DevStaR: High-Throughput Quantification of *C. elegans* Developmental Stages

Amelia G. White, Brandon Lees, Huey-Ling Kao, P. Giselle Cipriani, Eliana Munarriz, Annalise B. Paaby, Katherine Erickson, Sherly Guzman, Kirk Rattanakorn, Eduardo Sontag, Davi Geiger, Kristin C. Gunsalus*, and Fabio Piano

***Abstract*—We present DevStaR, an automated computer vision and machine learning system that provides rapid, accurate, and quantitative measurements of *C. elegans* embryonic viability in high-throughput (HTP) applications. A leading genetic model organism for the study of animal development and behavior, *C. elegans* is particularly amenable to HTP functional genomic analysis due to its small size and ease of cultivation, but the lack of efficient and quantitative methods to score phenotypes has become a major bottleneck. DevStaR addresses this challenge using a novel hierarchical object recognition machine that rapidly segments, classifies, and counts animals at each developmental stage in images of mixed-stage populations of *C. elegans*. Here, we describe the algorithmic design of the DevStaR system and demonstrate its performance in scoring image data acquired in HTP screens.**

***Index Terms*—*C. elegans*, computer vision, high-throughput phenotyping, object recognition.**

I. Introduction

C. elegans is one of the major model organisms used to study fundamental questions in animal development and neurobiology [1]. *C. elegans* was the first animal to have its genome completely sequenced [2], providing new opportunities for genome-wide analyses. For example, gene knockdown screens using RNA interference (RNAi) have been used to assess *in vivo* function for most of the 20 000 genes encoded in its genome [3], leading to the identification of entire sets of genes required for many key developmental and physiological processes in this organism [4].

Because it is small (∼1 mm as a mature adult) and can be cultured in liquid media in 96- or 384-well plate formats, *C. elegans* is highly amenable to a variety of high-throughput (HTP) functional genomic analyses. Numerous technical innovations in assay systems, robotic sample manipulation, and reagent libraries [5], [6] have rendered genome-wide HTP phenotypic screens increasingly feasible: with assistance from liquid handling robots, it is now possible to conduct thousands of experiments per week on intact organisms to assay different combinations of environmental and genetic perturbations.

The output of HTP phenotypic screens is often image data that must be analyzed to score resulting biological effects, such as viability, fitness, morphology, or behavior. Phenotypic analysis is the most challenging aspect of a large-scale screen and is usually rate limiting, since it typically requires expert manual annotation. This tends to be a painstaking process that is slow, qualitative, and potentially error-prone.

The burden of analyzing phenotypic readouts from HTP platforms can be alleviated by the application of image analysis software that can extract quantitative features from image data. Because analyzing bright-field images is very challenging, this task is commonly rendered more tractable by employing fluorescent reporters that provide easily identifiable markers of specific cell types, cellular compartments, or sub-cellular structures. Fluorescent markers have been used in combination with different image analysis tools to automatically extract numerous morphological features and measure quantitative phenotypes based on these "phenotypic characters" in the single-celled yeast *S. cerevisiae* [7] and in cell lines from *Drosophila* or human [8].

In whole-organism screens, one of the most fundamental phenotypes that can be assayed is survival or, conversely, lethality: this simple readout is often used as the first measure of different genetic or environmental perturbations, including temperature or chemicals (i.e., small molecules and drugs). Most mechanical steps required for HTP screening of *C. elegans* are relatively straightforward to optimize and streamline, but a major obstacle has been the lack of automated tools for the quantitative analysis of complex phenotypes in image data. Over the last few years, several groups have developed image analysis ap-

Manuscript received January 13, 2013; revised April 09, 2013; accepted May 16, 2013. Date of publication May 29, 2013; date of current version October 02, 2013. The work of K. C. Gunsalus and F. Piano was supported by the National Institutes of Health under Grant NICHD R01-HD046236. *Asterisk indicates corresponding author.*

A. G. White is with the Department of Computational Biology and Molecular Biophysics, Rutgers University, Piscataway, NJ 08854 USA, and also with the Center for Genomics and Systems Biology, Department of Biology, New York University, New York, NY 10003 USA (e-mail: white.amelia@gmail.com).

B. Lees, H.-L. Kao, P. G. Cipriani, E. Munarriz, A. B. Paaby, K. Erickson, S. Guzman, and K. Rattanakorn are with the Center for Genomics and Systems Biology, Department of Biology, New York University, New York, NY 10003 USA.

E. Sontag is with the Department of Mathematics, Rutgers University, Piscataway, NJ 08854 USA (e-mail: sontag@math.rutgers.edu).

D. Geiger is with the Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, NY 10003 USA (e-mail: geiger@cs.nyu.edu).

*K. C. Gunsalus is with the Center for Genomics and Systems Biology, Department of Biology, New York University, New York, NY 10003 USA, and also with the Division of Science, NYU Abu Dhabi, Abu Dhabi, UAE (e-mail: kcg1@nyu.edu).

F. Piano is with the Center for Genomics and Systems Biology, Department of Biology, New York University, New York, NY 10003 USA, and also with the Division of Science, NYU Abu Dhabi, Abu Dhabi, UAE (e-mail: fp1@nyu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMI.2013.2265092

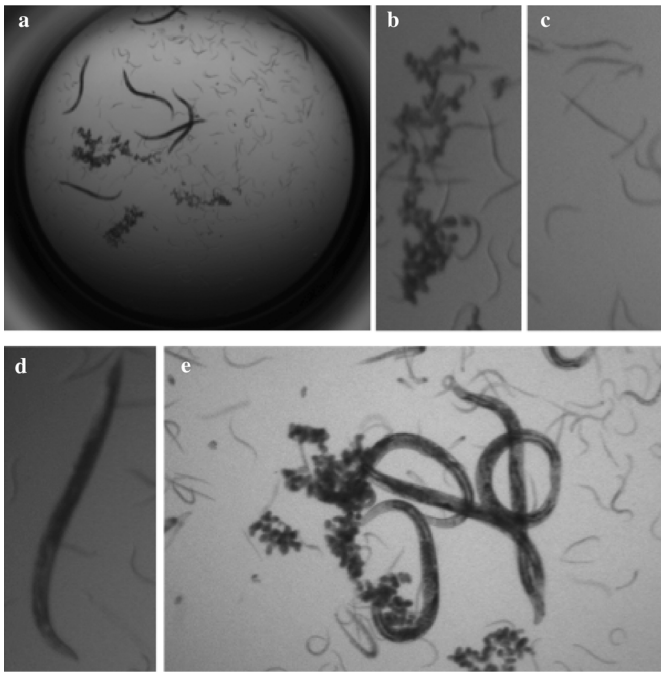


Fig. 1. Examples of microscopic images of *C. elegans*. (a) Typical image processed by DevStaR: a single well from a 96-well plate containing a *C. elegans* population of mixed developmental stages. (b)–(d) Magnified views of different developmental stages. (b) Embryos, seen as a clump of small dark ovals. (c) Larvae. (d) Adult worm. (e) Occlusions and deformations of objects: overlaps can represent different regions of a single animal or intersections between different individuals.

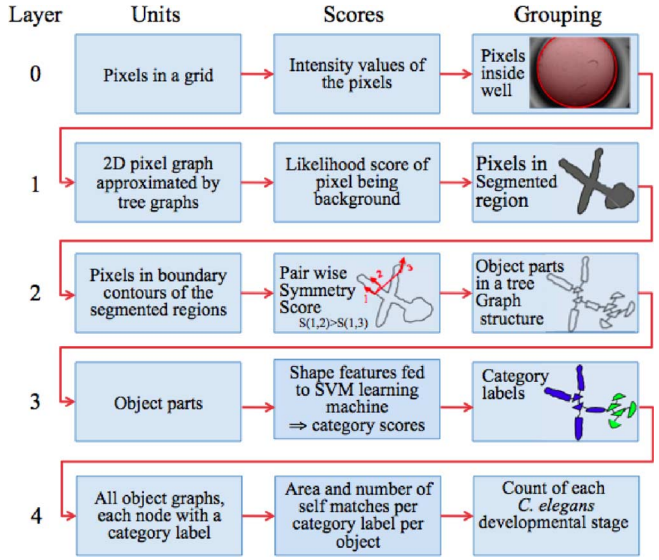


Fig. 2. Schematic of the DevStaR hierarchical recognition system for *C. elegans* developmental stages. The machine contains five layers (Layers 0–4), each comprising three steps (Units, Scores, Grouping). Units: Input units are the data on which each layer operates. Scores: Vision algorithm computes a score for each input unit based on explicit evaluation criteria. Grouping: Grouping vision algorithm produces a higher-order representation of the input units based on their scores and local topological relationships. The output of each preceding layer (grouped data) is used as input (unit data) for each successive layer.

proaches to measure different aspects of *C. elegans* biology, including adult morphology [9], locomotory behavior [10]–[13], and embryonic cell divisions [14], [15].

Since a first-level goal of many HTP screening assays is to assess developmental fitness, an efficient solution to this task is central to progress in this field. So far, however, no software solution addresses the problem of quantifying different developmental stages in bright-field images of mixed-stage populations. An alternative we considered is the COPAS BIOSORT (Union Biometrica), a flow-sorting instrument that automates physical sorting of *C. elegans*. However, this method is not practical for HTP screening because the counting is too slow: according to systems specifications, 35–150 individuals can be counted per second, translating to 8–32 min per 96-well plate (excluding sample preparation time, which further reduces throughput efficiency).

We have developed a hierarchical vision recognition system to automatically and quantitatively measure *C. elegans* developmental phenotypes from high-throughput microscope images, which we call DevStaR (developmental stage recognition). DevStaR, to our knowledge, is the only software that can segment and differentiate multiple developmental stages from images of populations of *C. elegans*, and it provides the first truly feasible, real-time solution for this highly relevant problem.

The specific problem we address is to recognize and count each developmental stage within mixed-stage populations of *C. elegans* animals, in order to determine fitness and viability. Taking bright-field microscope images as input (Fig. 1), DevStaR labels each pixel in an image as one of four classes: adult worm, larva, egg/embryo (used here interchangeably), or background. DevStaR then outputs quantitative measurements of classified objects that enable the calculation of lethality (or survival) in each sample. This task is particularly challenging both because illumination varies within and across images and because complicated occlusions and deformations of the animals are common [e.g., see Fig. 1(e)]. Furthermore, the objects may be in different focal planes since the animals are swimming in liquid, and the focal length may differ between images due to manual refocusing by the biologist. DevStaR addresses several major hurdles in the analysis of these biological image data using novel algorithmic solutions.

DevStaR is structured as a hierarchical object recognition system comprising a series of layers (numbered zero to four) that we call vision layers (Fig. 2). In each layer two vision algorithms are applied that: (i) take input units (nodes of the input graph) and perform an evaluation score on the units (typically using signal processing or a learning method), and then (ii) group the units based on spatial properties (geometrical and/or topological) and the evaluation score from (i), thus, reducing the size of the output graph. For example, in layer 0 we extract intensity features and then use the geometrical properties of circles to group the pixels that form the well. The grouping transforms the preceding, lower-level representation of the data (the input graph) into a new, higher-level representation (the output graph, which will contain a smaller number of nodes due to grouping). Thus, each layer outputs a new reduced graph to feed as input to the next layer. In this sense, the hierarchy presented here maybe viewed as a pipeline of sequential vision layers, where each layer reduces the complexity of the data, due to grouping, and thus delivers a new and more compact representation of the data.

DevStaR can be generalized to other image analysis problems concerning populations of *C. elegans*, for example to discriminate different larval stages or measure postembryonic growth rates and viability. The individual layers of DevStaR may also find application in other biological imaging problems: for example the techniques developed in layer 3 could be useful for recognizing morphological phenotypes of either whole animals or cultured cells.

Here, we present a full description of DevStaR and its application to HTP screening applications. We include a thorough performance analysis and show that the current version of DevStaR improves upon our earlier work [16] in the techniques used for several layers. Specifically, in Layer 1 we now apply a background removal step followed by a thresholding procedure to segment the image; we apply Layer 3 to all objects in the image (instead of first separating small and large objects); and in Layer 4 we output a count of animals at each developmental stage (rather than pixel areas).

DevStaR has been integrated into the analysis pipeline for ongoing genome-wide HTP genetic screens in our Center, and so far around two-dozen screens have been completed using automated DevStaR analysis as the primary scoring mechanism. DevStaR runs on average in 15 s per 1200×1600 pixel image on a single processor. For a 96-well plate, it takes about 1.5 min to acquire one image per well, and the same amount of time to analyze the images on a 16-core Linux box with 24 GB RAM. Thus, data can be fully scored by DevStaR in essentially real time.

II. Background: Visual Recognition in Image Analysis

Understanding the content of an image is arguably the primary goal of computer vision and, despite many decades of research, this challenge is far from solved. The main difficulty in understanding images derives from the many sources of variability in an object's appearance, e.g., pose transformations, lighting effects, intra-class variation, and occlusions. In addition, objects can be combined in a scene in an exponential number of ways.

Recent work in recognition has focused on building complex probabilistic models that explicitly model the many sources of variability in an image [17], [18]. These models typically contain many parameters, which are estimated from labeled sets of training data using machine learning techniques. These systems are not very complex: they are not hierarchical, and they typically rely on first applying a basic feature detector followed by a learning machine to perform a single recognition task. In particular, shape information and occlusions are typically not addressed, and images with overlapping objects and deformations are largely ignored [19], [20]. These systems would therefore not perform well in our application.

In principle, learning techniques such as SVM [21], Boosting [22], and Convolution Networks [19] can be applied to any data and problem. In practice, however, many manual decisions and intermediate steps are required, and these determine how well the systems work. For example, in SVM learning, one has to choose the features and the kernels; in Boosting, one has to choose the set of weak learners; and in Convolution Networks, one has to choose the number of layers and the proper architecture for the connections.

Recently, "deep learning" models [23] have engendered excitement. These are essentially hierarchical layered machines, in which each "feature" layer performs its own learning in turn [23]. While such models address some of the deficiencies of the above-mentioned learning methods, much more progress is needed in modeling hierarchical systems.

We note that a large number of effective algorithms used in vision are not derived from learning theory. These include all techniques based on signal processing (e.g., steerable pyramids [24] and wavelets [25]), as well as more geometric based vision methods (where spatial relations are at the core of the formulation) such as: Hough Transforms [26], graph-based methods (e.g., graph cuts [27], [28]), dynamic programming (e.g., [29]), normalized cuts (e.g., [30]), and shape extraction (e.g., [31], [32]). However, geometric based methods are typically applied to a specific specialized task and individually are insufficient to solve complex vision problems in real-world applications.

We argue that solutions to complex vision problems—in which image variability is large and performance equivalent to human interpretation is needed—will require combining learning techniques with other state-of-the-art vision algorithms. Moreover, we believe that developing methodology for the hierarchical organization of processing layers—each emitting successively higher-level units by applying a combination of scoring (rooted on learning method or signal processing methods) and grouping functions (rooted on spatial processes) to the preceding units—is needed for building complex computer vision systems. Our hierarchical design for DevStaR follows these principles.

III. DevStar Hierarchical Recognition System

The analysis performed by DevStaR begins with a graph representation of the image. A generic graph is represented by $G(v, e)$, where v is a vertex in the graph and e is an edge in the graph. In our application, the image is defined in a pixel graph $G_p(v, e)$, where v represents a pixel and stores the grey level, and e represents an edge between v and its eight neighboring pixels. The image grid is defined by the set F.

The DevStaR hierarchy, shown in Fig. 2, is described below in terms of the organization of the vision layers: Units, Scores, and Grouping of the Units.

A. Layer 0: Attention (Area of Interest)

Layer 0 performs the initial task of choosing within each image the area of interest (AOI) in which to perform the search for objects. The objects are all inside the well, which has a more or less circular shape in the image [see Fig. 1(a)].

Units: The nodes of the pixel graph $G_p(v, e)$.

Scores: The intensity at each pixel is stored in the nodes v in $G_p(v, e)$.

Grouping: We want to group the pixels inside the well (pw). We use a circle shape as the geometric constraint to group the pixels. Note that the search method proposed here is rooted on the geometric voting method of Hough transform. We define "mean intensity" as the mean value of intensity for all pixels along the circumference of a given circle, and we compute this

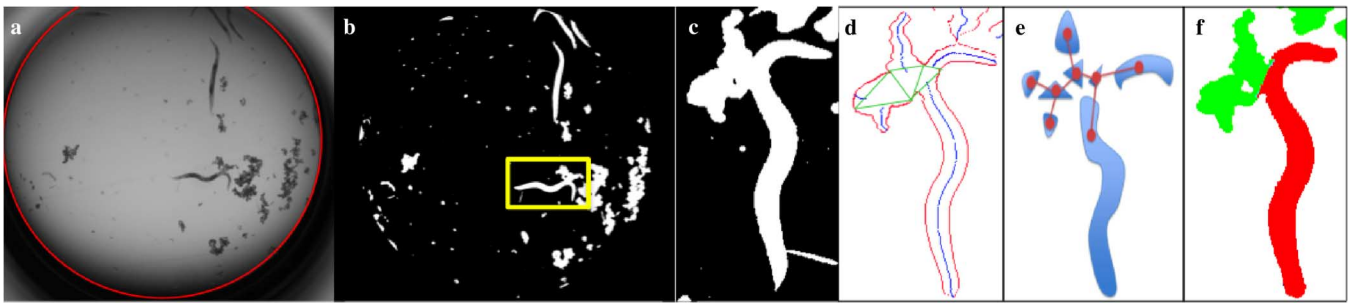


Fig. 3. Examples of the output from each DevStaR layer produced upon application of its grouping mechanism. (a) Layer 0 output: a graph $G_{pw}(v, e)$ comprising pixels inside the red circle, which define the AOI. (b) Layer 1 output: Segmentation of the image from (a) after background removal. (c) Magnified view of a large object from (b) (yellow box). (d), (e) Layer 2 output: for the shape shown in (c), a tree graph $GT_{op}(v, e)$, where nodes represent object parts and bifurcations, and edges connect adjacent regions. (d) Shown are the best pair-wise matching of boundary pixels for the object (red), the midpoint of the match (blue), and bifurcations that break the object into parts (green triangles). (e) Schematic of the tree graph $GT_{op}(v, e)$, showing nodes (red dots) superimposed on their respective object parts and bifurcations (blue) and edges connecting them (red lines). (f) Layer 3 output: Label assignment $GTL(v, e)$ for each node of the Layer 2 tree graph from (d), represented here by color: adult worm (red), embryo (green).

quantity for a range of radius values and center positions. For each center position considered, two circles with incremental radii are compared, and the signed difference in mean intensity is stored. We are interested in identifying changes from bright (high mean intensity) to dark (low mean intensity) values as the radii increase. The best circle (group) within a range of center positions and radii is defined as the outer circle representing the highest negative change in mean intensity between adjacent concentric circles.

Output Graph: We output the graph $G_{pw}(v, e)$, which is the graph $G_p(v, e)$ restricted to the nodes and edges corresponding to pixels inside the detected well (pw) (Fig. 3(a) shows an example).

Parameters: The parameters that need to be considered are the ranges of search for 1) the center position and 2) the radius of the circle. In this application, the ranges are fixed for all images based on an estimate of their maximum variation from a small sample of images (less than 100). The search ranges for the center coordinates of the circle are $x = 700$–900 and $y = 500$–700, and the range for radii is 550–940 pixels. Pairs of x, y coordinates and radii were sampled in intervals of 10 pixels.

B. Layer 1: Background Removal and Segmentation of Objects

Layer 1 first models and extracts the background from each image to allow accurate segmentation. Light microscopy images of multi-well plates contain large variations in intensity and contrast within and between wells, making background removal and segmentation difficult. Several factors conspire to cause a large but gradual decrease in intensity toward the edge of the well (see Fig. 1 for an example), including shadows from the walls and light diffraction by the meniscus (curvature of the liquid surface due to wetting of the walls). Moreover, the variation in illumination differs depending on the exact alignment of the well and settings of the microscope. These issues render the use of standard segmentation algorithms impractical, as the required parameters vary greatly both within and across images.

To overcome these challenges, we chose to remove the background by modeling the gradual changes in intensity as a continuous 3-D surface. We extract a model of the background for each image to accommodate the unique landscape of intensity variation. Another method that could be considered is the bias field removal [33], which is based on the EM algorithm and thus it is iterative and potentially slower. The approach we chose is guaranteed to be fast (on the order of a second for each image). Background removal enables us to accurately segment the objects across the well by a simple thresholding of the resulting images, as shown in Fig. 4.

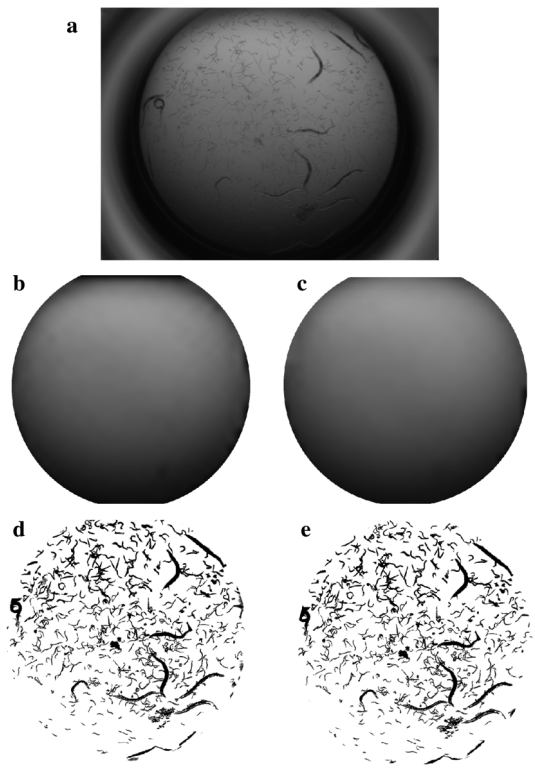


Fig. 4. Examples from Layer 1: background removal and image segmentation. (a) Light microscopy image of a population of *C. elegans*. (b) Background model, found by [36]–[38]. (c) Background model, found by layer 1. (d) Segmentation results using background from (b). (e) Segmentation results using background from (c). Results suggest layer 1 performs a good approximation of the optimal solution.

Units: The nodes of the graph $G_{pw}(v, e)$.

Scores: Our goal is to fit a 3-D surface, $s(x, y)$, to the background image data, ignoring foreground pixels. We use robust statistics to fit a piecewise linear model to the data. More

precisely, for a pair of neighbor pixels $\{(x,y),(x',y')\}$ where $(x',y') \in N(x,y)$, we write the robust potential

$$V\left(s(x,y),s(x',y')\right) = (1-\theta(Z))\,\lambda + \theta(Z)\left(\sum_{y''=y'}^{y}\sum_{x''=x'}\left(D(x'',y'')-s(x'',y'')\right)^2\right)$$
$$Z = \lambda - \Sigma_{y''=y'}^{y}\Sigma_{x''=x'}^{x}\left(D(x'',y'')-s(x'',y'')\right)^2$$

where $\theta(Z)$ is the Heaviside function, i.e., if $z \geq 0$ and $\theta(Z) = 1$ otherwise $\theta(Z) = 0$. For each pair of neighbor vertices and states $\{s(x,y),s(x',y')\}$ we have the "observed values" $D = \{D(x'',y');x=x',..,x;y=y',\ldots,y\}$ and a piece-wise linear model for $s(x,y)$

$$s(x'',y'') = s(x',y') + \omega(x'',y'')\left(s(x,y)-s(x',y')\right)$$

with weights $0 \leq \omega_1 \leq 1$

$$\omega_1 = \sqrt{\frac{\sqrt{(x''-x')^2+(y''-y')^2}}{\sqrt{(x''x')^2+(y''-y')^2}+\sqrt{(x''-x)^2+(y''-y)^2}}}.$$

Note that the neighbors $(x',y') \in N(x,y)$ are not the first pixel neighbors on the grid; they are pixels distant from each other by a window size $2W+1$, i.e., $x' = x \pm W$ and $' = y \pm W$ so that a piecewise linear model is meaningful. The parameter W needs to be estimated. We only consider the four neighbors

$$N(x,y) = \{(x_W,y),(x,y_W),x_{W^+},y),(x,y_{W^+})\}$$

where

$$x_W = x-W, y_W = y-W, x_{W^+} = x+W, y_{W^+} = y+W.$$

The eventual goal is to infer the states s given the observed D. The robust potential $V(s(x,y),s(x',y'))$ suggests that the better is the fit of the linear model to the data, the smaller is the potential. At places where the model does not fit well, i.e., the error is above λ, the cost of the fit is simply λ. In this way object pixel intensities are ignored during this procedure by assigning a constant weight (λ) when the distance from the fit line to the pixel intensity is large; this allows object pixels to be ignored without incurring an enormous fitting cost.

Grouping: A Smoothing term is added to the robust potential to group neighbor pixels and interpolate the data where background data is not available (at object pixels). We add the term

$$\text{Smooth}\left(s(x,y),s(x',y')\right) = \mu\left|s(x,y)-s(x',y')\right|$$

where neighbor states $s(x,y),s(x',y')$ with similar state values produce lower potentials, and μ controls the weight of the smoothing. In summary, the rationale for each pixel is that: 1) if the pixel intensity is far from the background state, it is because it belongs to the foreground and the model ignores this data, and 2) if it is a background pixel, a straight line segment between two neighbor state pixels separated by pixels should fit the data well.

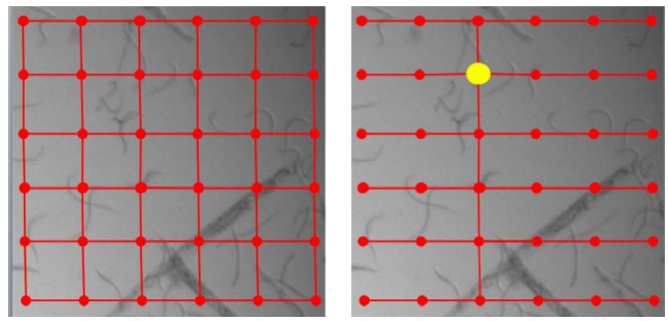


Fig. 5. Conversion of 2-D pixel graph into a tree graph. Left: 2-D pixel graph shown over the image in red. Right: all vertical edges in 2-D pixel graph have been removed except the ones in the column containing the root (yellow), creating a tree graph.

Finding the optimal background states solution is not possible in polynomial time; since there are loops in the image grid graph (edges connect all neighbor pixel nodes) and the robust statistics potential on node pairs is nonconvex, i.e., this problem is NP hard. We apply an approximation method. First we approximate the pixel graph by removing some of its edges and eliminating the loops, thus transforming the image graph into multiple tree graphs. More precisely, the tree graph approximation is made by creating one graph per node in $G_{pw}(v,e)$, using the node as the root of the tree. For each root node, the 2-D pixel graph is made into a tree by removing all vertical edges except the ones in the column containing the root (Fig. 5).

Second, we apply dynamic programming to this graph with one pass. The algorithm begins on the leaves of the tree (pixels on the image boundary) and moves toward the root node, where the optimal cost solution is obtained (standard dynamic programming forward algorithm). The running time of this algorithm is 0.6 s per 1200 pixels $\times$ 1600 pixels image on a 2.3-GHz Linux box with 12 GB of RAM.

Other algorithms are available to find the approximate solution to this NP hard problem, e.g., [34], [35]. A method described in the set of papers [36]–[38] is of particular interest since when it finds a solution it is guaranteed to be the optimal solution. Fig. 4(b) and (c) compares the background solution found using our approximation method to that found with the optimal method from [36]–[38]. The background solutions found are very similar, giving us evidence that our technique is comparable to the optimal solution. Our solution is much more efficient, taking less than a second to run on one image, where as the optimal solution from [36]–[38] takes minutes to run on one image on a 2.3-GHz Linux box with 12 GB of RAM.

Finally, once we have found the background state for the sub-sampled pixel locations, we fit a B-spline to these points to extract the entire background contour [Fig. 4(b) and (c)]. After the optimal background states are obtained, they can be subtracted or divided from the original image and a thresholdapplied to separate the regions containing objects from the background pixels [Fig. 4(d) and (e)].

Output Graph: The resulting output is a graph for each segmented region, with nodes belonging to pixels within the segmented region. We can describe the output simply by the region boundary of each segmented region. Say an image Z has N_Z

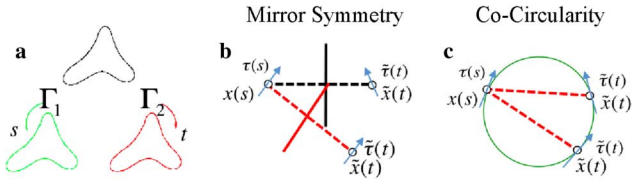


Fig. 6. Layer 2 Scoring function. (a) Shape boundary contour can be parameterized in two ways: counter-clockwise (Γ_1, green), and clockwise (Γ_2, red). Each pixel from Γ_1 is evaluated for pairwise symmetry with each pixel from Γ_2. Measure of symmetry of two elements on the shape boundary is given by a mirror symmetry score shown in (1). (b) $\tilde{\tau}(t)$ are both perfectly symmetric to $\tau(s)$, so the mirror symmetry score is zero. (c) Equivalent co-circularity measure where $\tilde{\tau}(t)$ are both perfectly co-circular to $\tau(s)$, so the mirror symmetry score is zero.

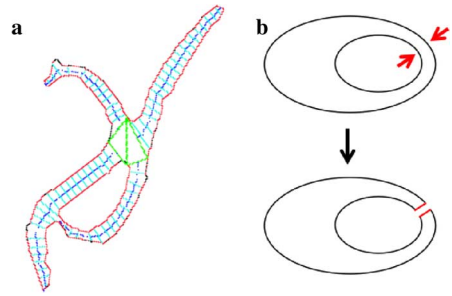


Fig. 7. Layer 2 Grouping mechanism and extension to multiple boundary contours. (a) Dynamic programming algorithm described in [31] finds the best pairwise matching of boundary contour pixels to obtain the optimal symmetry pairing for a given object (here, two overlapping adult C. elegans). Shown are the boundary contour pixels (red), every fifth matching pair (each connected by a cyan line), the bifurcation points that break the object into parts (green triangles), and the skeleton of each object part (dark blue lines). End of the skeleton is where the optimal matching is a self-match (i.e., a pixel matching to itself). The shape can now be represented as a tree graph $GT_{op}(v,e)$, where nodes represent object parts and bifurcations, and edges connect adjacent parts, as shown in Fig. 3(e). (b) To extend the symmetry axis method in [31] to objects with multiple boundary contours (here represented as concentric ovals), we select the point in each contour where the two contours are closest together (red arrows). We then apply a "topological surgery" (red lines) to link the two shape boundaries, creating a final long unique boundary shape.

segmented regions. A segmented region i can be represented by $\Gamma_Z^i(v,e)$, an ordered set of vertices representing the pixels at the boundary of region i and graph edges indicating a link between neighbor boundary pixels. The output is the union of all segmented contours, i.e., $G_{rb}(v,e) = \cup_{i=1}^{N_Z}(v,e)$.

Parameters: All parameters are chosen empirically. There are few parameters and their meaning in the model is clear, testing and modifying them worked well across all images so more sophisticated techniques such as learning were unnecessary. For the robust potential, the parameter λ indicates the error beyond which a foreground pixel is considered the local optimal solution state and is fixed at $\lambda = 5$ for all images in our dataset. Other parameters for the model are dictated by the size of the image and the scale of the objects to be segmented; here, the sampling interval between neighbor pixel-states is $2W+1 = 41$ pixels. The parameter μ is set to 4.

C. Layer 2: Object Parts

Layer 2 identifies and labels object parts. The inputs to this layer are the segmented regions, $G_{rb}(v,e) = \cup_{i=1}^{N_Z}\Gamma_Z^i(v,e)$. These regions can consist of overlapping objects, which may possibly have different labels. In this layer, we apply a shape mechanism to break regions into object parts, so that if a single region consists of multiple overlapping objects, the object parts can be identified and labeled correctly. Currently we do this using the symmetry axis method [31] to extract the symmetry axis, or skeleton, of the shape [Fig. 3(d) and (e)].

Units: The set of segmented contours described by the graph $G_{rb}(v,e) = \cup_{i=1}^{N_Z}\Gamma_Z^i(v,e)$. For each region boundary $\Gamma_Z^i(v,e)$ we have an ordered list of pixels. This list can be parameterized in two ways: counter-clockwise with parameter s or clockwise with parameter t, yielding $\Gamma_1(s)$ and $\Gamma_2(t)$ [Fig. 6(a)].

Scores: The symmetry scoring is created for each pair of nodes in the graph $\Gamma_Z^i(v,e)$ based on a mirror symmetry or equivalent co-circularity measure [Fig. 6(b) and (c)]. One can define this measure as resulting from a match of a node in $\Gamma_1(s)$ with a node in $\Gamma_2(t)$, expressed as

$$\boldsymbol{S}(s,t) = |((x(s)-\tilde{x}(t)) \odot (\tau(s)+\tilde{\tau}(t))| + \left|((x(s)-\tilde{x}(t)) \odot (\tau(s)+\tilde{\tau}(t))^{\perp}\right| \quad (1)$$

where $x(s)$ and $\tilde{x}(t)$ are the node coordinates for the respective parameterizations, $\tau(s)$ and $\tilde{\tau}(t)$ are the corresponding unit tangent vectors, $\odot$ is the dot product of two vectors, and $\perp$ denotes the orthogonal vector. Node pairs with more symmetry receive lower scores, and perfectly symmetric elements have score $\boldsymbol{S}(s,t) = 0$.

Grouping: We apply the dynamic programming algorithm from [31] to find the best pairwise matching of boundary contour elements to obtain the optimal symmetry pairing, taking into account the score cost of (1) and a penalty (parameter J) for the number of object parts created. Fig. 3(d) and Fig. 7(a) show examples of optimal symmetry pairings, where the midpoint between two paired pixels on the object contour is the skeleton of the object, and bifurcations of the skeleton break the object into parts. In order to obtain a reasonable number of skeleton branches and, equivalently, object parts for each object, we need to automatically determine the best J for each segmented region.

Output Graph: For each region boundary $\Gamma_Z^i(v,e)$, the output of the grouping mechanism is a tree graph [e.g., Fig. 3(e)], where nodes represent object parts and bifurcations in the skeleton and edges connect adjacent object parts and bifurcations (nodes). Leaf nodes are object parts that contain one self-match (i.e., a boundary pixel that matches to itself and represents a skeleton endpoint). We refer to this tree graph as $GT_{op}(v,e)$, where a vertex v represents part of an object or a bifurcation and an edge connects adjacent object parts and bifurcations. The tree representation, $GT_{op}(v,e)$, is a natural representation for the object categorization process. Again, each of the N_Z region boundaries $\Gamma_Z^i(v,e)$ outputs one $GT_{op}(v,e)$.

Parameter: The J parameter should scale with the size of the region, as the cost of (1) will scale with the size of the region. To find the relationship between the optimal J and object size, we manually created supervised data by choosing the best value of J for objects of varying size (area/perimeter). The best value of J was selected as the value that gave rise to a skeleton

closest to a manually annotated skeleton. We observed a linear relationship between object size and optimal parameter value, which we use to determine the J for each object.

A problem with the symmetry axis method occurs when shapes have holes, i.e., more than one boundary contour, as illustrated in Fig. 7(a). This occurs quite often in our application, due to overlaps of objects and even simply due to deformation of a single object. Therefore, it was necessary to extend the work in [31] to create a symmetry axis solution for shapes with more than one boundary contour. Our insight was to select a point in each boundary contour and then apply a "topological surgery" to link the two shape boundaries, creating a final long unique boundary shape [Fig. 7(b)]. The method extends to multiple shape boundaries as follows: first apply it to a pair of shape boundaries, merging them into one shape boundary via the "topological surgery;" apply the method again by merging the resulting shape boundary with another boundary shape; repeat the procedure successively until all shape boundaries have been merged.

In order to select to which pair of points the "topological surgery" should be applied, a greedy method is used, selecting the closest pair of points (in Euclidean distance) between two boundary contours (Fig. 7(b), arrows). Once the merge of the two boundary shapes is complete, we apply the same dynamic programming algorithm as in [31] to the new unique boundary shape. Fig. 7(a) shows a typical result.

D. Layer 3: Object Labeling

Layer 3 addresses the problem of object categorization. Our object label categories are "embryo," "larva," or "adult worm," represented, respectively, as $\boldsymbol{L} = 1, 2, 3$. We first focus on the problem of labeling the object parts, since a given contiguous region may contain overlapping objects of different categories (thus, it may be necessary to output more than one object label per object, which will be resolved to final categories later).

Units: Nodes in the object part tree graph $GT_{op}(v, e)$.

Scores: We use an SVM learning method [21], [39] to train three SVMs using the "one to one" multiclass SVM procedure. Each SVM represents a preference between two alternative labels in one pair of possible labels: ("embryo," "larva"), ("embryo," "adult worm"), and ("larva," "adult worm").

Visual inspection of many object part examples suggested that the shape characteristics that distinguish developmental stages are their size, thickness, elongation, and the smoothness of the boundary contour (which is informative since embryos tend to form clumps with rough boundaries, as seen in Fig. 1). We extract 13 features for each object part, derived from a variety of shape characteristics: area, length of the symmetry axis, length of the boundary contour, total change in width (calculated as $\sum(\Delta_w)$, where Δ_w is the difference between the widths of consecutive matches along the skeleton of the object part), and the number of times that Δ_w changes sign (this value will be high for a "bumpy" boundary and low for a smooth circle). Changes in width were calculated at three different scales, considering "consecutive" skeleton matches at 1, 3, or 5 pixel intervals. We then use the "one to one" multiclass SVM procedure for each of the three label pairs.

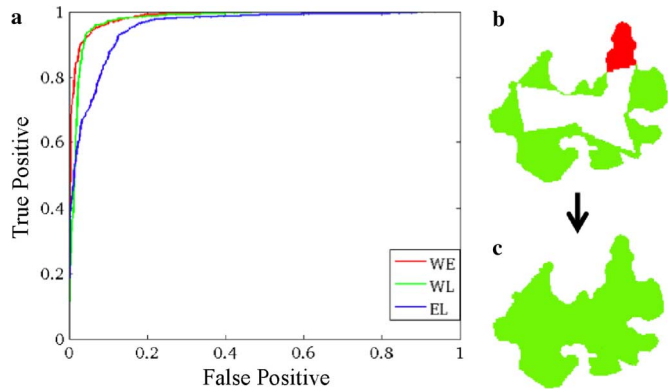


Fig. 8. Layer 3 object labeling by SVM and min-cut algorithms. (a) ROC curve of the multiclass SVM technique: "adult worm" versus "egg" SVM (WE, red); "adult worm" versus "larva" SVM (WL, green); "egg" versus "larva" SVM (EL, blue). (b) An object consisting of a clump of embryos. Object parts are assigned the "ajority vote" SVM label: "embryo" (green) or "adult" (red). The internal region (white) comprises bifurcations. Note that the SVM incorrectly labels part of the object. (c) The entire object is assigned a final category label ("embryo") after the min-cut algorithm corrects SVM labels and labels bifurcation areas.

To evaluate our SVMs, we performed 10-fold cross validation using manually labeled object parts in 100 images, including ∼100 000 object parts in total. The percent correct labels for each SVM were "adult worm" versus "larva," 93%; "adult worm" versus "embryo," 94%; and "embryo" versus "larva," 89%. Fig. 8(a) shows ROC curves for the three SVMs. We expect lower performance for the "embryo" versus "larva" SVM because it is more difficult to extract informative features from these objects, which are often very small and contain few pixels.

This procedure results in the construction of three new tree graphs for the three SVMs, $\{GT_p(v, e); \boldsymbol{p} = 1, 2, 3\}$, where $\boldsymbol{p} = 1$ ("egg" versus "larva"), $\boldsymbol{p} = 2$ ("egg" versus "adult worm") and $\boldsymbol{p} = 3$ ("larva" versus "adult worm"). The vertices and edges in these graphs are the same as in the object part graph $GT_{op}(v, e)$, except that each node is assigned a score output from the multiclass SVM procedure.

Grouping: When labeling object parts, errors in SVM label scores can occur; in particular, small object parts are more easily mislabeled due to their lower information content. However, the tree graph provides additional information about the proximity of object parts [Fig. 3(d) and (e)]. Our insight was to use this proximity in the tree graph to better resolve the scores: typically the nodes with scores that would give the wrong label had neighboring nodes with stronger scores that carried more discriminative power, and so could be used to help correct the score of their neighbor [Fig. 8(b) and (c)]. To accomplish this, we also make the graph $GT_{op}(v, e)$ a directed graph, where neighboring object parts that are connected are now connected by two directed edges.

In order to produce final scores that exploit this neighbor structure, for each SVM tree (indexed by $\boldsymbol{p} = 1, 2, 3$) we add a sink and a source node to the $GT_p(v, e)$ graph, with edges to each node in $GT_p(v, e)$ weighted according the SVM output, and we apply the min-cut algorithm [27], [28]. More precisely, given an SVM score on a label pair p, the edge weight from node n to the sink node is the $(\mathrm{SVM\ score} + \mathrm{C})$ and the edge weight

from node n to source node is $(-\text{SVM score} + \text{C})$, where C is a constant. The constant C ensures that all edge weights are non-negative (so C must be larger than the maximum magnitude of the SVM scores).

Bidirectional edges connecting two object parts (nodes) are assigned a weight μ based on the sizes of the neighboring nodes relative to the largest node in the graph (i.e., the largest object part in the object). For an edge connecting node n1 to node n2, $\mu = \text{A}_{\text{n1}}/\text{A}_{\text{nmax}}$, where A_{n1} is the area of node n1 and A_{nmax} is the area of the largest node. For an edge connecting node n2 to node n1, $\mu = \text{A}_{\text{n2}}/\text{A}_{\text{nmax}}$. Therefore, larger object parts exert greater influence on the labels of neighboring object parts. This grouping process gives a clear improvement in our results [Fig. 8(b) and (c)].

To assign a final label we use the Pareto optimal method, a simple "majority vote" procedure based on the output of the three graphs $\{GT_p(v,e); \boldsymbol{p} = 1,2,3\}$. For each object part (node), each graph provides one "winning" label: $\boldsymbol{L} = 1$ ("embryo"), $\boldsymbol{L} = 2$ ("larva"), or $\boldsymbol{L} = 3$ ("adult worm"). Since three graphs are associated with each object part, there are three winners per part. If one label wins twice, then this becomes the final label of the part. If all three labels receive one vote each, the final label is ambiguous, and we call it "no label."

Output Graph: The final output is presented by a unique graph $\boldsymbol{GTL(v,e)}$ that contains the final label assigned to each node v, including the possible label "no label." Note that each of the N_Z region boundaries $\Gamma_Z^i(v,e)$ outputs one $\boldsymbol{GTL(v,e)}$, i.e., the final output is one graph per object.

Parameters: The SVMs were trained using the radial basis function kernel. Cross-validation was used to select the best parameters for training. For all three SVMs, the optimal parameters were$\text{gamma} = 8$, $\text{epsilon} = 0.001$, and $\text{C} = 2^{15}$.

E. Layer 4: Counting Developmental Stages

Layer 4 computes counts of the different developmental stages using the size distributions of object labels provided by Layer 3. Our goal is to count the number of *C. elegans* larvae and embryos in each image, which form the basis for the quantitative phenotypic analysis of embryonic lethality in our HTP primary screens. The size of an embryo is expected to remain constant across images, and the expected number of pixels per embryo (which we call the embryo "unit size") can be empirically learned from manually labeled data. We can then estimate the number of embryos in each new image from the ratio (total area covered by embryos)/(embryo unit size).

In contrast, the sizes of adults and larvae can vary greatly, depending on the availability of food, the number of animals in the well, and the time the image is acquired. The final layer of our system therefore focuses primarily on counting the larvae and adult animals. In Layer 3, we trained SVMs to learn the labels "larva" and "adult" for each object part from a wide variety of images containing a range of animal sizes. This produces some overlap between the size distributions of adults and larvae, leading to the occasional misclassification of animals (see Fig. 9 for an example).

In Layer 4, we thus developed a method to obtain more accurate counts of adults and larvae by correcting for labeling errors using the size distributions of labeled objects in each image.

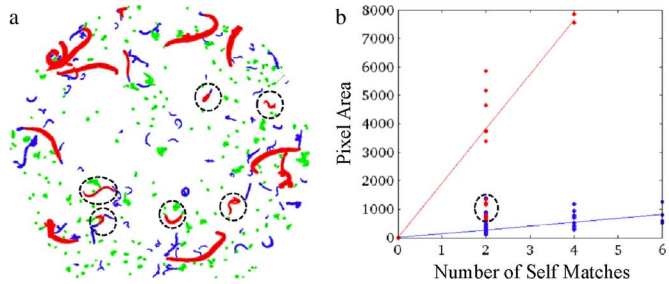


Fig. 9. Size distributions of labeled objects can be used to detect and correct mislabeled outliers. (a) Output from Layer 3, labeling of all objects: embryos (green), larvae (blue), adults (red). Larvae mislabeled as "adult" are highlighted (dashed circles). (b) Objects labeled as "larva" (blue points) or "adult" (red points) from (a), plotted by the area per self-match. Lines represent extracted size distributions from Layer 4 for larvae (blue) and adults (red). The red points falling within the distribution for larvae (dashed circle) correspond to the mislabeled larvae in (a) and are relabeled as "larva."

We know that the majority of images contain more larvae than adults and mislabeling events are infrequent, so we can correct for mislabeling between the two stages as follows: 1) utilizing their higher numbers, we first extract the size distribution of larvae for each image; 2) we then remove from the image all objects that fall within the variance of this distribution; and 3) we use the remaining objects to extract the size distribution of the adults. Specifically, we use the feature "animal area per self-match" to define the size distributions. We then relabel outliers with the label of the closest distribution.

After correcting for any mislabeling, we would like to convert the labeled objects (often consisting of multiple animals) into unit counts of individual animals. As described above (see Layer 2), self-matches in an object skeleton are associated to the tips of adults and larvae (Fig. 7(a) shows an example). Individual animals will have two self-matches, so we can use the number of self-matches to obtain a count of animals within each object.

Units: There are N_Z region boundaries $\Gamma_Z^i(v,e)$ and so N_Z labeled graphs $\boldsymbol{GTL(v,e)}$ with a label assigned to each node v, including the possible label "no label."

Scores: Given a region boundary $\Gamma_Z^i(v,e)$, the score for each label ($\boldsymbol{L} = 1$ or "larva," $\boldsymbol{L} = 3$ or "adult") is the total area of the nodes assigned that label divided by the number of self-matches in nodes assigned that label.

Grouping: If the image contains at least twice as many larvae as adult *C. elegans*, we first relabel the data with a grouping mechanism. A robust regression method is fit to the variable "area per self-match" for all data labeled "larva" and "adult" (Fig. 9). Due to the greater number of larvae than adults, the regression line will fit to the larva data and the adult worm data will be considered as outliers. At this point, we can relabel any data labeled "adult" that falls within the variance of the regression line for the larvae. We then remove all data labeled "larva" and perform robust regression a second time on the remaining data to extract the regression line for the adult animals (see Fig. 9). In this way, we are able to correct the labeling output for adults and larvae from Layer 3.

If the image does not contain more than twice as many larvae as adults, we do not have enough data to perform this grouping technique, so we do not attempt any relabeling.

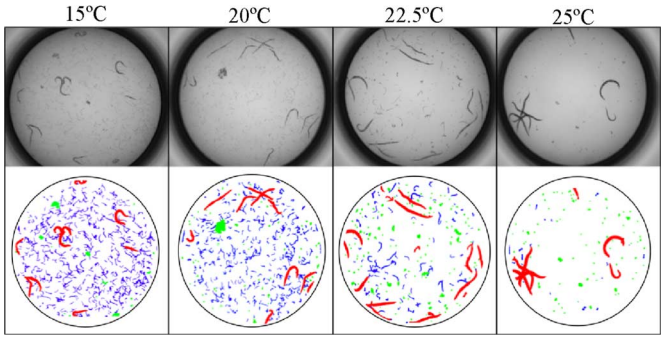


Fig. 10. Examples of results from DevStaR Layer 3. Shown are populations of a *C. elegans* strain carrying a *ts* allele of the essential gene *par-1*, at four different temperatures. With increasing temperature these animals exhibit higher levels of embryonic lethality, reflecting progressive loss of gene function (i.e., at 15 °C embryos hatch and produce larvae, whereas at 25 °C embryos die and no larvae hatch). Top: original images. Bottom: DevStaR Layer 3 output, in which each pixel is assigned a label: "adult" (red), "larva" (blue), "embryo" (green), or "background" (white).

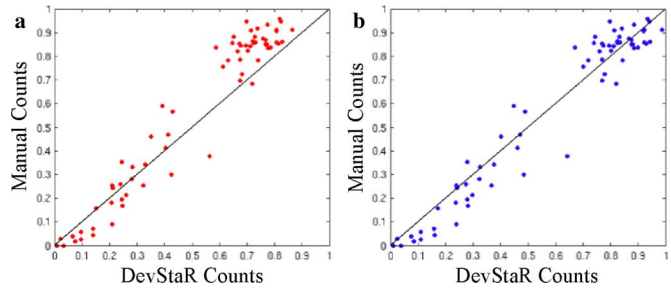


Fig. 11. Effect of bias correction on embryonic survival calculated using DevStaR versus manual counts. Each point represents a comparison of the survival rate based on DevStaR (x axis) and manual (y axis) counts for one image. Black lines represents expectation for perfect correlation; deviations from the line show differences in survival rates calculated by the two methods. (a) Results without bias correction, where sum of squared error = 17.4. Note consistent underestimation by DevStaR at high survival rates. (b) Results with bias correction (sum of squared error = 0.447).

Final Output: The final output of Layer 4 is the count of embryos, larvae, and adults per image. For embryos, the count estimate is found by dividing the total number of pixels labeled "embryo" by the embryo unit size (in our images, ∼70 pixels). To obtain count estimates for larvae and adult *C. elegans*, we divide the number of self-matches from all object parts with these respective labels by two. If an object has an odd number of self-matches, we round up the number of self-matches to the next even integer before dividing by two in order to account for occlusions.

IV. Results

To illustrate DevStaR performance, we applied it to measure the embryonic lethality of *C. elegans* strains carrying temperature-sensitive (*ts*) alleles of genes that are essential for embryonic development. Such *ts* alleles lead to reduced gene function, and therefore higher embryonic lethality, with increasing temperature. Fig. 10 shows examples of DevStaR pixel labeling results for a population of animals carrying a *ts* allele of *par-1* at four temperatures.

In this section, we present comparisons of the phenotype measured by DevStaR to the phenotype measured by manual scoring and show that DevStaR compares favorably at different levels of embryonic lethality, both for quantitative manual scoring of a small sample set and qualitative manual scoring of data from HTP screens. We evaluate each layer of DevStaR, with its associated parameters, to determine their individual contributions to performance. Finally, we show that the measurement error of DevStaR can be reduced by increasing either biological or image replicates.

A. Comparison With Quantitative Manual Scoring

We would like to compare embryonic survival calculated using DevStaR with the true survival rate. However, measuring the actual number of animals at each developmental stage in every well is nontrivial. Without access to a COPAS BIOSORT to obtain physical counts, manual annotation remains the most accurate alternative method available. We do this by labeling individual objects in an image of a well using an image-editing tool. For larvae, we draw one dot on each larva we observe in the image, and then count the dots. For embryos, we color all of the image pixels that represent embryos, and divide the total pixel area by an empirically determined constant for single embryo size (since, again, it is generally impractical to count individual embryos by eye due to clumping). This counting technique takes approximately 40 min per image.

Using this method of manual annotation, we counted the number of larvae and embryos in 70 images of populations of *C. elegans* exhibiting a range of embryonic survival, and we compared the results with DevStaR counts from the same images. Overall, we find that at higher embryonic survival, when growth is more robust and the wells become very crowded, DevStaR tends to underestimate embryonic survival [Fig. 11(a)]. Crowding contributes to overestimation of lethality in two ways: first, it creates many occlusions, which can lead to undercounting the number of larvae; and second, when many animals are in the well they can push any debris (such as eggshells) into clumps, which resemble—and therefore will typically be counted as—clumps of embryos, leading to overestimation of the number of embryos.

To compensate for this issue, we assume there exists a bias in DevStaR phenotype measurements that scales with the number of larvae, and that we can find this bias to correct survival estimates. We represent computed survival as

$$\mathrm{Surv}_{\mathrm{DevStar}} = \left(\frac{l}{l+e}\right)(1+\varepsilon)$$

where l is the count of the number of larvae and e is the count of the number of embryos obtained by DevStaR. We can then use the manual counts from the same images to findan that minimizes the error between the two

$$\Sigma_{\mathrm{Images}}(\mathrm{Surv}_{\mathrm{manual}} - \mathrm{Surv}_{\mathrm{DevStar}})^2 = \mathrm{error}(\varepsilon)$$

$$\text{so } \varepsilon = \frac{\Sigma_{\mathrm{Images}}\left(\mathrm{Surv}_{\mathrm{manual}}\left(\frac{l}{l+e}\right)\right)}{\Sigma_{\mathrm{Images}}\left(\frac{l}{l+e}\right)^2} - 1.$$

The bias correction [Fig. 11(b)] reduces the sum of the squared error based on manual and DevStaR counts from 17.14 to 0.447. The Pearson correlation coefficient is 0.97.

B. Performance Evaluation of Each DevStaR Layer

In order to evaluate the contribution that each layer and associated parameters make to DevStaR performance, we have compared results of each layer to manually annotated features in the same images. These features are well positions (for L0), object segmentation (for L1), or counts presented in the previous section (for L2, L3, and L4).

Layer 0: Attention (Area of Interest): To evaluate the performance of Layer 0 we manually annotated the well position for 13 images. We then compared DevStaR well location to the manually annotated location. The average precision and recall for DevStaR's identification of within-well pixels across the 13 images is 0.94 and 1.0, respectively, showing that DevStaR is able to locate the well accurately.

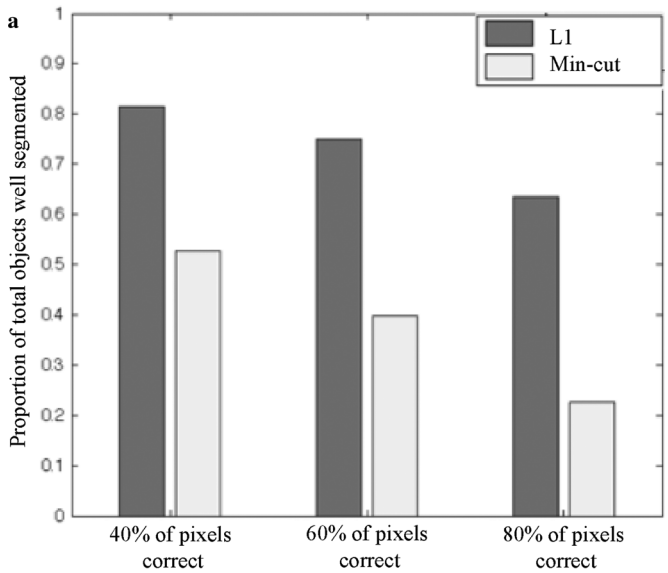
Layer 1:Background Removal and Object Segmentation: To evaluate Layer 1 we compared the performance of our new segmentation method with that of our previously published min-cut method [16]. Image segmentation presents two particular challenges. First, the method should accurately segment all images in the database without parameter adjustments, despite large variations in illumination and intensity. Second, to facilitate the labeling of developmental stages, segmentation accuracy must be as high as possible especially for the most difficult objects: small, thin objects whose boundary contour represents a large proportion of their total object area—in particular larvae, which are both small and of low contrast.

As an external standard, we manually segmented the objects in five images, totaling 1242 objects (connected components). We chose images with a wide variation in illumination and containing many larvae because these are the hardest images to segment (images are shown in supplementary Fig. 1). We perform manual segmentation by coloring individual pixels as foreground, which takes approximately 3–4 h per image. When comparing segmentation pixel-by-pixel, different methods vary slightly in boundary location detection due to small variations in intensity cut-off at object boundaries. These minor variations in object boundary do not change the shape structure so do not lead to labeling errors. Thus, when optimizing DevStaR we are more concerned with badly segmented objects with gross shape errors (missing large parts of the object, or segmenting noise) that can cause significant labeling errors.

We compared segmentation methods by counting the proportion of total objects across the five images that were "well-segmented" at different thresholds of segmentation accuracy, where accuracy is defined as the fraction of correctly labeled pixels within an object [Fig. 12(a)]. The results of these comparisons showed that DevStaR Layer 1 can more accurately segment greater numbers of objects without parameter adjustments over a wide range of image illumination and that it effectively segments even small, low contrast objects.

To demonstrate the performance of DevStaR layers L2, L3, and L4, we show the change in performance of DevStaR, on the manually-counted data presented in the previous section and in Fig. 11, as we adjust the parameters for these layers.

Layer 2: Object Parts: To evaluate the performance of the skeleton algorithm in breaking objects into parts and extracting features for the SVMs, we measured DevStaR performance



b

Layer	Parameter	Parameter value	Sum of squared error
2	J parameter	50	6.40
		100	3.90
		20,000	5.10
		auto	0.447
	Number of boundary contours	1	0.577
		>1	0.447
3	Min-cut mu parameter	0	0.526
		1	0.460
		3	0.496
		300	0.558
		auto	0.447
4	Count correction	No count	0.449
		count	0.447

Fig. 12. Performance analysis of DevStaR layers. (a) Layer 1: Performance of segmentation technique used in Layer 1 (L1) and min-cut method described in [16], with respect to manual annotation of five images (shown in Supplementary Fig. 1). The proportion of total objects that were "well-segmented" by each method was evaluated at three different thresholds of minimum segmentation accuracy, where accuracy is defined as the proportion of correctly labeled pixels within an object: (TP+TN)/(TP+FN). (b) Layers 2–4: Sum of squared error in DevStaR phenotype with respect to manually counted phenotype (using data shown in Fig. 11 for different parameter values in DevStaR Layers 2, 3, and 4.

using various fixed J parameters or using our new method to select J automatically, which depends on object size. As anticipated, automated J selection performed significantly better than any fixed J parameter [Fig. 12(b)]. This is because our images contain objects of significantly different scales (larvae and adults), and to obtain a good skeleton for both requires that we automatically determine J from the shape of the object.

We also evaluated performance with and without the capability of finding skeletons on shapes with more than one boundary contour (shapes with holes). Fig. 12(b) shows that applying the skeleton to multiple boundary contours improves

performance, though to a smaller extent than tuning the J parameter. This is because there are often few shapes with holes, whereas an incorrect J parameter usually affects many objects in the image.

Layer 3: Object Labeling: The performance of the SVMs used in Layer 3 is shown by the ROC curves in Fig. 8(c). The min-cut grouping technique used by DevStaR attempts to correct any SVM mislabeling events by allowing the influence of neighboring labels. The μ parameter determines the strength of neighbor influence, and we evaluated DevStaR performance at different values of μ in comparison with manual annotation [Fig. 12(b)]. For fixed μ, the best performing value is $\mu = 1$, where the sum of the squared error = 0.46 (versus a value of 0.526 when $\mu = 0$, i.e., no neighbor influence). Automatic selection of μ based on the sizes of neighboring parts (the method used by DevStaR and described above) achieves better performance than any fixed $\sum$ (sum of squared error = 0.447). These results demonstrate that neighbor influence does affect DevStaR performance and that using the size of neighboring parts to determine the strength of the influence gives the greatest performance improvement. The μ parameter has a smaller effect than the J parameter because of the already high performance of the SVMs; thus label correction might only affect a small proportion of the objects in the image, leading to small changes in measured phenotype. However, in cases where an object that is a large clump of eggs is incorrectly labeled, label correction can have a huge effect on the resulting phenotype, and this is where the min-cut grouping with neighbor influence can most prevent phenotype measurement errors.

Layer 4: Counting Developmental Stages: The reduction in error with respect to manual annotation that is achieved by the label correction mechanism of Layer 4 is relatively small [Fig. 12(b)]. This is due both to infrequent mislabeling of larvae as adult worms and to the small proportion of images where larvae have grown larger than usual. Large larvae typically result from greater food availability (e.g., due to few adults and high embryonic lethality) or delayed recording of images (allowing more growth time). Thus, while label correction may have a small overall effect on error, for certain images it will be an important step to achieving an accurate phenotype.

C. Comparison With Semi-Quantitative Manual Scoring

In order to validate the utility of DevStaR in the context of HTP applications, we compared DevStaR developmental stage counts with semi-quantitative manual scoring of image data acquired in an HTP RNAi screen. The data set used for comparison comprised 30 000 manually scored images of *C. elegans* populations spanning a large range of embryonic survival. For manual scoring, each image was assigned two integers ranging from 0 to 9 (one for larvae and one for embryos), representing zero animals at the respective developmental stage to the maximum number of animals at that stage observed across all the images. This graded scoring scale was selected, after testing larger and smaller ranges, to provide the most intuitive level of subjective discrimination by a human. The manual scores represent $\sim$200 h of work and were all assigned by the same individual. A static reference panel showing representative images for all

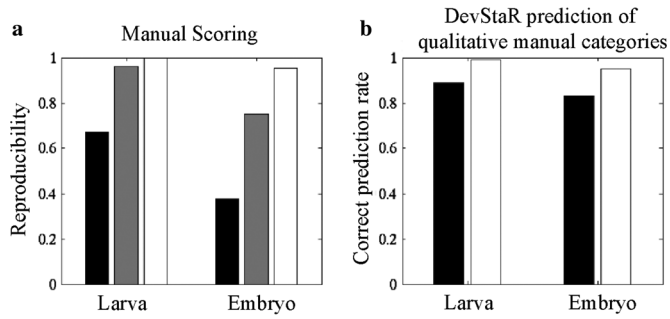


Fig. 13. Comparison of DevStaR results with qualitative manual scoring. (a) Reproducibility of manual estimates of survival in two rounds of blind scoring of 200 images. Each image was assigned one of 10 qualitative categories representing 0%–100% survival. Bar heights represent the proportion of images with "consistent" category labels, as defined by three different methods: an exact label match, e.g., {3,3} (black bars); an exact or first-neighbor category match, e.g., {8,9} (gray bars); or an exact, first-neighbor, or second-neighbor (e.g., {5,7}) category match (white bars). Reproducibility increases as the stringency for a "consistent" match is relaxed. (b) DevStaR performance compared with qualitative manual scoring on 30 000 images, performed as outlined above. DevStaR "correct" prediction rate on ten-fold cross validation, defined as an exact or first-neighbor overlap for larvae, and as an exact, first-neighbor, or second-neighbor overlap for embryos. Bar heights represent the "correct" prediction rate using one image (black) or three images (white) for training and prediction. The prediction rate by DevStaR using three images is on par with the reproducibility of manual scoring at the same stringency.

embryo and larva score categories was visible for all analyses (Supplementary Fig. 2).

We emphasize that, in practice, it is not feasible for a human to count precisely the developmental stages in HTP experiments, since it takes on average around 40 min for a human to count the objects in one image. Instead these qualitative scores represent estimates of embryo and larva counts, which, after training, take 15–30 s per image. Depending on the experimental setup and level of detail required, other scoring systems may be used for semi-quantitative estimates.

In order to obtain meaningful comparisons between DevStaR and semi-quantitative manual scoring, it is therefore important to consider only the resolution at which the manual scoring is reproducible. We first evaluated how consistently a human assigns categories to each developmental stage, based on blind scoring of 200 images in duplicate [Fig. 13(a)]. We found that the proportion of images assigned the same category in the two rounds of scoring (the "match rate") was low: 37% for embryos and 66% for larvae. These results indicate that a human cannot reliably discriminate between 10 categories for these developmental stages. Next, we relaxed the criteria for a "consistent" match by allowing neighboring categories to count as a match. For example, a first-neighbor match would allow an image assigned 3 in one round of scoring to be considered as consistent if it received 2, 3, or 4 in the second round. If we allow matching between both first- and second-neighbor categories, then for example a 3 would count as a consistent match with any value spanning 1–5. With these relaxed criteria, we were able to achieve above 90% correct match rate for larvae by allowing first-neighbor matches and for embryos by allowing first- or second-neighbor matches. These results show us the resolution at which semi-quantitative manual scoring of larvae and embryos is reliable using this 10 point system.

In order to compare the results from DevStaR with this semi-quantitative scoring scheme, we first performed 10-fold cross-validation to learn the best correspondence between DevStaR counts and manual category assignments. Specifically, we used 90% of the 30 000 scored images to learn the distribution of DevStaR counts for each of the 10 manually defined categories for each developmental stage (Supplementary Fig. 3). With the remaining 10% of the data, we calculated the probability that the DevStaR counts for each image could represent a sample from the distribution for each category, and we assigned to the image the category with the highest probability.

Using the relaxed criteria described above that define the resolution of manual scoring (first-neighbor matches for larvae and first/second-neighbor matches for embryos), we then compared the predicted and human categories [Fig. 13(b)] and found that DevStaR achieved a correct rate of 89% for larvae and 83% for embryos. We also tested using the average DevStaR count over three images (with the same manual score) to learn and predict the categories—the rationale being that biological experiments are often performed with replicates, so usually more than one image is available to estimate the underlying phenotype. Using the average of three images (and again using the relaxed criteria that define the resolution of manual scoring), we were able to obtain a correct match rate greater than 95% for both embryos and larvae [Fig. 13(b)]. These results show that DevStaR can be used to score HTP screens in place of manual scoring without a loss in accuracy.

D. Measurement Error of DevStaR

We evaluated the reproducibility and resolution of DevStaR results by examining the variation in measurements for technical or biological replicates. To this end, we acquired a total of 3840 images: 10 image replicates for every well in individual 96-well plates of populations carrying the *mbk-2 ts* allele, acquired at each of four temperatures (thus, 960 images at each temperature). The resulting data are similar to the examples shown in Fig. 10.

We first computed embryonic survival at each temperature using the 10 replicate images per well for each of the 96 wells as our estimate of the survival in the population, and then used this population estimate to analyze the sampling error among technical and biological replicates. For technical replicates, we analyzed multiple snapshots of a single well: since the animals inside the well are moving, each picture is different, but the actual number of each developmental stage inside the well remains constant. We evaluated our measurements as a function of the number of independent snapshots per well, considering sample sizes ranging from 1–10 images, and used the Student's t distribution to obtain a 95% confidence interval around the sample mean for the true mean of embryonic survival (Fig. 14, blue line). For biological replicates, we performed the same exercise but compared single snapshots of different wells from the same plate (Fig. 14, red line). From these results we observe that the confidence interval shrinks as the sample size increases for both image and biological replicates. Thus, the measurement error of an underlying phenotype decreases as either biological or technical replicates are increased.

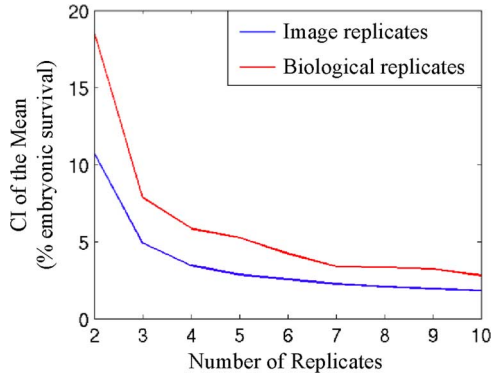


Fig. 14. Confidence interval of the mean from DevStaR measurements. CI decreases as the number of either image replicates (blue) or biological replicates (red) is increased.

We can also use these data to measure the resolution that can be obtained with DevStaR in measuring embryonic lethality. Here, for example, we expect to achieve a measurement error of $\pm 4\%$ embryonic lethality using four image replicates in an experiment. Using four biological replicates, the confidence interval is $\pm 6\%$ embryonic lethality, and this value comprises both the measurement error of the entire experimental system and natural biological variation. This suggests that the difference in embryonic lethality between two populations of *C. elegans* must be greater than 6% in order to be distinguishable as two distinct phenotypes that are not attributable to measurement error or biological variation. Moreover, this resolution exceeds that which can be obtained by even semi-quantitative manual scoring.

V. Conclusion

DevStaR can rapidly and accurately segment, label and count developmental stages in populations of *C. elegans*, and therefore is a useful tool to measure embryonic lethality in mixed populations in a high-throughput context. In our lab, we have so far used DevStaR to score over four million images from multiple genetic interaction screens, producing results with much greater resolution and speed than would otherwise be possible by manual analysis. DevStaR thus enables high-throughput screens in *C. elegans* that were previously impractical or simply unfeasible, opening up many new opportunities for genetic and chemical genomic screening.

Acknowledgment

The authors would like to thank R. Cole for insightful discussions on extending the skeleton to shapes with multiple boundary contours, M. Rockman for conversations on manual quantification of image data, and D. Sontag for discussions on MRF algorithms. The authors would also like to thank the reviewers whose comments and suggestions helped them greatly improve their paper in terms of clarity and organization.

References

[1] S. Brenner, "The genetics of Caenorhabditis elegans," *Genetics*, vol. 77, pp. 71–94, 1974.

[2] Consortium, "Genome sequence of the nematode C. elegans: A platform for investigating biology," *Science*, vol. 282, pp. 2012–2018, 1998.

[3] K. C. Gunsalus and F. Piano, "RNAi as a tool to study cell biology: Building the genome-phenome bridge," *Curr. Opin. Cell Biol.*, vol. 17, pp. 3–8, 2005.

[4] K. L. B. Sönnichsen, A. Walsh, P. Marschall, B. Neumann, M. Brehm, A. M. Alleaume, J. Artelt, P. Bettencourt, E. Cassin, M. Hewitson, C. Holz, M. Khan, S. Lazik, C. Martin, B. Nitzsche, M. Ruer, J. Stamford, M. Winzi, R. Heinkel, M. Röder, J. Finell, H. Häntsch, S. J. Jones, M. Jones, F. Piano, K. C. Gunsalus, K. Oegema, P. Gönczy, A. Coulson, A. A. Hyman, and C. J. Echeverri, "Full-genome RNAi profiling of early embryogenesis in Caenohabditis elegans," *Nature*, vol. 434, pp. 462–469, 2005.

[5] P. G. Cipriani and F. Piano, "RNAi methods and screening: RNAi based high-throughput genetic interaction screening," *Methods Cell Biol.*, vol. 106, pp. 89–111, 2011.

[6] F. Zanella, J. B. Lorens, and W. Link, "High content screening: Seeing is believing," *Trends Biotechnol.*, vol. 28, pp. 237–249, 2010.

[7] Y. Ohya *et al.*, "High-dimensional and large-scale phenotyping of yeast mutants," *Proc. Nat. Acad. Sci. USA*, vol. 102, pp. 19015–19020, 2005.

[8] T. R. Jones *et al.*, "Scoring diverse cellular morphologies in image based screens with iterative feedback and machine learning," *Proc. Nat. Acad. Sci. USA*, vol. 106, pp. 1826–1831, 2009.

[9] C. Wahlby, L. Kamentsky, Z. H. Liu, T. Riklin-Raviv, A. L. Conery, E. J. O'Rourke, K. L. Sokolnicki, O. Visvikis, V. Ljosa, J. E. Irazoqui, P. Golland, G. Ruvkun, F. M. Ausubel, and A. E. Carpenter, "An image analysis toolbox for high-throughput C. elegans assays," *Nature Methods*, vol. 9, pp. 714–6, 2012.

[10] W. Geng *et al.*, "Automatic tracking, feature extraction and classification of C elegans phenotypes," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 10, pp. 1811–1820, Oct. 2004.

[11] G. D. Tsibidis and N. Tavernarakis, "A computational tool for analyzing nematode locomotion," *BMC Neurosci.*, vol. 8, 2007.

[12] C. J. Cronin, Z. Feng, and W. R. Schafer, "Automated imaging of C. elegans behavior," *Methods Molecular Biol.*, vol. 351, pp. 241–251, 2006.

[13] K. M. Huang, P. Cosman, and W. R. Schafer, "Automated detection and analysis of foraging behavior in Caenorhabditis elegans," *J. Neurosci. Methods*, vol. 171, pp. 153–164, 2008.

[14] Z. Bao, J. I. Murray, T. Boyle, S. L. Ooi, M. J. Sandel, and R. H. Waterston, "Automated cell lineage tracing in Caenorhabditis elegans," *Proc. Nat. Acad. Sci. USA*, vol. 103, pp. 2707–2722, 2006.

[15] A. Kimura and S. Onami, "Computer simulations and image processing reveal length-dependent pulling force as the primary mechanism for C. elegans male pronuclear migration," *Develop. Cell*, vol. 8, pp. 765–775, 2005.

[16] A. G. White, P. G. Cipriani, H. L. Kao, B. Lees, D. Geiger, E. Sontag, K. C. Gunsalus, and F. Piano, "Rapid and accurate developmental stage recognition of C. elegans from high-throughput image data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 3089–3096.

[17] A. T. Erik, B. Sudderth, W. T. Freeman, and A. S. Willsky, "Describing visual scenes using transformed objects and parts," *Int. J. Comput. Vis.*, vol. 77, pp. 291–330, 2008.

[18] R. Fergus, P. Perona, and A. Zisserman, "Weakly supervised scale-invariant learning of models for visual recognition," *Int. J. Comput. Vis.*, vol. 71, pp. 273–303, 2007.

[19] Y. LeCunn, K. Kavukvuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proc. Int. Symp. Circuits Syst.*, 2010.

[20] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vis.*, vol. 2004, pp. 137–154, 2004.

[21] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

[22] R. E. Schapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, pp. 197–227, 1990.

[23] G. Hinton, "Learning multiple layers of representation," *Trends Cognit. Sci.*, vol. 11, pp. 428–434, 2007.

[24] W. T. Freeman and E. H. Adelsonm, "The design and use of steerable filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 9, pp. 891–906, Sep. 1991.

[25] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 7, pp. 674–693, Jul. 1989.

[26] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York: Wiley, 2001.

[27] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 11, pp. 1222–1239, Nov. 2001.

[28] H. Ishikawa and D. Geiger, "Segmentation by grouping junctions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 1998, pp. 125–131.

[29] U. Montanari, "On the optimal detection of curves in noisy pictures," *Commun. ACM*, vol. 14, pp. 335–345, 1971.

[30] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–895, Aug. 2000.

[31] T. Liu, D. Geiger, and R. Kohn, "Representation and self-similarity of shapes," in *Proc. Int. Conf. Comput. Vis.*, 1998.

[32] D. L. Geiger, Tyng-Luh, and R. V. Kohn, "Representation and self similarity of shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 1, pp. 86–99, Jan. 2003.

[33] F. M. K. Van Leemput, D. Vandermeulen, and P. Suetens, "Automated model-based bias field correction of MR images of the brain," *IEEE Trans. Med. Imag.*, vol. 18, no. 10, pp. 885–896, Oct. 1999.

[34] D. Tarlow, D. Batra, P. Kohli, and V. Kolmogorov, "Dynamic tree-block coordinate descent," presented at the Int. Conf. Mach. Learn., Bellevue, WA, 2011.

[35] J. Yarkony, C. Fowlkes, and A. Ihler, "Covering trees and lower-bounds on quadratic assignment," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 887–894.

[36] A. Globerson and T. Jaakkola, "Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations," *Adv. Neural Inf. Process. Syst.*, 2007.

[37] D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss, "Tightening LP Relaxations for MAP using message passing," in *Conf. Uncertain. Artif. Intell.*, Helsinki, Finland, 2008.

[38] D. Sontag, D. K. Choe, and Y. Li, "Efficiently searching for frustrated cycles in MAP inference," in *28th Conf. Uncertain. Artif. Intell.*, Catalina Island, CA, 2012.

[39] C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowl. Discov.*, vol. 2, pp. 127–167, 1998.