Automatic adaptation of filter sequences for cell counting

Uroš Čibej†, Jasna Lojk*, Mojca Pavlin*, and Luka Šajn†
†Faculty of computer and information science, University of Ljubljana
*Faculty of electrical engineering, University of Ljubljana
Ljubljana, Slovenia

***Abstract*—Manual cell counting in microscopic images is usually tedious, time consuming and prone to human error. Several programs for automatic cell counting have been developed so far, but most of them demand some specific knowledge of image analysis and/or manual fine tuning of various parameters. Even if a set of filters is found and fine tuned to the specific application, small changes to the image attributes might make the automatic counter very unreliable. The goal of this article is to present a new application that overcomes this problem by learning the set of parameters for each application, thus making it more robust to changes in the input images. The users must provide only a small representative subset of images and their manual count, and the program offers a set of automatic counters learned from the given input. The user can check the counters and choose the most suitable one. The resulting application (which we call Learn123) is specifically tailored to the practitioners, i.e. even though the typical workflow is more complex, the application is easy to use for non-technical experts.**

I. Introduction

Cell counting on microscopic images of various types is one of the fundamental tasks for many researchers and practitioners in all life-sciences. Being such a fundamental task, many automated solutions exist. Despite this abundance of tools for automated counting, the majority of counting tasks are still done manually by the practitioners. The main reasons for this can be boiled down to two problems. The first problem is the inaccuracy of the automated counters and the second one the difficulty to use for non-technical experts. Our work is focused on overcoming both of these problems, i.e. to develop an accurate tool, and a tool that is simple to use out-of-the-box.

Currently several programs for cell counting are already available. From many commercial solutions, such as MetaMorph, BioQuant, Image-Pro and SymenTec to free software, such as ITCN ImageJ Plugin [5], CellProfiler [4], UTHSCSA ImageTool [1], and CellC [10]. However, all of the existing solutions require some additional input or even some image preprocessing of the raw images, which is not only time-consuming but can also be too complex to some biology field experts. [3].

We already developed a cell counting tool called CellCounter [7], with which we demonstrated that for a particular application (i.e. cell viability estimation), a fixed set of filters with specifically tailored parameters can give great results, making the human counting almost obsolete and thus speeding up the analysis of the experiment results in life-sciences by an order of magnitude. This result is very important in practice, however it is still not the silver bullet for all the problems one might encounter in every such application.

The program was developed in collaboration with a research group, who used cell counting as a way to determine cell viability [2] and transfection efficiency [8] for certain experimental settings. By fluorescently staining cell nuclei they obtained fluorescent images with higher contrast compared to BF or PC images, but it still left them with large number of microscopic images to count, requiring a more efficient way to process the images than manual counting.

To get an intuitive picture of all the problematic features of the counting of cells, let us mention some of the problems, that typically arise in practice: different types of assays resulting in a large variety of images which are hard to generalize, inability to detect individual nuclei in multinucleated cells, overlapping cells, uneven illumination and other equipment related factors, such as electronic and/or optical noise [6], that lead to images having variable contrast and quality, low contrast of the image, different cell shapes, internal cell structures, extracellular debris, and varying density of cell culture.

Due to all these possibilities, there is a large set of variations that prevent a general solution for any circumstance to be developed. The optimal tool would detect all these variations without any user intervention. However, the many failed attempts to find such a tool demonstrate that a different approach is needed. The main goal of this article is to present such an application, which is able to adapt to the specific use-case, with (as little as possible) guidance from the user. The user only provides a set of examples, from which the application learns the general counting as accurately as possible.

Since one of the major objectives is also a widespread usability of this application, the following goals are sought:

- the learning should be done as intuitively as possible,
- the result of the learning should be easily verifiable, to ensure that the obtained results are sound,
- the workflow must not be too complex, avoiding any technology related terms,
- the learning has to be a repeatable and incremental process,
- the results of the learning and counting must be usable also in other de-facto standard applications of the user-base.

Automatic learning is a widespread area of research, so there are plenty of possible approaches to developing such an application. We define the learning as an optimization problem,

where the goal is to optimize the difference between the counting results defined by the user, and the counting obtained by the automatic counter. As mentioned above, the CellCounter application used a predefined sequence of filters, where the parameters were fine-tuned for a particular application. We used this sequence as a general framework, and the parameters of the image filters represent the vector which needs to be optimized.

As the optimization method we chose an evolutionary approach, because of the many positive features. Namely, the evolutionary approach is very flexible, we can easily change the goal function, stop the process at any point and get feasible solutions, start from existing solutions and improve the solution by interacting with the user.

The remainder of the article is structured as follows. Section II. presents the learning application, section III. describes all the implementation details, the framework, and the technologies used, section IV. describes the experimental evaluation of the application, and finally, section V. concludes the paper and gives directions for our future work.

II. Description of the application

In this section we will first describe the application from a purely user perspective. Since one of our basic goals is to make the application as simple as possible for the end-user, we managed to strip down all the unnecessary concepts. The user does not have to be aware of any underlying algorithm and technology and in the workflow it can only focus on the images and the task at hand. This is an extremely important aspect for the practical usability of such application. Many similar applications overflow the user with technical terms and make them specify various parameters of algorithms, of which the end user should not be aware of.

The application is subdivided into two sub-application. The first one being the learning part (Learn123) and the second the counting part (Count123) of the application. In what follows we describe in more detail the tasks that constitute a typical workflow, starting with the learning part of the application.

A. Choosing a subset of images

The first task of the user is to select a representative subset of images. The subset needs to be relatively small (typically around 10 images) and the images spawn as many different aspects of the entire set (i.e. different sizes of object, different lighting conditions, different shapes, etc.).

B. Manual count

The second step for the user is to manually count the objects in the images. The information that it provides are only the approximate centers of the object. This is another very useful simplification, since some counting tools require the user to provide the contours, or at least the center and the radius of the object. Providing only the centers speeds up this step significantly.

C. The learning process

Once all the objects in the images have been manually annotated, the learning process can be initiated. The length of the learning process depends mostly on the size of the image subset, the typical span of the learning is from 5 minutes to about an hour. But the user can also participate in this ongoing process, since a set of candidate counters is immediately offered. This set is the set of currently best known counters and the user can assess the given counters. If the user is satisfied with the results he/she can interrupt the learning and use the counter.

D. Assessing the quality of the counters

When the program finishes its learning process, the set of 10 best counters is presented to the user. The user can see the count of the automatic counter visually and compare it to the counts made manually. If none of the counters is satisfactory, the user can add new images to the testset, remove some old images, and restart the learning process. Fig. 1 shows an example of a counter assessment. Notice that the learning does not start from scratch, it uses most of the information learned from the previous learn run.

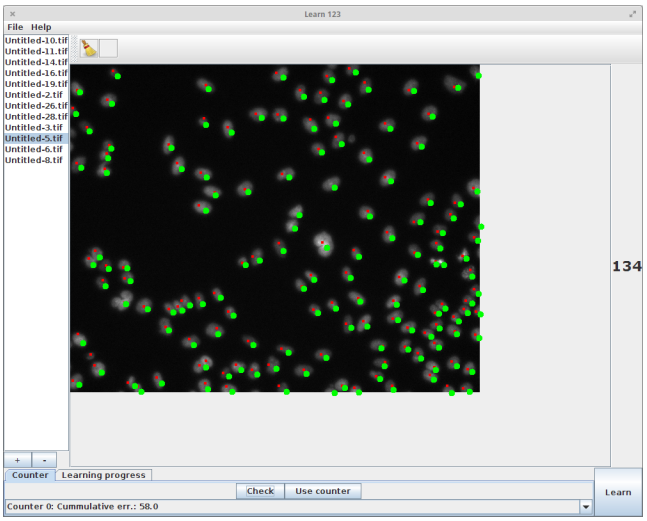

Fig. 1. The comparison of the manual and automatic count in a test image. The small red dots represent the manual count, the larger green dots represent the automatic counter.

E. Counting application

When a suitable counter is found by the above process, it can be used in the second sub-application, the Count123. The user can choose all the images in the set and the application will visually present the results. All the results can thus be checked and any point can be added or removed to correct any errors.

The assessment can also find some images on which the automatic counter behaves badly. These images (or one typical representative) can be added to the testset and the new learning process will adapt also to this type of images.

If the obtained results are satisfactory, they can be easily exported and used in other applications for further data analysis.

Fig.2 show an overview of the workflow. Every part of the workflow is technically neutral, agnostic about any algorithms used behind the scenes.

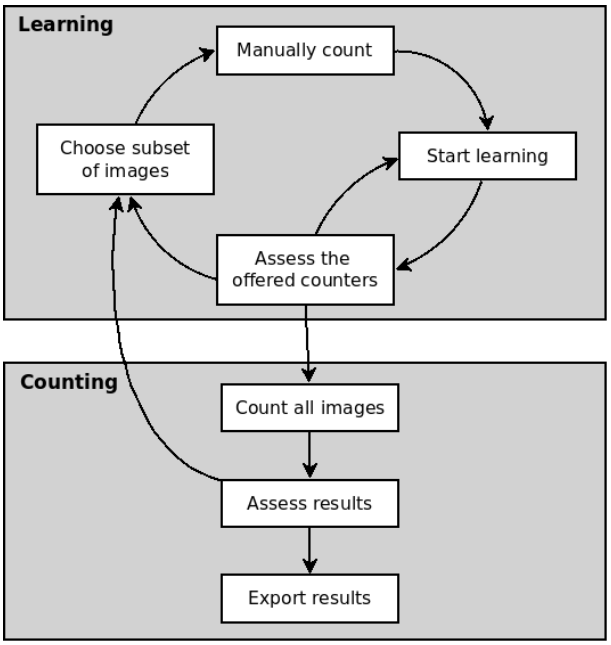

Fig. 2. The user workflow in the Learn123 application.

III. Implementation details

The implementation details can be boiled down to three aspects: (1) the modeling of the learning as an optimization problem, (2) the algorithm used for the optimization, and (3) the technologies used for the implementation.

A. The optimization problem

Intuitively, the objective of the learning process is to find a counter (specified by a (numeric) vector) with minimal counting error on the learning set. There are many possibilities how to encompass this problem into a mathematical model, the presented optimization model empirically demonstrated very good results, showing its good applicability to real problems.

As the input, a set of k images is given:

$$Img = \{img_1, img_2, \ldots, img_k\}.$$

For every image img, the manual count is given by a sequence of points in the plane, indicating the (approximate) centers of the identified objects:

$$C_{man}(img) = \{p_1, p_2, \ldots, p_n\}$$

The subject of the optimization is a vector of parameters

$$P = \{param_1, param_2, \ldots, param_l\},$$

which uniquely define the automatic counter. These parameters can be integer values, boolean values, real numbers, or certain objects from specific sets, but without any loss in generality we can assume only integer valued vectors.

The result of applying the counter defined by the vector P to the image img will be denoted by the set of points:

$$C_P(img) = \{p_1, p_2, \ldots, p_m\}.$$

Now for the most important part of the model, we have to define the goal function (criteria), which will model what we consider to be an accurate counter. Intuitively, there are two criteria that a good counter should fulfill:

- the difference of the obtained counts (the numbers) should be as small as possible, and
- the positions of the found objects should be as close as possible.

These two intuitive criteria have to be modeled appropriately and combined into a suitable goal function. We will first define this function for one image img. The first criterion is simple to model as a simple absolute difference between the number of points found with the manual counts and number of points with automatic counts on the same image:

$$\Delta(C_{man}(img), C_P(img)) = ||C_{man}(img)| - |C_P(img)||.$$

The second criterion compares all the points found with the manual count and see how far are they from the points found with the automatic counter.

The goal function for the second criterion was chosen as

$$D(C_{man}(img), C_P(img)) = \sum_{p \in C_{man}(img)} d(p, closest(p, C_P(img)))$$

,

where d is the Euclidean distance between the two points, and $closest$ is a function that finds the closest point from a given point to the given set.

This goal function is simple and fast to compute, but it can run into problems on some corner cases where it prefers degenerate counters. An example of the problem with this function is given in Fig. 3. This problem will be amended by combining the two goal functions. How these two goal functions are combined is described together with the algorithm, since it is more specific to the way the algorithm works.

The functions Δ and D model the differences in counts for a single image. Now we need to choose a suitable function that combines the results of these two function on all images. We experimented with a few variations, such as the maximum value, the median value, and the average value over all images. Empirically the simple average gives the best results, so the goal functions which combine Δ on all images is

$$G_\Delta(P, Img) = \sum_{im \in Img} \Delta(C_{man}(img), C_P(img)),$$

and an equivalent definition is used for G_D.

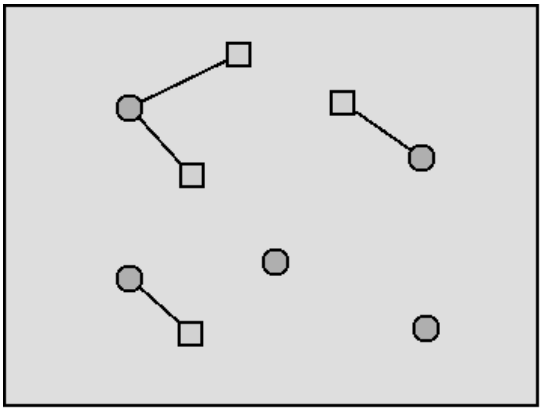

Fig. 3. An example of the D criterion matching between manual counts and automatic counts. The circles represent automatic counts, the square represent manual counts. This example also demonstrates the shortsightedness of this goal function, which has to be amended with the function Δ.

1) Evolutionary algorithm: An evolutionary algorithm is used for the minimization of the above defined optimization problem. The main reason for this choice is the general applicability of this algorithm, robustness, and the fact that at any point there are feasible solutions available, which can be immediately presented to the user for assessment. The framework of each counter is a sequence of filters for which we demonstrated in [7] to be a very adequate sequence for counting cells in electronic microscope images. This sequence of filters was manually fine-tuned for the specific application. Namely, each filter has a set of parameters, and the final count is very sensitive to the specific values of these parameters. The sequence of filters (i.e. image-processing algorithms) used for counting is as follows:

1) Enhance contrast - the images have typically a very low intensity and a very low contrast. By enhancing the contrast, many more features of the image are visible, and the results of the counting improve dramatically.
2) Threshold - in the second step, the image is converted to binary format. For this a threshold is set, by using either a specific value of the threshold or one of the 13 possible algorithms.
3) Watershed - by thresholding we typically loose the boundary between different cells, and by watershedding [9] this merged objects are divided again.
4) Holefill - after the application of the above filters, the objects are not uniformly filled. With this filter, the holes in the objects are filled, making it more suitable for the next detection step.
5) Particle analysis - the last step of the algorithm detects the remaining blobs of active pixels in the image and decides whether to add an object to the count or not. There are a few parameters upon which this decision is made - the most important being the minimum radius, the maximum radius, and the eccentricity of the object. With this parameters the counter can adapt to count only specifically shaped objects or objects with specific sizes.

The parameters for these filters represent the chromosome, as it is shown in Fig. 4, and a typical evolutionary algorithm

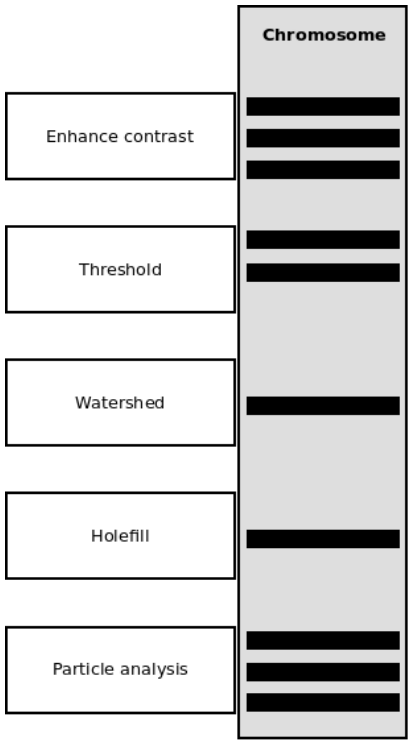

Fig. 4. An overview of the basic chromosome.

is used to evolve from completely random counters to very successful ones. The pseudocode for the algorithm is shown in Algorithm 1

Algorithm 1 The outline of the learning algorithm

```
1:  population = randomPopulation(populationSize);
2:  for gen ∈ [1, 2, ..., numGen] do
3:      if gen ≤ numGen/10 or gen%2 = 0 then
4:          goal = G_Δ
5:      else
6:          goal = G_D
7:      end if
8:      rankedPop = sortByFitness(population, goal)
9:      pop1 = select(rankedPop,populationSize)
10:     pop2 = select(rankedPop, populationSize)
11:     population = crossover(pop1, pop2)
12:     population = mutate(population)
13:     population = take(population, populationSize)
14: end for
```

Lines 3-6 of the pseudocode demonstrate the aforementioned combination of functions Δ and D. The first 10% of the generations, the fitness function is only Δ. After this starting generations, the D function is used in every other generation, eliminating possibly corrupt counters that accidentally count a similar number of cells, but on completely different positions that those obtained by the manual count.

The selection (lines 8-10) of the counters which are used for creating a new generation is done with rank selection, i.e. the probability that a counter is chosen is proportional to its rank in the current population.

The crossover (line 11) used is a one-point crossover, i.e. a random position in the chromosome is chosen and the two chromosomes exchange the parameters to the left of that position. The mutation (line 12) is not a bit-level mutation (as it is usually the case with genetic algorithms), but we used a mutation on the parameter level, where a uniformly random value is chosen from the parameter domain.

Ultimately, to obtain the next generation, only the best counters are chosen in line 13.

B. The technologies used

The program was developed in the programming language Scala. Scala compiles to the JVM, making the Learn123 application completely platform independent. Furthermore, a wide range of de-facto standard Java tools can be used as a part of the application. We integrated the entire ImageJ application into Learn123; for the users to help them further manipulate the images as well as for the algorithm itself, since most of the filters in the counting algorithm are directly used in ImageJ. Since a lot of researchers use this tool for their everyday work, they can use Learn123 just to find the appropriate counter and then use it entirely in ImageJ.

IV. Experiments

The algorithm was developed and optimized based on microscopic images obtained from several experiments the collaborating research group provided, but one experiment was selected for the evaluation of the program efficiency.

The experiments were done on Chinese hamster ovary cells (CHO) grown in Ham's tissue culture medium for mammalian cells (HAM) supplemented with 10% fetal bovine serum (FBS) at 37 °C in 5% CO_2-enriched air at saturation humidity. All experiments were performed on 24h old cell cultures in exponential growth phase.

The cell viability experiment was performed as described previously [2]. Briefly, cells were incubated with increasing concentration of polycationic polymer coated magnetic nanoparticles for 24 h and stained with two fluorescent dyes; Hoechst 33342, which stained cell nuclei, and propidium iodide (PI), which differentially stained dead cells. At least 15 visual fields at 200 × magnification were taken of each sample for each used fluorescent dye using a fluorescent microscope (Zeiss 200, Axiovert, Germany). The images were recorded by MetaMorph imaging system software (Visitron, Germany) and saved in lossless TIFF format.

For the Hoechst dye 173 images were obtained and for the PI dye 159 images were obtained. All these images were manually counted. Then (for each dye separately) a small training set was chosen (15 images for PI and 13 images for Hoechst). The manual counting was done again using Learn123 and the learning process offered a set of the best counters (according to the function G_Δ). The best offered counters were used to perform the automatic count on the entire set of images. Figures 5 and 6 show a scatterplot of the comparison between automatic count (x-axis) and manual count (y-axis).

The results show a very good correlation between the manual and automatic counts. Only a couple of images in the entire set exhibit a somewhat larger error. But even these errors can be easily detected by quickly skimming through the results in Learn123, and manually add or remove the few erroneous points.

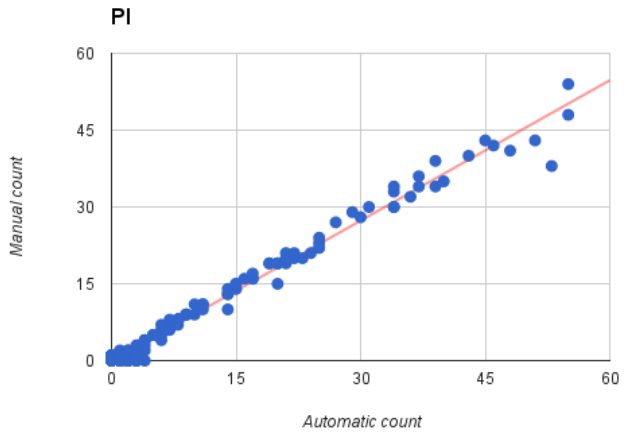

Fig. 5. Scatterplot showing a very good correlation between manual counting and automatic counting.

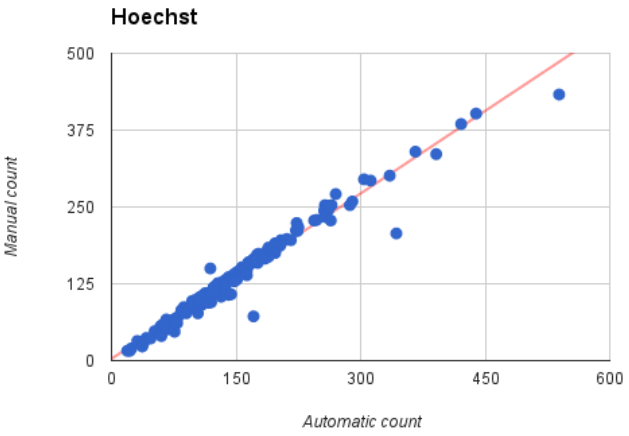

Fig. 6. Scatterplot showing a very good correlation between manual counting and automatic counting.

V. Conclusion

We presented a novel application for cell counting. The application enables regular, non-technical users to find better suited automatic counters, tailored specifically for their set of images. This is one step towards overcoming the problem of huge variations of the problems set, which can only be solved by a great adaptability of the application. Learn123 offers a simple workflow in which the user can teach the program how to count, by only giving example counts on a small subset of all the images.

The presented method was used on a fixed sequence of filters, however it can easily be extended to any sequence. Our future goal is to extend this application for any sequence

of filters that can be defined as an ImageJ macro. With this extension Learn123 will become a truly flexible tool, usable not only for cell counting, but for many other counting applications.

ACKNOWLEDGEMENT

This work was supported by Slovenian Research Agency within projects J4-4324,J2-6758,J3-6794 and young researchers program.

REFERENCES

[1] Uthscsa imagetool, home page. web, 2014. http://compdent.uthscsa.edu/dig/itdesc.html.

[2] Vladimir Boštjan Bregar, Jasna Lojk, Vid Šuštar, Peter Veranič, and Mojca Pavlin. Visualization of internalization of functionalized cobalt ferrite nanoparticles and their intracellular fate. *International journal of nanomedicine*, 8:919–931, 2013.

[3] Jiyun Byun, Mark R Verardo, Baris Sumengen, Geoffrey P Lewis, BS Manjunath, and Steven K Fisher. Automated tool for the detection of cell nuclei in digital microscopic images: application to retinal images. *Mol Vis*, 12:949–960, 2006.

[4] Anne Carpenter, Thouis Jones, Michael Lamprecht, Colin Clarke, In Kang, Ola Friman, David Guertin, Joo Chang, Robert Lindquist, Jason Moffat, Polina Golland, and David Sabatini. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome Biology*, 7(10):R100, 2006.

[5] Sean R Gallagher. Digital image processing and analysis with imagej. *Current Protocols Essential Laboratory Techniques*, pages A–3C, 2010.

[6] Kang Li, Eric D Miller, Lee E Weiss, Phil G Campbell, and Takeo Kanade. Online tracking of migrating and proliferating cells imaged with phase-contrast microscopy. In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on*, pages 65–65. IEEE, 2006.

[7] J. Lojk, L. Šajn, U. Čibej, and M. Pavlin. Automatic cell counter for cell viability estimation. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*, pages 239–244, May 2014.

[8] Igor Marjanovič, Maša Kandušer, Damijan Miklavčič, Mateja Manček Keber, and Mojca Pavlin. Comparison of flow cytometry, fluorescence microscopy and spectrofluorometry for analysis of gene electrotransfer efficiency. *Submitted for publication*, 2014.

[9] Roerdink and Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *FUNDINF: Fundamenta Informatica*, 41, 2000.

[10] Jyrki Selinummi, J Seppala, Olli Yli-Harja, and Jaakko A Puhakka. Software for quantification of labeled bacteria from digital microscope images by automated image analysis. *Biotechniques*, 39(6):859, 2005.