

# **fastER: a user-friendly tool for ultrafast and robust cell segmentation in large-scale microscopy**

Oliver Hilsenbeck<sup>1</sup>, Michael Schwarzfischer<sup>2</sup>, Dirk Loeffler<sup>1</sup>, Sotiris Dimopoulos<sup>1</sup>, Simon Hastreiter<sup>1</sup>, Carsten Marr<sup>2</sup>, Fabian J. Theis<sup>2,3</sup>, and Timm Schroeder<sup>1</sup>

<sup>1</sup>Department of Biosystems Science and Engineering, ETH Zurich, Basel, Switzerland

<sup>2</sup>Institute of Computational Biology, Helmholtz Zentrum München, Neuherberg, Germany

<sup>3</sup>Department of Mathematics, Technische Universität München, Garching, Germany

# Supplementary Notes

## 1 Additional algorithmic details

### 1.1 Splitting of extremal regions

fastER considers each extremal region  $x \subseteq D$  for splitting in two sub-regions  $x_1 \subset x$  and  $x_2 \subset x$  such that  $x_1 \cup x_2 = x$  and  $x_1 \cap x_2 = \emptyset$  if  $x$  has exactly two child-nodes in the component tree (with corresponding extremal regions  $c_1 \subset x$  and  $c_2 \subset x$ ) and if both  $c_1$  and  $c_2$  were not split themselves. The pair of sub-regions is then together also considered as candidate regions for segmentation (compare equation 2 in main text). Each sub-region contains all pixels of the corresponding extremal region (i.e.  $c_1 \subseteq x_1$  and  $c_2 \subseteq x_2$ ), and each remaining pixel  $p \in x$  (i.e. each pixel  $p \in x$  with  $p \notin c_1$  and  $p \notin c_2$ ) is added to the sub-region  $x_i$  (with  $i = 1$  or  $i = 2$ ) whose corresponding extremal region  $c_i$  is closer to  $p$ . Since fastER does not maintain a list of contained pixels for each extremal region, the closest sub-region  $x_i$  is chosen based on the distance to the bounding box (denoted by  $b_i$ ) of the corresponding extremal region  $c_i$  using Manhattan distance (note that  $b_1$  and  $b_2$  are bounding boxes of the extremal regions  $c_1$  and  $c_2$ , but not of  $x_1$  and  $x_2$  – they are only used to distribute remaining pixels of  $x$  among  $x_1$  and  $x_2$ , but they remain unchanged when new pixels are added to  $x_1$  and  $x_2$ ). To enable this, fastER considers  $x$  for splitting only if the bounding boxes  $b_1$  of  $c_1$  and  $b_2$  of  $c_2$  do not overlap. Additionally, fastER has to guarantee that the sub-regions  $x_1$  and  $x_2$  remain connected after distributing the remaining pixels of  $x$  among them. To ensure this without computationally expensive calculations, fastER requires that for each remaining pixel  $p \in x$  there must be at least one pixel  $q$  adjacent to  $p$  (i.e.  $pAq$ ) that is closer to a randomly chosen pixel of the corresponding nested extremal region  $c_i$  (unless  $p$  itself is adjacent to the random pixel from  $c_i$ ) and has a pixel intensity that is lower or equal to the maximal pixel intensity in  $x$ . If this constraint is violated,  $x$  is not considered for splitting anymore.

If extremal region  $x$  meets these constraints, it is split into  $x_1$  and  $x_2$ . If the parent extremal region  $par(x)$  of  $x$  has no other child nodes except  $x$  in the component tree,  $par(x)$  is also split by analogously distributing its remaining pixels (i.e. each pixel  $p \in par(x)$  with  $p \notin x$ ) among  $x_1$  and  $x_2$  based on their distances to  $b_1$  and  $b_2$  (and so on for the parent extremal region of  $par(x)$ ). Like an extremal region can be specified with a single seed pixel (see main text), an extremal region  $x$  with sub-regions  $x_1$  and  $x_2$  can be specified with a seed pixel of  $x$  and the bounding boxes  $b_1$  and  $b_2$  that were used to distribute the remaining pixels of  $x$  among  $x_1$  and  $x_2$ .

### 1.2 Features

In this section, we formally define the features extracted from candidate regions (like in equations 6 to 14 in the main text, but without using the sufficient statistics for their definition). Let  $x \subseteq D$  denote a region,  $cov(x)$  the sample covariance matrix of the x- and y-coordinates of the pixels in  $x$ , and  $G(x)$  all pairs of pixels in  $x$  and  $D \setminus x$  that are adjacent:

$$G(x) = \{(p, q) \mid p \in x \wedge q \in D \setminus x \wedge pAq\} \quad (1)$$

With this, the features of  $x$  are defined as follows:

$$\begin{aligned}
\text{Size}(x) &= |x| \\
\text{Major axis length}(x) &= e_1(\text{cov}(x)) \\
\text{Minor axis length}(x) &= e_2(\text{cov}(x)) \\
\text{Avg. intensity}(x) &= \frac{1}{|x|} \sum_{p \in x} I(p) \\
\text{Heterogeneity}(x) &= s(\{I(p) | p \in x\}) \\
\text{Avg. gradient}(x) &= \frac{1}{|G(x)|} \sum_{(p,q) \in G(x)} |I(p) - I(q)| \\
\text{Std. dev. of gradient}(x) &= s(\{|I(p) - I(q)| | (p,q) \in G(x)\}) \\
\text{Eccentricity}(x) &= \frac{|G(x)|}{8\sqrt{|x|/\pi}} \\
\text{Avg. background}(x) &= \frac{1}{|G(x)|} \sum_{(p,q) \in G(x)} I(q), \tag{2}
\end{aligned}$$

where  $e_1$  and  $e_2$  denote the major and minor Eigenvalue, respectively, and  $s$  the sample standard deviation.

### 1.3 Inference method

In this section, we further describe the inference method used by fastER, which we defined with the recursive formula  $F(x)$  in the main text (equations 2 to 4, main text). As stated in the main text,  $F(x)$  returns a set of non-overlapping candidate regions and maximizes its average score under the additional constraint that for any extremal region  $y$  nested in  $x$  (i.e.  $y \subseteq x$ ) with  $S(y) > 0$ , it holds that either  $y$  itself, a parent extremal region containing  $y$  or at least one child region contained in  $y$  is included in the solution. Without this constraint, only extremal regions with maximal score would be returned (i.e. only one region in most cases), because adding any region with non-maximal score to the solution would reduce the average score of the solution. To segment an image  $I$ , the segmentation formula  $F$  is applied on the whole image, i.e. on the whole image domain  $D \subset \mathbb{Z}^2$ , which is an extremal region by definition (equation 1, main text).

There are some specific cases, where the solution that is extracted by fastER does not have globally maximal average score under the additional constraint – and where this would actually lead to undesirable results. This can best be shown with a concrete example: assume fastER returns as solution for the segmentation of an image the set of extremal regions  $X = \{a, b, c\}$  with the scores  $S(a) = 40, S(b)=10$  and  $S(c)=10$ . The average score of  $X$  is thus  $\bar{S}(X) = \frac{1}{3}(40 + 10 + 10) = 20$ . Furthermore, assume that the extremal regions  $b$  and  $c$  have a parent extremal region  $d$ , i.e.  $b \subset d \wedge c \subset d$ , with  $S(d) = 5$ . Thus,  $b$  and  $c$  in  $X$  could be replaced with  $d$  without violating the additional constraint mentioned above, which would result in the alternative solution  $X' = \{a, d\}$ . This solution is obviously undesirable, because it chooses region  $d$  with score 5 instead of the regions  $\{b, c\}$  which both have the much higher score 10. However, the average score of  $X'$  is  $\bar{S}(X') = \frac{1}{2}(40 + 5) = 22.5$  which is higher than the average score  $\bar{S}(X) = 20$  of  $X$ .

Thus, when extracting a solution with globally maximal average score under the constraint mentioned above, the undesirable solution  $X'$  would be correct in this case. However, fastER still

returns  $X$  as solution in this case, because it does not alter solutions for sub-problems later on. In the example, the segmentation of extremal region  $d$  is a sub-problem for which the solution is  $F(d) = \{b, c\}$  and not  $F(d) = \{d\}$ .

## 1.4 Efficient calculation

### 1.4.1 Overview

fastER employs the algorithmic framework from Nistér and Stewénus (2008) to efficiently enumerate all extremal regions (including sub-regions generated by splitting) from an image, calculate their features, and apply the recursive segmentation formula. As described in Nistér and Stewénus (2008), their algorithm can best be explained with an analogy: interpret the pixel intensities of a given grayscale image as heights and imagine the resulting image landscape is flooded with water starting at a random point. Water flows downward until a local minimum is reached (which corresponds to a leaf node in the component tree). There, water accumulates and then flows upward again until the basin corresponding to the local minimum is filled and a local maximum is reached (which corresponds to a node in the component tree with multiple child nodes). From there, water flows downward again to the next local minimum and fills the corresponding basin. Eventually, the whole image landscape is flooded and the root node of the component tree is reached. In this way, the algorithm processes each node of the component tree beginning at a leaf node and ending at the root node containing the whole image (see Nistér and Stewénus (2008) for implementation details). In this section, we describe the changes that are required to adapt the algorithm for fastER (compare C++ source code files `faster.h`, `faster.cpp` and `faster_defs.h`).

### 1.4.2 Handling of components

The algorithm from Nistér and Stewénus (2008) processes all nodes in the component tree without ever calculating the whole tree: at most one branch of it is constructed at a time. The corresponding components are stored in a stack with at most as many entries as there are different graylevels in the image (plus one dummy component). Each entry corresponds to one node in the component tree and thus one extremal region  $x$  and, if available, its sub-regions  $x_1$  and  $x_2$ . After a component has been completely processed (i.e. after all its pixels have been processed), it is either merged into its parent component (if pixels of the parent component have already been processed) or it becomes the parent component (if no pixel of the parent component has been processed yet; see Nistér and Stewénus (2008) for details). We now describe the information fastER stores for each component, how components are initialized and merged, how new pixels are added to them and how they are processed after all their pixels have been added.

**Information stored in components.** Unlike the original algorithm, fastER does not store a size and seed history for nested extremal regions, but the following additional information: sufficient statistics  $s_i(x)$  (with  $i = 0, \dots, 11$ ) for the extremal region  $x$  corresponding to the component; the merged segmentation results for all nested extremal regions corresponding to child nodes of  $x$  in the component tree that have been processed so far (i.e. a list with the seeds of the extracted (sub-) regions) and their sum of scores; and, if  $x$  has sub-regions  $x_1$  and  $x_2$ , sufficient statistics  $s_i(x_1)$  and  $s_i(x_2)$  for them and the bounding boxes  $b_1$  and  $b_2$  used to distribute the remaining pixels of  $x$  among them (see above).

**Initializing components.** To initialize a new (and thus empty) component, the sufficient statistics for the corresponding extremal region  $x$  are set to zero (i.e.  $s_i = 0$  for  $i = 0, \dots, 11$ ). Initially,  $x$  has no sub-regions.

**Adding pixels to components.** To add a new pixel  $p \in D$  to a component, the sufficient statistics  $s_i(x)$  for the corresponding extremal region  $x$  are updated. If  $x$  has sub-regions  $x_1$  and  $x_2$ , the sub-region  $x_i$  the pixel should be added to is selected (see above) and the corresponding sufficient statistics are updated as well.

Let  $s_i$  and  $s'_i$  denote the sufficient statistics for a (sub-) region before and after adding the new pixel  $p$ , respectively. Updating the sufficient statistics is trivial for  $s_0$  to  $s_7$ :

$$\begin{aligned} s'_0 &= s_0 + 1 \\ s'_1 &= s_1 + I(p) \\ s'_2 &= s_2 + I(p)^2 \\ s'_3 &= s_3 + p_x \\ s'_4 &= s_4 + p_y \\ s'_5 &= s_5 + p_x^2 \\ s'_6 &= s_6 + p_y^2 \\ s'_7 &= s_7 + p_x p_y \end{aligned} \tag{3}$$

Updating  $s_8$  to  $s_{11}$  is more complicated, because the neighbouring pixels of  $p$  have to be considered. Let  $N(p)$  denote the neighborhood of pixel  $p$ :

$$N(p) = \{q \in D \mid pAq\} \tag{4}$$

As before, let  $G(x)$  denote all edges between pixels in  $x$  and  $D \setminus x$  (equation 1). Then, let  $G_{add}(p, x)$  and  $G_{rem}(p, x)$  denote edges that have to be added to and removed from  $G(x)$ , respectively, when adding pixel  $p$  to  $x$ . For extremal region  $x$ , these can easily be calculated by considering the gray level of each neighbouring pixel:

$$\begin{aligned} G_{add}(p, x) &= \{(p, q) \mid q \in N(p) \wedge I(q) > I(p)\} \\ G_{rem}(p, x) &= \{(q, p) \mid q \in N(p) \wedge I(q) < I(p)\} \end{aligned} \tag{5}$$

For a sub-region  $x_i$  of extremal region  $x$  (let  $j$  denote the index of the other sub-region  $x_j$ ), in addition to the gray levels of the neighboring pixels, their distances to the bounding boxes  $b_1$  and  $b_2$  used for distributing the remaining pixels of  $x$  (see above) among  $x_1$  and  $x_2$  have to be considered to calculate  $G_{add}(p, x_i)$  (let  $d_i(p)$  denote the Manhattan distance of pixel  $p$  to the bounding box  $b_i$ ):

$$G_{add}(p, x_i) = \{(p, q) \mid q \in N(p) \wedge (I(q) > I(p) \vee d_j(q) < d_i(q))\} \tag{6}$$

With this,  $s_8$  to  $s_{11}$  can be updated in  $O(1)$  as follows (assuming that the number of neighbouring pixels is constant, i.e. that  $N(p)$  can be enumerated in constant time):

$$\begin{aligned} s'_8 &= s_8 + |G_{add}| - |G_{rem}| \\ s'_9 &= s_9 + \sum_{(p,q) \in G_{add}} |I(p) - I(q)| - \sum_{(p,q) \in G_{rem}} |I(p) - I(q)| \\ s'_{10} &= s_{10} + \sum_{(p,q) \in G_{add}} (I(p) - I(q))^2 - \sum_{(p,q) \in G_{rem}} (I(p) - I(q))^2 \\ s'_{11} &= s_{11} + \sum_{(p,q) \in G_{add}} I(q) - \sum_{(p,q) \in G_{rem}} I(q) \end{aligned} \tag{7}$$

This guarantees that the sufficient statistics for an extremal region  $x$  (and for its sub-regions  $x_1$  and  $x_2$ , if available) are correct once all pixels of  $x$  have been processed. Assuming that the neighbours  $N(p)$  of a pixel  $p$  can be enumerated in constant time, adding a pixel to a component is in  $\mathcal{O}(1)$ .

**Merging components.** To merge a component corresponding to extremal region  $x$  into its parent component corresponding to extremal region  $par(x)$ , the corresponding sufficient statistics  $s_i$  and  $\hat{s}_i$  are added together:

$$\hat{s}'_i = s_i + \hat{s}_i \text{ with } i = 0, \dots, 11 \quad (8)$$

Sub-regions are handled as follows:

- If  $par(x)$  has no sub-region yet:
  - o If  $x$  has sub-regions,  $par(x)$  takes them over
  - o Otherwise,  $x$  becomes the first sub-region of  $par(x)$
- If  $par(x)$  has exactly one sub-region so far:
  - o If  $x$  has sub-regions or if the bounding boxes of  $x$  and the other sub-region of  $par(x)$  overlap,  $par(x)$  is not considered for splitting anymore
  - o Otherwise,  $x$  becomes the second sub-region of  $par(x)$
- Otherwise:  $par(x)$  is not considered for splitting anymore

Additionally, the segmentation results for extremal region  $x$  are merged into the segmentation results list of the parent component, and the corresponding sums of scores are added together. All this can also be implemented to run in  $\mathcal{O}(1)$ .

**Processing complete components.** When all pixels of a component have been processed, fastER extracts the features from the corresponding extremal region  $x$  and calculates its score  $S(x)$  (and does the same for its sub-regions, if available). Then it applies the recursive segmentation formula (equation 2, main text). Unlike the original algorithm, it does not test for stability. With the sufficient statistics, the features can be calculated in  $\mathcal{O}(1)$ , and therefore the processing of a component is also in  $\mathcal{O}(1)$ . After that, the component is either merged into or becomes the parent component (see above).

### 1.4.3 Runtime analysis

The algorithm from Nistér and Stewénus (2008) runs in  $\mathcal{O}(nm)$ , where  $n$  is the number of pixels and  $m$  the number of possible graylevels in the image (Carlinet and Geraud, 2014). With the adaptations for fastER, the operations to add a pixel to a component, merge two components, and process a component each run in  $\mathcal{O}(1)$  (see above). Since fastER does not perform any additional operations, the overall complexity of the algorithm remains  $\mathcal{O}(nm)$  after it has been adapted for fastER.

## 2 Additional evaluation details

### 2.1 Generation of synthetic fluorescence images

The set of synthetic fluorescence images was created with *SimuCell* v1.0 (Rajaram et al., 2012). In total, 20 images (each with 1200x1200 pixels) were generated (10 for the training set and 10 for the test set). It was specified that within 50 pixels of the image border no cells should be placed. In each image, one subpopulation with 250 bigger cells (shape mode: “cytoplasm”, radius: 20, eccentricity: 0.7, randomness: 0.3) and one with 250 smaller cells (shape mode: “cytoplasm”, radius: 10, eccentricity: 0.2, randomness: 0.2) were placed randomly without overlap. A constant (per cell) marker level of on average 0.3 was used with the following standard deviations (for cell to cell variability) for the ten images in both the training and the test set: 0.055, 0.060, 0.065, 0.070, 0.075, 0.080, 0.085, 0.090, 0.095, and 0.100. The “Distance\_To\_Edge\_Marker\_Gradient” operation was used to make the graylevels of pixels within cells increase with their distance to the cell edges (falloff\_type: 'Exponential', increasing\_or\_decreasing: “increasing”, falloff\_radius: 4 and 2 for subpopulation 1 and 2, respectively). Then, a constant background intensity of 0.2 and additive Gaussian noise (mean: 0, standard deviation: 0.1) were added to each pixel. Finally, the graylevels were normalized from the range 0.0 to 1.0 to 8-bit integers, i.e. values in  $[0, \dots, 255]$ .

### 2.2 Parameters used for evaluation

In this section, we specify the processing steps and parameters that were used for each method and dataset for the quantitative evaluation of segmentation quality (all datasets) and speed (brightfield dataset only).

#### 2.2.1 fastER

Our implementation of fastER always uses the same segmentation pipeline: images are first denoised with bilateral filtering (Tomasi and Manduchi, 1998) and then segmented with fastER. Then, after filling holes in the detected objects, a size filter is applied to remove objects that are too small or too big. The parameters for all these steps are learned automatically from the training labels - the user can only choose between disabling, normal, and strong denoising (see below for details). As in the main text, let  $P_i$  (with  $i = 1, \dots, n$ ) denote the set of positive labels (i.e. a set of regions labeled as corresponding to single cells) for a training image  $I$ , and let  $l$  and  $u$  denote the size of the smallest and biggest positive label, respectively:  $l = \min(|P_i|)$  and  $u = \max(|P_i|)$  for  $i = 1, \dots, n$ . The size filter parameters  $minSize$  and  $maxSize$  are then set to 70% and 130% of the smallest and biggest label, respectively:

$$\begin{aligned} minSize &= \lfloor 0.7 * l \rfloor \\ maxSize &= \lfloor 1.3 * u \rfloor, \end{aligned} \tag{9}$$

where  $\lfloor \cdot \rfloor$  denotes rounding down to the closest integer. To learn the parameters  $sigmaColor$ ,  $sigmaSpace$ , and  $kernelSize$  of the bilateral filter, let  $\hat{d}$  denote the diameter of a perfect circle with the same area as the smallest positive label:

$$\hat{d} = 2 \sqrt{\frac{l}{\pi}} \tag{10}$$

The parameters  $\sigma_{Color}$  and  $\sigma_{Space}$  are then calculated using an affine transformation of  $\hat{d}$  that we found to yield good results in most cases (a value of 10 is used as minimum):

$$\begin{aligned}\sigma_{Color} &= \lfloor \max(10, 4\hat{d} - 10) \rfloor \\ \sigma_{Space} &= \lfloor \max(10, 4\hat{d} - 10) \rfloor,\end{aligned}\tag{11}$$

If strong denoising is enabled,  $\sigma_{Color}$  is doubled:

$$\sigma_{Color_{strong}} = \lfloor 2 * \max(10, 4\hat{d} - 10) \rfloor\tag{12}$$

The kernel size is derived from  $\sigma_{Space}$ , also using an affine transformation:

$$kernelSize = \left\lceil 7 + \frac{3}{50} \sigma_{Space} \right\rceil\tag{13}$$

This pipeline was used for all datasets (with strong denoising in the fluorescence dataset) for the quantitative evaluation. Therefore, to reproduce the segmentation results of fastER for the evaluation, it suffices to import the corresponding training labels into fastER and enable strong denoising for the fluorescence dataset. The files with the used training labels are included in the supplementary data in the “Evaluation” folder. Note that even when using identical training labels, very small variations of the segmentation results are possible due to random partitioning of the training data by LIBSVM (Chang and Lin, 2011) when estimating class probabilities during training. However, the variations are generally very small and not strong enough to change the interpretation of the evaluation results: we repeated training, segmentation and evaluation of fastER five times, and in all datasets F-score, Jaccard index and merge error rate varied within a range of at most 0.4 %, 0.1 % and 0.07 %, respectively.

## 2.2.2 CellProfiler

We used CellProfiler v2.1.1 (Carpenter et al., 2006) for evaluation. For each dataset, a segmentation pipeline with specific processing steps and parameters was created and tuned (using only the training set of each dataset) to achieve best results. All analysis steps were performed in CellProfiler without using external tools. For the brightfield and the phase contrast datasets, *CorrectIlluminationCalculate* and *CorrectIlluminationApply* were used to correct uneven illumination before segmentation. In the fluorescence dataset, *Smooth* was used to denoise images with Gaussian blurring before segmentation. For complete details, refer to the CellProfiler pipeline files included in the supplementary data in the “Evaluation” folder. In all pipelines, after the specific pre-processing steps, cells were detected with *IdentifyPrimaryObjects* using Watershedding (Vincent and Soille, 1991) to separate clumping cells. Then, the segmentation results were refined with *IdentifySecondaryObjects*, also using Watershedding. Finally, after filling holes in the detected objects, a size filter was applied with mostly identical size limits as for fastER (i.e. with the ranges 14 to 1046, 10 to 494, and 50 to 1413 pixels for the phase contrast, brightfield, and fluorescence dataset, respectively).

## 2.2.3 Ilastik

We used Ilastik v1.1.0 (Sommer et al., 2011) for evaluation. No additional pre-processing steps were used, because Ilastik proved robust against uneven illumination and noise. For the phase contrast



and the fluorescence datasets, all features with all sizes were used. To improve segmentation speed for the brightfield dataset, which was also used for evaluation of practical runtime, the largest features (size: 10) were excluded, because they seemed not to improve segmentation. To segment the images, a foreground and a background class were created and labels were added to the training set of each dataset until segmentation did not improve anymore. The results were then exported as *Simple Segmentation*. We post-processed the results by filling holes in the detected objects and applying a size filter with mostly identical size limits as for fastER (i.e. with the ranges 14 to 1046, 25 to 494, and 50 to 1413 pixels for the phase contrast, brightfield, and fluorescence dataset, respectively). For more details (e.g. the used training labels), refer to the Ilastik project files included in the supplementary data in the “Evaluation” folder.

## 2.2.4 CellDetect

We used CellDetect v1.0 (Arteta et al., 2012) for evaluation. Since the phase contrast dataset was already used by the authors of CellDetect for its evaluation (Arteta et al., 2012), we directly used their provided classifier and parameters to segment this dataset. The only additional post-processing step we performed for this dataset was filling of holes in detected objects. For the fluorescence dataset, images were first denoised with Gaussian blurring (kernel size: 9x9, standard deviation: 3) using Matlab (MATLAB, 2013), because segmentation of the raw images was not possible. The centroids of all cells in one image of the brightfield and one of the fluorescence training set were completely labeled to train one classifier for each dataset (compare image and “wStruct\_alpha\_0.mat” files in the “Evaluation/DatasetFL\_Fluorescence\_Synthetic/CellDetect” and “Evaluation/DatasetBF\_BrightField\_MurineHSCs/CellDetect” folders included in the supplementary data). The size limits used by the CellDetect implementation were set to similar values as for fastER (i.e. 25 to 494, and 50 to 1413 pixels for the brightfield and fluorescence dataset, respectively). In these datasets, holes in the detected objects were filled, too.

## 2.2.5 CellX

We used CellX v1.12 (Dimopoulos et al., 2014; Mayer et al., 2013) for evaluation. Like for CellDetect (see above), the images of the fluorescence dataset were denoised with Gaussian blurring (kernel size: 9x9, standard deviation: 3) using Matlab (MATLAB, 2013) before seeding and segmentation. For the phase contrast and fluorescence datasets, we selected the CellX option to pre-process images with contrast limited adaptive histogram equalization (CLAHE) for segmentation and seeding. In the brightfield dataset, CLAHE was also used, but only for segmentation. In the brightfield dataset, results were post-processed by filling holes and applying a size filter (range: 5 to 494 pixels). For the other datasets, post-processing was not necessary. For more details, refer to the CellX project files included in the supplementary data in the “Evaluation” folder.

## 2.2.6 U-Net

We evaluated the 2D U-Net (Ronneberger et al., 2015) using the extended version of the Caffe library (Jia et al., 2014) included with the 3D U-Net (Çiçek et al., 2016). It contains custom layers for the data augmentation that is required also by the 2D U-Net to generate sufficient training data and it supports learning from sparse annotations (i.e. not every pixel has to be labelled in each training image). The library was compiled from source, and the 2D U-Net network structure *prototxt* file was upgraded so it could be parsed by the Caffe version used by the 3D U-Net. We then incorporated the 'CreateDeformation' and 'ApplyDeformation' layers included with the 3D U-Net to augment training data with random elastic deformations, mirroring, rotations and offsets. The sizes of the extracted image patches were set to 588 x 588, 572 x 572 and 348 x 1100 (height x width) pixels for datasets

FL, PC and BF, respectively. Thus, in the lowest resolution the resulting networks contained  $33 \times 33$ ,  $32 \times 32$  and  $18 \times 65$  pixels. Bigger image patches could be extracted from the images in datasets FL and BF, but this was not possible without running out of memory (using the GeForce GTX 1080, 8 GB). The high width was chosen for dataset BF, because its images contain a strong gradient of illumination and cell density along the x-axis. Due to the high width, each extracted image patch thus contained areas with low and high illumination as well as low and high cell densities. Additionally, for dataset BF dropout layers were disabled, and input image data was centred by subtracting the mean image intensity from each pixel. This seemed to improve results strongly for this dataset. The elastic deformation parameters were tuned for each dataset so that deformations were noticeable, but the images still looked realistic. For datasets PC and FL, grey level variations were induced using the ‘ValueAugmentation’ layer included with the 3D U-Net. This was not done for dataset BF, because the layer seems to be incompatible with centred input data. However, the training data for this dataset already contains strong grey level variations.

A custom Matlab script was used to convert pixel values to single precision floating point numbers between 0 and 1; to normalize the pixel values for dataset BF; to add separating pixels between adjacent cells as described in Ronneberger et al. (2015) (this was only required for dataset FL); to calculate weight maps as described in Ronneberger et al. (2015) to force the network to learn correct classification of pixels separating adjacent cells (i.e. to avoid merge errors); and to save the resulting training data in the required HDF5 format. The parameter  $w_0$  for the weight map generation was set to 10 as in Ronneberger et al. (2015) for datasets FL and BF, and to 20 for dataset PC due to the very large number of merge errors produced for this dataset when using  $w_0 = 10$ . The number of merge errors was still comparably high, but further increasing  $w_0$  did not seem to improve this. The  $\sigma$  parameter was set to 5 as in Ronneberger et al. (2015) for all datasets. Inverse class frequencies were used for the parameter  $w_c(x)$ .

For dataset FL, the complete training set, including labels for all pixels, was used for training. For datasets PC and BF, sparse training labels were generated with a modified version of the fastER interface using manual labelling only. This was done in several iterations for each dataset (i.e. labels were used for training and segmentation of the training data, and new labels were then added according to encountered segmentation errors).

After training a classifier for each dataset, images were segmented using the segmentation script included with the 2D U-Net using the lowest possible value for the  $nTiles$  parameter without causing an out of memory error (i.e. 7, 1 and 8 for datasets FL, PC and BF, respectively). The script was adjusted to include the image normalization performed for dataset BF.

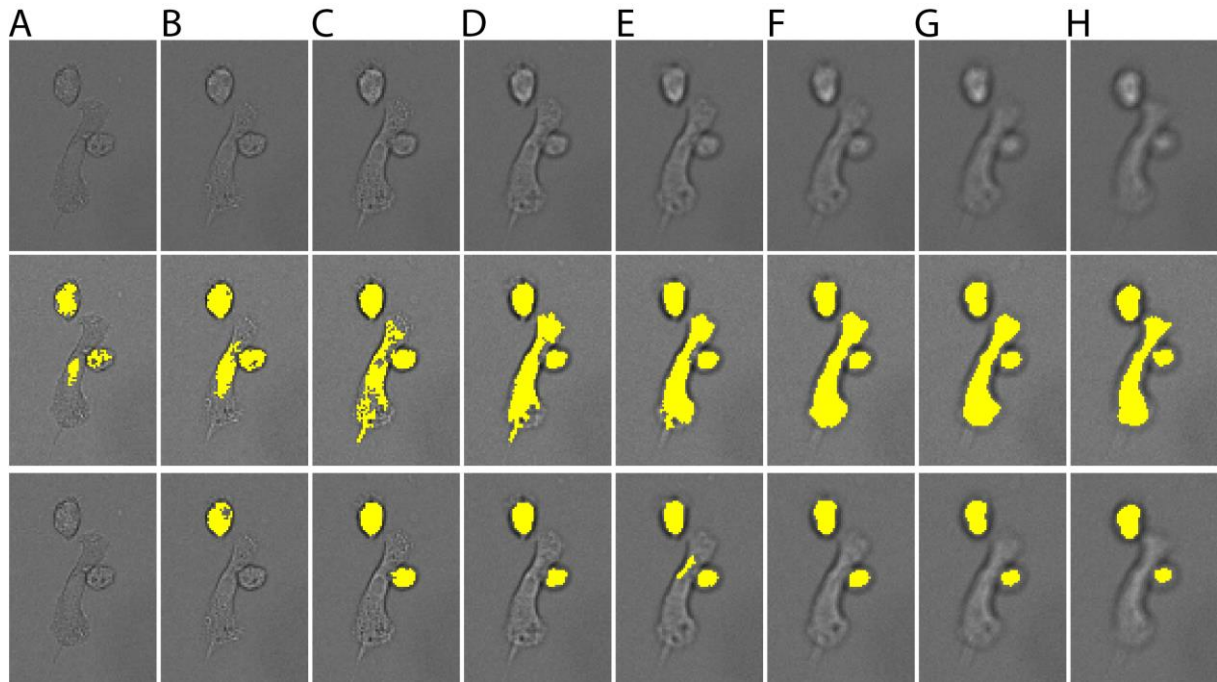
Segmentation results were post-processed for datasets PC and BF by filling holes and applying a size filter afterwards using the same size limits as for fastER (i.e. with the ranges 14 to 1046 and 25 to 494 pixels for the PC and BF dataset, respectively). No post-processing was required for dataset FL.

For more details (i.e. the network prototxt files, the used training labels and the used solver parameters), refer to the files included in the supplementary data in the “Evaluation” folder.

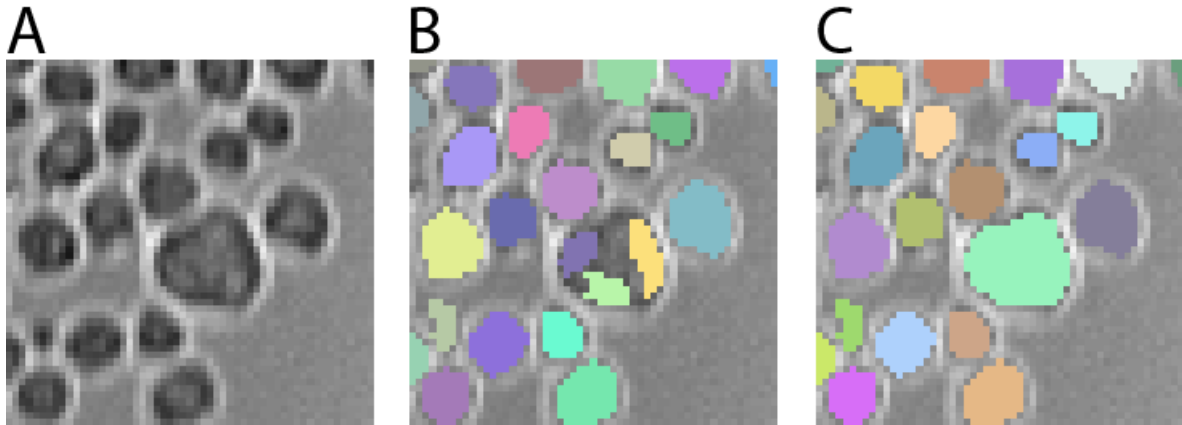
### **3 Population dynamics of in-vitro blood formation**

The raw data of the long-term time-lapse imaging experiment we used for the analysis of population dynamics of in-vitro blood formation as well as the training labels that were used for fastER are available on request. In the beginning, the experiment contains only about 250 cells, which are very small (i.e. only about 30 pixels per cell) and distributed over 48 positions with in total almost 70 million pixels. This small number of cells and the presence of dead cells as well as other debris with similar morphologies as living cells causes a high false-positive rate in the beginning of the experiment: we estimate that at the first time point around 304 detections (i.e. 52 %) are false positives. However, the total number of false positives remains relatively constant over time, because they are mostly caused by the same dead cells and debris rather than occurring randomly. Therefore, we subtracted 304 of the number of detections at each time point to correct for false positives and thus obtain an unbiased estimate for the actual cell counts.

# Supplementary Figures



**Supplementary Fig. 1. Acquiring transmitted light microscopy images slightly out-of-focus facilitates automatic cell segmentation with extremal regions (ERs).** Patches of brightfield microscopy images of murine blood cells acquired with different focus levels (top row) are shown with best matching ERs (middle row) and maximally stable extremal regions (MSER, bottom row), if available (a custom C++ implementation of the linear time MSER algorithm described in Nistér and Stewénus (2008) was used with the parameters:  $\delta=3$ ,  $\text{maxVariation}=0.25$ ). (A) When acquired in-focus, cells have low and varying contrast (especially, when they are big and complexly shaped) and fit only poorly to extremal regions. (B-G) As images are acquired more and more out-of-focus, contrast improves and cells appear as homogeneous regions that are brighter than background, thus enabling segmentation with ERs. However, only cells with high contrast can be accurately segmented using MSER. (H) When acquired too much out-of-focus, morphological details disappear, thus decreasing segmentation accuracy. Images were denoised with bilateral filtering (Tomasi and Manduchi, 1998) before extracting ERs and MSER, and holes in found regions were filled. Contrast of all images was enhanced for display only.



**Supplementary Fig. 2. Extracting a set of extremal regions with maximal average score avoids over-segmentation errors.** One image (an image patch with 55 x 55 pixels is shown) from the test set of dataset BF (A) was segmented with a modified version of fastER that extracts a solution with maximal sum of scores (B) and with the original version of fastER that extract a solution with maximal average score (C) using identical training labels and parameters. The example illustrates that extracting a solution with maximal sum of scores can indeed cause over-segmentation errors. Splitting of extremal regions was disabled in both segmentation runs. Contrast was enhanced for display only.

# Supplementary Movies

**Supplementary Movie 1. Interactive learning with fastER.** The high training and segmentation efficiency of fastER allows showing segmentation results in real-time as the user labels cells and background for training, thus facilitating efficient and effective adaption of the method to specific image sets.

# References

- Arteta, C. *et al.* (2012) Learning to Detect Cells Using Non-overlapping Extremal Regions. In *Medical Image Computing and Computer-Assisted Intervention*, pp. 348–356. Springer.
- Carlinet, E. and Geraud, T. (2014) A Comparative Review of Component Tree Computation Algorithms. *IEEE Trans. Image Process.*, **23**, 3885–3895.
- Carpenter, A. E. *et al.* (2006) CellProfiler: image analysis software for identifying and quantifying cell phenotypes. *Genome Biol.*, **7**, R100.
- Chang, C.-C. and Lin, C.-J. (2011) LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.*, **2**, 1–27.
- Çiçek, Ö. *et al.* (2016) 3d u-net: Learning dense volumetric segmentation from sparse annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 424–432. Springer.
- Dimopoulos, S. *et al.* (2014) Accurate cell segmentation in microscopy images using membrane patterns. *Bioinformatics*, **30**, 2644–2651.
- Jia, Y. *et al.* (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014. S. 675–678.
- MATLAB and Image Processing Toolbox Release 2013a, The MathWorks, Inc., Natick, Massachusetts, United States.
- Mayer, C. *et al.* (2013) Using CellX to Quantify Intracellular Events. *Current Protocols in Molecular Biology*, **14**, 1–21.
- Nistér, D. and Stewénius, H. (2008) Linear Time Maximally Stable Extremal Regions. In, *ECCV 2008*, pp. 183–196. Springer.
- Rajaram, S. *et al.* (2012) SimuCell: a flexible framework for creating synthetic microscopy images. *Nat. Methods*, **9**, 634–635.
- Ronneberger, O. *et al.* (2015) U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of LNCS, pages 234–241. Springer.
- Sommer, C. *et al.* (2011) Ilastik: Interactive learning and segmentation toolkit. In, *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. IEEE, pp. 230–233.
- Tomasi, C. and Manduchi, R. (1998) Bilateral filtering for gray and color images. In, *Sixth International Conference on Computer Vision*. IEEE, pp. 839–846.

Vincent,L. and Soille,P. (1991) Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Mach. Intell.*, **13**, 583–598.