Tracking cell lineages in 3D by incremental deep learning

Ko Sugawara[1,2,*], Cagri Cevrim[1,2] and Michalis Averof[1,2,*]

[1] Institut de Génomique Fonctionnelle de Lyon (IGFL), École Normale Supérieure de Lyon, 32 avenue Tony Garnier, 69007 Lyon, France

[2] Centre National de la Recherche Scientifique (CNRS), France

[*] Authors for correspondence (ko.sugawara@ens-lyon.fr, michalis.averof@ens-lyon.fr)

Abstract: Deep learning is emerging as a powerful approach for bioimage analysis, but its wider use is limited by the scarcity of annotated data for training. We present ELEPHANT, an interactive platform for cell tracking in 4D that seamlessly integrates annotation, deep learning, and proofreading. ELEPHANT's user interface supports cycles of incremental learning starting from sparse annotations, yielding accurate, user-validated cell lineages with a modest investment in time and effort.

Source code: https://github.com/elephant-track/

Main text

Recent progress in deep learning has led to significant advances in bioimage analysis [1-4]. As deep learning is data-driven, it is adaptable to a variety of datasets once an appropriate model architecture is selected and trained with adequate data [1]. In spite of its powerful performance, deep learning remains challenging for non-experts to utilize, for three reasons. First, pre-trained models can be inadequate for new tasks and the preparation of new training data is laborious. Because the quality and quantity of the training data are crucial for the performance of deep learning, users must invest significant time and effort in annotation at the start of the project [1]. Second, an interactive user interface for deep learning, especially in the context of cell tracking, is lacking. Third, deep learning applications are often limited by accessibility to computing power (high-end GPU). We have addressed these challenges by establishing ELEPHANT (Efficient learning using sparse human annotations for nuclear tracking), an interactive web-friendly platform for cell tracking, which seamlessly integrates manual annotation with deep learning and proofreading of the results. ELEPHANT implements two algorithms optimized for incremental deep learning using sparse annotations, one for detecting nuclei in 3D and a second for linking these nuclei across timepoints in 4D image datasets. Incremental learning allows models to be trained in a stepwise fashion on a given dataset, starting from sparse annotations that are incrementally enriched by human proofreading, leading to a rapid increase in performance (Figure 1a). ELEPHANT is implemented as an extension of Mastodon (https://github.com/mastodon-sc/mastodon), an open-source framework for large-scale tracking based on Fiji [5]. It implements a client-server architecture, in which the server provides a deep learning environment equipped with sufficient GPU (Supplementary Figure 1).

ELEPHANT employs the tracking-by-detection paradigm [6], which involves initially the *detection* of nuclei in 3D and subsequently their *linking* over successive timepoints to generate tracks. In both steps, the nuclei are represented as ellipsoids, using the data model of Mastodon (Figure 1b, c). In the detection phase, voxels are labelled as *background*, *nucleus center* or *nucleus periphery*, or left unlabelled (Figure 1b, top right). The *nucleus center* and *nucleus periphery* labels are generated by the annotation of nuclei, and the *background* can be annotated either manually or by intensity thresholding. Sparse annotations (e.g. of a few nuclei in a single timepoint) are sufficient to start training. A U-Net convolutional neural network (U-Net CNN; [7,8], Supplementary Figure 2) is then trained on these labels (ignoring the unlabelled voxels) to generate voxel-wise probability maps for *background*, *nucleus center*, or *nucleus periphery*, across the entire image dataset (Figure 1b, bottom right). Post-processing on these probability maps yields predictions of nuclei which are available for visual inspection and proofreading (validation or rejection of each predicted nucleus) by the user (Figure 1b, bottom left). Human-computer interaction is facilitated by color coding of the annotated nuclei (as true positive, false positive, true negative, false negative, or unevaluated, see Supplementary Figure 3) based on the proofreading. The cycles of training and prediction are rapid because only a small amount of training data are added each time (in the order of seconds, see Supplementary Table 1). As a result, users can enrich the annotations by

proofreading the output almost simultaneously, enabling incremental training of the model in an efficient manner.

In the linking phase, we found that nearest neighbour approaches for tracking nuclei over time [9] perform poorly in challenging datasets when the cells are dividing; hence we turned to optical flow modeling to improve linking [10-12]. A second U-Net CNN, optimized for optical flow estimation (Supplementary Figure 4), is trained on manually generated/validated links between nuclei in successive timepoints (Figure 1c, top left). Unlabelled voxels are ignored, hence training can be performed on sparse linking annotations. The flow model is used to generate voxel-wise 3D flow maps, representing predicted x, y and z displacements over time (Figure 1c, bottom right), which are then combined with nearest neighbour linking to predict links between the detected nuclei (see Methods). Users proofread the linking results to finalize the tracks and to update the labels for the next iteration of training (Figure 1c, bottom left).

We evaluated the performance of ELEPHANT using two types of 4D confocal microscopy datasets in which nuclei were visualized by fluorescent markers: the first type of dataset captures the embryonic development of *Caenorhabditis elegans* (CE datasets), which has been used in previous studies to benchmark tracking methods [13,14], and the second type captures limb regeneration in *Parhyale hawaiensis* (PH dataset, imaging adapted from [15]), which presents greater challenges for image analysis (see below, Supplementary Video 1). For both types of dataset, we find that fewer than ten annotated nuclei are sufficient to initiate a virtuous cycle of training, prediction and proofreading, which efficiently yields cell tracks and validated cell lineages in highly dynamic tissues.

Interactive cycles of manual annotation, deep learning and proofreading on ELEPHANT reduce the time required to detect and validate nuclei (Figure 2a). On the CE1 dataset, a complete cell lineage was built over 195 timepoints, from scratch, using ELEPHANT's semi-automated workflow (Figure 2c). The detection model was trained incrementally starting from sparse annotations (four nuclei) on the first timepoint. On this dataset, linking could be performed using the nearest neighbor algorithm (without flow modeling) and manual proofreading. In this way, we were able to annotate in less than 8 hours a total of 23,829 nuclei (across 195 timepoints), of which ~2% were manually annotated (483 nuclei) and the remaining nuclei were collected by validating predictions of the deep-learning model.

Although ELEPHANT works efficiently without prior training, cell tracking can be accelerated by starting from models trained on image data with similar characteristics. To illustrate this, we used nuclear annotations in a separate dataset, CE2, to train a model for detection, which was then applied to CE1. This pre-trained model allowed us to detect nuclei in CE1 much more rapidly and effortlessly than with an untrained model (Figure 2a, blue versus orange curves). For benchmarking, the detection and linkage models trained with the annotations from the CE1 and CE2 lineage trees were then tested on unseen datasets with similar characteristics (without proofreading), as part of the Cell Tracking Challenge [6,14]. In this test, our models with assistance of flow-based interpolation (see Methods) outperformed

state-of-the-art tracking algorithms [16,17] in detection (DET) and tracking (TRA) metrics (Figure 2b).

The PH dataset presents greater challenges for image analysis, such as larger variations in the shape, intensity, and distribution of nuclei, lower temporal resolution, and more noise (Supplementary Figure 5). ELEPHANT has allowed us to grapple with these issues by supporting the continued training of the models through visual feedback from the user (annotation of missed nuclei, validation and rejection of predictions). Using ELEPHANT, we annotated and validated over 260,000 nuclei in this dataset, across 504 timepoints spanning 168 hours of imaging.

We observed that the conventional nearest neighbor approach was inadequate for linking in the PH dataset, resulting in many errors in the lineage trees (Figure 2d). This is likely due to the lower temporal resolution in this dataset (20 minutes in PH, versus 1-2 minutes in CE) and the fact that daughter nuclei often show large displacements at the end of mitosis. We trained optical flow using 1,162 validated links collected from 10 timepoints (including 18 links for 9 cell divisions). These sparse annotations were sufficient to generate 3D optical flow predictions for the entire dataset (Supplementary Video 2), which significantly improved the linking performance (Figure 2d, Supplementary Table 2): the number of false positive and false negative links decreased by ~57% (from 2,093 to 905) and ~32% (from 1,991 to 1349), respectively, among a total of 259,071 links.

By applying ELEPHANT's human-in-the-loop semi-automated workflow, we succeeded in reconstructing 109 complete and fully-validated cell lineage trees encompassing the duration of leg regeneration in *Parhyale*, each lineage spanning a period of ~1 week (504 timepoints, Supplementary Figure 6). Using analysis and visualization modules implemented in Mastodon and ELEPHANT, we could capture the distribution of cell divisions across time and space (Figure 2e) and produce a fate map of the regenerating leg of *Parhyale* (Figure 2f). This analysis, which would have required several months of manual annotation, was achieved in ~1 month of interactive cell tracking in ELEPHANT, without prior training. Applying the best performing models to new data could improve tracking efficiency even further.

Methods

Image datasets

The PH dataset (dataset li13) was obtained by imaging a regenerating T4 leg of the crustacean *Parhyale hawaiensis*, based on the method described by [15] (Supplementary Video 1). The imaging was carried out on a transgenic animal carrying the *Mi(3xP3>DsRed; PhHS>H2B-mRFPRuby)* construct [18], in which nuclear-localised mRFPRuby fluorescent protein is expressed in all cells following heat-shock. The leg was amputated at the distal end of the carpus. Following the amputation, continuous live imaging over a period of 1 week was performed on a Zeiss LSM 800 confocal microscope equipped with a Plan-Apochromat 20x/0.8 M27 objective (Zeiss 420650-9901-000), in a temperature control chamber set to 26˚C. Heat-shocks (45 minutes at 37˚C) were applied 24 hours prior to the amputation, and 65

and 138 hours post-amputation. Every 20 minutes we recorded a stack of 11 optical sections, with a z step of 2.48 microns. Voxel size (in xyz) was 0.31 x 0.31 x 2.48 microns.

The CE1 and CE2 datasets were from [13], obtained via the Cell Tracking Challenge [14] (datasets Fluo-N3DH-CE).

ELEPHANT platform architecture

ELEPHANT implements a client-server architecture (Supplementary Figure 1), which can be set up on the same computer or on multiple connected computers. This architecture brings flexibility: allowing the client to run Mastodon (implemented in Java) while the deep learning module is implemented separately using Python, and releasing the client computer from the requirements of high GPU needed to implement deep learning. The client side is implemented by extending Mastodon, a framework for cell tracking built upon the SciJava ecosystem (https://scijava.org/) and is available as a Fiji [5] plugin. Combining the BigDataViewer [19] with an efficient memory access strategy (https://github.com/mastodon-sc/mastodon/blob/master/doc/trackmate-graph.pdf), Mastodon enables fast and responsive user interaction even for very large datasets. ELEPHANT leverages the functionalities provided by Mastodon, including the functions for manual annotation of nuclei, and extends them by implementing modules for deep learning-based algorithms.

The server side is built using an integrated system of a deep learning library (PyTorch [20]), tools for tensor computing and image processing (Numpy [21], Scipy [22], Scikit Image [23]) and web technologies (Nginx, uWSGI, Flask). The client and the server communicate by Hypertext Transfer Protocol (HTTP) and JavaScript Object Notation (JSON). To reduce the amount of data exchanged between the client and the server, the image data is duplicated and stored in an appropriate format on each side. An in-memory data structure (Redis) is used to organize the priorities of the HTTP requests sent by the client. A message queue (RabbitMQ) is used to notify the client that the model is updated during training. The client software is available as a standalone Java executable packaged with Mastodon and other dependencies (https://github.com/elephant-track/elephant-client). The server environment is provided as a Docker container to ensure easy and reproducible deployment (https://github.com/elephant-track/elephant-server).

Computer setup and specifications

In this study, we set up the client and the server on the same desktop computer (Dell Alienware Aurora R6) with the following specifications: Intel Core i7-8700K CPU @ 3.70GHz, Ubuntu 18.04, 4x16 GB DDR4 2666 MHz RAM, NVIDIA GeForce GTX 1080 Ti 11 GB GDDR5X (used for deep learning), NVIDIA GeForce GTX 1650 4 GB GDDR5, 256 GB SSD and 2 TB HDD. System requirements for the client and the server are summarized in the user manual (https://elephant-track.github.io/).

Dataset preparation

Images were loaded in the BigDataViewer (BDV, [19]) format on the client software. The CE1 and CE2 datasets were converted to the BDV format using the BigDataViewer Fiji plugin (https://imagej.net/BigDataViewer)without any preprocessing. Because the PH dataset

showed non-negligible variations in intensity during long-term imaging, the original 16-bit images were intensity normalized per timepoint before conversion to the BDV format; intensity values were re-scaled so that the minimum and maximum values at each timepoint become 0 and 65535, respectively. The PH dataset also showed 3D drifts due to heat-shocks. The xy drifts were corrected using an extended version of image alignment tool [24] working as an ImageJ [25] plugin, where the maximum intensity projection images were used to estimate the xy displacements, subsequently applied to the whole image stack (https://github.com/elephant-track/align-slices3d), and the z drifts were corrected manually by visual inspection using Fiji.

On the server, images, annotation labels and outputs were stored in the Zarr format, allowing fast read/write access to subsets of image data using chunk arrays. At the beginning of the analysis, these data were prepared using a custom Python script that converts the original image data from HDF5 to Zarr and creates empty Zarr files for storing annotation labels and outputs (https://github.com/elephant-track/elephant-server).

Algorithm for detection

Detection of nuclei relies on three components: (i) a U-Net CNN that outputs probability maps for *nucleus center*, *nucleus periphery*, and *background*, (ii) a post-processing workflow that extracts *nucleus center* voxels from the probability maps, (iii) a module that reconstructs nuclei instances as ellipsoids. We designed a variation of 3D U-Net [8] as illustrated in Supplementary Figure 2. In both encoder and decoder paths, repeated sets of 3D convolution, ReLU activation [26] and Group Normalization [27] are employed. Max pooling in 3D is used for successive downsampling in the encoder path, in each step reducing the size to half the input size (in case of anisotropy, maintaining the z dimension until the image becomes nearly isotropic). Conversely, in the decoder path, upsampling with nearest-neighbor interpolation is applied to make the dimensions the same as in the corresponding intermediate layers in the encoder path. As a result, we built a CNN with 5,887,011 trainable parameters. The weights are initialized with the Kaiming fan-in algorithm [28] and the biases are initialized to zero for each convolution layer. For each group normalization layer, the number of groups is set as the smallest value between 32 and the number of output channels, and the weights and biases are respectively initialized to one and zero. When starting to train from scratch, the CNN is trained using the cropped out 3D volumes from the original image prior to training with annotations. In this prior training phase, a loss function L_{prior} is used that penalizes the addition of the following two mean absolute differences (MADs): (i) *nucleus center* probabilities c_i and the [0, 1] normalized intensity of the original image y_i, (ii) *background* probabilities b_i and the [0, 1] normalized intensity of the intensity-inverted image $1 - y_i$, where i stands for the voxel index of an input volume with n voxels $i \in V := \{1, 2, \ldots, n\}$.

$$L_{prior} = \frac{1}{n}\sum_{i=1}^{n}|y_i - c_i| + \frac{1}{n}\sum_{i=1}^{n}|(1 - y_i) - b_i|$$

The prior training is performed on three cropped out 3D volumes generated from the 4D datasets, where the timepoints are randomly picked, and the volumes are randomly

cropped with random scaling in the range (0.8, 1.2). The training is iterated for three epochs with decreasing learning rates (0.01, 0.001, and 0.0001, in this order) with the Adam optimizer [29]. The prior training can be completed in ~20 seconds for each dataset.

Training with sparse annotations is performed in the following steps. First, the client application extracts the timepoint, 3D coordinates and covariances representing ellipsoids of all the annotated nuclei in the specified time range. Subsequently, these data, combined with user-specified parameters for training, are embedded in JSON and sent to the server in an HTTP request. On the server side, training labels are generated from the received information by rendering *nucleus center*, *nucleus periphery*, *background* and unlabeled voxels with distinct values. The *background* labels are generated either by explicit manual annotation or intensity thresholding, resulting in the label images as shown in Figure 1b. To render ellipsoids in the anisotropic dimension, we extended the draw module in the scikit-image library [23] (https://github.com/elephant-track/elephant-server). Training of the CNN is performed using the image volumes as input and the generated labels as target with a loss function L_{vclass} that consists of three terms: (i) a class-weighted negative log-likelihood (NLL) loss, (ii) a term computed as 1 minus the dice coefficient for the *nucleus center* voxels, and (iii) a term that penalizes the roughness of the *nucleus center* areas. We used the empirically-defined class weights $\mathbf{wc}$ for the NLL loss: *nucleus center* = 10, *nucleus periphery* = 10, *background* = 1; the unlabelled voxels are ignored. The first two terms accept different weights for the true annotations wt (i.e. true positive and true negative) and the false annotations wf (i.e. false positive and false negative). The third term is defined as the MAD between the voxel-wise probabilities for *nucleus center* and its smoothed representations, which are calculated by the Gaussian filter with downsampling ($Down$) and upsampling (Up). Let i stand for the voxel index of an input volume with n voxels $i \in V := \{1, 2, \ldots, n\}$, x_i for the input voxel value, $\mathbf{h}_i$ for the output from the CNN before the last activation layer for the three classes, $y_i \in Y := \{1, 2, 3\}$ for the voxel class label (1: *nucleus center*, 2: *nucleus periphery*, 3: *background*, respectively), and $z_i \in Z := \{true, false, unlabeled\}$ for the voxel annotation label. We define the following subsets: the voxel index with true labels $T = \{i \mid i \in V,\ z_i = true\}$, with false labels $F = \{i \mid i \in V,\ z_i = false\}$, and the *nucleus center* $C = \{i \mid i \in V,\ y_i = 1\}$. In the calculation of the L_{dice}, a constant $\epsilon = 0.000001$ is used to prevent zero division. Using these components and the empirically-defined weights for each loss term ($\alpha = 1,\ \beta = 5,\ \gamma = 1$), we defined the L_{vclass} as below.

$$L_{vclass} = \alpha L_{nll} + \beta L_{dice} + \gamma L_{smooth}$$

$$L_{nll} = wt \frac{\sum_{i \in T} (NLL(\mathbf{h_i}, y_i) \cdot wc[y_i])}{\sum_{i \in T} wc[y_i]} + wf \frac{\sum_{i \in F} (NLL(\mathbf{h_i}, y_i) \cdot wc[y_i])}{\sum_{i \in F} wc[y_i]}$$

$$L_{dice} = wt \left(1 - \frac{2 \sum_{i \in T} (Prob(\mathbf{h_i}, 1) \cdot Onehot(y_i, 1))}{max\left(\sum_{i \in T} \left((Prob(\mathbf{h_i}, 1))^2 + (Onehot(y_i, 1))^2 \right), \epsilon \right)} \right)$$

$$+ wf \left(1 - \frac{2 \sum_{i \in F} (Prob(\mathbf{h_i}, 1) \cdot Onehot(y_i, 1))}{max\left(\sum_{i \in F} \left((Prob(\mathbf{h_i}, 1))^2 + (Onehot(y_i, 1))^2 \right), \epsilon \right)} \right)$$

$$L_{smooth} = \frac{1}{n}\sum_{i \in V} |Prob(\mathbf{h_i}, 1) - Up(Down(Prob(\mathbf{h_i}, 1)))|$$
$$Prob(\mathbf{h}, c) = \frac{exp(\mathbf{h}[c])}{\sum_{j=1}^{3} exp(\mathbf{h}[j])}$$
$$NLL(\mathbf{h}, y) = -\log(Prob(\mathbf{h}, y))$$
$$Onehot(y_i, c) = \begin{cases} 0 & (y_i \neq c) \\ 1 & (y_i = c) \end{cases}$$

Training of the CNN is performed on the image volumes generated from the 4D datasets, where the volumes are randomly cropped with/without random scaling, random contrast and random rotation, which are specified at runtime. There are two modes for training: (i) an interactive mode that trains a model incrementally, as the annotations are updated, and (ii) a batch mode that trains a model with a fixed set of annotations. In the interactive training mode, sparse annotations in a given timepoint are used to generate crops of image and label volumes, with which training is performed using the Adam optimizer with a learning rate specified by the user. In the batch training mode, a set of crops of image and label volumes per timepoint is generated each iteration, with which training is performed for a number of epochs specified by the user (ranging from 1 to 1,000) using the Adam optimizer with the specified learning rates. In the prediction phase, the input volume can be cropped into several blocks with smaller size than the original size to make the input data compatible with available GPU memory. To stitch the output blocks together, the overlapping regions are seamlessly blended by weighted linear blending.

As post-processing for the CNN output, voxel-wise probabilities for *nucleus center* class are denoised by subtracting edges of *background* class that are calculated with the Gaussian filter and the Prewitt operation for each z-slice. After denoising, the voxels with *nucleus center* probabilities greater than a user defined value are thresholded and extracted as connected components, which are then represented as ellipsoids (from their central moments). These ellipsoids representing the *nucleus center* regions are enlarged so that they cover the original nucleus size (without excluding its periphery), where the ellipsoids with radii smaller than r_{min} are removed and the radii are clamped to r_{max} specified by the user, generating a list of center positions and covariances that can be used to reconstruct the nuclei. On the client application, the detection results are converted to Mastodon spots and rendered on the BDV view, where the existing and predicted nuclei are tagged based on its status: labeled as positive and predicted (true positive), labeled as positive and not predicted (false negative), labeled as negative and not predicted (true negative), labeled as negative and predicted (false positive) and newly predicted (non-validated). If more than one nucleus is predicted within a user-specified threshold, the one with human annotation is given priority, followed by the one with the largest volume.

Algorithm for linking

Linking of nuclei relies on two components: (i) estimation of the positions of nuclei at the previous timepoint by optical flow estimation using deep learning, which is skipped in the case of the nearest neighbor algorithm without flow support, (ii) association of nuclei based

on the nearest neighbor algorithm. We designed a variation of 3D U-Net for flow estimation as illustrated in Supplementary Figure 4. In the encoder path, the residual blocks [30] with 3D convolution and LeakyReLU [31] activation are applied, in which the outputs are divided by two after the sum operation to keep the consistency of the scale of values. In the decoder path, repeated sets of 3D convolution and LeakyReLU activation are employed. Downsampling and upsampling are applied as described for the detection model. Tanh activation is used as a final activation layer. As a result, we built a CNN with 5,928,051 trainable parameters. The weights and biases for convolution layers are initialized as described for the detection model. Training of the flow model with sparse annotations is performed in a similar way as for the detection model. First, on the client application, for each annotated link, which connects the source and target nuclei, the following information gets extracted: the timepoint, the backward displacements in each of the three dimensions, and the properties of the target nucleus (3D coordinates and covariances). Subsequently, these data, combined with parameters for training, are embedded in JSON and sent to the server in an HTTP request. On the server side, flow labels are generated from the received information by rendering backward displacements for each target nucleus in each of three dimensions, where the displacements are scaled to fit the range (-1, 1). In this study, we used fixed scaling factors (1/80, 1/80, 1/10) for each dimension, but they can be customized to the target dataset. Foreground masks are generated at the same time to ignore unlabelled voxels during loss calculation. Ellipsoid rendering is performed as described for the detection training. Training of the CNN for flow estimation is performed using the two consecutive image volumes (I_{t-1}, I_t) as input, and the generated label as target. A loss function L_{flow} is defined with the following three terms; (i) a dimension-weighted MAD between the CNN outputs and the flow labels, (ii) a term computed as 1 minus the structural similarity (SSIM) [32] of I_{t-1} and $\tilde{I}_t$, where the estimated flow is applied to I_t [33], (iii) a term penalizing the roughness of the CNN outputs. Let i stand for the voxel index of an input volume with n voxels $i \in V := \{1, 2, \ldots, n\}$, $\boldsymbol{x}_i$ for the input voxel value, $\hat{\mathbf{y}}$ for the output of the CNN, $\mathbf{y}$ for the flow label, $m \in M \subset V$ for the index of the annotated voxels, $d \in D := \{0, 1, 2\}$ for the dimension index for three dimensions and $\mathbf{wd}$ for the dimension weights. In the SSIM calculation, we defined a function $Gauss$ as a 3D Gaussian filter with the window size (7, 7, 3) and standard deviation of 1.5. Using these components and the empirically-defined weights for each loss term ($\alpha = 1$, $\beta = 0.0001$, $\gamma = 0.0001$), we defined the L_{flow} as below.

$$L_{flow} = \alpha L_{mad} + \beta L_{ssim} + \gamma L_{smooth}$$

$$L_{mad} = \frac{1}{n} \sum_{d \in D} wd_d \sum_{m \in M} |\mathbf{y}_{md} - \hat{\mathbf{y}}_{md}|$$

$$L_{ssim} = 1 - SSIM\left(I_{t-1}, \hat{I}_t\right)$$

$$L_{smooth} = \frac{1}{3n} \sum_{i \in V} \sum_{d \in D} |\hat{\mathbf{y}}_{id} - Up(Down(\hat{\mathbf{y}}_{id}))|$$

$$\mu_{I_1} = Gauss(I_1),\ \mu_{I_2} = Gauss(I_2)$$

$$\sigma_{I_1}^2 = Gauss\left(I_1^2\right) - \mu_{I_1}^2,\ \sigma_{I_2}^2 = Gauss\left(I_2^2\right) - \mu_{I_2}^2,\ \sigma_{I_1 I_2} = Gauss(I_1 I_2) - \mu_{I_1}\mu_{I_2}$$

$$SSIM(I_1, I_2) = \frac{\left(2\mu_{I_1}\mu_{I_2} + C_1\right)\left(2\sigma_{I_1 I_2} + C_2\right)}{\left(\mu_{I_1}^2 + \mu_{I_2}^2 + C_1\right)\left(\sigma_{I_1}^2 + \sigma_{I_2}^2 + C_2\right)}$$

, where $C_1 = 0.0001$ and $C_2 = 0.0009$.

The training is performed on the image volumes generated from the 4D datasets, where the sets of two consecutive images and corresponding flow labels are randomly cropped with/without random scaling and random rotation, which are specified at runtime. The training is performed for a fixed number of epochs using the Adam optimizer and with learning rates specified by the user, generating a set of images and labels for each timepoint in each epoch. The CNN outputs are rescaled to the original physical scale and used to calculate the estimated coordinate of each nucleus center at the previous timepoint. Let $K \subset V$ stands for a subset of voxel index of a nucleus and $\mathbf{p}$ for its center coordinate. Using the output of the CNN $\hat{\mathbf{y}}$ and the scaling factor s, the estimated coordinate at the previous timepoint $\hat{\mathbf{p}}$ is calculated.

$$\hat{\mathbf{p}} = \mathbf{p} + \frac{s}{n(K)} \sum_{k \in K} \hat{\mathbf{y}}_k$$

These estimated coordinates are subsequently used to find the parent of the nucleus at the previous timepoint by the nearest neighbor algorithm (a similar concept was introduced for 2D phase contrast microscopy data; [34,35]). The pairs with a distance smaller than d_{search} are considered as link candidates, where the closer the Euclidean distance between the two points, the higher their priority of being the correct link. Each nucleus accepts either one or two links, determined by the estimated displacements and actual distances. Briefly, given that a single nucleus has two possible links, it can accept both if at least one of the estimated displacements is larger than the threshold d_{disp} or both distances are smaller than the threshold d_{dist}. In this study, we used ad hoc thresholds $d_{disp} = 1.0$ and $d_{dist} = 1.0$. If there are competing links beyond the allowed maximum of two links, the links with smaller d_{disp} are adopted and the remaining nucleus looks for the next closest nucleus up to N_{max} neighbors. The links are generated by repeating the above procedure until all the nuclei get linked or the iteration count reaches to five. We optionally implement an interpolation algorithm, in which each orphan nucleus tries to find its source up to T_{max} timepoints back and is linked with a nucleus at the estimated coordinate based on the flow prediction, interpolating the points in between.

Detection and tracking in the CE datasets

On the CE1 and CE2 datasets, training of detection and flow models was performed with volumes of 384 x 384 x 16 voxels that were generated by preprocessing with random scaling in the range (0.5, 2) and random cropping. For training of a detection model, preprocessing with random contrast in the range (0.5, 1) was also applied. In the label generation step, the center ratio was set to 0.3 and the background threshold was set to 0.1 and 1 (i.e. all voxels without manual annotations are background), for the interactive mode and the batch training mode, respectively. In the interactive training of detection models, 10 labelled cropped out volumes were generated per iteration, with which training was performed using the Adam optimizer with a learning rate between 5×10^{-5} and 5×10^{-6}. In the batch training of detection models, training was performed for 100 epochs using the Adam optimizer with learning rates of 5×10^{-5}. In the training of a flow model, training was performed for 100 epochs using the

Adam optimizer with learning rates of $5x10^{-5}$ for the first 50 epochs and $5x10^{-6}$ for the last 50 epochs. wt and wf were set to 1 and 5, respectively, and $\mathbf{wd}$ was set to (1/3, 1/3, 1/3). In the prediction phase, the input volumes were cropped into 2x2x2 blocks with size (544, 384, 28) for CE1 or (544, 384, 24) for CE2, and stitched together to reconstruct the whole image of (708, 512, 35) for CE1 or (712, 512, 31) for CE2. In the preprocessing of the prediction for detection, we corrected the uneven background levels across the z-slices by shifting the slice-wise median value to the volume-wise median value. In the postprocessing of the prediction for detection, a threshold for the *nucleus center* probabilities were set to 0.3, and r_{min} and r_{max} were set to 1 and 3, respectively. In the nearest-neighbor linking with/without flow prediction, d_{search} was set to 5 μm and N_{max} was set to 3. In the results submitted to the CTC organizer (Figure 2b), the linking was performed by the nearest-neighbor linking with flow support and an optional interpolation module, where T_{max} was set to 5.

Detection and tracking in the PH dataset

On the PH dataset, training of detection and flow models was performed with volumes of 384 x 384 x 12 voxels generated by preprocessing with random rotation in the range of +/− 180 degrees and random cropping. For training a detection model, preprocessing with random contrast in the range (0.5, 1) was also applied. In the label generation step, the center ratio was set to 0.3, and the background threshold was set to 0.03. In the interactive training of a detection model, 10 crops of image and label volumes were generated per iteration, with which training was performed using the Adam optimizer with a learning rate between $5x10^{-5}$ and $5x10^{-6}$. In the batch training of a detection model, training was performed for 100 epochs using the Adam optimizer with learning rates of $5x10^{-5}$. In the training of a flow model, training was performed for 100 epochs using the Adam optimizer with learning rates of $5x10^{-5}$ for the first 50 epochs and $5x10^{-6}$ for the last 50 epochs. wt and wf were set to 1 and 3, respectively, and $\mathbf{wd}$ was set to (1, 1, 8). In the prediction phase, the input volumes were fed into the CNNs without cropping or further preprocessing. In the postprocessing of the prediction for detection, a threshold for the *nucleus center* probabilities were set to 0.3, and r_{min} and r_{max} were set to 1 and 3, respectively. In the nearest-neighbor linking with/without flow prediction, d_{search} was set to 5 μm and N_{max} was set to 3.

Analysis of CE and PH datasets

On the CE1 and CE2 datasets, the detection and link annotations were made starting from timepoint 0 and proceeding forward until timepoints 194 (CE1) and 189 (CE2), respectively. In the CE1 dataset, the detection was made from scratch, based on manual annotation and incremental training, and the linking was performed by the nearest neighbor algorithm without flow prediction. After completing annotation from timepoint 0 to 194 on the CE1 dataset, the detection and flow models were trained by the batch mode with the fully-labeled annotations. In the CE2 dataset, the detection was performed in a similar way as for CE1, by extending the model trained with CE1, and the linking was performed by the nearest neighbor algorithm with flow support using the pre-trained model followed by proofreading.

Incremental training of the detection model was performed when there were annotations from nuclei that were not properly predicted.

On the PH dataset, the annotations were made by iterating the semi-automated workflow. In general, the nuclei with high signal-to-noise ratio (SNR) were annotated early, while the nuclei with low SNR were annotated in a later phase. The detection model was updated frequently to fit the characteristics of each region and timepoint being annotated, while the flow model was updated less frequently. The CE1 dataset was used to evaluate the speed of detection and validation (Figure 2a). All workflows started at timepoint 0 and proceeded forward in time, adding and/or validating all the nuclei found in each timepoint. To evaluate the manual workflow, we annotated nuclei using hotkeys that facilitate the annotation of a given nucleus at successive timepoints. To evaluate the ELEPHANT from scratch workflow, we performed prediction with the latest model, followed by proofreading, including add, modify or delete operations, and incremental training. At each timepoint, the model was updated with the new annotations added manually or by proofreading. To evaluate the ELEPHANT pre-trained workflow, we performed predictions with a model trained on the CE2 dataset, followed by proofreading without additional training. The numbers of validated nuclei associated with time were counted from the log data. We measured the counts over 30 minutes after the start of each workflow and plotted them in Figure 2a.

To compare the linking performances (Figure 2d), we trained the flow model with 1,162 validated links, including 18 links corresponding to 9 cell divisions, from 108 lineage trees collected between timepoints 150 and 159. It took around 30 hours to train the flow model from scratch using these links. Starting from a pre-trained model, the training time can be decreased to a few minutes, providing a major increase in speed compared with training from scratch (Supplementary Table 2).

The results shown in Figure 2e and Figure 2f were generated based on the tracking results with 260,600 validated nuclei and 259,071 validated links. In the analysis for Figure 2e, nuclei were categorised as dividing or non-dividing depending on whether the lineages to which they belong contain at least one cell division or not during the period of cell proliferation (timepoints 100 to 350). Nuclei that did not meet these criteria were left undetermined. For Figure 2f, the complete lineages of 109 nuclei were tracked through the entire duration of the recording, from 0 to 167 hours post-amputation, with no missing links.

Evaluation of cell tracking performance

We submitted our results and executable software to the Cell Tracking Challenge organizers, who evaluated our algorithm's performance, validated its reproducibility using the executable software that we submitted, and provided us with the scores. The details of the detection accuracy (DET), tracking accuracy (TRA), and segmentation accuracy (SEG) metrics can be found in the original paper [36] and the website (http://celltrackingchallenge.net/evaluation-methodology/). Briefly, the DET score evaluates how many *split*, *delete* and *add* operations are required to achieve the ground truth starting from the predicted nuclei, reflecting the accuracy of detection; the TRA score evaluates how many *split*, *delete* and *add* operations for nuclei, and *delete*, *add* and *alter the semantics* operations for links are required to reconstruct

the ground truth lineage trees from the predicted lineage trees, reflecting the accuracy of linking; the SEG score evaluates the overlap of the detected ellipsoids with fully segmented nuclei, reflecting the precision of nucleus segmentation. All three scores range from 0 (poorest) to 1 (best).

Code availability

The source code for the ELEPHANT client is available at https://github.com/elephant-track/elephant-client, for the ELEPHANT server at https://github.com/elephant-track/elephant-server, and for the Align Slices 3D+t extension ImageJ plugin at https://github.com/elephant-track/align-slices3d. The user manual for ELEPHANT is available at https://elephant-track.github.io/.

Acknowledgements

We are grateful to Anna Kreshuk and Constantin Pape for training in machine learning, to Jean-Yves Tinevez and Tobias Pietzsch for support in developing ELEPHANT as a Mastodon plugin, to the NEUBIAS community for feedback on the software, to the Cell Tracking Challenge organizers for support in our submission to the challenge, and to Sebastien Tosi for extensive feedback on the manuscript. We also thank Jan Funke, Carsten Wolff, Martin Weigert, Jean-Yves Tinevez, Philipp Keller, Irepan Salvador-Martínez, Severine Urdy, and Mathilde Paris for comments on the manuscript. This research was supported by the European Research Council, under the European Union Horizon 2020 programme, grant ERC-2015-AdG #694918; CC was supported by a doctoral fellowship from Boehringer Ingelheim Fonds.

Author contributions

KS and MA conceived the project; KS designed and produced the software, and evaluated its performance; CC acquired the image dataset on regenerating limbs; KS and CC generated the annotations and tested the software; KS and MA wrote the manuscript.

References

1. Moen, E. *et al.* Deep learning for cellular image analysis. *Nat Methods* **16,** 1233–1246 (2019).
2. Ouyang, W., Aristov, A., Lelek, M., Hao, X. & Zimmer, C. Deep learning massively accelerates super-resolution localization microscopy. *Nat Biotechnol* **36,** 460–468 (2018).
3. Weigert, M. *et al.* Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nat Methods* **15,** 1090–1097 (2018).
4. Caicedo, J. C. *et al.* Nucleus segmentation across imaging experiments: the 2018 Data Science Bowl. *Nat Methods* **16,** 1247–1253 (2019).
5. Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis. *Nat Methods* **9,** 676–682 (2012).
6. Maška, M. *et al.* A benchmark for comparison of cell tracking algorithms. *Bioinformatics* **30,** 1609–1617 (2014).
7. Ronneberger, O., Fischer, P. & Brox, T. in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* 234–241 (Springer, Cham, 2015). doi:10.1007/978-3-319-24574-4_28
8. Cicek, O., Abdulkadir, A., Lienkamp, S. S., Brox, T. & Ronneberger, O. in *Medical Image Computing and Computer-Assisted Intervention - MICCAI* (eds. Ourselin, S., Joskowicz, L., Sabuncu, M. R., Unal, G. & Wells, W.) 1–9 (2016). doi:10.1007/978-3-319-46723-8
9. Crocker, J. C. & Grier, D. G. Methods of Digital Video Microscopy for Colloidal Studies. *Journal of Colloid and Interface Science* **179,** 298–310 (1996).
10. Horn, B. & Schunck, B. G. Determining Optical Flow Artificial Intelligence Vol. 17. *Artificial Intelligence* **17,** 185–203 (1981).
11. Lucas, B. D. & Kanade, T. An iterative image registration technique with an application to stereo vision. *Proceedings of the international joint conference on Artificial intelligence* **2,** 674–679 (1981).
12. Amat, F., Myers, E. W. & Keller, P. J. Fast and robust optical flow for time-lapse microscopy using super-voxels. *Bioinformatics* **29,** 373–380 (2012).
13. Murray, J. I. *et al.* Automated analysis of embryonic gene expression with cellular resolution in C. elegans. *Nat Methods* **5,** 703–709 (2008).
14. Ulman, V. *et al.* An objective comparison of cell-tracking algorithms. *Nat Methods* **14,** 1141–1152 (2017).
15. Alwes, F., Enjolras, C. & Averof, M. Live imaging reveals the progenitors and cell dynamics of limb regeneration. *Elife* **5,** 73 (2016).
16. Scherr, T., Löffler, K., Böhland, M. & Mikut, R. Cell Segmentation and Tracking using CNN-Based Distance Predictions and a Graph-Based Matching Strategy. *arXiv.org* (2020).
17. Magnusson, K. E. G., Jaldén, J., Gilbert, P. M. & Blau, H. M. Global linking of cell tracks using the Viterbi algorithm. *IEEE Trans Med Imaging* **34,** 911–929 (2015).
18. Wolff, C. *et al.* Multi-view light-sheet imaging and tracking with the MaMuT software reveals the cell lineage of a direct developing arthropod limb. *Elife* **7,** 375 (2018).
19. Pietzsch, T., Saalfeld, S., Preibisch, S. & Tomancak, P. BigDataViewer: visualization and processing for large image data sets. *Nat Methods* **12,** 481–483 (2015).

20. Paszke, A. *et al.* PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* **32,** (2019).
21. Harris, C. R. *et al.* Array programming with NumPy. *Nature Publishing Group* **585,** 357–362 (2020).
22. Virtanen, P. *et al.* SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat Methods* **17,** 261–272 (2020).
23. van der Walt, S. *et al.* scikit-image: Image processing in Python. *Peer J* **cs.MS,** e453 (2014).
24. Tseng, Q. *et al.* A new micropatterning method of soft substrates reveals that different tumorigenic signals can promote or reduce cell contraction levels. *Lab Chip* **11,** 2231–2240 (2011).
25. Schneider, C. A., Rasband, W. S. & Eliceiri, K. W. NIH Image to ImageJ: 25 years of image analysis. *Nat Methods* **9,** 671–675 (2012).
26. Nair, V. & Hinton, G. E. Rectified linear units improve Restricted Boltzmann machines. *ICML 2010 - Proceedings, 27th International Conference on Machine Learning* (2010).
27. Wu, Y. & He, K. Group Normalization. *International Journal of Computer Vision* **128,** (2020).
28. He, K., Zhang, X., Ren, S. & Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *IEEE International Conference on Computer Vision* 1026–1034 (2015). doi:doi:10.1109/ICCV.2015.123
29. Kingma, D. P. & Ba, J. L. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015).
30. He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. *arXiv.org* (2015).
31. Maas, A. L., Hannun, A. Y. & Ng, A. Y. Rectifier nonlinearities improve neural network acoustic models. *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing* (2013).
32. Wang, Z., Bovik, A. C., Sheikh, H. R. & Simoncelli, E. P. Image quality assessment: from error visibility to structural similarity. *IEEE Trans Image Process* **13,** 600–612 (2004).
33. Ilg, E. *et al.* Flownet 2.0: Evolution of optical flow estimation with deep networks. in 1647–1655 (2017).
34. Hayashida, J. & Bise, R. Cell tracking with deep learning for cell detection and motion estimation in low-frame-rate. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2019).
35. Hayashida, J., Nishimura, K. & Bise, R. MPM: Joint representation of motion and position map for cell tracking. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2020). doi:10.1109/CVPR42600.2020.00388
36. Matula, P. *et al.* Cell Tracking Accuracy Measurement Based on Comparison of Acyclic Oriented Graphs. *PLoS ONE* **10,** e0144959 (2015).

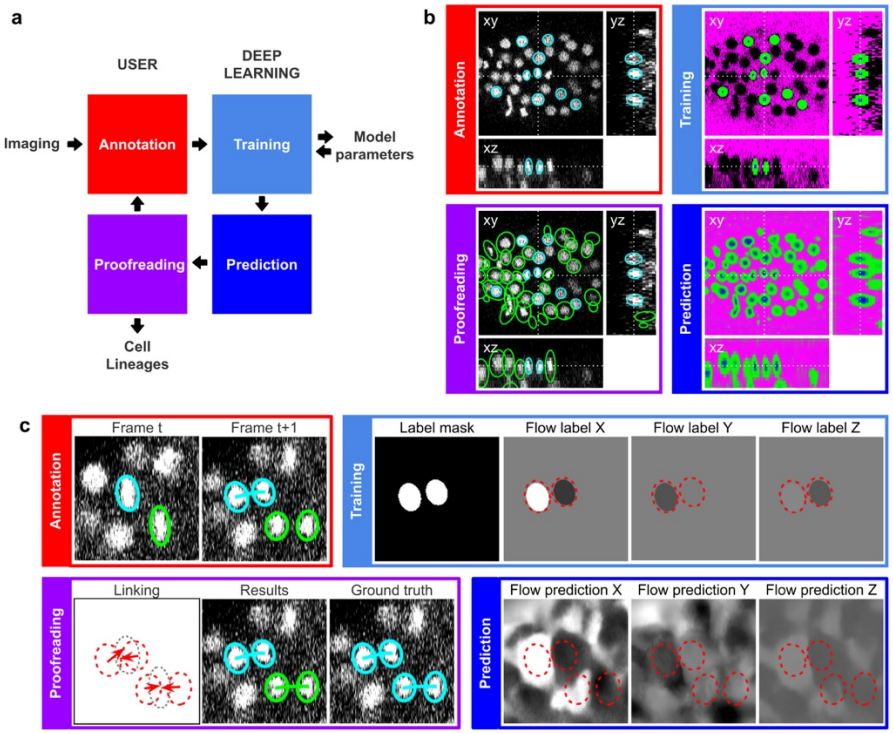
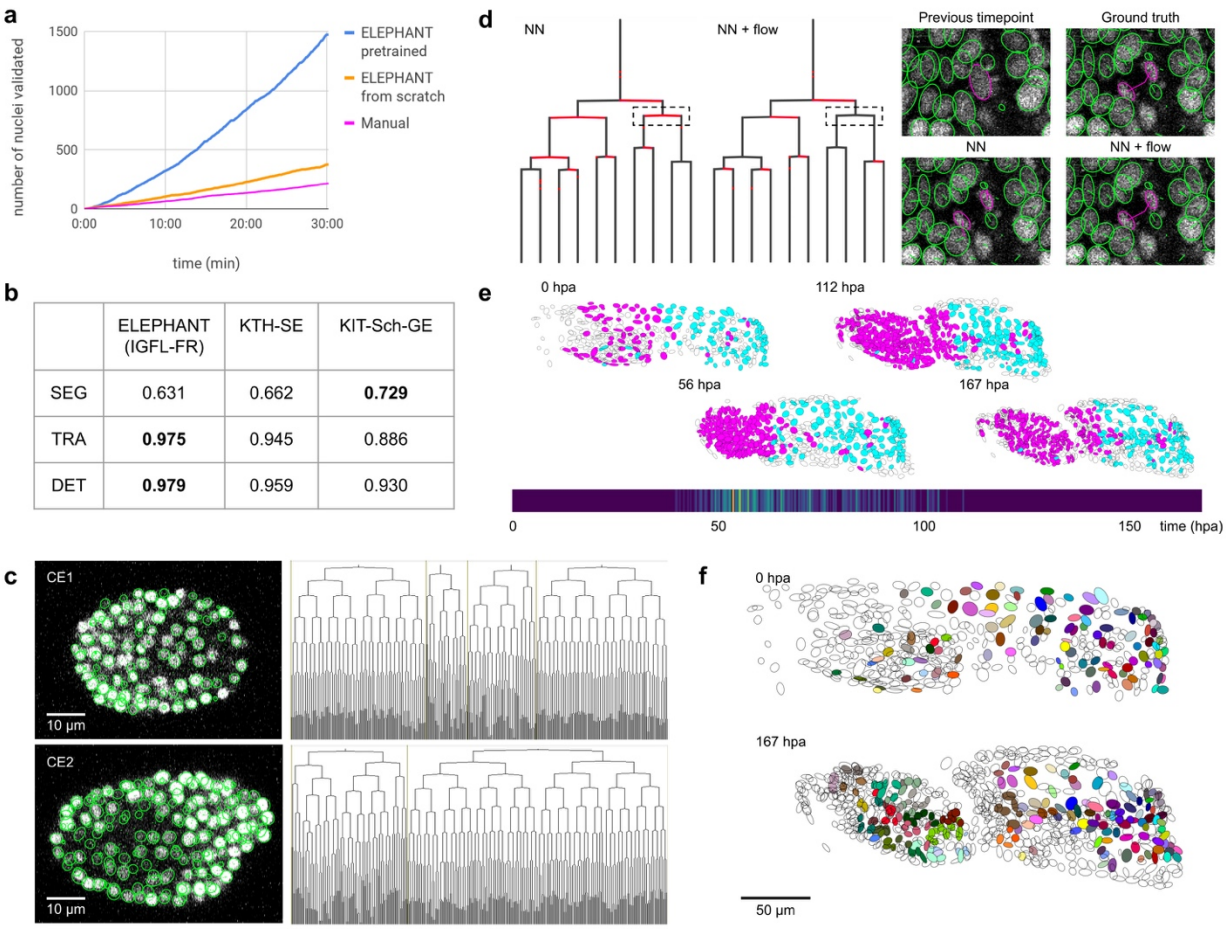


Figure 1. Overview of ELEPHANT

a, Schematic illustration of incremental learning with ELEPHANT. Imaging data are fed into a cycle of annotation, training, prediction and proofreading to generate cell lineages. At each iteration, model parameters are updated and saved. This workflow applies to both detection and linking phases. **b**, Detection workflow, illustrated with orthogonal views on the CE1 dataset. Top left: The user annotates nuclei with ellipsoids in 3D; newly generated annotations are colored in cyan. Top right: The detection model is trained with the labels generated from the sparse annotations of nuclei and from the annotation of *background* (in this case by intensity thresholding); *background*, *nucleus center*, *nucleus periphery* and unlabelled voxels are indicated in magenta, blue, green and black, respectively. Bottom right: The trained model generates voxel-wise probability maps for *background* (magenta), *nucleus center* (blue), or *nucleus periphery* (green). Bottom left: The user validates or rejects the predictions; predicted nuclei are shown in green, predicted and validated nuclei in cyan. **c**, Linking workflow, illustrated on the CE1 dataset. Top left: The user annotates links by connecting detected nuclei in successive timepoints; annotated/validated nuclei and links are shown in cyan, non-validated ones in green. Top right: The flow model is trained with optical flow labels coming from annotated nuclei with links (voxels indicated in the label mask), which consist of displacements in X, Y and Z; greyscale values indicate displacements along a given axis, annotated nuclei with link labels are outlined in red. Bottom right: The trained model generates voxel-wise flow maps for each axis; greyscale values indicate displacements, annotated nuclei are outlined in red. Bottom left: The user validates or rejects the predictions; predicted links are shown in green, predicted and validated links in cyan.



	ELEPHANT (IGFL-FR)	KTH-SE	KIT-Sch-GE
SEG	0.631	0.662	**0.729**
TRA	**0.975**	0.945	0.886
DET	**0.979**	0.959	0.930

Figure 2. Evaluation of performance and applications

a, Comparison of the speed of detection and validation of nuclei on the CE1 dataset, by manual annotation (magenta), semi-automated detection without a pre-trained model (orange) and semi-automated detection using a pre-trained model (blue) using ELEPHANT. **b**, Performance of ELEPHANT compared with two state-of-the-art algorithms [16,17], using the metrics of the Cell Tracking Challenge on unseen CE datasets. ELEPHANT outperforms the other methods in detection and linking accuracy (DET and TRA metrics); it performs less well in segmentation accuracy (SEG). **c**, Tracking results obtained with ELEPHANT. Left panels: Tracked nuclei in the CE1 and CE2 datasets at timepoints 194 and 189, respectively. Representative optical sections are shown with tracked nuclei shown in green; out of focus nuclei are shown as green spots. Right panels: Corresponding lineage trees. **d**, Comparison of tracking results obtained on the PH dataset, using the nearest neighbor algorithm (NN) with and without optical flow prediction (left panels); linking errors are highlighted in red on the correct lineage tree. The panels on the right focus on the nuclear division that is marked by a dashed line rectangle. Without optical flow prediction, the dividing nuclei (in magenta) are linked incorrectly. **e**, Spatial and temporal distribution of dividing cells in the regenerating leg of *Parhyale* over a 1-week time course (PH dataset), showing that cell proliferation is concentrated at the distal part of the regenerating leg stump and peaks after a period of

proliferative quiescence, as described in [15]. Top: Nuclei in lineages that contain at least one division are colored in magenta, nuclei in non-dividing lineages are in cyan, and nuclei in which the division status is undetermined are blank (see Methods). Bottom: Heat map of the temporal distribution of nuclear divisions; hpa, hours post amputation. The number of divisions per 20-minute time interval ranges from 0 (purple) to 9 (yellow). **f**, Fate map of regenerating leg of *Parhyale*, encompassing 109 fully tracked lineage trees (202 cells at 167 hpa). Each clone is assigned a unique color and contains 1-9 cells at 167 hpa. Partly tracked nuclei are blank. In panels e and f, the amputation plane (the distal end of the limb) is located on the left.

Supplementary Information

Supplementary Table 1: Processing speed of the detection model

Supplementary Table 2: Comparison of linking performances

Supplementary Figure 1: ELEPHANT client-server architecture

Supplementary Figure 2: 3D U-Net architecture for detection

Supplementary Figure 3: Proofreading in detection

Supplementary Figure 4: 3D U-Net architecture for flow

Supplementary Figure 5: Image quality issues in the PH dataset

Supplementary Figure 6: Complete and fully-validated cell lineage trees in a regenerating leg of *Parhyale*

Captions for Supplementary Videos

Supplementary Table 1: Processing speed of the detection model

Dataset		CE1	CE2	PH
Input size	voxels timepoints	708x512x35 195	712x512x31 190	1024x500x12 504
Prediction	Patch size	544x384x28	544x384x24	1024x500x12
	Number of patches	8	8	1
	Speed	6 sec/timepoint	5 sec/timepoint	2 sec/timepoint
Training	Patch size	384x384x16	384x384x16	384x384x12
	Number of patches per epoch	10	10	10
	Speed	24 sec/epoch	23 sec/epoch	22 sec/epoch

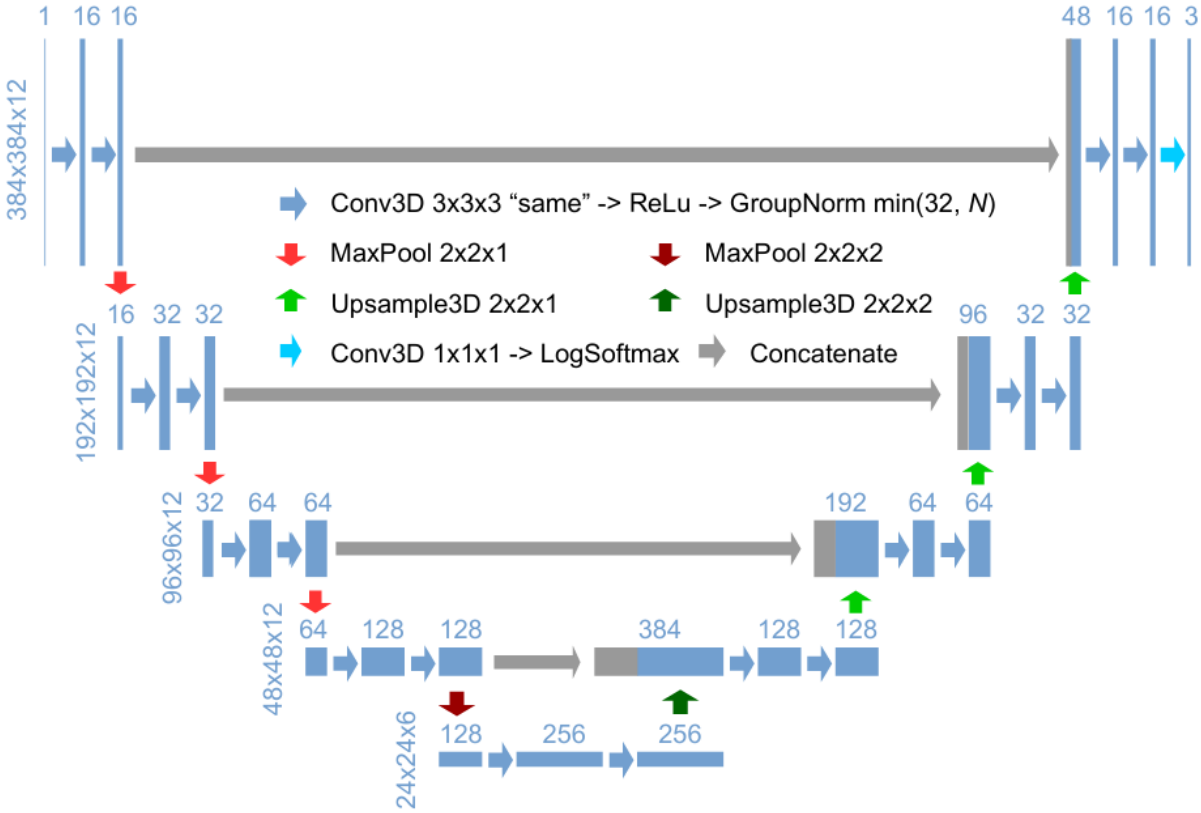
The table shows a summary of the processing speed of the deep learning model for the detection of nuclei, applied to three datasets. The training speed is affected by the distribution of annotations because the algorithm contains a try-and-error process for cropping, in which the *nucleus periphery* labels are forced to appear with the *nucleus center* labels.

Supplementary Table 2: Comparison of linking performances

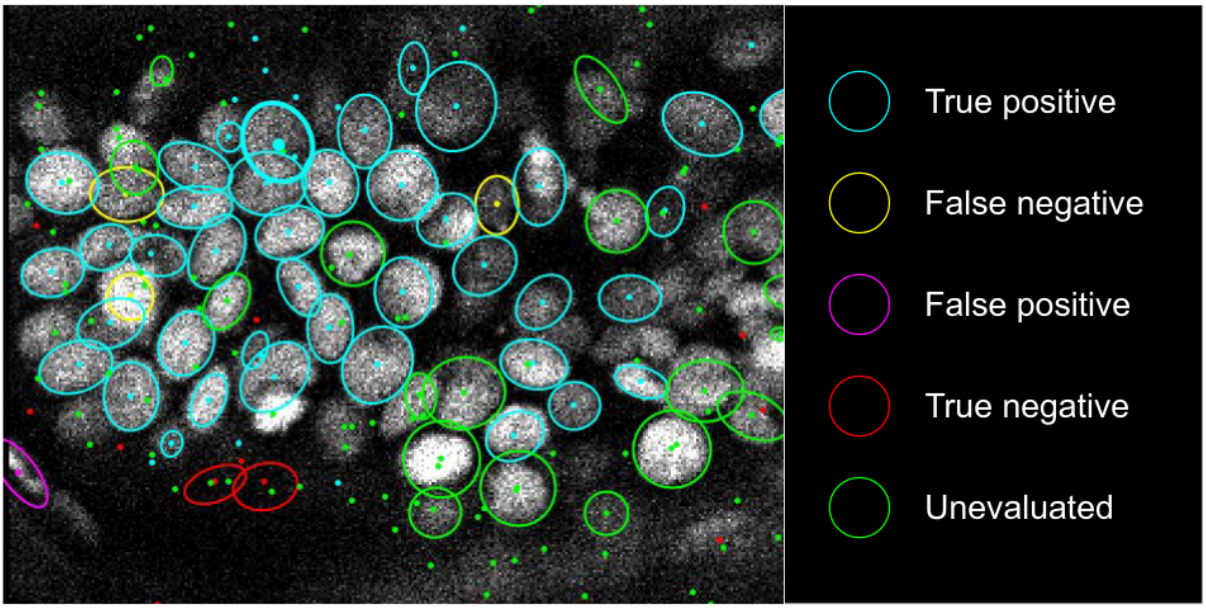
		Training from scratch	Incremental training from pre-trained model	Nearest neighbour
False positive	all	905	959	2,093
	cell division	11	16	16
False negative	all	1,349	1,329	1,991
	cell division	232	306	327
Training time		31 h	4 min	-

The table shows a summary of linking performances tested on the PH dataset, on a total number of 259,071 links (including 688 links on cell divisions). Incremental training was performed by transferring the training parameters from the model pre-trained with the CE datasets. Linking performance on dividing cells is scored separately.

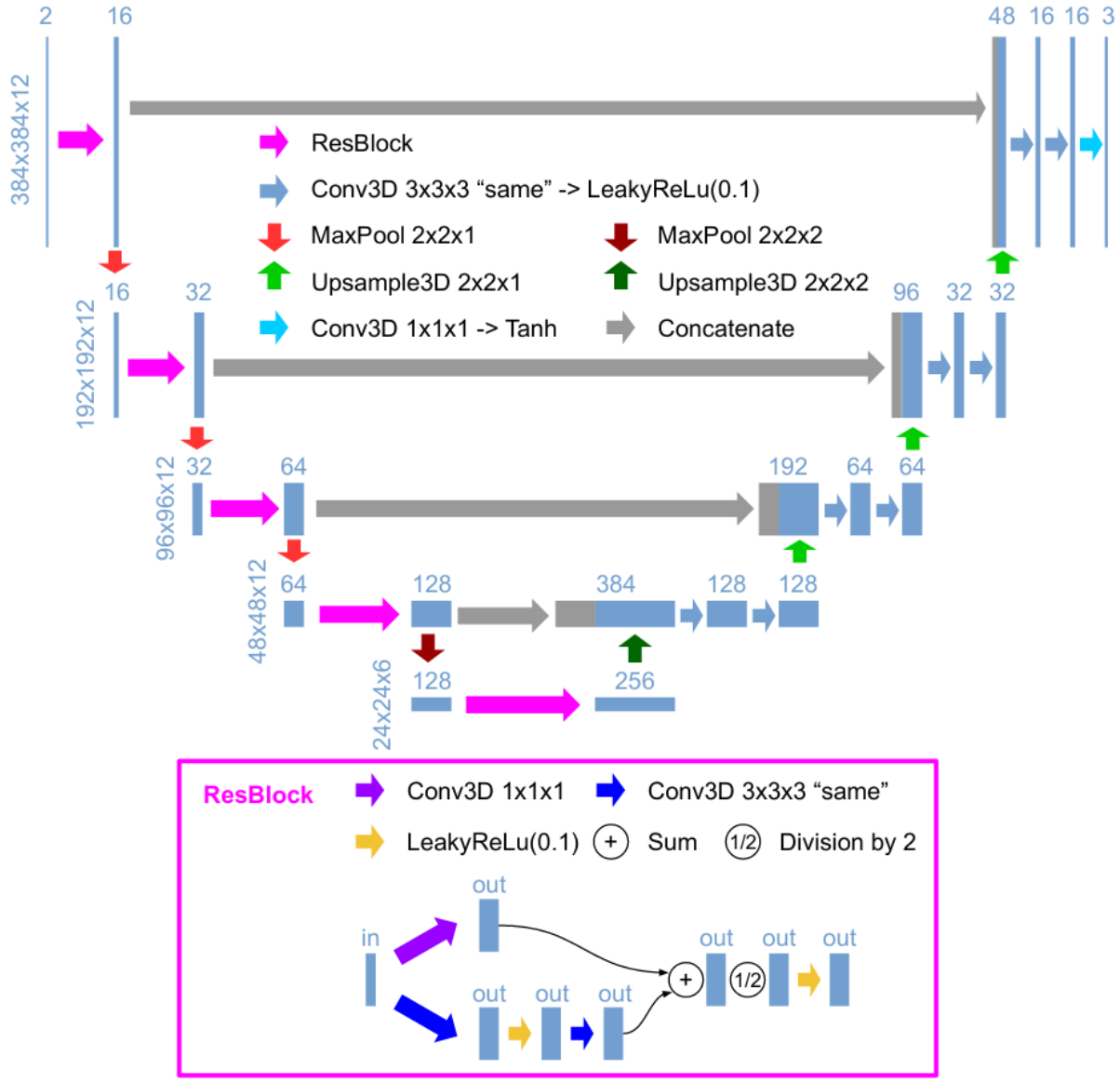
Supplementary Figure 1: ELEPHANT client-server architecture
The client provides an interactive user interface for annotation, proofreading and visualization. The server performs training and prediction with deep learning. The client and server communicate using HTTP and JSON.



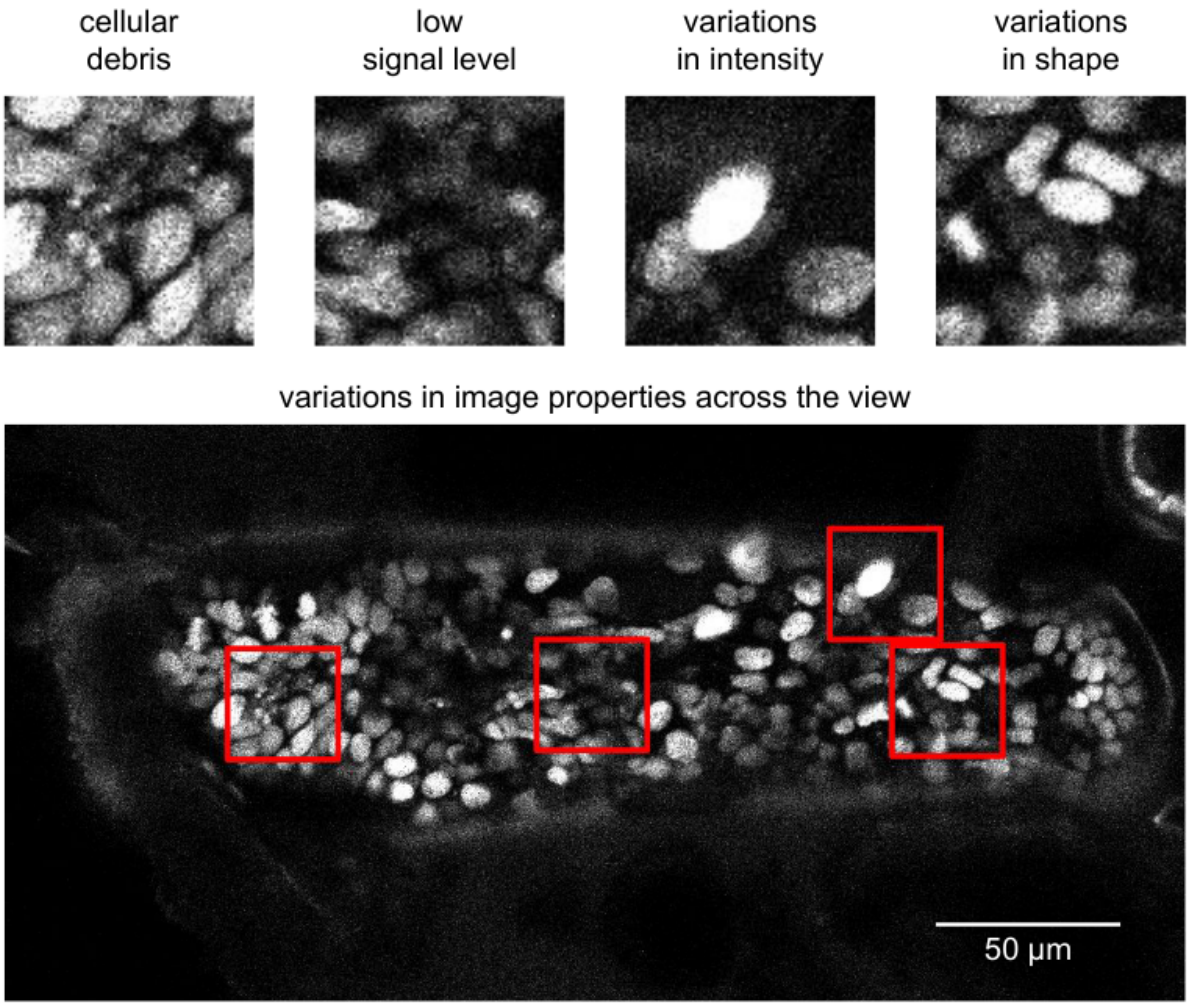
Supplementary Figure 2: 3D U-Net architecture for detection
Schematic illustration of the 3D U-Net architecture for detection, using an input image with a size of 384x384x12 and a ratio of lateral-to-axial resolution of 8 as an example. Rectangles show the input/intermediate/output layers, with the sizes shown on the left of each row and the number of channels shown above each rectangle. Block arrows represent different operations as described in the figure. The resolution of the z dimension is maintained until the image becomes nearly isotropic (ratio of lateral-to-axial resolution of 1, in the bottom layers in this example).



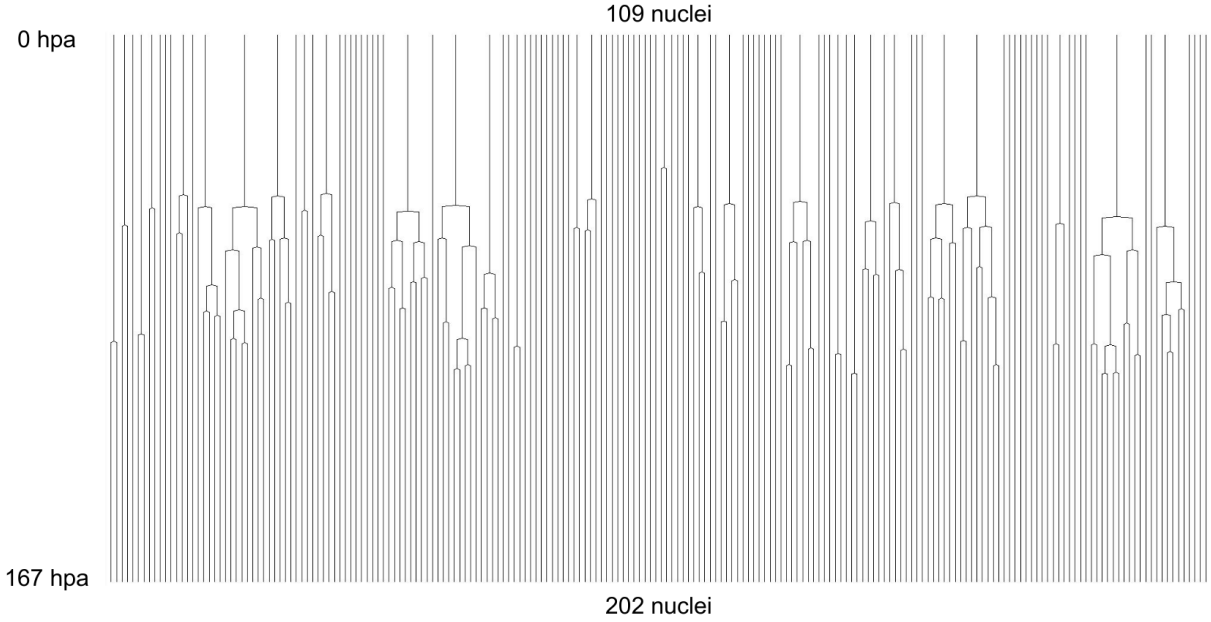
Supplementary Figure 3: Proofreading in detection
The ellipses show the sections of nuclear annotations in the xy plane and the dots represent the projections of the center position of the annotated nuclei, drawn in distinct colours; colour code explained on the right. Nuclei that are out of focus in this view appear only as dots.



Supplementary Figure 4: 3D U-Net architecture for flow
Schematic illustration of the 3D U-Net architecture for the flow model, depicted as in Supplementary Figure 2. The structure of ResBlock is shown on the bottom.



Supplementary Figure 5: Image quality issues in the PH dataset
Snapshots represent the image characteristics of the PH dataset that render cell tracking more challenging: fluorescence from cellular debris, low signal, variations in nuclear fluorescence intensity and nuclear shape, and variations in image quality across the imaged sample. The top panels show parts of a field of view indicated with red squares; the bottom panel shows an entire xy plane.



Supplementary Figure 6: Complete and fully-validated cell lineage trees in a regenerating leg of *Parhyale*

The displayed trees contain 109 complete and fully-validated cell lineages in a regenerating leg of *Parhyale* (PH dataset), corresponding to Figure 2f.

Captions for Supplementary Videos

Supplementary Video 1: Live imaging of *Parhyale* leg regeneration (PH dataset)
https://doi.org/10.5281/zenodo.4557870
A maximum intensity projection of the PH dataset captures the regeneration of a *Parhyale* T4 leg amputated at the distal end of the carpus, over a period of 1 week. hpa, hours post amputation

Supplementary Video 2: Incremental training of the detection model in ELEPHANT
https://doi.org/10.5281/zenodo.4557867
A cycle of incremental training of the ELEPHANT detection model is shown, including the annotation, training and prediction steps. The color-coding of the annotations is the same as shown in Supplementary Figure 3.

Supplementary Video 3: ELEPHANT flow predictions in 3D
https://doi.org/10.5281/zenodo.4557858
The PH dataset is shown in parallel with the corresponding flow predictions of the ELEPHANT optical flow model (in three dimensions), over the entire duration of the recording. Gray values for flow predictions represent displacements between timepoints as introduced in Figure 1c.