Automated Detection of Nucleoli Using ImageJ

Nico Hochberger

November 2014

Contents

List of Figures

List of Tables

1 Introduction

1.1 Preface

1.2 Objective

1.3 Motivation

Currently nucleoli are detected using an application called CellProfiler[1]. While this application yields reliable results, it also takes pretty long to complete the analysis. Runtimes up to 45 seconds are common.

Due to the fact that CellProfiler is a very general approach, applicable to a large variety of tasks related to detecting nuclei and nucleoli, the results of its analysis have to be checked manually to reduce the amount of false-positives.

- TODO -

Consequently, this leads to a more specialized way of analyzing the cells, which does not do all the analysis performed by CellProfiler, but on the other hand is much faster and thus helps to prevent cells dying before the analysis is finished.

[1] http://www.cellprofiler.org

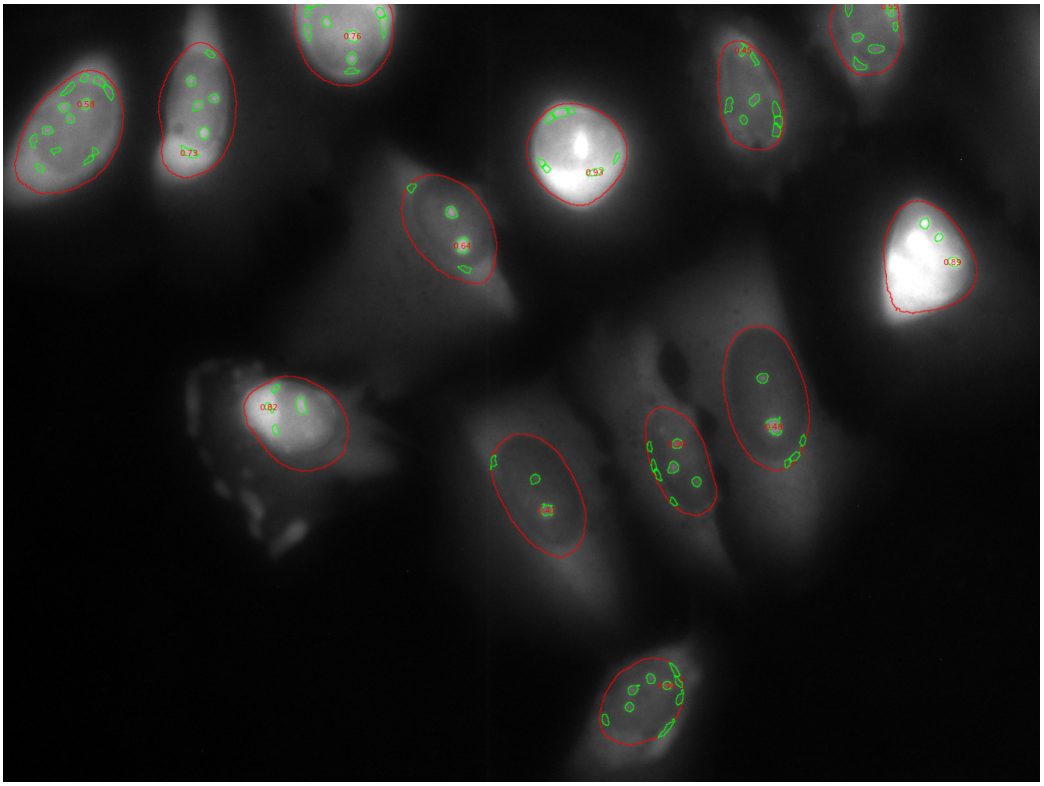


Figure 1: Example analysis performed with CellProfiler

2 Requirements

The application has to meet the following requirements:

- **Reliable nucleoli detection:** Stable and reliable detection of nucleoli is the main purpose of the application. Hence, it is supposed to detect at least 75% of the nucleoli CellProfiler can detect. This includes a certain degree of stability concerning fuzzy pictures or pictures with unequally distributed or diffuse brightness.

- **Fast analysis:** As the application is tailored to this single task, it is expected to detect a suitable amount of nucleoli in only a small fraction of the time required by CellProfiler. The topmost time that the application may require to complete the analysis of one image is five seconds.

- **Fallback in case of empty nuclei:** Each nucleus is expected to containing at least one nucleolus. Yet, this expectation cannot always be fulfilled due to potentially damaged nuclei, fuzzy images, or other

reasons. In this case, the center of the nucleus has to be provided as fallback target.

- **Visualizer:** In order to quickly check the results directly after running the analysis and to provide a way to quickly present the results to a potential audience, the application has to provide the possibility to be configured so that it shows the results as an image. This image has to contain all detected neuleoli targets, fallback targets and the regions of interest, e.g. the nuclei.

- **Versatility:** Since the appearance of different specimen can vary in various ways, all analysis parameters have to be configurable. Among others, this includes the minimum and maximum sizes of nuclei and nucleoli. The configuration is supposed to be achieved via an understandable, interchangeable, text-based file[2].

- **Statistics:** To determine the most suitable parameters for different kinds of specimen, another feature may be configured. This statistics feature has to include:

  - The amount of detected nuclei
  - The amount of detected nucleoli
  - Nuclei to nucleoli ratio as percentage
  - The distance of each detected nucleolus to the center point of the containing nuclei and the average distance in pixels
  - The area of each detected nucleus and the average area in square pixels
  - The area of each detected nucleolus and the average area in square pixels

- **Serialization of the results:** All results have to be stored in their accordant files in a subfolder *results* of the folder containing the original data. The name of each result file has to contain a the timestamp indicating the application's execution in the format `<year>_<month>_<day>_<hour>_<minute>_<second>`.
  E.g. `targets_2014_11_20_14_50_35.txt`.
  In the following, the accordant formats and files are described.

[2]Configurable parameters are explained in detail in the User's Manual section

- **Targets:** Real targets and fallback targets sre to be saved in one txt-file named `targets_<timestamp>.txt` in the following format:

```
# nucleoli targets
<target number> : [<x-coord>, <y-coord>]
...
# targets in center of empty nuclei
<target number> : [<x-coord>, <y-coord>]
...
```

Listing 1: Format of results txt-file

Example:

```
# nucleoli targets
1 : [468, 43]
2 : [1183, 14]
# targets in center of empty nuclei
3 : [87, 174]
4 : [769, 198]
```

Listing 2: Example of results file

- **Statistics:** The statistics as mentioned above have to be stored in a txt-file named `statistics_<timestamp>.txt`. An example of the statistics file can be found in the appendix.
- **Result image:** The image as it would be displayed by the visualizer has to be stored to a file named `targets_<timestamp>.<original image filetype>`. See Figure 2 for an example the image.

- **Quick and easy deployment:** The application is supposed to be deployed and run as easily as possible. Since the environment and the machines this will happen on may vary drastically, a versatile and independent way to do this is required. The Java framework delivers this possibility. Hence, the application will be developed using Java.
- **Use of an established graphics framework:** In order to keep maintenance effort as small as possible and, in case of potential future development, time it takes the developer to familiarize with the project, as short as possible, an established and well documented graphics framework has to be used. ImageJ meets these requirements and furthermore can be easily used as a library within a Java application.

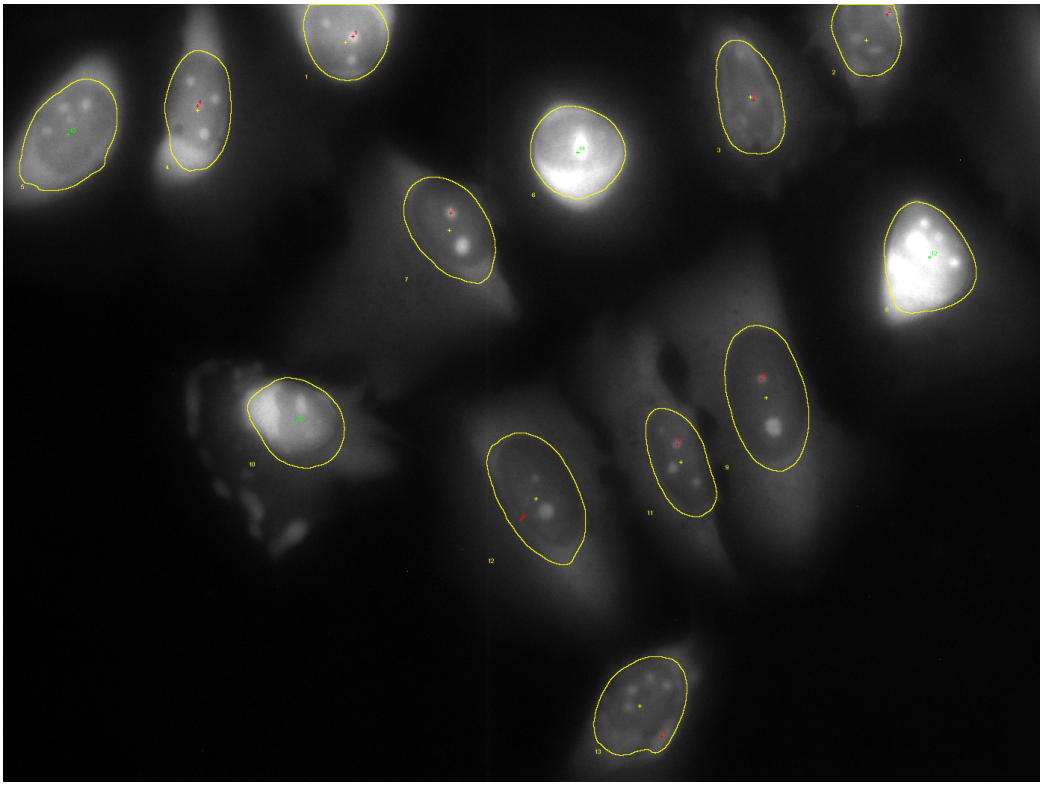


Figure 2: Example of the image containing the results

3 User's Manual

3.1 System Requirements

The application requires the Java Runtime Environment 1.7 or higher. For Microsoft Windows 64bit operating system this results in the following minimum system requirements [3]:

Memory:	128MB
Processor:	Pentium 2 @ 266MHz
Disc space:	124 MB + 2 MB

Table 1: Minimum system requirements of Java 1.7 on Windows 64bit

As it is good practice and vital to the security of any system, the Java framework should be updated on a regular basis. Note that the requirements

[3]Further details on the system requirements can be found at: https://docs.oracle.com/javase/7/docs/webnotes/install/

may change with any update.

3.2 Starting the Application

Starting the application is accomplished using the command line interface via Java's `java -jar` or `javaw -jar` command[4]. Furthermore, the application expects the configuration file to be passed as single parameter.

Considering a scenario in which the configuration file (see 3.3) is stored as `configuration.txt` in the same folder as the `NuFi.jar`, the following command will properly start the application:
`java -jar NuFi.jar configuration.txt`.

3.3 Configuration

Configuration of the application is performed using a simple text file containing the information listed in the following. Note that a missing parameter will result in a fatal error, preventing the application from properly working.

A complete example of the configuration file can be found in the appendix.

3.3.1 General Parameters

This section contains information on filetypes, source folders and naming convention of source files.

- **Source folder:** The folder containing the source files (e.g. image files) on which the analysis will be based. The source folders are expected to be provided as absolute path or relative to `NuFi.jar`.

  Note that the application expects exactly three files of the type defined as channel filetype. If there are more or less than three files of that type, a fatal error will occur.

  Key:
  `source.folder`

  Examples:
  `source.folder = C:/example/images`
  `source.folder = ../../example/images`
  `source.folder = example/images`

[4]See https://docs.oracle.com/javase/7/docs/technotes/tools/windows/java.html for further details.

- **Used channels:** Each specimen is photographed three times using different colorization. This results in three different images referred to as different channels. Channels 1 and 3 are used in the process of detecting targets, thus, the correct specification and order of the files is vital for image analysis. Position one determines channel 1, position 3 determines channel 3. The source folder is scanned for files containing these channels determination. If these are not found, the application will terminate with a fatal error.

  Key:
  `used.channels`

  Examples:
  `used.channels = Kanal1, Kanal2, Kanal3`
  `used.channels = channel1, channel2, channel3`

- **Channel filetype:** Though using the png-file format is advised, the application is built to be able to analyze various image formats. If it is necessary to analyze images that are not provided in the png-format, the accordant format can be provided using this key.

  Key:
  `channel.filetype`

  Examples:
  `channel.filetype = png`
  `channel.filetype = jpg`

3.4 General Picture Analysis Settings

This section contains information of the expected size and shape of nuclei and nucleoli, as well as the width of the in depth-analysis.

- **Size settings:** Since the size of the nuclei and nucleoli in the specimen may vary and since the magnification of the microscope used in taking the pictures may also change, the size in the photographs may need to be adapted. This can be achieved by giving an average size in square-pixels and the minimum and maximum sizes as multiples of that size.

  Keys:
  `nucleus.average`
  `nucleus.min.factor`
  `nucleus.max.factor`
  `nucleolus.average`

`nucleolus.min.factor`
`nucleolus.max.factor`

Examples:
`nucleus.average = 10000`
`nucleus.min.factor = 0.5`
`nucleus.max.factor = 1.5`
`nucleolus.average = 100`
`nucleolus.min.factor = 0.75`
`nucleolus.max.factor = 1.8`

- **Minimum circularity:** This parameter pays respect to the variation of the objects in their shape. A value of 1 indicates a perfect circle, while a 0 indicates that there are no requirements to the shape of the object. Since nuclei typically are elliptical in shape, a value of approximately 0.6 is advised. Nucleoli are typically close to a perfect circle. Thus, a value of 0.9 includes most of them while it excludes a variety of anomalies and prevents false-positives on nuclei borders.

  Keys:
  `nucleus.min.circularity`
  `nucleolus.min.circularity`

  Examples:
  `nucleus.min.circularity = 0.6`
  `nucleolus.min.circularity = 0.9`

- **In-depth width:** This parameter defines the variation in thresholding used during in-depth analysis.

  Key:
  `indepth.range`

  Examples:
  `indepth.range = 25`
  `indepth.range = 10`

3.4.1 Improved Image Detection Parameters

In the standard setting the application will use the improved image detection algorithm. In this section, the parameters required for this algorithm can be defined. These include setting for brightness correction (`<object>.background.blur` and `<object>.thresholding.blur`) and the value used for correcting discrepancies between channel 3 colorization and the actual size of nuclei (`nucleus.boundary.width`

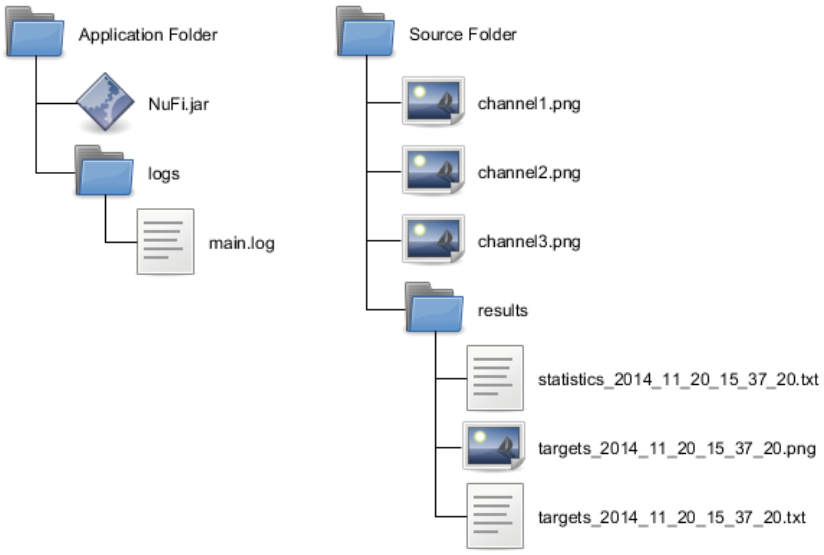


Figure 3: Structure of directories and files created in the application's folder and the source folder

- **Brightness correction:**

Keys:

`nucleus.background.blur`
`nucleus.thresholding.blur`
`nucleus.boundary.width`
`nucleolus.background.blur`
`nucleolus.thresholding.blur`

Examples:

```
nucleus.background.blur = 100
nucleus.thresholding.blur = 3
nucleus.boundary.width = 5
nucleolus.background.blur = 10
nucleolus.thresholding.blur = 1
```

3.5 Structure of Files and Folders

The results of the image analysis will be stored into a directory `results` which is created as a subfolder of the directory defined as source folder. This directory will contain the targets file, the result image and, in case statistics

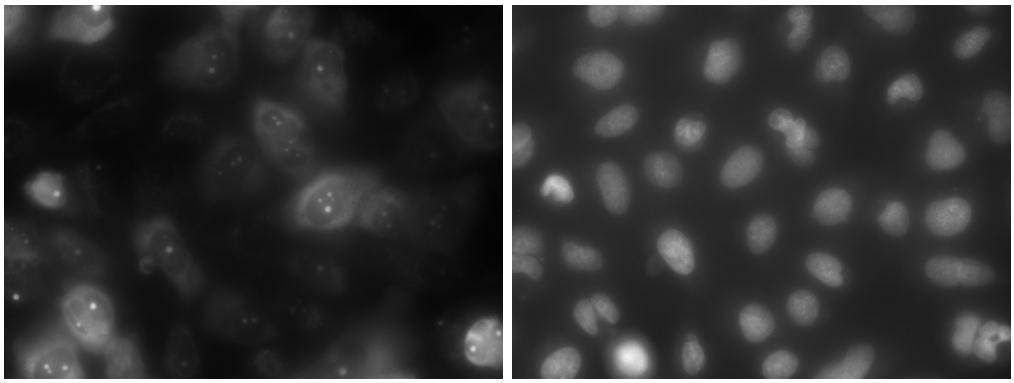
are enabled, the statistics file. See figure 3.

The application will neither delete nor overwrite files created by a previous execution since each file contains a timestamp as defined in 2.

Besides this, the application will create a directory `logs` on the same level as `NuFi.bat` is located. This directory contains the log-file of the application itself. If any unexpected behaviour is encountered while using the application, refer to this file since it might contain helpful information. See figure 3.

4 Image Analysis

In this section the procedure of how the application processes and analyzes images is explained. Details such as how the applications checks its configuration are omitted in favor of more emphasis on image analysis.



(a) Channel 1 example image (b) Channel 3 example image

Figure 4: Example images of channel 1 and 3

4.1 Target Detection Procedure

The target detection process can basically divided in two parts. At first, the regions of interest are detected using channel 3. These regions are then transferred to channel 1 in order to only analyze those regions that may actually contain nucleoli.

In the following, these two parts are explained step-by-step using the example images in figure 4.

4.1.1 Determining Regions of Interest

Finding the regions of interest is performed using only the image referred to as channel 3, as this is the image which best shows the nuclei due to the colorization used.

Typically the images show very uneven exposure to light, resulting in the necessity of brightness correction as a very first step. This is achieved by using a gaussian blur with a radius approximately the size of the objects that are supposed to be found, in this example, a value be-

Figure 5: Channel 3 after Gaussian Blur with a radius of 100 pixels has been applied

tween 100 and 120 yields pretty useful results. The result of this operation can be seen in 5 and is referred to as background image. The original channel 3 image is then divided by the background image, which means that each pixel value of the original image is divided by the value of the pixel of the background image that is at the same location. This leads to an image with very small pixel values, typically close to zero, e.g. the resulting image is all black. Consequently, the next step is to multiply each pixel value with a constant factor, e.g. 178. As a last step in this preparation process, a gaussian blur with a small radius, e.g. 2px, is applied to the image in order to smooth the edges of the nuclei and to reduce noise. The result of these steps can be seen in 6.

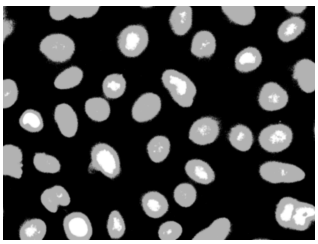


Figure 6: Channel 3 after brightness correction has been performed

Having finished this preparation, channel 3 now shows evenly distributed brightness and only little variation in these. This makes the following step, finding a suitable threshold which is described in detail in 4.2, easier and much more stable across different images. Figure 7 shows channel 3 after an auto default threshold was applied.

In the last part of determining the regions of interest, the areas to which the threshold applies, e.g. are marked in red, are analyzed using ImageJ's Particle Analyzer plugin. Details on how Particle Analyzer works are explained in 4.3. Based on configured values for minimum and maximum area a nucleus may cover and minimum circularity, the regions of interest are determined. Figure 8 shows the result of running a particle analysis on figure 7 using a minimum area of 8000 pixel^2, a maximum area of 15000 pixel^2 and a minimum circularity of 0.6. Obviously, not all nuclei are recognized as reagions of interest. In this example, the area of these nuclei are too large or too small. If the specimen contains a vast amount of too large or too small nuclei, changing the configured minimum and maximum sizes should be adapted.

The regions of interest determined in this first part of the image analysis are the basis of finding nucleoli in the next step. If these regions cannot be properly determined, for example due to low quality images or flawed colorization, the subsequent analysis will not yield feasible results.

See appendix for high resolution versions of the images used in this section.

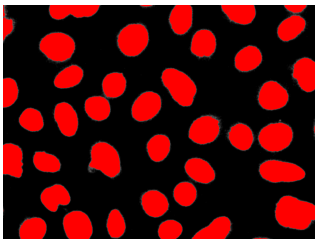


Figure 7: Channel 3 with auto threshold applied

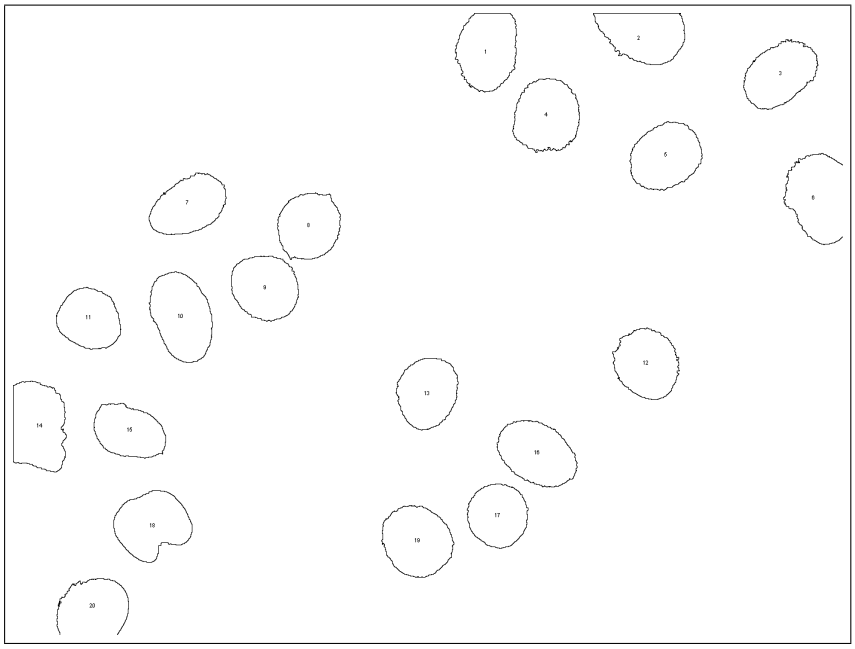


Figure 8: Regions of interest found by Particle Analyzer in thresholded channel 3 image

4.1.2 Finding Nucleoli

Having found the regions of interest, the area
which has to be searched is drastically reduced. Furthermore, since each region is analyzed separately, influence from uneven exposure to light is negligible. Hence, in favor of faster execution no further brightness correction is performed while detecting nucleoli. The following describes the process by the example of one region of interest, e.g. one nucleus. Nonetheless, in the whole process of target detection, this process is performed for each region of interest. The example nucleus can be seen in figure 9.

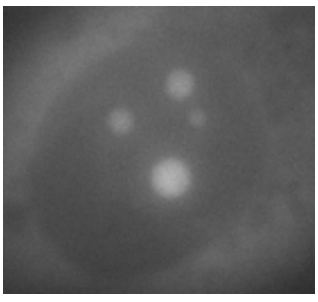


Figure 9: Example nucleus

The first step in order to analyze a single nucleus is to crop the affected region as seen in 9. Apparently, the region of interests border does not exactly match the boundaries of the nucleus. This is caused by the differences in colorization of channel 1 and 3 and by a delay between taking the pictures. Furthermore, the colorization of channel 1 causes the boundaries of nuclei to be nearly as bright as the nucleoli. Without proper correction false-positives are likely to appear on the bound-

aries. To counter this, the extend of the region of interest is shrunk by the configured amount of pixels. Additionally, in order to prevent the brightness outside the region of interest, all pixels that are not inside the region, are set to black. This results in the image depicted in figure 10.

After preparing the image of the nucleus like this, the analysis is continued by applying an auto determined threshold using ImageJ's auto thresholding procedures. In contrast to determining the regions of interest, where all thresholding methods lead to comparable results, different methods lead to significantly different results when applied here. Figure 11 shows the effects of different thresholding methods.

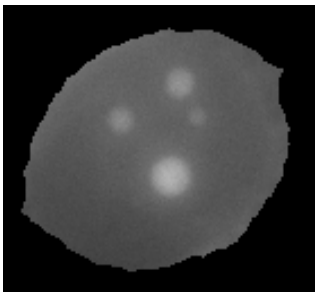
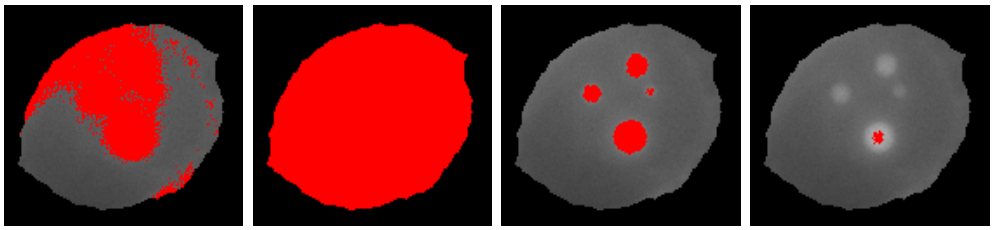


Figure 10: Region of interest shrunk by 8 pixels, everything outside the region is set to black

By comparing the results of different thresholding methods applied to numerous nuclei found in several pictures, the MaxEntropy method, depicted in figure 11c, was determined as the most suitable one.

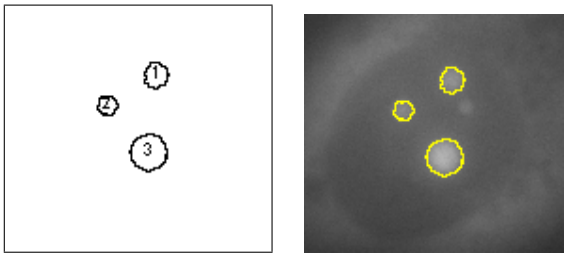


(a) Default (b) Huang (c) MaxEntropy (d) Shanbhang

Figure 11: Comparisson of different thresholding methods

Based on the threshold achieved by using the MaxEntropy method, a particle analysis is performed similar to the one performed to determine the regions of interest, but with parameters that pay respect to the size and shape of nucleoli rather than nuclei. Figure 10 shows the result of an analysis with a minimum size of 80 pixel^2, a maximum size of 350 pixel^2 and a minimum circularity of 0.8. Note that the nucleolus on the right is not recognized due to the fact that it is smaller than the configured minimum size.

To determine which of the nucleoli found is the most suitable target, the sizes of all nucleoli within one nucleus are compared. In order to ensure high accuracy during exposure to radiation later on in the process of analyzing the specimen, the largest nucleolus, e.g. covers the largest area, is chosen. Note that this will not result in choosing too large objects, since these are



(a) Plain regions of interest

(b) Regions of interest as overlay

Figure 12: Regions of interest found by particle analysis

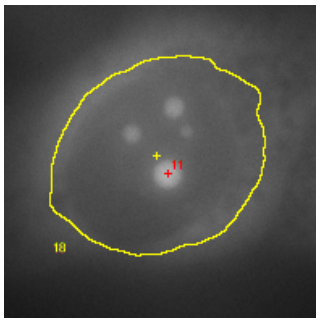


Figure 13: Extract from the result image showing the example nucleus as well as the chosen target

ignored due to the maximum size defined in particle analysis. In the example nucleus, region 3 is determined the most suitable choice, which can bee seen in figure 13.

In case no nucleoli are found in a certain nucleus, the application acts on the assumption that all nuclei contain at least one nucleolus and thus performs an in-depth analysis of the nucleus. This analysis is performed by using different thresholds based on the automatically found threshold and the configured width of the in-depth analysis. The nucleus will be thresholded with values of `threshold - inDepthWidth` to `threshold - inDepthWidth`. By stepwise increasing the threshold's value the possibility of not finding a nucleolus due to the fact hat it is just outside the minimum or maximum size because of a sub-optimal threshold is reduced. As soon as one nucleolus is found, the in-depth analysis is finished.

4.2 Thresholding

Despite the possibility of determining the threshold by hand or, respectively newly implement ways of automatically finding a suitable threshold, the application relies on ImageJ's AutoThresholder. This ImageJ plugin provides the possibility to use different algorithms(also: methods) to determine the threshold automatically. These include algorithms such as Huang (Hua93), Otsu (Gre10), MaxEntropy (Sac04) and others[5].

This section is intended to give a brief overview of how thresholding in general and the two methods used here, default and MaxEntropy, in particular, work. For a more detailed explanation please refer to the cited sources as this is not within the scope of this work.

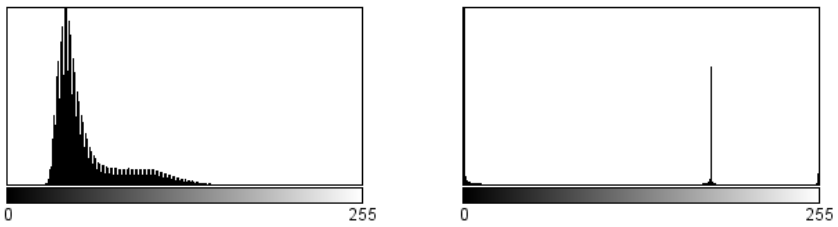
Thresholding is used to simplify image analysis by reducing the information contained within the image data. Typically this results in a binary, e.g. black and white, image. This may be achieved by simply comparing a pixel's (brightness) value to a defined value, the threshold. If the pixel's value is below that threshold, the value is set to 0 (black), if it is above, the value is set to 1 (white). Note that in some cases black and white may be switched in order to ease further processing. A very basic approach to do this can be seen in the below Java source code:

```
/*
width, height refer to the image's width and height

threshold refers to the predefined threshold

image.value(int x, int y) returns the value of the pixel at
  position (x, y)

image.setValue(int x, int y, double newValue) sets the value
  of the pixel at position (x, y) to newValue
*/

for (int x = 0; x < width; x++) {
  for (int y = 0; y < height; y++) {
    if (threshold > image.value(x, y)) {
      image.setValue(x, y, 0d);
    } else {
      image.setValue(x, y, 1d);
    }
  }
}
```

Listing 3: Basic thresholding algorithm

[5](Lan13) provides an elaborated overview on the AutoThresholder plugin and the available methods.

While this is pretty much the idea behind thresholding, there are several sophisticated approaches on how to determine a suitable value to be used as threshold based on an image's histogram and assumptions on how pixels are related to each other. Figure 14 shows the histograms of the original channel 3 image as shown in 4b and the brightness corrected version as shown in figure 6. Apparently, finding a suitable threshold based on the histogram is much easier in the brightness corrected version of the image.



(a) Original image (b) After brightness correction

Figure 14: Histograms of channel 3 prior to brightness correction and afterwards

The thresholding algorithm that is used in ImageJ by the name of *default* is basically a slight variation of the IsoData algorithm, compare (Lan13), which is also known as *iterative intermeans* described in (Rid78). This iterative algorithm defines an initial threshold as a basis for the first iteration, typically half the dynamic range, e.g. 178 if the range is 0 to 255. The algorithm then determines the average value of the pixels below and above the threshold. The threshold for the next iteration is then calculated as the arithmetic mean of these two values. The algorithm terminates when the value does not change anymore. The below source code shows a possible implementation of this algorithm:

The second method used in the application is based on the entropy found in the histogram (Kap84). According to (Sac04) the MaxEntropy threshold of a normalized histogram can be calculated as follows:

$$\sum_{i=0}^{i_{max}} h(i) = 1$$

Indicating that the histogram h is normalized, e.g. the sum of all values $h(i)$ equals 1. The entropy H_B of black pixels is defined as:

$$H_B(t) = -\sum_{i=0}^{t} \left(\frac{h(i)}{\sum_{j=0}^{t} h(j)} log \frac{h(i)}{\sum_{j=0}^{t} h(j)} \right)$$

while the entropy H_W of white pixels is defined as:

$$H_W(t) = -\sum_{i=t+1}^{i_{max}} \left(\frac{h(i)}{\sum_{j=t+1}^{i_{max}} h(j)} log \frac{h(i)}{\sum_{j=t+1}^{i_{max}} h(j)} \right)$$

The optimal threshold T_{opt} is defined as the maximum of the sum of these entropies:

$$T_{opt} = Max(H_B(t) + H_W(t)) \quad , \quad t \, \epsilon \, [0, i_{max}]$$

In order to determine the optimal threshold, the basic form of this algorithm calculates the entropies of black and white pixels for each possible threshold, e.g. 256 different values for t. This might seem extensive and time-consuming. Yet, compared to other operations to be performed while analyzing the image, this is put into perspective. Furthermore, the results this algorithms yields when applied to images of single nuclei beat that of other algorithms considering stability and effectiveness. Additionally, the possible values of t can be reduced by an analysis of the histogram prior to calculating the analysis.

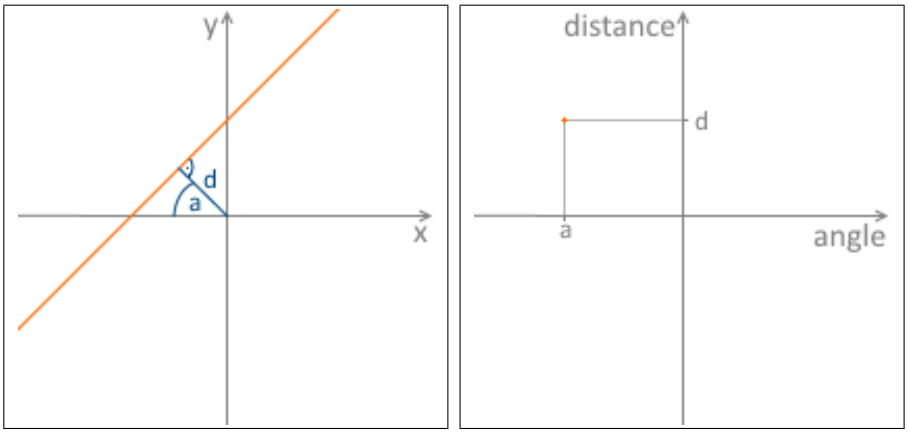
4.3 Particle Analysis

The ImageJ documentation on this topic is rather superficial. Nonetheless, the basics of particle detection is some kind of Hough transformation as described in (Dud71). Since this is a rather elaborated technique with various existing enhancements and modifications, this section will only describe the basic technique by the example of detecting straight lines.

Nonetheless, ImageJ's documentation does state that ParticleAnalyzer works with the following algorithm:

```
for each line do
    for each pixel in this line do
        if pixel value "inside" threshold range then
            trace the edge to mark the object
            do measurements
            fill the object with a color outside threshold range
        else
            continue scan
        end if
    end for
end for
```

Where *"trace the edge to mark the object"* may refer to various tracing algorithms, for example a Hough transformation. The basic idea behind Hough transformations is to find a space in which the same information can be represented in a more useful way. For example, if a simple straight line has to be detected, the coordinates of the Hough space are not x and y coordinates, but instead the angle between the line and the x-Axis and its distance to the center of coordinates. By doing so, a line in carthesian space can be represented by a single point in Hough space, as seen in figure 15.



(a) Straight line in carthesian coordinates

(b) Single point in Hough space representing the same line

Figure 15: The same line represented in carthesian coordinates and in Hough space

While this is a neat way of representing a straight line with known angle and distance to the x-axis, finding a straight line within the data of an image requires some more effort. The basic idea behind edge tracing with hough transformations is to look at each pixel in the image[6] and, given the pixel has the desired foreground value, e.g. white, determines the distance d between the coordinate center a straight line for every possible angle α that line's normal can enclose with the x-axis by calculating $d(\alpha) = x \cdot \cos(\alpha) + y \cdot \sin(\alpha)$ for $\alpha \, \epsilon \, [0, \pi]$. The graphs of $d(\alpha)$ for each point are then accumulated. This can be achieved by the following algorithm:

[6]Note that the image was thresholded beforehand thus is binary, e.g. only includes black and white pixel, by the time the transformation is applied

$maxDistance \leftarrow \sqrt{(\frac{1}{2} \cdot imageHeight)^2 + (\frac{1}{2} \cdot imageWidth)^2}$
$minDistace \leftarrow -maxDistance$
$houghSpace[0..\pi][minDistance..maxDistance] \leftarrow 0$
for each line **do**
  for each pixel in that line **do**
    if $pixelValue = 0$ **then** skip that pixel
    end if
    for $\alpha \leftarrow 0$ to π **do**
      $distance \leftarrow x_{pixel} \cdot \cos(\alpha) + y_{pixel} \cdot \sin(\alpha)$
      $houghSpace[\alpha][distance] + +$
    end for
  end for
end for

To determine the detected line's distance and angle, that point in Hough space which has the overall maximum value has to be found, which can be simply done by iterating over the values and comparing them. From that values angle α and distance d the straight line can be reconstructed in the original carthesian coordinate system. This is illustrated in figure 16[7].

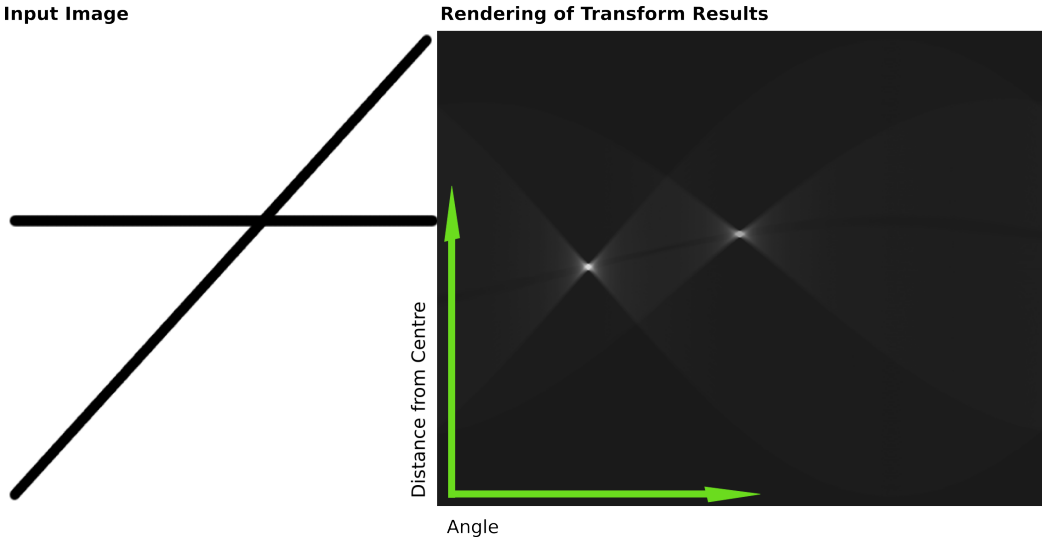


Figure 16: Example of detecting lines using Hough transformation

In order to detect circles or arbitrary shapes, other parameters have to be found. In the case of (overlapping) circles that are to be detected, Hough

[7]Image taken from http://upload.wikimedia.org/wikipedia/commons/1/1c/Hough-example-result-en.png

space would be three dimensional, consisting of the center coordinates and radii of the circles. Besides that and corresponding changes, the procedure stays the same.

5 Conclusion and Prospect

5.1 Evaluation

Evaluation happened on basis of 15 different sets of images taken in May 2014 which were analyzed using the following settings:

```
nucleus.average = 10000
nucleus.min.factor = 0.4
nucleus.max.factor = 2.6
nucleus.min.circularity = 0.6

nucleolus.average = 100
nucleolus.min.factor = 0.5
nucleolus.max.factor = 4
nucleolus.min.circularity = 0.8

indepth.range = 30

# settings for improved mode
nucleus.background.blur = 100
nucleus.thresholding.blur = 3
nucleus.boundary.width = 5

nucleolus.background.blur = 10
nucleolus.thresholding.blur = 1
```

Listing 4: Settings used during evaluation

This lead to the results shown in 2. With the success rate calculated as $\frac{(\#found - \#wrong)}{(\#found - \#wrong + \#missing)}$. Note that all results have been cross-checked and validated by manually determining targets in the different images. Hence, the application is compared to what a human with good eyesight is able to recognize. Also note that the evaluation was run on a computer with an AMD Phenom II X6 1100T processor @ 3.30GHz and 16GB of RAM. All images had resolutions of 1388 pixel x 1040 pixel. Execution times may vary on other systems.

5.2 Conclusion

The application developed during this assignment presents itself as a reliable and fast tool to detect nucleoli as targets. Nonetheless, there is still plenty of potential for improvements concerning stability of the results and execution speed, which will make the application even more reliable and useful.

During the first test in operating conditions, serious shortcomings were revealed when flawed colorization lead to extremely bright and thick nucleus

Specimen	Targets	wrong	missing	success rate	time [ms]
1	15	0	2	88.24%	3149
2	15	0	3	83.33%	3193
3	12	1	2	84.62%	2763
4	9	0	2	81.82%	2956
5	10	2	2	80.00%	2949
6	10	0	3	76.92%	3246
7	10	0	1	90.91%	2831
8	11	0	8	57.89%	2484
9	8	0	1	88.89%	3074
10	17	0	3	85.00%	3072
11	10	0	3	76.92%	2662
12	8	0	5	61.54%	2835
13	6	0	2	75.00%	2220
14	14	1	3	81.25%	2551
15	10	1	2	81.82%	2746
Mean:	11.00	0.33	2.80	79.61%	2848.73

Table 2: Evaluation results

boundaries. Though situations like this can be overcome by increasing the amount of pixels by which regions of interest are shrunk before the nuclei are analyzed, the application should be able to handle them itself.

Furthermore, while the applied measures of brightness correction are a great way to deal with uneven exposure to light, the application still has serious issues with images that show distinct fuzziness. In case the analyzed channel 3 image if too fuzzy, this might even result in too few or even false targets to be detect even though channel 1 shows a perfectly clear image. This also shows the application's dependence of a good image of channel 3, as the whole process of target detection strongly depends on reliable determination of regions of interest in this image. Later versions should be able to double check the results gained from analysis of channel 3 images. Probably later versions a capable of using information from channel 2 images to detect the regions of interest as well.

A major hindrance while developing the application was posed by ImageJ itself. As the most common way of using ImageJ seems to be as a stand-alone application, most expansions and use-cases are developed as plug-ins. This leads to the majority of documentation being created for this way of development. Using ImageJ as a library in another application seems to be performed very scarcely. Hence, most parts of ImageJ's API are only docu-

mented rudimentarily. Thus, lots of trial and error development and reading (partially cryptic) source code and commentaries was necessary, which drastically slowed down development speed.

Despite the mentioned shortcomings, based on the exemplary data provided in 5.1 and the experiences made during tests, the application fulfills the requirements specified in 2 to a satisfying extent and in some aspects even exceeds them. Thus, the project can be considered as a success.

In addition, though it might not be a perfectly user-optimized library, ImageJ still is a suitable choice to work as a basis for image analysis. Considering the fact that ImageJ2 will be released in 2015, which among other objectives, aims at providing better documentations and developer resources, ImageJ might become an even better choice in the future.

5.3 Prospect

In the near future the main task is certainly to derive a method to provide the most suitable configuration in order to pay respect to deviations in composition and quality of specimen and microscope-photography. To achieve this, already existing images as well as those taken during coming examinations have to be analyzed thoroughly.

Considering future development there are several possible extensions and enhancements. The most prominent way of extending the application certainly is to implement further image analysis algorithms and target detection procedures, e.g. Deriche or Canny edge detections. Another very obvious advancement is to enhance and expand the possibilities of the statistics module, as this part is kept pretty simple up to now.

As speed is a major requirement to the developed application, thinking about even further reducing the execution time by using multithreading and parallelization techniques are certainly worth thinking about.

References

[Dud71] Duda, Peter E. Richard O.; Hart H. Richard O.; Hart: *Use of the Hough Transformation to Detect Lines and Curves in Pictures*. 1971

[Gre10] Greensted, Andrew: *Otsu Thresholding*. `http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html`. Version: 2010. – The Lab Book Pages

[Hua93] Huang, Mao-Jiun J. Gliang-Kai; Wang W. Gliang-Kai; Wang: *Image Thresholding by Minimizing the Measures of Fuzziness*. 1993

[Kap84] Kapur, P. K. Wong A. K. C. J. N.; Sahoo S. J. N.; Sahoo: *A New Method fpr Gray-Level Picture Thresholding Using the Entropy of the Histogram*. 1984

[Lan13] Landini, Gabriel: *Auto Threshold*. `http://fiji.sc/Auto_Threshold`. Version: 2013. – Fiji Manual

[Mor00] Morse, Bryan S.: *Lecture 4: Thresholding*. 2000. – Brigham Young University

[Rid78] Ridler, S. T. W.; C. T. W.; Calvard: *Picture Thresholding Using an Iterative Selection Method*. 1978. – EEE Transactions on Systems, Man, and Cybernetics, Vol. sMC-8, No. 8

[Sac04] Sacha, Jerek: *Maximum Entropy Threshold*. `http://fiji.sc/Maximum_Entropy_Threshold`. Version: 2004. – Fiji Manual