pipeline. This comes from the way we were constructing it. In each stage of the development, we intuitively considered, what step to take next. We took the step which we thought would improve our situation the most.

The criterion for how much the situation would improve was based on simple observance and gut feeling at first. We knew what the final segmentation should look like. We estimated, how far from it we were before and after the considered step. Finally we compared these two estimations.

In the further stages of the development, we focused on addressing the most common problems of the method by editing the existing steps and also by adding some additional steps.

Generally, the steps described in Chap. 3 were included on the basis of gut feeling - these are the core steps of the method. The steps described in Chap. 4 were included in order to fix some frequently occurring problems.

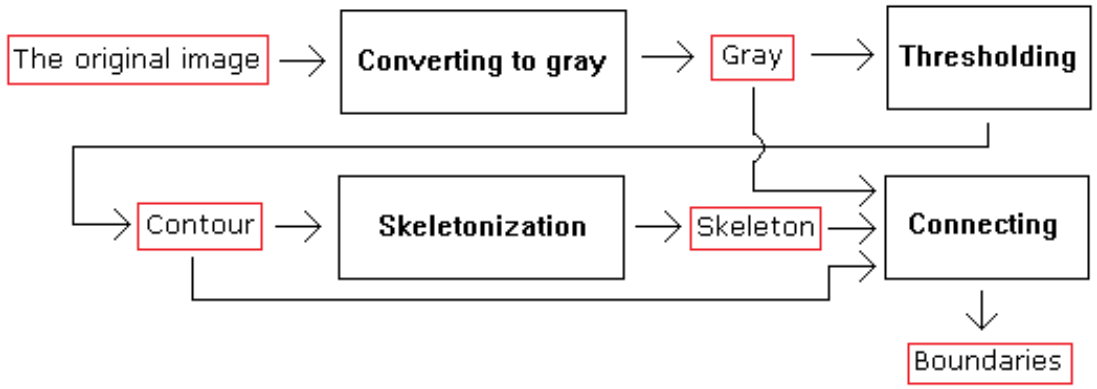


Figure 3.1 A diagram displaying the main steps of the method.

3. The main steps of the method

Our method has only one color image of cells at the input. This image is processed through a pipeline of steps. In this chapter, we describe the four main steps of the method. We also give a name to the output of each step. Let us summarize the steps and their outputs (see Fig. 3.1):

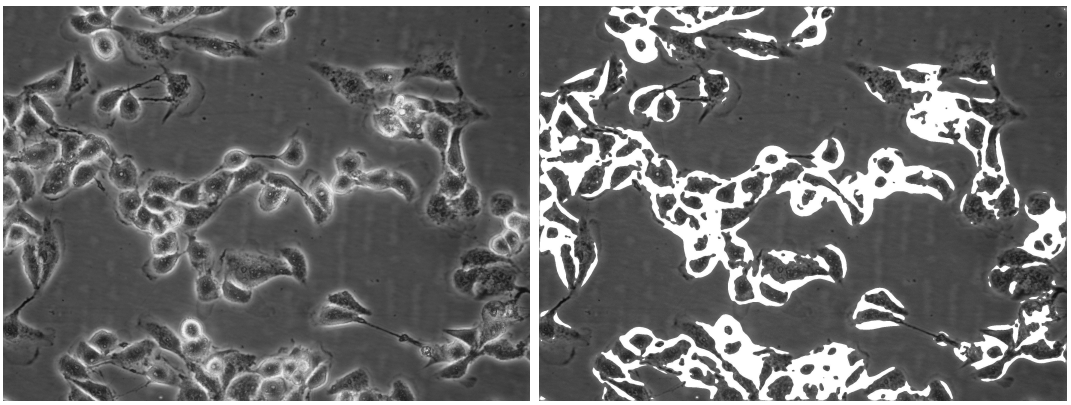
1. *Converting to gray-scale* - its output is called **Gray**. It is a gray-scale image.
2. *Thresholding* of **Gray** - its output is called **Contour**. It is a binary image.
3. *Skeletonization* of **Contour** - its output is called **Skeleton**. It is a binary image.
4. *Connecting* of **Skeleton** - its output is called **Boundaries**. It is a binary image.

3.1 Converting to gray-scale

We use the function *rgb2gray* in MATLAB R2013a [13] to transform the image to gray-scale. This function firstly converts the image from RGB to HSL color space [15]. Just like in RGB color space, in HSL, every pixel has three values. Unlike in RGB, these values do not represent individual basic color components. They stand for *hue*, *saturation* and *luminance*. The first two of them are measures of color and the third one is a measure of intensity. The converting from RGB to HSL is a simple linear transformation - multiplying the vector of the color values by a 3×3 regular matrix. Because this matrix is invertible, no information is lost in this process. This function keeps only the value of *luminance* in the resulting **Gray**. Subsequently, we linearly scale the values of pixels in **Gray** into the range [0, 1], so that the maximum value present in **Gray** is 1, the minimum one is 0.

3.2 Thresholding and Skeletonization

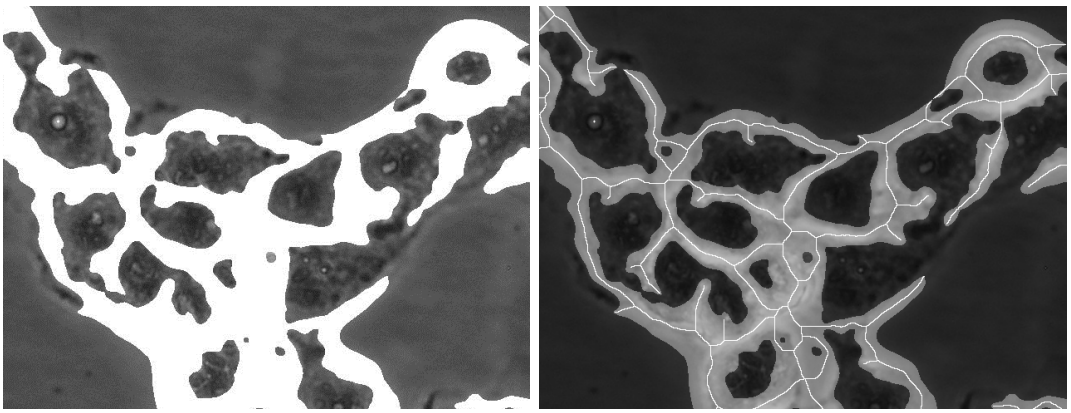
We suppose that halos accompany the majority of the boundaries between the cells. We utilize thresholding to isolate them in **Contour** - see Fig. 3.2. We use Otsu's method to compute the threshold.



(a) Gray. **(b) Contour** superimposed on **Gray**.

Figure 3.2 An example of *Thresholding*

Contour is thick, and thus hard to use in any subsequent operations. We *Skeletonize* it to get **Skeleton** - an approximation of the real boundaries (Fig. 3.3). Our aim is to make **Skeleton** run along the main course of **Contour**. We do not want it to have branches in places, where there is no dominant branch visible in **Contour**. The idea is that the branches should be located only where **Skeleton** ends, but the real boundaries continue. As discussed in Preliminaries, Hilditch's algorithm [12] is more suitable for this purpose than the one provided by MATLAB [13] (see Fig. 7), which is why we use it to produce **Skeleton**.



(a) Contour. **(b) Skeleton** is white, **Contour** is light-gray, they are superimposed on **Gray**.

Figure 3.3 An example of *Skeletonization*

3.3 Connecting

Skeleton is incomplete - many times, some sections of the real boundaries are missing. When this happens, there is often a branch present in **Skeleton** (Fig. 3.4).

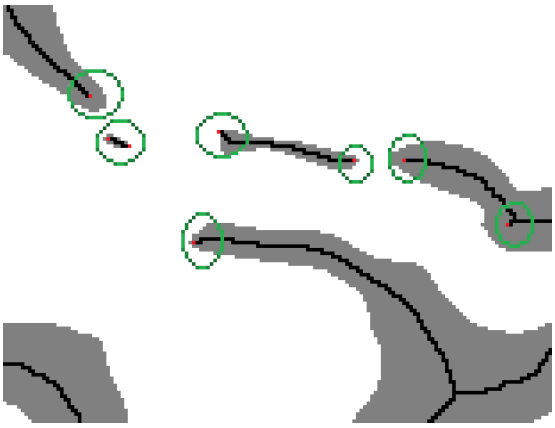


Figure 3.4 End points of **Skeleton**.

The idea of this step is to fix this problem. Our aim is to reasonably connect the end-point of every branch of **Skeleton** to another point of **Skeleton** and subsequently add the interconnecting paths to **Skeleton**. For this purpose, we will construct an interconnecting algorithm which will be launched individually from every end-point of **Skeleton** to find a suitable interconnecting path.

3.3.1 The criteria for the interconnecting path

Our aim is to make the interconnecting path lie as close to the real boundaries as possible, which we express by the list of the criteria which it must satisfy. Every one of them reflects a practical requirement. We introduce a simplified name for each criterion, with the help of which we will refer to it later on. This name is written in parentheses after the description of the criterion.

1. **It must finish in Skeleton. Except for that, it cannot cross Skeleton (*skel_finish*)** - the real boundaries do not cross either.

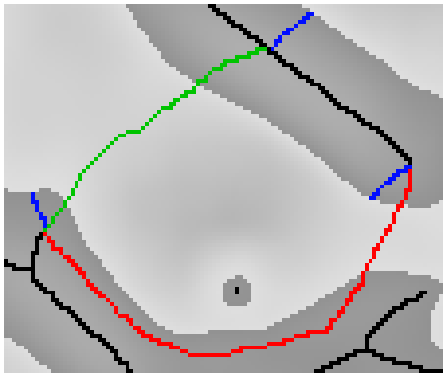
2. **It must start and finish in Contour. It must leave Contour exactly once (*contour_leave*)** - it should interconnect two ends of a missing part of the boundaries. Such a part is also missing in **Contour**, otherwise there would be no gap in **Skeleton**. Thanks to this criterion, the path does not finish in a neighbor of the starting point. This criterion forbids paths like the red one in Fig. 3.5b. It also gives the required logical foundation to the criterion n. 3.

3. **The length of the starting and the finishing section inside Contour is *limited* (*contour_length*)** - it forbids paths like the red one in Fig. 3.5a.

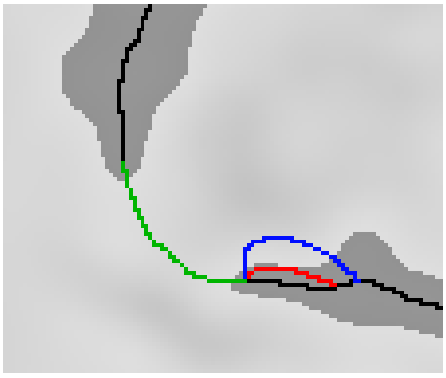
4. **It must be as light as possible (*light*)** - the boundaries are mostly lighter than the rest. We want the path to be as close to them as possible.

5. **It must be as short as possible (*short*)** - the section of the boundaries which is missing in **Skeleton** rarely extends the length of one cell. Together with the criterion n. 4, this criterion makes up the *price* of the path. The *price* comprises a weighted sum of these two criteria.

6. **It must be the one with the lowest *price* among the paths which satisfy the criterion n. 1 and have the same end-points (*cheapest*)** - it forbids paths like the blue one in Fig. 3.5b. A narrow path interconnecting the same pixels, but staying in **Contour** for the whole time, would be cheaper. It would be surely shorter and also lighter, because the pixels in **Contour** are lighter than the rest.



(a) The red path is forbidden; its starting section in **Contour** is too long. The green one is OK.



(b) Samples of three types of interconnecting paths; only the green one is allowed.

Figure 3.5 A demonstration of the purpose of some of the criteria for the interconnecting path. **Skeleton** is black, **Contour** is dark-gray.

3.3.2 The connecting algorithm

As we have mentioned, this is the algorithm for connecting an end-point of **Skeleton** with another point of **Skeleton** via a path which satisfies the enumerated criteria. The criterion *light* requires, that the algorithm utilizes the pixel intensities, which can be found in **Gray**.

The criterion *cheapest* can be elegantly satisfied by using Dijkstra's algorithm [8]. As mentioned in Preliminaries, this algorithm successively visits the nodes in a graph with prices edges. We can be sure, that at the moment of visit, the algorithm has found the cheapest path to the visited node, supposing that all prices of the edges are non-negative. The criterion *cheapest* would be hard to check in another way.

In order to utilize this algorithm, we need to define a suitable transformation of a gray-scale image to a graph with priced edges. We want to look for interconnecting paths in the image, so we will build edges between 8-connected pixels and make the diagonal edges appropriately more expensive[1]. The edges will have

[1] An alternative would be to build edges only between 4-connected pixels, but this would not reflect the natural distance between diagonal pixels as good.

non-negative prices in order to guarantee the proper functionality of the algorithm. Thanks to this, the criterion *short* will be satisfied, because the longer a path is, the more expensive it gets. We can view a pixel as an obstacle on the path which is the same, no matter where we come from to walk through it. It follows, that all edges leading to the same pixel will have the same prices.

Now, it remains to suitably define the prices of the edges in order to satisfy the criterion *light*. The lighter a pixel is, the cheaper the edges leading to it should be. We can postpone this problem and state, that the price of an edge will be equal to the intensity of the pixel which the edge leads to. You might have noticed, that if such a graph was built now directly atop **Gray**, we would not achieve the desired result, because the lighter a pixel, the more expensive the edges leading to it would be. However, this is what we meant by "postponing": a suitable transformation of the intensities in **Gray** can be performed, so that if we build a graph with the described characteristics atop the image after the transformation, the pricing of the edges will reflect the criterion *light*.

So, let us accurately describe how we will build a graph from the transformed image:

- Every pixel becomes a *node*. No other *nodes* exist.
- For each pixel, there are *edges* to all of its 8-connected neighbors. No other *edges* exist.
- The *price* of an *edge* $(a,\ b)$ is $dist(a,\ b) \cdot v(b)$, where *v(b)* is the intensity of the pixel b and $dist(a, b) = \sqrt{2}$, if $a,\ b$ are diagonal neighbors, otherwise $dist(a,\ b) = 1$ (Fig. 3.6). This reflects the real distance between diagonal neighbors.
- Then, the *price* of a path is the sum of the *prices* of the *edges* contained by it.
- We also define the *length* of a path as the sum of dist$(a,\ b)$ for every *edge* it contains.

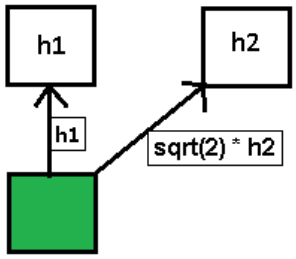


Figure 3.6 The value of an *edge* between two neighbors.

Now, it remains to specify the function to transform the intensities of pixels in **Gray**. The intensities in **Gray** are scaled into the range [0, 1], so this function needs to look like this:

$$f:\ [0,1]\ \longrightarrow\ [0,\infty] \tag{3.1}$$

With the respect to the criterion *light*, a satisfactory function could be for example $f(x) = -x$. Different functions give us different ratios between the criteria *light* and *short*. We tested several of them, including various forms of the exponential function, polynomial functions and some custom-made functions.

The function $f(x) = -log(x)$ turned out to be the most appropriate one of them[2]. As you can see in Fig. 3.7, the new intensity of a pixel rises very quickly to the infinity as its original intensity decreases to 0, which greatly emphasizes the difference between dark and light pixels (Fig. 3.8). This might be the reason for the suitability of this function.

The zero intensities are transformed to the infinity, the practical consequence of which is that Dijkstra's algorithm never visits such pixels, as long as there are still some pixels with non-zero original intensities reachable from the starting pixel before the algorithm finishes. It follows, that the prices of the paths crossing pixels with zero original intensities are mutually incomparable.

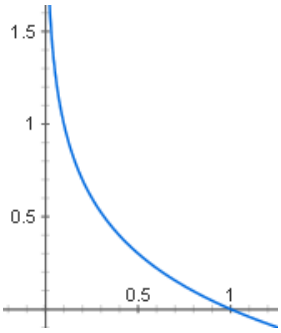
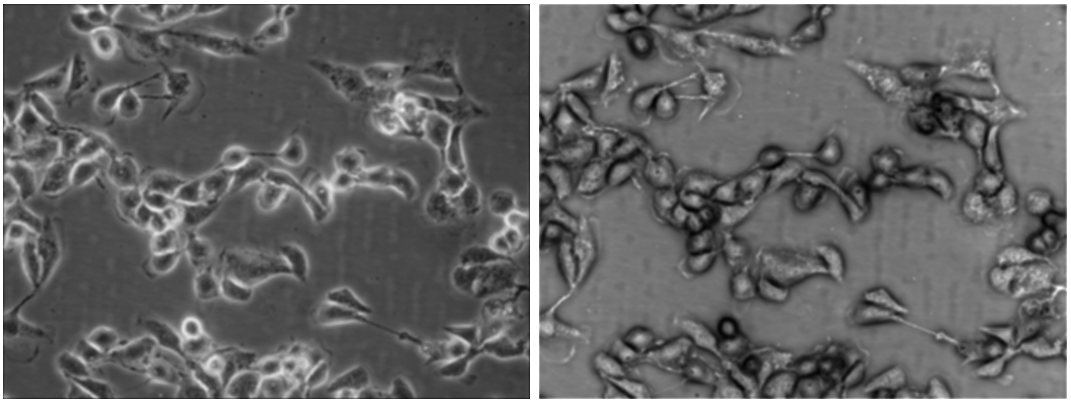


Figure 3.7 The function $f = -log(x)$ in the range [0, 1].



(a) The original image. **(b)** The transformed image.

Figure 3.8 The function $f = -log(x)$ applied to the intensities in a gray-scale image.

By far, our algorithm satisfies the criteria *light*, *short* and *cheapest*. The remaining ones will not be so hard to satisfy. The criterion *skel_finish* can be satisfied by suitable formulations of the *end condition* (see Algorithm 5) and the *continuing condition* (see Algorithm. 6) of Dijkstra's algorithm.

[2]According to the further mentioned evaluation of segmentation (Chap. 5)

However, the criteria *contour_leave* and *contour_length* require some additional information about the found paths to be computed - the *path records* need to be extended, which must be accompanied by the extension of *assigning a new path record* (see Algorithm. 4). In this algorithm, dist(a, b) is 1, if a and b are 4-connected, otherwise it is $\sqrt{2}$. On the basis of the information in the extended *path records*, the *end condition* will then determinate, whether the path leading to the currently visited node satisfies all the criteria (see Algorithm 5).

So, let us enumerate the fields which we need to include in the *path record* besides the *price* and the link to the *previous node*. For each one of the fields and also for each value of a field, its shorter name is written in the parentheses, with the help of which we will refer to it later on.

- The *state* of the path with the respect to **Contour** (*state*) - it can be one of:
  - It is whole in **Contour** (*contour_first*).
  - It has left **Contour** once and is currently out of it (*contour_between*).
  - It has left **Contour** once and is currently in it (*contour_second*).
  - Any other state (*contour_out*).
- The *length* of the first continuous section of the path in **Contour** (*length_first*).
- The *length* of the second continuous section of the path in **Contour** (*length_second*).

In Algorithm 5, we use the expression *certain limits*. Let us describe what we mean by it. For each pixel p of **Skeleton** we compute d(p): the *length* of the shortest path from p to the closest border of **Contour**. In Fig. 3.5a, these paths are drawn with blue color.

We define **tolerance**: an inner parameter of our method, such as: **tolerance** $\in \mathbb{R}$ & **tolerance** > 1. Let us consider a path interconnecting two pixels of **Skeleton** - a and b. Then, the length of its starting section in **Contour** must be less than **tolerance** $\cdot\, d(a)$ and the length of its finishing section in **Contour** must be less than **tolerance** $\cdot\, d(b)$. This approach ensures the adaptiveness to various thickness of **Contour**.

Algorithm 4 Procedure
assignNewPathRecord(current_node, previous_node, new_price)

```
new_record ← copyOf(previous_node.path_record)
new_record.price ← new_record.price + new_price
new_record.previous_node ← previous_node
dist ← dist(current_node, previous_node)
if new_record.state = contour_first then
    if current_node ∈ Contour then
        new_record.length_first ← new_record.length_first + dist
    else
        new_record.state ← contour_between
    end if
else if new_record.state = contour_between then
    if current_node ∈ Contour then
        new_record.state ← contour_second
        new_record.length_second ← dist(current_node, previous_node)
    end if
else if new_record.state = contour_second then
    if current_node ∈ Contour then
        new_record.length_second ← new_record.length_second + dist
    else
        new_record.state ← contour_out
    end if
end if
current_node.path_record ← new_record
```

Algorithm 5 Function
endCondition(visited_node)

```
return visited_node ∈ Skeleton and
visited_node.path_record.state = contour_second and
visited_node.path_record.length_first < certain limit and
visited_node.path_record.length_second < certain limit
```

Algorithm 6 Function
continuingCondition(visited_node)

```
return visited_node ∉ Skeleton
```

4. The additional steps and the modifications to the method

In this chapter, we identify some of the frequently occurring issues of the described method and address them by suitably modifying it or by including some more steps in it. So far, the method has not utilized the separation of the cells from the background; equivalently, the output of the detection of clusters of cells. It is a binary image, the pattern in which denotes the areas, where the cells are likely to appear. From now on, we will call it **Clusters**. This chapter includes the utilization of it. Soukup's algorithm [23] focuses on the detection of clusters and its result are close to the ground truth. No other methods addressing this detection are publicly available. Thus, **Clusters** used in our method is the output of this algorithm.

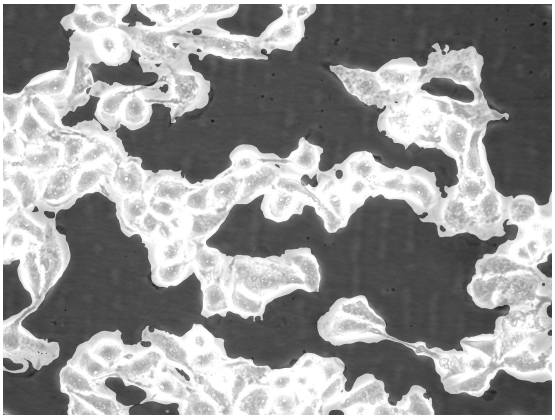


Figure 4.1 Clusters superimposed on **Gray**.

Several parameters are introduced in this chapter. The settings of the parameters of this method are discussed in Sec. 6.1. We divide the parameters into two groups:

- The inner parameters - the ones which are not modifiable by the user, because they only weakly depend on the input data and we tuned them properly according to the evaluation. They are marked as **blue**. One of them is also **tolerance** at the end of Sec. 3.3.
- The regular parameters - the ones which are modifiable by the user. Their optimal settings depend on the input image; mainly on the sizes and shapes of the cells and on how much the image is blurred. They are marked as **red**.

4.1 Blurring

If **Gray** is blurred after its creation, the subsequently created **Contour** has a more definite shape, less influenced by noise and the inner structure of the cells. The *Gaussian convolution core* turned out to be suitable for this purpose[1]. Such

[1] According to the evaluation (Chap. 5)

a core has two parameters - **diameter** and **variance**. The result of the method greatly depends on their settings. The higher their values are, the less detail is preserved in **Gray**, which is only profitable to a certain extent. Their optimal settings mainly depend on the size of the image and on how much the image is already blurred.

4.2 Multiple thresholding

This is a modification to *Thresholding* (see Sec. 3.2). The issue is that **Contour** sometimes covers the whole cell. This happens when the interior of the cell is too light (Fig. 4.2b). The aim of this modification is to reduce the number of such cases. Because **Contour** is subsequently skeletonized, it is quite sufficient to ensure that at least a small spot of the interior of the cell lies out of **Contour** (see Fig. 4.2d).

Using a higher threshold (Fig. 4.2c) solves this problem, but a lot of information about the boundaries is lost this way. The idea of this modification is to combine the advantages of the lower threshold (Fig. 4.2b) with the advantages of the higher threshold. We want to preserve all the information about the boundaries provided by the lower threshold and only make "some holes" in the places, where it is likely, that the output of the lower threshold is covering the whole cell.

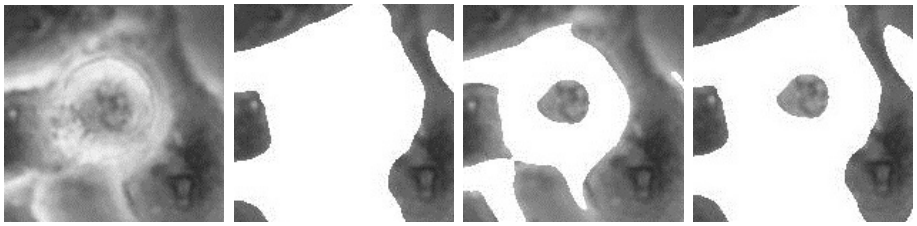
Let t be the threshold computed by Otsu's method [20]. We define **steps**: an inner parameter of our method, such as: **steps** $\in \mathbb{N}$ & **steps** ≥ 1. The value of this parameter is the number of the steps of the multiple thresholding. **Gray** is successively thresholded with **steps** thresholds higher than t. Their values are evenly distributed in the range $(t, 1)$ (note that the intensities in **Gray** are in the range $[0, 1]$).

Let us call C the result of thresholding **Gray** with the threshold t (computed by Otsu's method). Now, we will simply denote the output of the a-th step of the multiple thresholding, where $a \in \{1, .., \textbf{steps}\}$: let C_a be the result of thresholding **Gray** with the threshold $t + a(1-t)/(\textbf{steps}+1)$. With this notation, the algorithm 7 describes how the resulting **Contour** is produced. Refer to Preliminaries for the definitions of holes in binary images.

In this algorithm, the function addHoles(**Contour**, C_a) looks for the holes in C_a which are whole in **Contour**. Then, it inserts such holes into **Contour** (sets the values of all their pixels to 0 in **Contour**). In Fig. 4.2, you can see an example of an application of this function. You can imagine that addHoles(4.2b, 4.2c) returns 4.2d.

Algorithm 7 The creation of **Contour** by multiple thresholding.

```
Contour ← C
for a ← 1 to steps do
    Contour ← addHoles(Contour, C_a)
end for
```



(a) **Gray**. **(b)** The lower threshold. **(c)** The higher threshold. **(d)** Their combination.

Figure 4.2 An example of multiple thresholding. The outputs of thresholding are superimposed on **Gray**.

4.3 Enhanced connecting

This section describes modifications to *Connecting*. They include:

- Modifications to **Contour** and **Skeleton** before the connecting routine starts.
- Modifications to the connecting algorithm which is launched from every end-point of **Skeleton**.

4.3.1 Enriching Contour and Skeleton

It is a common problem, that boundaries of some cells at the border of a cluster are not detected by *Thresholding*, because they are not light enough (Fig. 4.3a). Here we can use **Clusters** (Fig. 4.3b). From there, we can extract the boundary pixels of the clusters - let us call them B. You can view B as the borders of the bright area in Fig. 4.3b. The idea is to include B to **Skeleton**

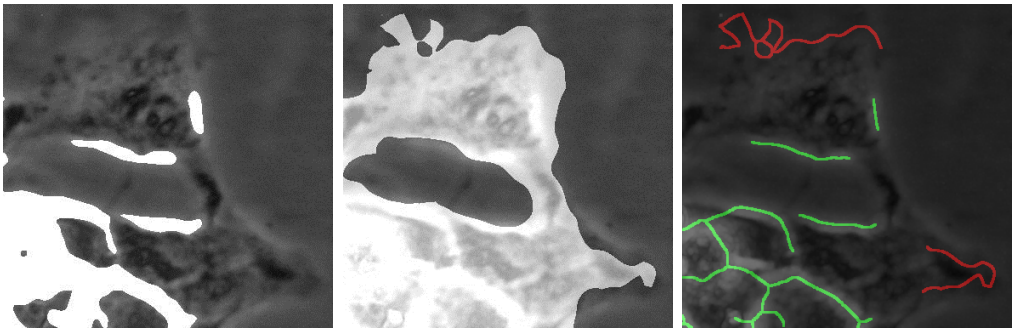
However, B often lies close to **Skeleton** (the green lines in Fig. 4.3c) and the real boundaries around the cells are approximated differently in B and in **Skeleton**. This stems from the difference in the way, how these approximations are constructed. **Skeleton** is mostly based on the light intensity, whereas B is based on the movement of the cells between the subsequent images in the series. Many times, these two approximations variously cross each other.

Thus, adding the whole B to **Skeleton** creates the following problem: an end-point of **Skeleton** which was formerly connected to another point of **Skeleton** via a path running along an inner boundary between the cells can now be connected to a nearby point in B instead, if the path to it is cheaper. For example, this may happen, when B crosses the formerly found interconnecting path. Thus, if the boundary between the cells approximated by the former interconnecting path is not discovered by another path, undersegmentation occurs.

This is why we include only the parts of B which are "far enough" from **Skeleton**. It seems that we can now hardly get by without introducing a parameter expressing, what "far enough" means. This is why we introduce the parameter **distance**. We select the pixels in B which are further than **distance** from the pixel of **Skeleton** which is the closest one to them. Let S be the result of this selection. Then we enrich **Contour** and **Skeleton** by S (Fig. 4.3c):

$$\begin{aligned} \textbf{Contour} &= \textbf{Contour} \cup S \\ \textbf{Skeleton} &= \textbf{Skeleton} \cup S \end{aligned} \tag{4.1}$$

It is important to enrich **Contour** too, otherwise the connecting algorithm would not connect the enriched parts of **Skeleton**; it demands, that the interconnecting path starts and finishes in **Contour** (see the criterion *contour_leave* in Sec. 3.3.1). The setting of the parameter **distance** has a considerable impact on the performance of this method. Its optimal setting depends on the input image, mainly on the size of the cells in proportion to the size of the image.



(a) Contour. **(b) Clusters.** **(c) Skeleton** is green, the red paths are the boundaries are added from **Clusters**.

Figure 4.3 Enriching **Skeleton** by the distant boundaries of **Clusters** - all the displayed binary images are superimposed on **Gray**.

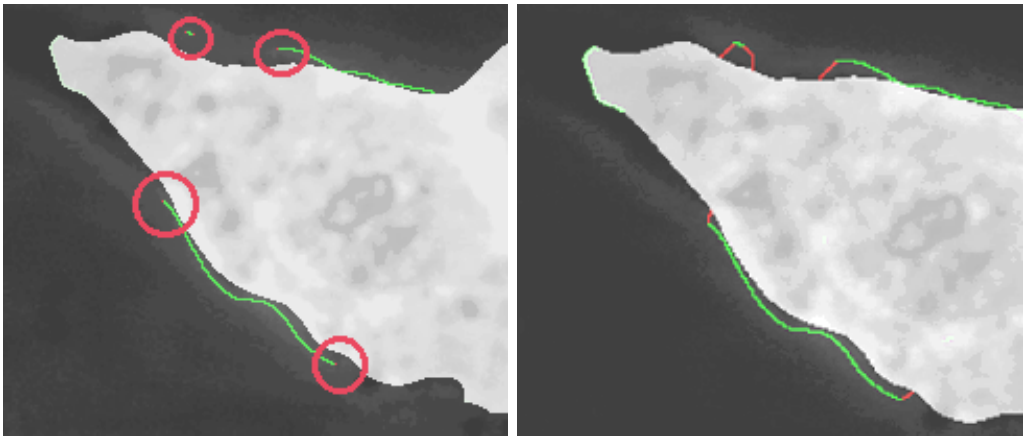
4.3.2 Connecting only inside Clusters

This is a modification to the connecting algorithm. It represents another utilization of **Clusters**.

In the image, the boundaries between the cells never occur outside **Clusters**, supposing that **Clusters** are close enough to the ground truth. This is why we look for the interconnecting paths only inside **Clusters**. This also speeds up the connecting algorithm, because it reduces the number of the pixels which can be visited by it.

Unfortunately, this constraint brings about a small problem: some end-points of **Skeleton** are located out of **Clusters**, and thus cannot be directly interconnected through **Clusters** (Fig. 4.4a). This is caused by the phenomenon discussed in the previous section: the approximations of the real boundaries around the cells by **Skeleton** and **Clusters** differ.

Luckily, this problem can be easily fixed by extending **Clusters**. We use the previously described connecting algorithm for this purpose. We launch it from every end-point of **Skeleton** which lies out of **Clusters**, and let it find the cheapest path from it to **Clusters**. The found paths are then added to **Clusters** (Fig. 4.4b). Now, every end-point of **Skeleton** can be interconnected with another point of **Skeleton** through **Clusters**.



(a) Some end-points of **Skeleton** are out of **Clusters**.

(b) The red lines denote the extension of **Clusters**.

Figure 4.4 The issue when connecting only inside **Clusters**.

4.3.3 Connecting inside a clipped space

This is another modification to the connecting algorithm.

We observed, that the shape of the cells is generally not extremely curvy and the cells rarely have sharp spurs. We utilize this observation in the modification. Its idea is to reasonably clip the space which is searched by the connecting algorithm, so it is unable to find interconnecting paths which form too sharp angles with the branches of **Skeleton** they grow from. As a side benefit, this modification also speeds the connecting algorithm up, because it reduces the number of pixels which can be visited by it.

We employ a cone-shaped symmetric clipping (Fig. 4.5). We define **angle**: a regular parameter of our method. The meaning of this parameter is depicted in Fig. 4.5a by the letter u. It defines, how much clipped is the searched space. The greater its value, the smaller the searched space. Its optimal setting depends mainly on the shape of the cells.

In order to implement this modification, we firstly need to be able to find out the tail angle of a branch of **Skeleton**. We define **dist**: an inner parameter of the method. Then, this is how we compute the tail angle of a branch:

Let us call E the end-pixel of the branch. We make **dist** steps along the branch from E to its other end. One step stands for moving from a pixel P to its not yet *visited* neighbor and declaring P *visited*. If we come across the root of the branch before making **dist** steps, we stop there. Let us call S the pixel at which we have stopped. Then, S sufficiently represents the tail angle of the branch ending at E (Fig. 4.5b).

It can also happen, that the branch is very short and it has no root (see the blue circle in Fig. 4.5a). If we come across another end of the branch before making **dist** steps from E, we declare E *indeterminate*, which means that the space is not going to be clipped from this branch.

Secondly, we need to modify the *continuing condition* of the connecting algorithm (Algorithm 6 in Chap. 3). We add the following sub-condition to it (a *node* stands for a pixel - see Sec. 3.3.2):

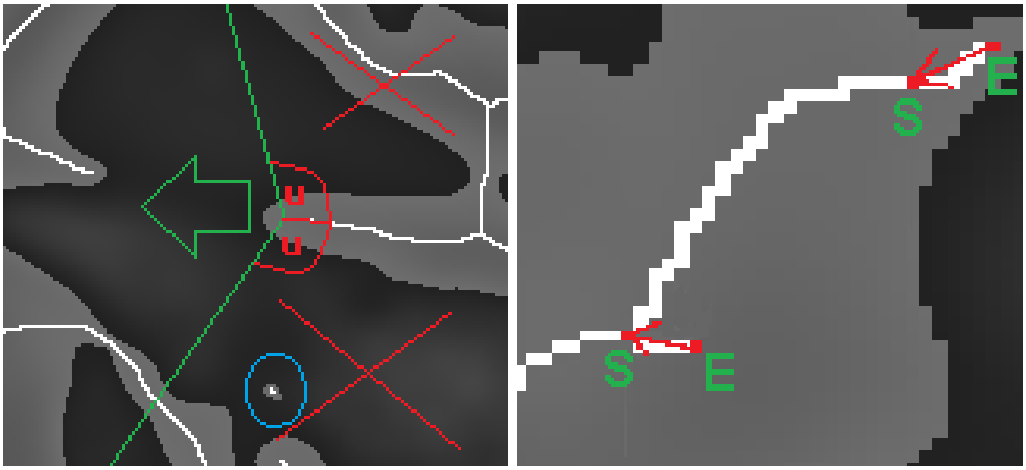
- If E is not *indeterminate*, the angle (the *node*, E, S) must be greater than

angle.

The *continuing condition* then becomes the conjunction of the existing sub-condition (that the *node* cannot be in **Skeleton**) and the new sub-condition.

In this moment, you might question the existence of the criterion *contour_length* in Sec. 3.3.1 and the existence of all the elements of the algorithm stemming from this criterion (the extra fields in the *path records* and the extra sub-conditions in the *end condition*). The main purpose of this criterion is similar to the purpose of this modification - to eliminate too "sharp" connections.

The fact is, that the method could now work without this criterion. However, the graph 7.1 in Attach. **??** shows, that increasing the value of **tolerance** results in worse performance of the method. Note that increasing the value of **tolerance** is similar to weakening this criterion and setting it to infinity is equivalent to omitting the criterion. From this, we concluded that keeping this criterion in our method is beneficial.



(a) The meaning of the parameter **angle**. **(b)** The computation of the tail angle of a branch in **Skeleton**.

Figure 4.5 The cone-shaped clipping of the space searched by the connecting algorithm.

4.3.4 Skeletonization after the connecting

The output of the connecting algorithm is oversegmented quite much (Fig. 4.6a). However, it often occurs that very small objects are segmented - much smaller ones than the cells, especially near the boundaries of the cells. This is mainly caused by:

1. Holey shape of **Contour** near the boundaries - subsequently, small objects are around these holes in **Skeleton** (tags "1" in Fig. 4.6a).

2. Branches of **Skeleton** (caused by spurs of **Contour**) heading towards the interior of a cell - it often happens that they are connected back to

a pixel of the boundary of the cell, which is close to they place they "grow" from, creating a small object this way (tags "2a" in Fig. 4.6a)[2].

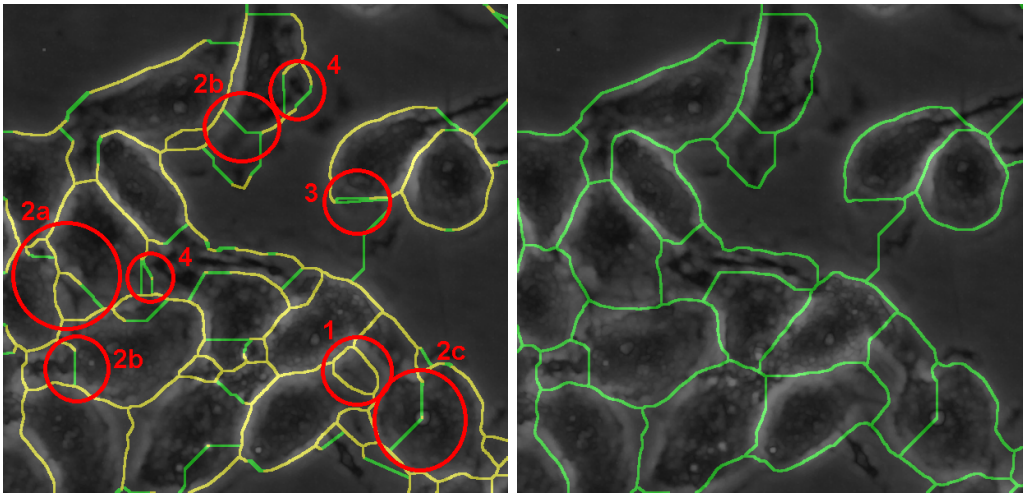
3. The phenomenon when two end-pixels of **Skeleton** are interconnected with each other via distinct paths with the same prices. A small object is delimited by these paths (tags "3" in Fig. 4.6a).

4. Other inaccuracies near the boundaries (tags "4" in Fig. 4.6a).

In this case, the knowledge of the minimum area of a cell can help a lot. If we are sure, that a cell cannot be smaller than a certain specified minimum, we can eliminate objects smaller than that. We introduce **area**: a regular parameter of the method. It specifies the minimum size of a cell in pixels. Its optimal setting depends on the size of the observed cells in proportion to the size of the image.

In the output of the connecting algorithm (the **Skeleton** enriched by the interconnecting paths), we identify all the objects the area of which is smaller than **area**. We "fill them up" - set the values of their interior pixels to 1. Then we skeletonize the whole image (see Fig. 4.6b) using the same algorithm as in Sec. 3.2. Note that the skeletonization shifts the boundaries closer to the middle of the small objects.

An alternative to this approach could be merging such objects with their neighbors. However, this would introduce the question, which neighbor to choose. Regarding the small sizes of the considered objects and also the fact, that they are often created due to some sort of inaccuracy, we think that the proposed solution based on skeletonization is the more elegant one.



(a) The output of the connecting algorithm. **Skeleton** is yellow, the paths found by the connecting algorithm are green.

(b) The boundaries in Fig. 4.6a after the skeletonization of the small objects.

Figure 4.6 Skeletonization of the small objects.

[2]Unfortunately, it also happens, that they are connected to the opposite boundary of the cell via a path crossing the cell (tags "2b" in Fig. 4.6a). This also occurs when **Skeleton** is present inside a cell (tags "2c" in Fig. 4.6a). Neither this section, nor this thesis present solutions to these problems, if the objects segmented this way are not distinctly small.

4.3.5 Double connecting

This modification addresses the following problem:

Sometimes, there is only a short fragment of **Skeleton** present in the middle of a real boundary between the cells. If the fragment is short enough, its end-points are declared *indeterminate* (see Sec. 4.3.3). When looking for optimal connecting paths from them, the searched space is not clipped. This is why they might be connected to the same side of the real boundary (Fig. 4.7), which is not the desired result. We would like one of them to connect to another side of the boundary.

However, after the subsequent skeletonization (see Sec. 4.3.4), one of the end-points of the former short fragment becomes the end-point of a new, longer branch. It is possible that it is not *indeterminate* now. For this reason, we run the connecting routine for the second time, which is followed by the second skeletonization of its output. During the second run of the algorithm, the end-point marked as red in Fig. 4.7 will be connected to the other side of the real boundary.

Let us call **Boundaries** the output of the first iteration of the connecting algorithm and the subsequent skeletonization (Sec. 4.3.4). We must be aware that **Boundaries** will most probably exceed the enriched **Contour** (Sec. 4.3.1)[3]. We need to include them in **Contour** before using **Contour** as the input to the second iteration of the connecting algorithm. More formally, we need to perform the following operation:

$$\mathbf{Contour} = \mathbf{Contour} \cup \mathbf{Boundaries} \tag{4.2}$$

Otherwise, the algorithm would not connect the parts of **Boundaries** exceeding **Contour** (see the criterion *contour_leave* in Sec. 3.3.1). To the contrary, **Clusters** do not need to be enriched anymore[4].

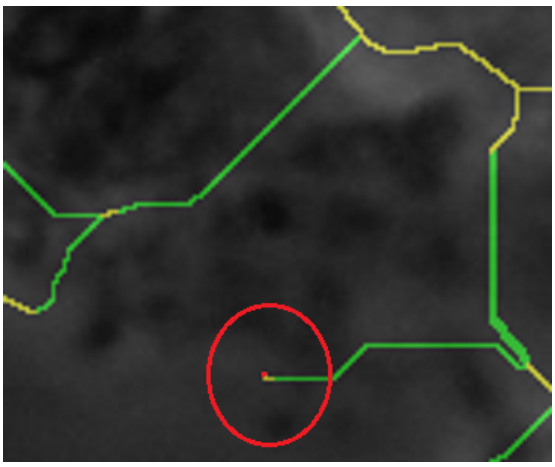


Figure 4.7 The reason for the second run of the connecting algorithm.

[3]Because the connecting algorithm adds new interconnecting paths and the skeletonization moves the resulting boundaries.

[4]All the interconnecting paths added by the first iteration of the connecting algorithm lie inside **Clusters** (see Sec. 4.3.2). Skeletonization of small objects "shrinks" the boundaries - most of the time, it does not move them out of **Clusters**.

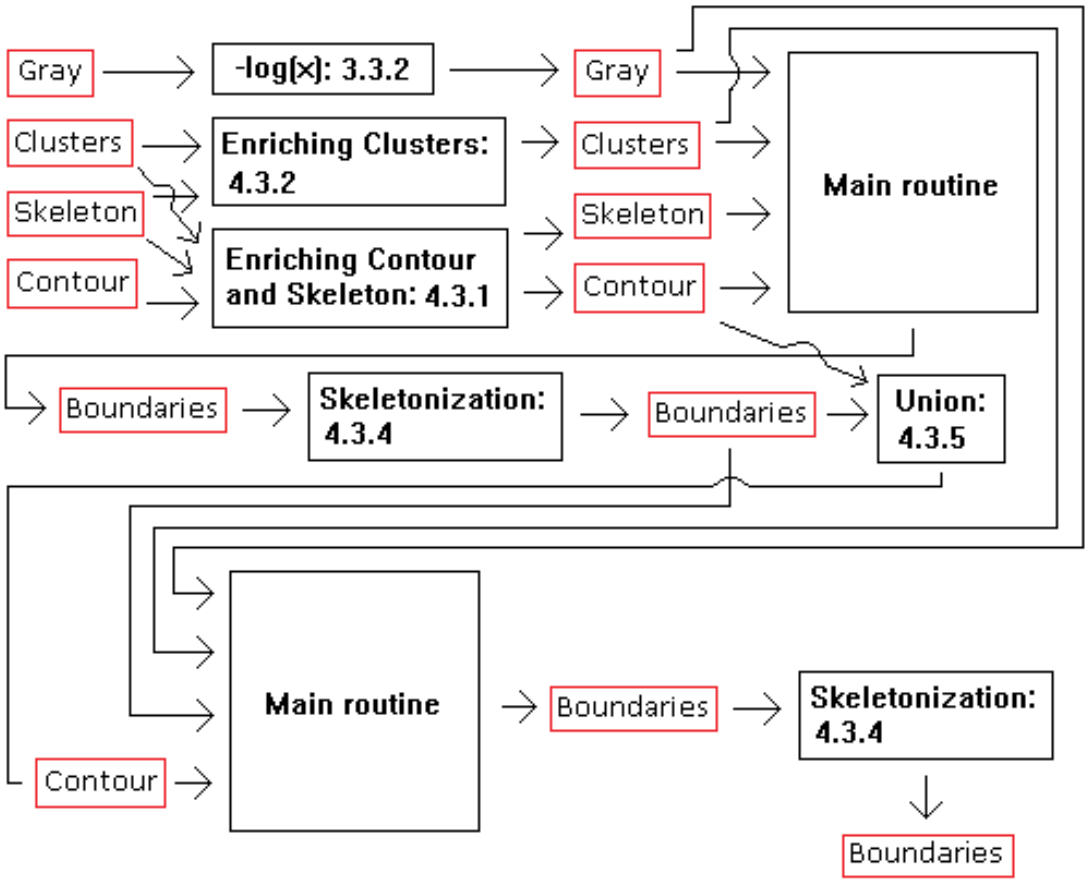


Figure 4.8 A diagram of Enhanced connecting.

4.3.6 A diagram of Enhanced connecting

In this section, we present a diagram of *Connecting* after the modifications to it described in the previous sections. You can view this diagram as an expansion of the box labeled "Connecting" in the main diagram of this method - Fig. 3.1 in Chap. 3.

Compared to the former diagram, *Enhanced connecting* has one more input now - **Clusters** (see Sec. 4.3.1 and 4.3.2 of this chapter). Unlike in the main diagram, **Gray** is blurred now (see the beginning of this chapter - below the list of the types of the method's parameters). In the diagram, the box called *Main routine* comprises launching the *connecting algorithm* from all branches of **Skeleton** (**Boundaries** in the second iteration). The mentioned *connecting algorithm* is the one described in Chap. 3, including the modifications in this chapter, namely in Sec. 4.3.2 and 4.3.3.

4.4 The final steps

The following sections describe the steps, which we placed after *Connecting* to improve the results. The first one of them adjusts the boundaries separating the cells from the background. The second one of them identifies the objects which are not likely to be cells and marks their whole interiors as the boundaries. Both these routines utilize **Clusters**.

4.4.1 Adjusting the outermost boundaries

In the outputs of the previously described steps, we observed, that the outermost boundaries of the cells (equivalently the boundaries between the cells and the background) were quite distant from the outermost boundaries of the *test objects* (the objects segmented by the human experts). In addition to that, the boundaries of the *test objects* were often located somewhere between the boundaries of *our objects* (the objects segmented by our method) and the boundaries of **Clusters** (see Fig. 4.9a). This is because our method reacts to the intensities of pixels, whereas Soukup's method for detecting clusters reacts to the movement of the cells. It seems that the truth is often "somewhere between". This inspired us to push **Boundaries** (the output of the previous steps of our method) a bit towards the boundaries of **Clusters**.

In order to do this, we firstly insert the boundaries of **Clusters** into **Boundaries**. Secondly, we detect the *border objects*, which were created by this insertion. Those are the objects, which lay inside **Clusters** and which have some parts of their boundaries common with the boundaries of **Clusters** (see Fig. 4.9c). The idea is, that the *border objects* should be completely new objects squeezed between the boundaries of *our objects* and the boundaries of **Clusters**. We deduce it from the observation, that these two boundaries differ and that **Boundaries** are most of the time surrounded by the boundaries of **Clusters**. However, as we see in the red circles in Fig. 4.9c, it is not always like this and the *border objects* sometimes comprise the cells. This is why we must *filter* these *border objects*.

The circularity turns out to be a reasonable *filtering* criterion, because the *border objects*, which we actually want to detect, are often much more elongated than the cells. This is why we introduce another regular parameter of the method: **circularity**. Then, we throw away all the objects with the circularity higher than **circularity** from the *border objects* (see Fig. 4.9d).

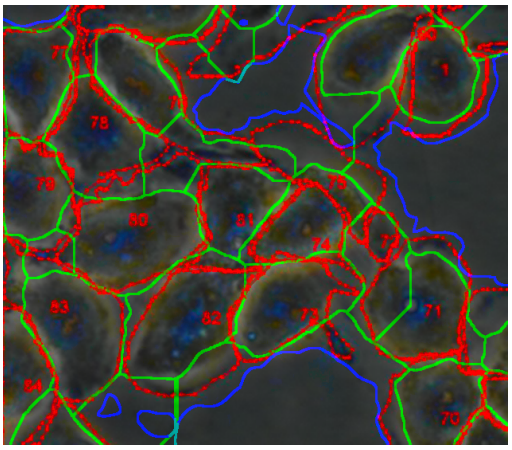
Subsequently, we fill the interiors of the *border objects* up - we mark them as boundaries - and we skeletonize **Boundaries**. This is the same principle as in Sec. 4.3.4. The skeletonization causes the outermost boundaries of *our objects* to move a bit outwards (see Fig. 4.9b) As you can see in this figure, **Boundaries** contain branches inside objects (see the red circles). These inner branches are unnecessary, so they are subsequently erased.

The optimal setting of the parameter **circularity** mainly depends on the dominant shape of the observed cells.

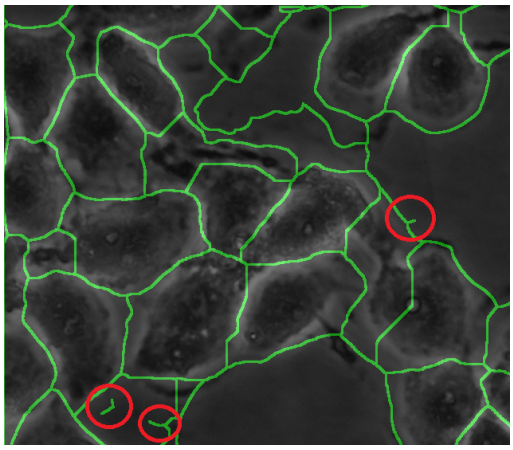
4.4.2 Identifying the background

At this moment, **Boundaries** (the output of the previous steps of this method) also contain segmented objects, which are out of **Clusters**, and thus are in the background. We throw away the objects, which are "not sufficiently" covered.

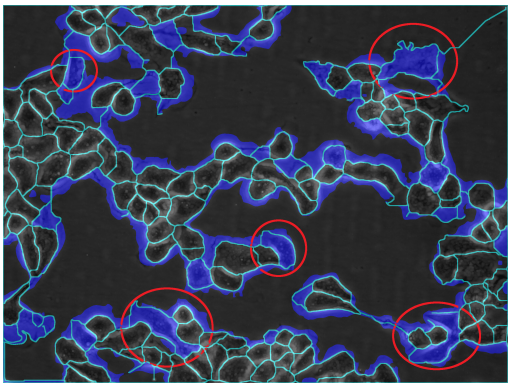
For this purpose, we introduce an inner parameter of the method: **cover**. For a certain object, let us denote S_C the area which is covered by **Clusters**. Let us denote S the total area of the object. Then, for every object segmented by **Boundaries**, we compute the ratio of its S_C to its S. If it is less than **cover**, we declare the object a *background object*. In Fig. 4.10, you can see an example of the selection of the *background objects*. The optimal setting of **cover** depends on the accuracy of both our segmentation and the boundaries of **Clusters**.



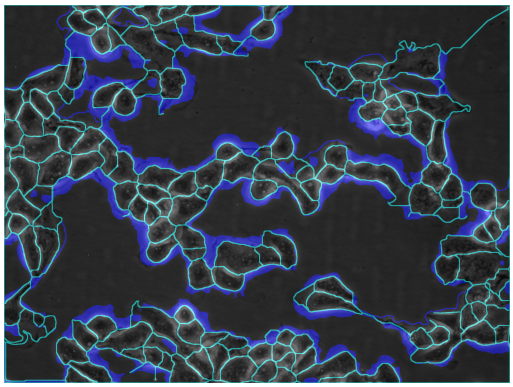
(a) A comparison of the outermost boundaries. The green lines are **Boundaries**, the blue ones are the boundaries of **Clusters** and the red ones are the boundaries of the *test objects*.



(b) **Boundaries** after the skeletonization of the filtered *border objects*.



(c) The firstly selected *border objects* - they are marked as blue.



(d) The filtered *border objects* - they are marked as blue.

Figure 4.9 An example of adjusting the outermost boundaries.

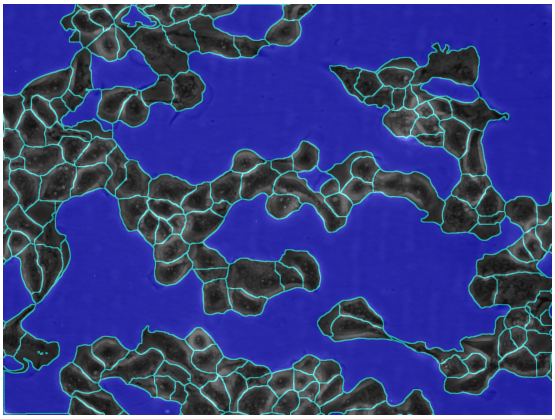


Figure 4.10 An example of identifying the background - it is marked as blue.

Finally, we mark the whole interior of every *background object* as boundaries (set the values of its pixels to 1). This way, we get a sufficient representation of the segmentation. Hence, the final output of our method is a binary image.

4.5 The final diagram of the method

In this section, we present the final diagram of our method: Fig. 4.11. This diagram replaces the former main diagram (Fig. 3.1 at the beginning of Chap. 3). The changes are following:

- *Converting to gray* is substituted by *Converting to gray and blurring* - see Sec. 4.1.
- *Thresholding* is replaced by *Multiple thresholding* - see Sec. 4.2.
- *Connecting* is replaced by *Enhanced connecting* - see Sec. 4.3.
- Two more steps after *Connecting* are added - see Sec. 4.4.1 and 4.4.2.

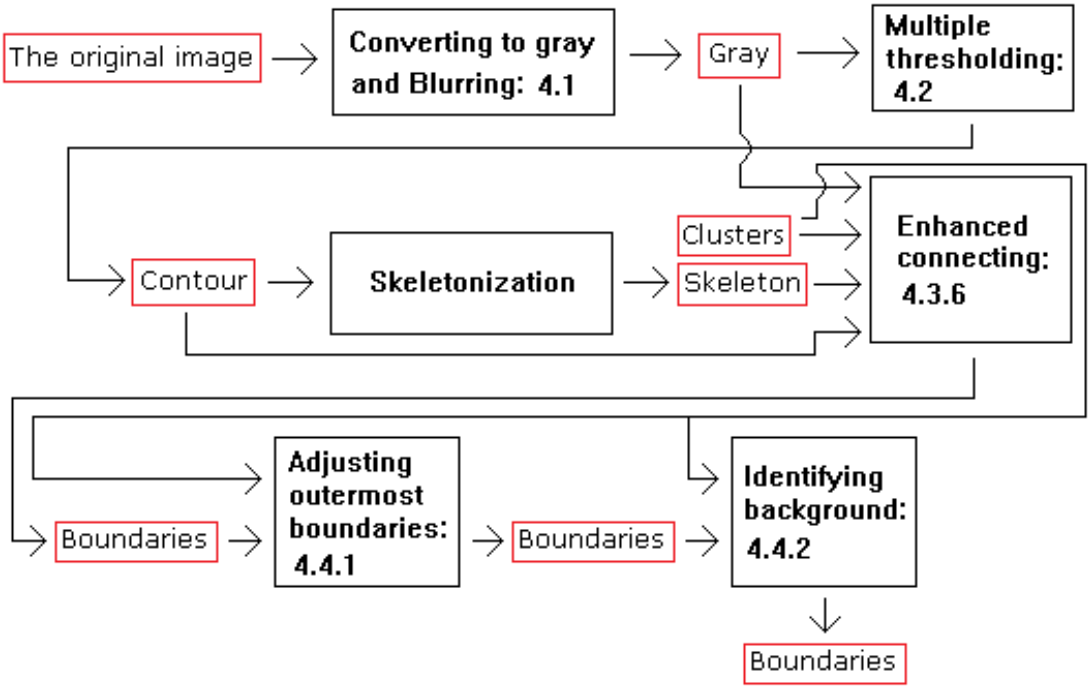


Figure 4.11 The final diagram of the method.

5. The evaluation metrics of the segmentation

We had a few reference images available - the images, a manual segmentation of which has been performed by human experts. Let us call the segmented areas in such a segmentation the *test objects*. Let us call the segmented areas determined by **Boundaries** *our objects*.

Let us describe the evaluation metrics. Firstly, we initialize the global variables Fn, Tp, Fp (false negative, true positive, false positive) to zeros. For each test object **t**, we do the following (see Fig. 5.1):

- We find *our object* **o** with the maximum area of intersection with **t**.
- We compute t_{Fn} - the size of the area of **t**, which is not included in **o** (false negative).
- We compute t_{Tp} - the size of the area of **t**, which is included in **o** (true positive).
- We compute t_{Fp} - the size of the area of **o**, which is not included in **t** (false positive).
- We increment the global variables:
  - $Fn = Fn + t_{Fn}$
  - $Tp = Tp + t_{Tp}$
  - $Fp = Fp + t_{Fp}$

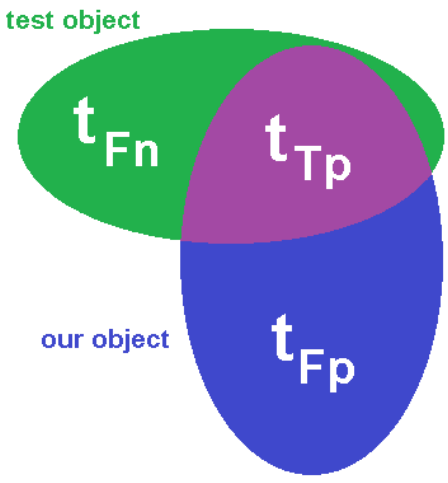


Figure 5.1 A demonstration of the evaluation of one *test object*.

After we perform these steps for every *test object*, we compute:

- The *precision*: $P = Tp/(Tp + Fp)$.

- The *recall*: $R = Tp/(Tp + Fn)$.
- The *f-score*: $F = 2PR/(P + R)$.

We use the value of F as the number expressing the quality of the segmentation. Besides this, we also consider the ratio of the number of *our objects* to the number of the*test objects* as another measure. The closer their values are to 1, the more accurate the segmentation is.

The imperfections of the segmentation can be basically divided into two types:

- *Undersegmentation* - when *our object* covers several *test objects*.
- *Oversegmentation* - when a *test object* covers several *our objects*.

The lower the *undersegmentation*, the higher the *precision*, because *Fp* decreases (imagine the blue object in Fig. 5.1 shrinking). The lower the *oversegmentation*, the higher the *recall*, because *Fn* decreases (imagine the green object in Fig. 5.1 shrinking). The *f-score* is the harmonic mean of the *precision* and the *recall*, so it comprises the influence of the both types of imperfection evenly.

6. The results

In this chapter, we show the results of the evaluation of this method (see Chap. 5 for details). We also evaluate the standard watershed algorithm the same way and compare the results. We do not evaluate any of the methods mentioned in Chap. 1, as none of them is publicly available.

We evaluated these methods on 8 manually labeled images - we call them the *test set*. For each one of the images, we calculated the precision, the recall, the f-score and the ratio of the number of *our objects* to the number of the *test objects* (*Counts ratio*). We also measured, how long it took to segment each one of them (*Time elapsed*). Then, we computed the means and the standard deviations of these quantities. We took the mean f-score followed by the mean *Counts ratio* as the main measures of quality of the segmentation.

The *test set* contained several types of cells - they were taken from 4 time-lapse series. Even though the number of the images was not very high, they contained the total of 1907 cells. Thus, the average number of cells in one image was 238.375. The standard deviance of the count of cells in one image was 107 and the minimum count in one image was 122, so the weight of every image did not vary a lot. This is why we think that the presented results are credible.

We use the implementation of our method, which is provided on the attached CD. We use the implementation of the watershed provided in MATLAB R2013a [13].

6.1 Tuning the parameters

Skeletonization of the small ones:

We tuned the parameters of the both tested methods (our method and the watershed algorithm) manually by an alternating optimization of the f-score on the whole test set. In every iteration of it, we did the following for every parameter: we shifted its value until the f-score reached a local optimum. We performed several such iterations until the f-score stopped improving. In every iteration, we firstly optimized the inner parameters and then the regular ones (see Chap. 5). The order of the parameters within each one of these two sets was the same as in the lists below, which show their values.

We then adjusted the value of **area** (see Sec. 4.3.4) for every image separately (its value was the same for images from the same series). It is common that the smallest cells are the ones in mitosis, which are mostly the lightest ones in the image. If these cells are detected, their sizes can be helpful for setting the value of **area**.

We are aware that we possibly did not reach the global optimum this way. However, the meaning of the individual parameters suggests, that they do not depend on each other very much (except the parameters of the convolution core used for blurring). Thus, it is probable, that the function of dependency of the f-score on the parameters is separable. However, we have not proven it, because an experiment verifying this hypothesis would be extremely computationally demanding[1]

[1]Our method has nine parameters. Sampling of just 10 values of each one yields the need

In Attach. 7.1, the shown graphs demonstrate, that at the presented settings of the parameters, the functions of the mean f-score of the methods on the *test set* have local maxima. We verify it by performing this for every parameter: we move its value to both sides while fixing the values of the remaining parameters. If the mean f-score decreases after moving the value of every parameter, we have reached a local optimum[2]. The graphs demonstrate this fact.

We came to the following values of the inner parameters of our method:

- **tolerance** = 2.5 (Sec. 3.3.2)
- **dist** = 12 (Sec. 4.3.3)
- **cover** = 0.8 (Sec. 4.4.2)
- **steps** = 5 (Sec. 4.2)

We came to the following values of the regular parameters of our method:

- **diameter** = 14 (Sec. 4.1)
- **variance** = 8 (Sec. 4.1)
- **angle** = 120 *degrees* (see Sec. 4.3.3)
- **distance** = 60 (Sec. 4.3.1)
- **circularity** = 0.48 (see Sec. 4.4.1)
- **area** depended on the image (Sec. 4.3.4)

The performance of our method greatly depends on the settings of all of its parameters.

We came to the following values of the parameters of the watershed method:

- **diameter** = 103
- **variance** = 17

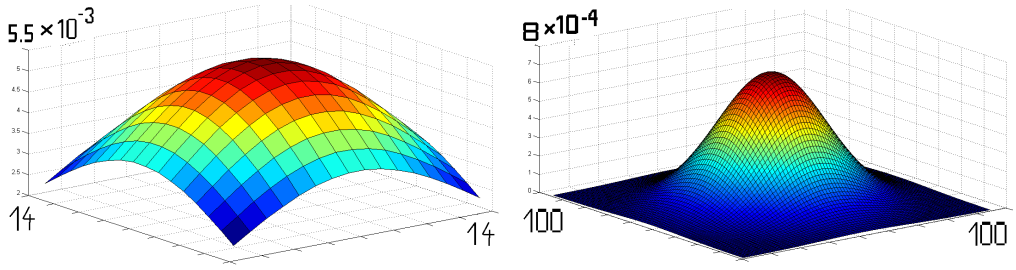
The parameters of the watershed method are the parameters of the Gaussian convolution core, which is used to blur the image converted to gray-scale before the algorithm is launched. The final performance of the watershed greatly depends on their values. If they are the same as the ones used for our method (14, 8), the mean f-score of the watershed reaches only about 0.4 on the test set.

In Fig. 6.1, you can see a comparison of the Gaussian convolution cores used for our method and for the watershed method. Even though the maximum value of the second one is much smaller, it causes a more substantial blurring, because it is the ratio of the weights between the pixel and its neighbors what counts

of 10^9 evaluations. The evaluation of one image takes approx. 40 seconds, which sums up to 320 seconds for eight images - the whole test set. This sums up to $10^9 \cdot 320$ seconds in total, which is approx. 10147 years. Of course, we could accelerate the implementation of the method or parallelize the evaluation, but this would not shorten the total amount of time to a reasonably bearable length.

[2]Among the set of sampled values of parameters.

during the convolution. The watershed works better with a more substantial blurring than our method. This might be because the watershed profits from the increased smoothness of the interiors of the cells towards their borders. However, the halos are not so distinct after a more substantial blurring, which decreases the performance of our method.



(a) The core used for our method. **(b)** The core used for the watershed algorithm.

Figure 6.1 A comparison of the Gaussian cores used for our method and for the watershed method.

6.2 The performance of our method and the watershed algorithm

As we mentioned, we gave the watershed method a suitably blurred image. Moreover, we also identified the background objects in its output segmentation, the same way as we do in our method (see Sec. 4.4.2). This way, we remarkably reduced the number of objects segmented by it to make the comparison more relevant.

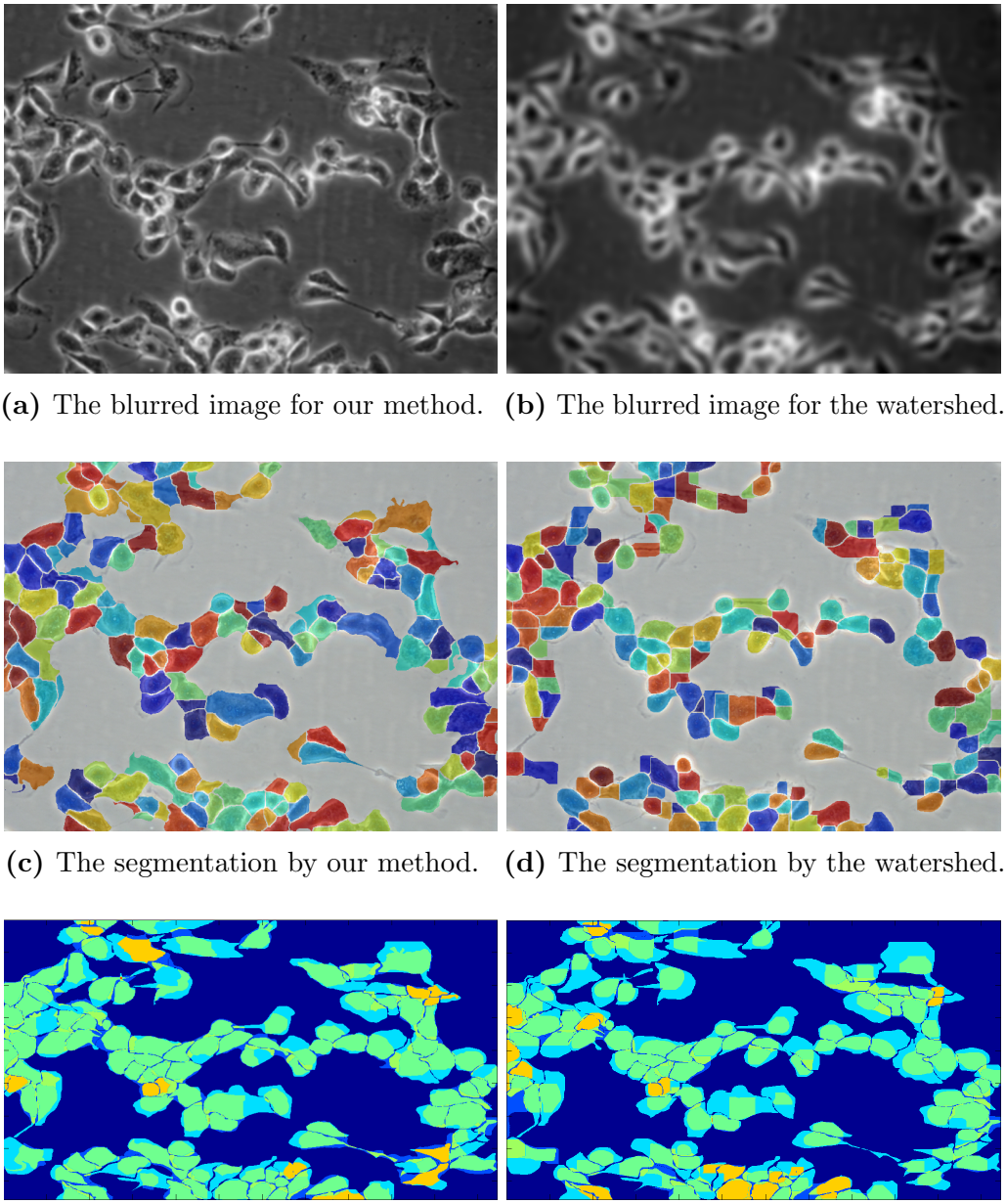
In Fig. 6.2, you can see a graphical comparison of the two methods on one image. In Fig. 6.2e and 6.2f, the green color indicates matches with the test objects (true positive), the light-blue color indicates the parts of the test objects, which were not matched by a segmented object (false negative) and the orange color indicates the parts of the segmented objects, which were matched to more than one test object (false positive).

Thus, the larger the total light-blue area, the higher the oversegmentation and the lower the recall. The larger the total orange area, the higher the undersegmentation and the lower the precision. From the presented matching maps (Fig. 6.2e and 6.2f), we can form a hypothesis that the watershed suffers from oversegmentation more than our method.

Let us take a look at the performance of these methods on the *test set*.

Tab. 6.1 and Tab. 6.2 show the mean values of the measured quantities over the images in the *test set*, along with their standard deviations. Fig. 6.3 shows these quantities graphically.

These results confirm the hypothesis based on the displayed matching maps (Fig. 6.2e and 6.2f). The mean recall of the watershed is significantly lower, which means that it tends to oversegment more. The mean f-score of our method is higher, but the standard deviations of this quantity prevent us from stating that our method has outperformed the watershed in this aspect. Both methods tend to similarly overestimate the number of cell in the image. The mean running



(a) The blurred image for our method. **(b)** The blurred image for the watershed.

(c) The segmentation by our method. **(d)** The segmentation by the watershed.

(e) The matching map of our method. **(f)** The matching map of the watershed.

Figure 6.2 A comparison of the results our method and the watershed method.

time of the watershed is about 2.5 times lower, with a much smaller deviation. The watershed is also a lot simpler method.

	F-score	Precision	Recall	Counts ratio	Time elapsed
Mean value	69.28 %	67.47 %	71.99 %	131.32 %	44.7824 s
Standard deviation	5.74 %	8.20 %	6.95 %	24.85 %	13.3272 s

Table 6.1 The results of our method on the test set.

	F-score	Precision	Recall	Counts ratio	Time elapsed
Mean value	65.00 %	67.31 %	64.09 %	131.11 %	17.1270 s
Standard deviation	5.60 %	7.59 %	9.69 %	36.22 %	5.2811 s

Table 6.2 The results the watershed method on the test set.

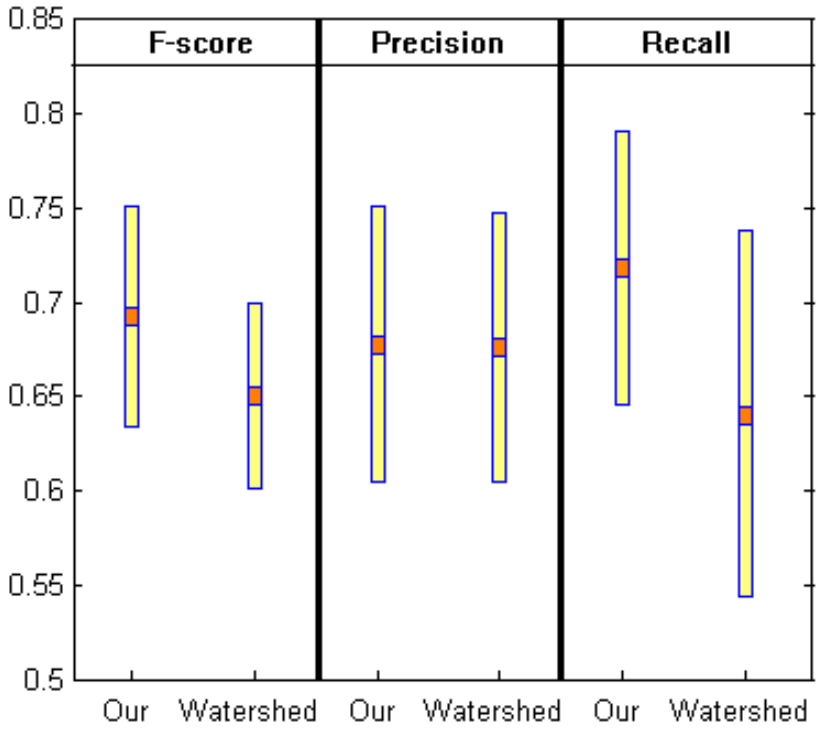


Figure 6.3 A graphical comparison of the results of the two methods.

7. Conclusion

We proposed a new method for the segmentation of live mammalian cancer cells from microscopic images. This method is composed of simple methods from the field of the mathematical morphology and the theory of graphs. Its functionality is dependent on the presence of the halos between the cells. Its main assumption is that the boundaries are lighter than the rest. It is not based on machine learning, which limits its universality. On the other hand, it does not require any training set and is not so computationally demanding. The method forms quite a complicated pipeline of mostly simple operations, many times just some heuristics. It is highly dependent on the settings of its parameters. It ignores much of the information present in the input image. However, it works well on the images, where the boundaries between the cells are distinctly light.

We proposed an evaluation metrics for the overall quality of segmentation based on the accuracy of the segmented areas. According to this metrics, we compared the results of our method to the results of the publicly available watershed algorithm. The results were not very convincing due to the deviations, however, our method reached a higher mean f-score than the watershed. As our evaluation showed, both these methods heavily rely on the settings of their parameters. Despite the fact, that the watershed originally does not require any parameters, its results significantly improved after a heavy blurring of the input image. Our method utilizes blurring too, but in addition, it contains several more parameters, which need to be tuned in order to optimize its performance. This is definitely its serious drawback.

We implemented our method and the evaluation based on the proposed metrics in MATLAB [13], including subprograms written in Java for some low-level subroutines, such as Dijkstra's algorithm [8] or matching the objects segmented by our method to the objects segmented by human experts. We also created a GUI for this method.

We implemented a GUI segmentation editor in Java. With this program, a user can easily correct the mistakes in the output of our method. Moreover, this program can also serve as an intuitive and straightforward tool for manual segmentation of cells, which can be subsequently utilized for training a machine-learning classifier.

7.1 Future work

The quality of the segmentation performed by this method could improve by exploiting the time-lapse character of the images somehow. Further improvements could be achieved by integrating more advanced method. For example, an active contour model could be used to improve the position of the boundaries between cells provided by our method. Moreover, machine learning methods could be used to reduce the undersegmentation and the oversegmentation.

However, our method is heavily dependent on its parameters, it is far from universal and it is prone to various inaccuracies. It would be ideal to develop a method which does not suffer from these problems. In the future, we would like to focus on developing a machine-learning method for this purpose. Our

idea is to start experimenting with neural networks. Firstly, we plan to get up to date with the knowledge available in this area. Secondly, we want to choose an appropriate approach. An inspiring one is present in a paper by IDSIA about segmenting neuronal membranes with the help of deep neural networks [6]. They used graphical processing units to implement fast neural networks. They also employed the concept of deep learning [1]. They fed the networks directly with the pixel intensities and achieved remarkable results.

Bibliography

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives, 2012.

[2] S. Beucher and Centre De Morphologie Mathmatique. The watershed transformation applied to image segmentation. In *Scanning Microscopy International*, pages 299–314, 1991.

[3] Harry Blum et al. A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form*, 19(5):362–380, 1967.

[4] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: 10.1145/130385.130401. URL `http://doi.acm.org/10.1145/130385.130401`.

[5] C. K. Chui. *An introduction to wavelets*. Academic Press, Boston, 1992. ISBN 0121745848.

[6] Dan Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2843–2851. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4741-deep-neural-networks-segment-neuronal-membranes-in-electron-microscop pdf`.

[7] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. ISSN 0885-6125. doi: 10.1007/BF00994018. URL `http://dx.doi.org/10.1007/BF00994018`.

[8] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. ISSN 0029-599X. doi: 10.1007/BF01386390. URL `http://dx.doi.org/10.1007/BF01386390`.

[9] I. Ersoy, F. Bunyak, M. A. Mackey, and K. Palaniappan. Cell segmentation using hessian-based detection and contour evolution with directional derivatives. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 1804–1807, Oct 2008. doi: 10.1109/ICIP.2008.4712127.

[10] Dong-Chen He and Li Wang. Texture unit, texture spectrum, and texture analysis. *Geoscience and Remote Sensing, IEEE Transactions on*, 28(4): 509–512, 1990.

[11] Weijun He, Xiaoxu Wang, Dimitris Metaxas, Robin Mathew, and Eileen White. Cell segmentation for division rate estimation in computerized video time-lapse microscopy. In *Biomedical Optics (BiOS) 2007*, pages 643109–643109. International Society for Optics and Photonics, 2007.

[12] J. Hilditch. Linear skeletons from square cupboards. *Machine Intelligence*, 4: 404–420, 1969.

[13] The MathWorks Inc.

[14] Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. ISBN 0-13-336165-9.

[15] George H. Joblove and Donald Greenberg. Color spaces for computer graphics. *SIGGRAPH Comput. Graph.*, 12(3):20–25, August 1978. ISSN 0097-8930. doi: 10.1145/965139.807362. URL `http://doi.acm.org/10.1145/965139.807362`.

[16] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[17] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL `http://dl.acm.org/citation.cfm?id=645530.655813`.

[18] Kang Li, Eric D Miller, Mei Chen, Takeo Kanade, Lee E Weiss, and Phil G Campbell. Cell population tracking and lineage construction with spatiotemporal context. *Medical image analysis*, 12(5):546–566, 2008.

[19] WarrenS. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. ISSN 0007-4985. doi: 10.1007/BF02478259. URL `http://dx.doi.org/10.1007/BF02478259`.

[20] Nobuyuki Otsu. A Threshold Selection Method from Gray-level Histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62–66, 1979. doi: 10.1109/TSMC.1979.4310076. URL `http://dx.doi.org/10.1109/TSMC.1979.4310076`.

[21] J. Pan, T. Kanade, and Mei Chen. Heterogeneous conditional random field: Realizing joint detection and segmentation of cell regions in microscopic images. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2940–2947, June 2010. doi: 10.1109/CVPR.2010.5540037.

[22] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Machine Learning*, pages 80–91, 1999.

[23] Jindřich Soukup, Petr Císař, and Filip Šroubek. Segmentation of time-lapse images with focus on microscopic images of cells. In Alfredo Petrosino, editor, *Image Analysis and Processing – ICIAP 2013*, volume 8157 of *Lecture Notes in Computer Science*, pages 71–80. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-41183-0. doi: 10.1007/978-3-642-41184-7_8. URL `http://dx.doi.org/10.1007/978-3-642-41184-7_8`.

[24] Zhaozheng Yin and Takeo Kanade. Restoring artifact-free microscopy image sequences. In *Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on*, pages 909–913. IEEE, 2011.

[25] F. Zernike. Phase contrast, a new method for the microscopic observation of transparent objects. *Physica*, 9:686–698, July 1942. doi: 10.1016/S0031-8914(42)80035-X.

List of Figures

Attachments

A - CD

The attached CD contains:

- Demonstrating data - a few images of cells, together with their respective outputs of the detection of clusters.
- A GUI enabling a user to segment the images by this method. A segmentation editor can be launched from there in order to correct the outputs of the method.
- An installation manual together with all files needed for the installation.
- A user manual for the whole GUI, together with the minimum requirements of the program.
- All source files - (MATLAB: .fig, .mat, Java: .class) needed for the compilation of the program. Note that deploytool by MATLAB is needed to compile the MATLAB files.

B - The graphs justifying the values of the parameters

The following graphs show how the value of the mean f-score and its deviation change, when we shift the value of a single parameter at a time.

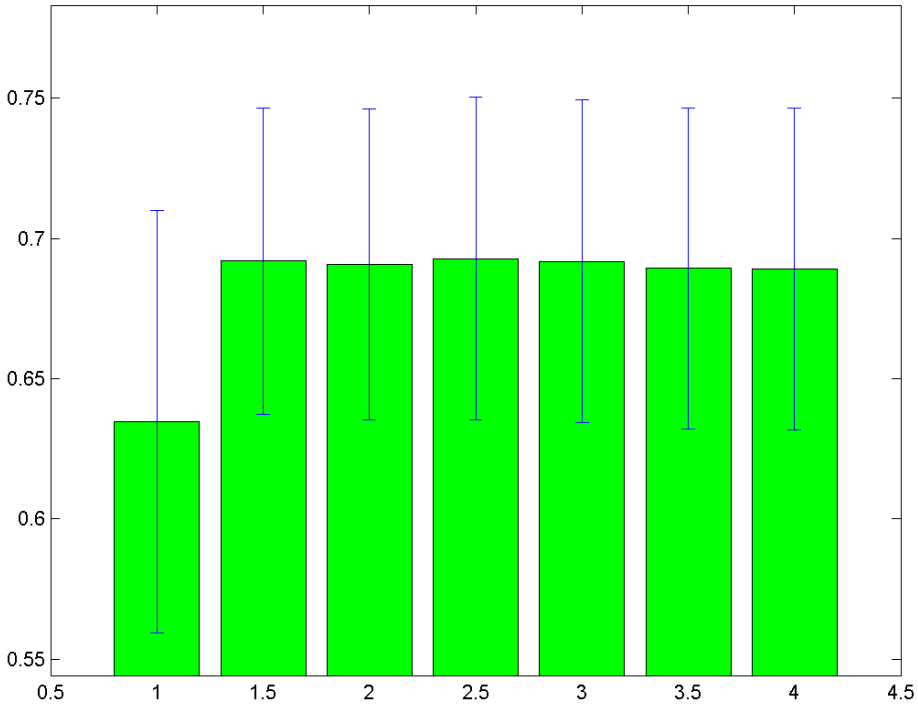


Figure 7.1 Our method: **tolerance**

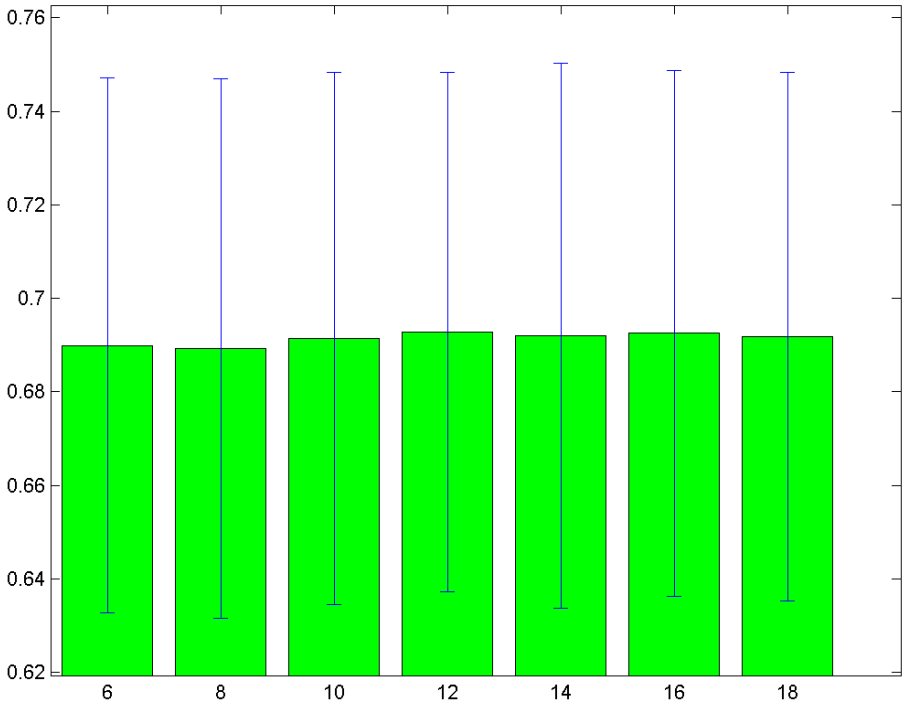


Figure 7.2 Our method: **dist**

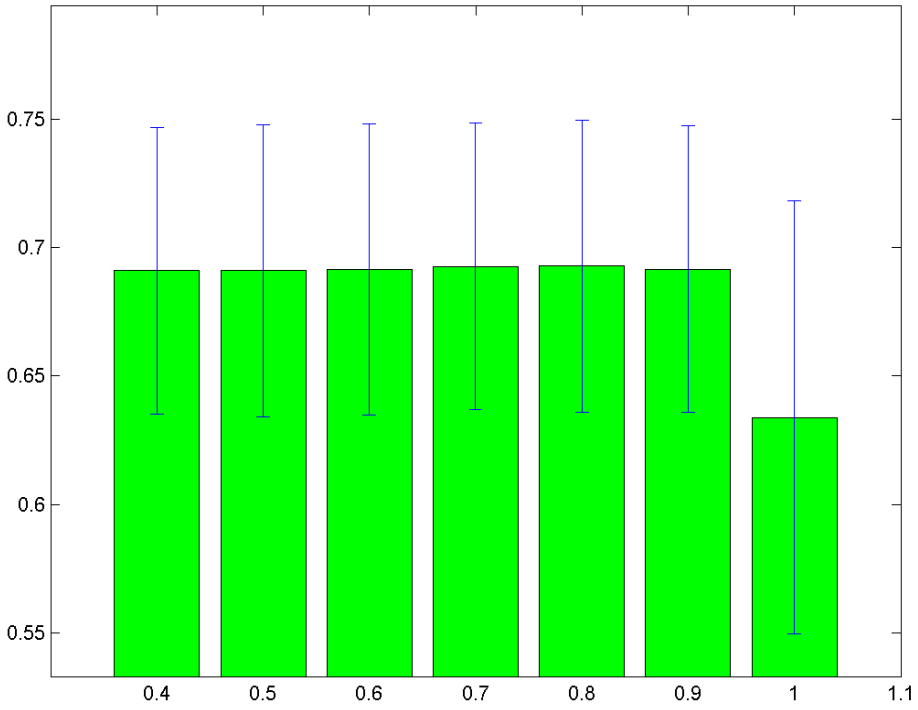


Figure 7.3 Our method: **cover**

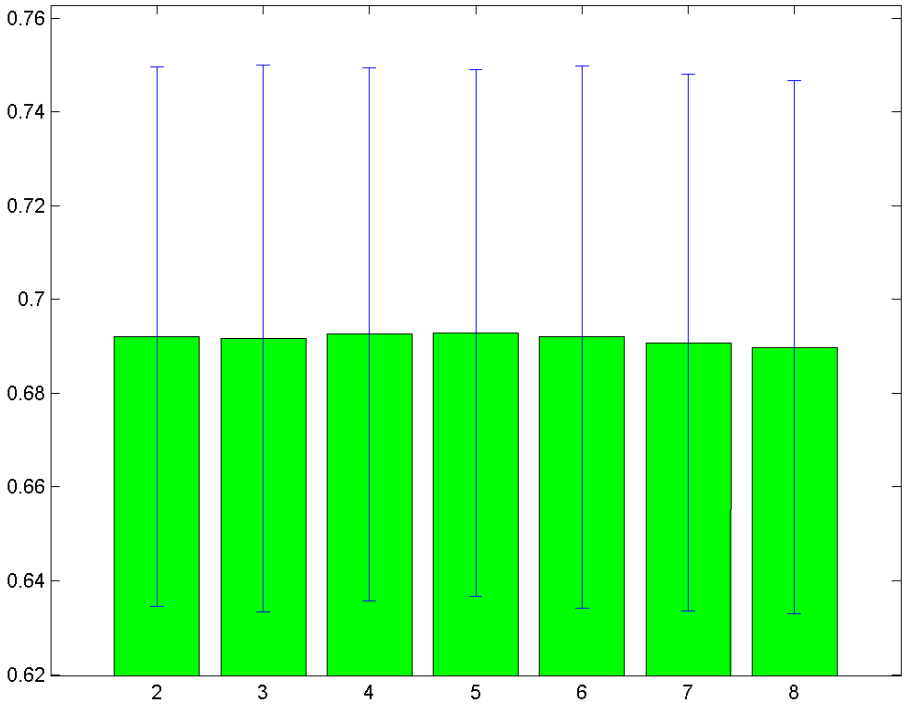


Figure 7.4 Our method: **steps**

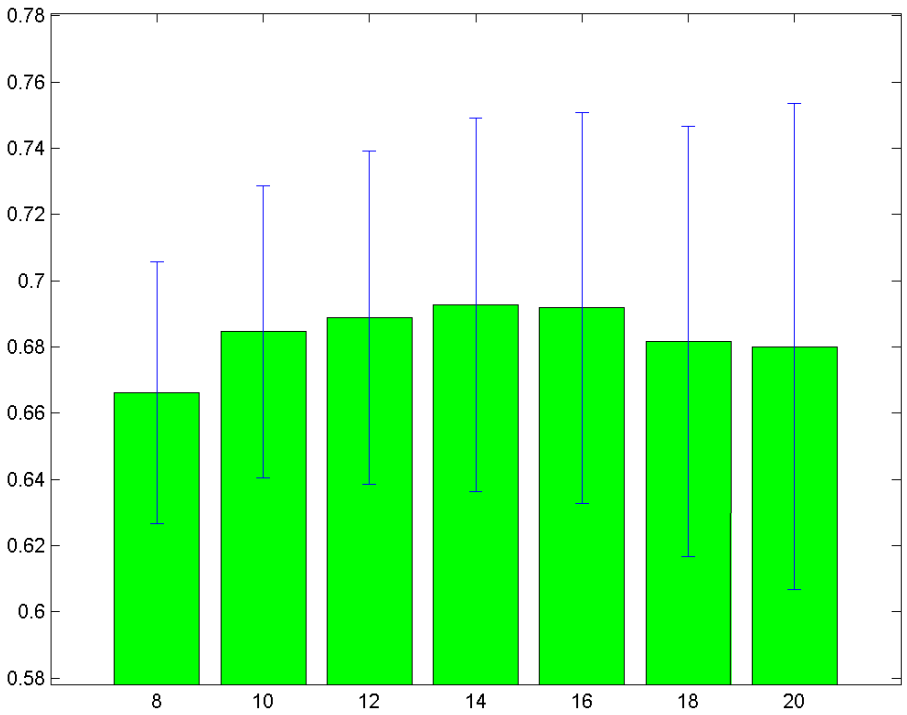


Figure 7.5 Our method: **diameter**

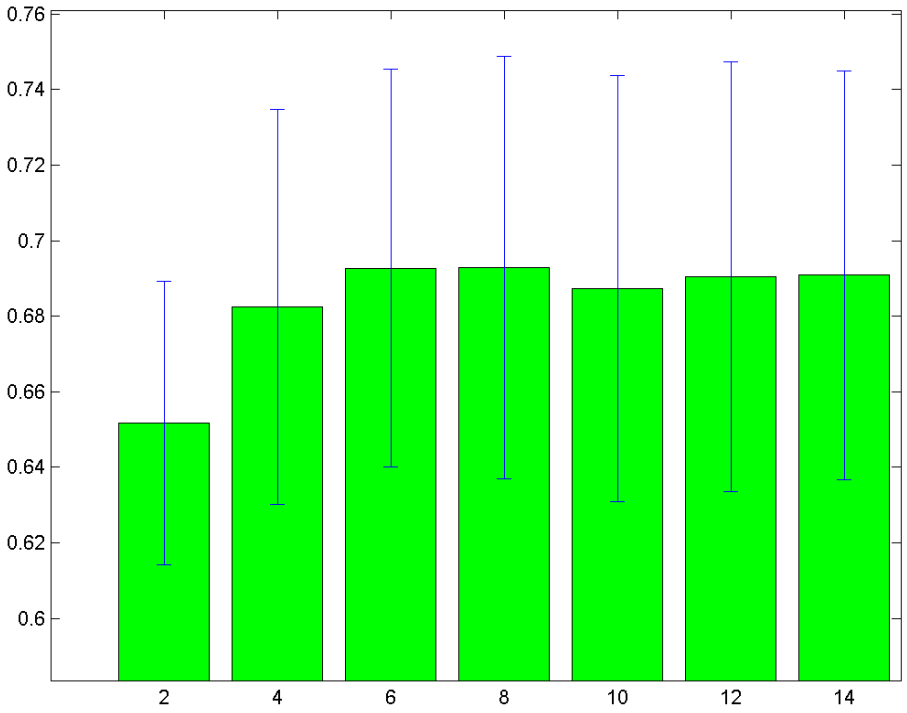


Figure 7.6 Our method: **variance**

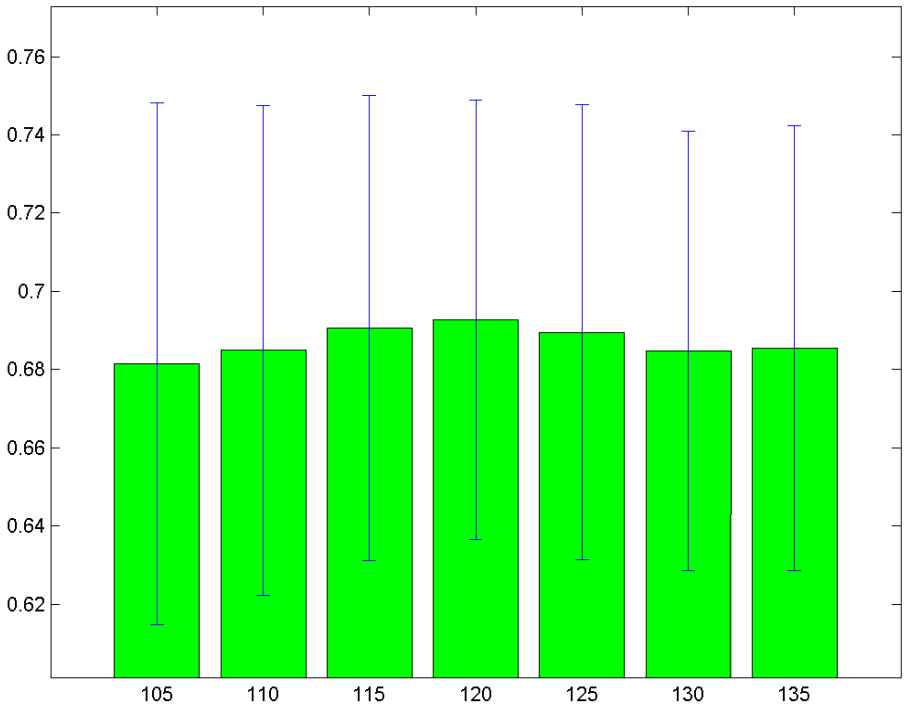


Figure 7.7 Our method: **angle**

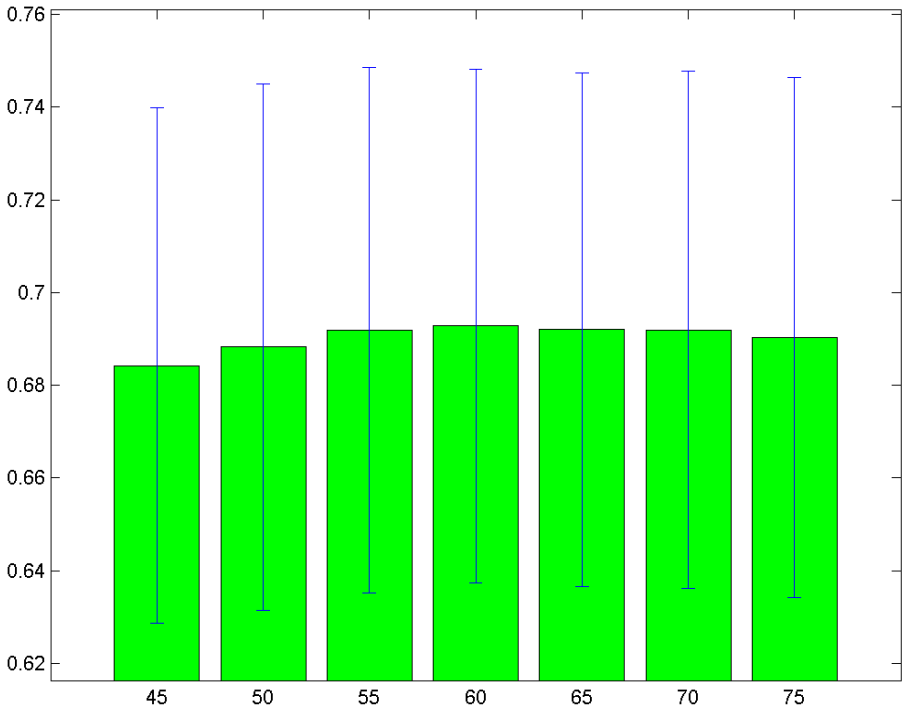


Figure 7.8 Our method: **distance**

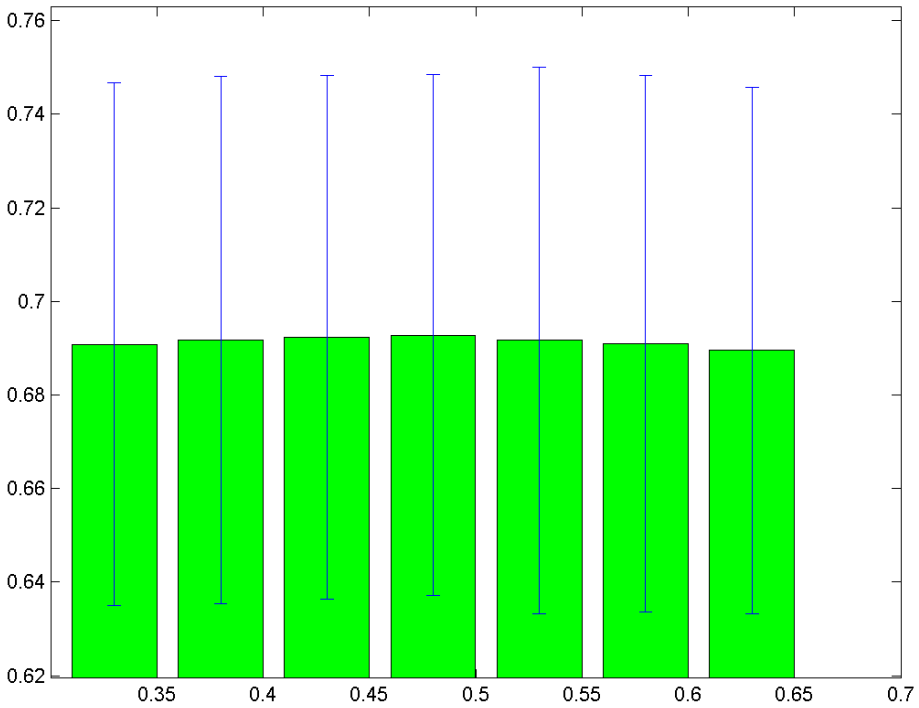


Figure 7.9 Our method: **circularity**

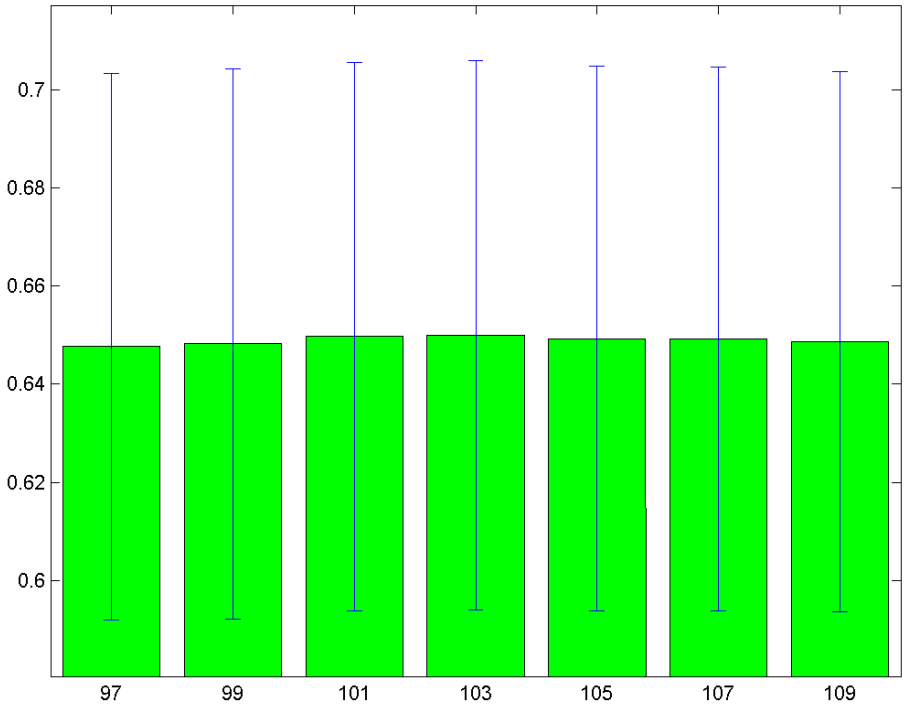


Figure 7.10 The watershed: **diameter**

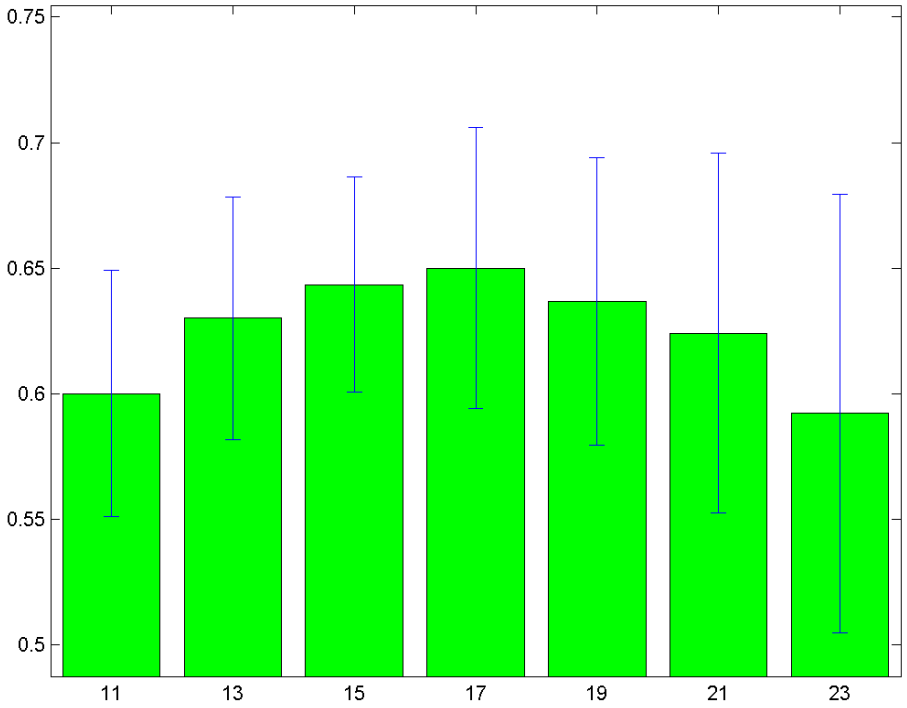


Figure 7.11 The watershed: **variance**