BGU-IL (5) DESCRIPTION

Segmentation Authors: Assaf Arbelle[1,2], Shaked Cohen[1,2] and Tammy Riklin Raviv[1,2]
Tracking Authors: Tal Ben-Haim[1] and Tammy Riklin Raviv[1,2]

[1]School of Electrical and Computer Engineering, Ben-Gurion University
[2]The Zlotowski Center for Neuroscience, Ben- Gurion University of the Negev.

The details about the segmentation method are provided in Section 1, while the tracking algorithm is explained in Section 2.

1. WEAKLY SUPERVISED MICROSCOPY CELL SEGMENTATION VIA CONVOLUTIONAL LSTM NETWORKS

1.1. Method

We address individual cells' segmentation from microscopy sequences. The main challenge in this type of problems is not only foreground-background classification but also the separation of adjacent cells. We apply two orthogonal approaches to overcome the multiple instance problem. From the segmentation perspective, we adopt the three class loss used by [1], [2]. The segmentation representation is designed to enhance individual cells' delineation by a partitioning of image domain into three classes: foreground, background and cell contours. From the detection perspective, we get our inspiration from [3], and aim to detect rough cell markers. The markers, as opposed to the full segmentation, do not cover the entire cell, but are rather a small "blob" somewhere within the cell. The markers have two desirable properties. First, they are much smaller than the object and thus are easier to separate between instances. One marker will never overlap or touch boundaries with a neighboring marker. Second, the markers are easy to annotate, as the annotator does not need to be precise, making data acquisition a simpler task. Often, for microscopy image sequences, the only available annotation is in the form of markers or approximate cell centers. We train the proposed network to estimate both the segmentation and the markers and merge the two using the Fast Marching Distance (FMD) [4]. The entire framework is illustrated in Figure 1.

1.1.1. Input and Output

The input to the method is a sequence of live cell microscopy images of arbitrary length T. We define the d dimensional (2 or 3) image domain by $\Omega \in \mathbb{R}^d$. We denote a frame in the input image sequence as $I_t : \Omega \to \mathbb{R}$, where the subscript $t \in [0, T-1]$ denotes the frame index and $I_t(v)$ is the intensity of a pixel (or voxel), $v \in \Omega$. The output of the network consists of two components, the scalar marker map (Section 1.1.4) $M_t : \Omega \to [0,1]$ which represents the probability of a pixel (voxel) to belong to a marker (cell segmentation core) and the soft segmentation map (Section 1.1.3) denoted as $S_t : \Omega \to [0,1]^3$ which represents the probabilities of each pixel (voxel) to belong to either the foreground, background or cell boundary. These two maps are then passed to an instance segmentation block (Section 1.1.6) which outputs the final labeled segmentation, map $\Gamma_t : \Omega \to \mathbb{N}^+$. Figure 1 shows an overview of the proposed method with visualization of the intermediate steps.

1.1.2. LSTM-UNet

The proposed network incorporates C-LSTM [5] blocks into the U-Net [6] architecture. This combination, first suggested in our preliminary work [1], is shown to be powerful. The UNet architecture, built as an encoder-decoder with skip connections, enables to extract meaningful descriptors at multiple image scales. However, this alone does not account for the cell specific dynamics that can significantly support the segmentation. The introduction of C-LSTM blocks into the network allows considering past cell appearances at multiple scales by holding their compact representations in the C-LSTM memory units. We propose here the incorporation of C-LSTM layers in every scale of the encoder section of the U-Net. Applying the C-LSTM on multiple scales is essential for cell microscopy sequences since the frame to frame differences might be at different scales, depending on cells' dynamics. The specific architecture was selected based on preliminary work which shows the empirical advantage over other alternatives [1]. The network is fully convolutional and, therefore, can be used with any image size[1] during both training and testing. Figure 1 illustrates the network architecture detailed in Section 1.2. The network is composed of

[1]In order to avoid artefacts it is preferable to use image sizes which are multiples of eight due to the three max-pooling layers.

two sections of N blocks each, the encoder recurrent block $E_{\theta_n}^{\{n\}}(\cdot)$ and the decoder block $D_{\theta_n}^{\{n\}}(\cdot)$ where θ_n are the network's parameters. The input to the C-LSTM encoder layer $n \in [0, \dots, N-1]$ at time $t \in T$ includes the down-sampled output of the previous layer; the output of the current layer at the previous time-step; and the C-LSTM memory cell. We denote these three inputs as $x_t^{\{n\}}$, $h_{t-1}^{\{n\}}$, $c_{t-1}^{\{n\}}$ respectively. Formally we define:

$$(h_t^{\{n\}}, c_t^{\{n\}}) = E_{\theta_n}^{\{n\}}(x_t^{\{n\}}, h_{t-1}^{\{n\}}, c_{t-1}^{\{n\}}) \tag{1}$$

where,

$$x_t^{\{n\}} = \begin{cases} I_t, & n = 0 \\ MaxPool(h_t^{\{n-1\}}), & 0 < n < N \end{cases} \tag{2}$$

The inputs to the decoder layers $n \in [N, \dots, 2N-1]$ are the up-sampled [2] output of the previous layer and the output of the corresponding layer from the encoder denoted by $y_t^{\{n\}}$ and $h_t^{\{2N-1-n\}}$ respectively. We denote the decoder output as $z_t^{\{n\}}$. Formally,

$$y_t^{\{n\}} = \begin{cases} h_t^{\{n-1\}}, & n = N \\ UpSample(z_t^{\{n-1\}}), & N < n < 2N-1 \end{cases} \tag{3}$$

where:

$$z_t^{\{n\}} = D_{\theta_l}(y_t^{\{n\}}, h_t^{\{2N-1-n\}}) \tag{4}$$

In [1] the network had a single output for the segmentation map while here, the last layer of the network is split to produce outputs for the soft segmentation map and marker map, S_t and M_t, to be discussed in the following sections. We define the full network as the composition of N encoder blocks followed by N decoder blocks. Note, that the encoder blocks, $E_{\theta_n}^{\{n\}}$, encode high-level spatio-temporal features at multiple scales and the decoder blocks, $D_{\theta_n}^{\{n\}}$, refines that information into full scale segmentation and marker maps.

1.1.3. The Soft Segmentation map

The output of the soft segmentation layer is defined as follows:

$$S_t = D_{\Theta_S}(y_t^{\{2N-1\}}, h_t^{\{0\}}) \tag{5}$$

where Θ_S are the parameters of this layer. S_t has the same spatial dimension as the input and has three channels corresponding to the un-normalized evidence for the three classes, background, foreground and cell boundary, encoded in the entries $l \in \{0, 1, 2\}$, respectively. We define the soft segmentation, S_t by the following softmax equation:

$$S_t(v) \triangleq p(l|S_t(\mathbf{v})) = \frac{\exp\{[S_t(\mathbf{v})]_l\}}{\sum_{l' \in \{0,1,2\}} \exp\{[S_t(\mathbf{v})]_{l'}\}} \tag{6}$$

[2]We use bi-linear interpolation

1.1.4. The Marker map

Similarly to the soft segmentation layer, the marker layer is defined as:

$$M_t = D_{\Theta_M}(y_t^{\{2N-1\}}, h_t^{\{0\}}) \tag{7}$$

where Θ_M are the weights of this layer. Note that the weight Θ_M are different from the soft segmentation weights Θ_S. M_t has the same spatial dimension as the input and has one channel corresponding to the un-normalized evidence for the marker. We define the marker map, M_t by the following sigmoid function:

$$M_t(\mathbf{v}) \triangleq \sigma(M_t) = \frac{1}{1 + \exp^{-M_t}} \tag{8}$$

1.1.5. Addressing 3D Data

In this paper, we address both 2D and 3D image sequences. Due to the heavy computational load of the 3D data, we generate the volumetric M_t and S_t from 2D slices. Let $\{I_t^{(k)}\}_{k=1}^K$ be the input It decomposed into K slices along the depth dimension. Similarly, we denote the corresponding soft segmentation and marker maps $S_t = \{S_t^{(k)}\}_{k=1}^K$, respectively. In order to keep the spatial coherence we process $I^(k)_t$ by feeding the network with three consecutive slices, $\{I^(k-1)_t, I^(k)_t, I^(k+1)_t\}$. These three slices are concatenated along the channel axis. The outputs S_t and M_t are constructed by stacking the respective 2D slices.

1.1.6. Instance Segmentation

The instance segmentation block merges the two network outputs, S_t and M_t, into a single, multi-label map. We define the semantic cell region by:

$$\mathcal{R}_t^{cell} = \{\mathbf{v} | \arg\max_{l \in \{0,1,2\}} [S_t(\mathbf{v})]_l = 1\} \tag{9}$$

The boundary class in S_t is used to partition the cell region, $\mathcal{R}_t^{cell}$, into connected components, where each component should represent a single cell instance. Yet, due to possible under-segmentation, the number of connected components may not necessarily correspond to the number of blobs in the marker map, which more faithfully represents the true number of cells. We, therefore, use M_t to facilitate the separation of merged cell instances. Let the marker regions be the thresholded marker map such that:

$$\mathcal{R}_t^{blob} = \{\mathbf{v} | M_t(\mathbf{v}) \geq 0.5\} \tag{10}$$

$\mathcal{R}_t^{blob}$ is partitioned into N_t connected components, $\{m_i\}_{i=1}^{N_t}$. We construct the final label map Γ_t for each $\mathbf{v} \in \mathcal{R}_t^{cell}$, by

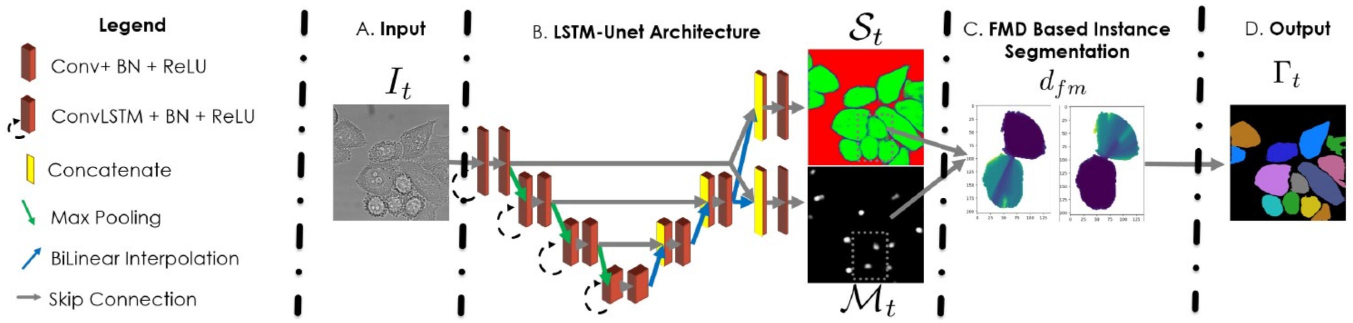


Fig. 1. **Method outline**: A. The input to the method is a time-lapse sequence of microscopy images. B. The network's down-sampling path consists of a C-LSTM layer followed by a convolutional layer with ReLU activation, the output is then down-sampled using max pooling and passed to the next layer. The up-sampling path consists of a concatenation of the input from the lower layer with the parallel layer from the down-sampling path followed by two convolutional layers with ReLU activations. The last layer is split to produce the soft segmentation and marker maps, S_t and M_t, respectively. C. The two network outputs are merged in to perform instance segmentation utilizing the FMD. The figures in the "Instance Segmentaiton" block show a zoom-in view of the gray dashed line. The distances to the two relevant centers allows fore optimal separation. D. The final label map, Γ_t is the extracted by finding the minimal distance to the centers.

searching for the nearest m_i with respect to a geodesic distance defined by the soft segmentation foreground map $[S_t]_1$:

$$\Gamma_t(\mathbf{v}) = \begin{cases} \arg\min_{i=1,\dots,N_t} d_{fm}(\mathbf{v}, m_i|[S_t]_1) & \mathbf{v} \in \mathcal{R}_t^{cell} \\ 0 & otherwise \end{cases} \tag{11}$$

The function d_{fm} denotes the FMD, which computes the shortest distance between a source point, $\mathbf{v}$, and a target point m_i, given a speed map $[S_t]_1$ in either 2D or 3D, with respect to the input domain. Areas in a speed map with high values will result in shorter paths, whereas areas with lower values will constrain the solution to longer distances. Specifically, pixels (voxels) with high boundary probability imply low foreground probability and thus resulting in very low speed.

1.1.7. *Data and Annotations*

Manual segmentation of microscopy sequences is a tedious and time consuming task, in particular where 3D+t data is considered. The annotation task can be simplified if the annotator, instead, marks the approximate cell centers. While ideally we wish to have one-hot encoded (background, foreground and boundary) labels, $\bar{S}_t : \Omega \rightarrow \{0,1\}^3$, for the entire training sequence, in practice, for most of the sequence cells are only weakly annotated by markers, $\bar{M}_t : \Omega \rightarrow \{0,1\}$. Specifically, we assume that at least one frame (or 2D slice in 3D) is fully annotated.

1.1.8. *Training and Loss*

The network is trained using Truncated Back Propagation Through Time (TBPTT) [7]. At each back propagation step the network is unrolled to τ time-steps. The total loss for training is a sum of two losses, one for the marker map, L_M, and one for the soft segmentation map, L_S. For the marker map we use single class cross entropy loss:

$$L_M = -\sum_{t'=t}^{t+\tau}\sum_{\mathbf{v}\in\Omega}[\bar{M}_t \cdot \log(M_t(\mathbf{v})) + (1-\bar{M}_t)\cdot\log(1-M_t(\mathbf{v}))] \tag{12}$$

The soft segmentation map is penalized by the weighted, multi-class cross entropy loss, giving higher weights, w_l to the cell contour pixels, due to the class imbalance:

$$L_S = -\sum_{t'=t}^{t+\tau}\sum_{\mathbf{v}\in\Omega}\sum_{l\in\{0,1,2\}} w_l \cdot [\bar{S}_t(\mathbf{v})]_l \cdot \log([S_t(\mathbf{v})]_l) \tag{13}$$

Frames, or pixels within frames, which did not have GT annotations were not taken into account in the loss calculation. The final loss is set to be the sum of the two:

$$L = L_M + L_S \tag{14}$$

1.2. Implementation Details

1.2.1. *Libraries*

The method is implemented in python utilizing Tensorflow for the neural networks section and the scikit-fmm python library for the FMD.

1.2.2. *Architecture*

The network include N = 4 encoder and decoder blocks. Each block in the encoder section is composed of C-LSTM layer, leaky ReLU, convolutional layer, batch normalization [8], leaky ReLU and finally down-sampled using maxpool operation. The decoder blocks consist of a bi-linear interpolation, a concatenation with the parallel encoder block followed by two sets of convolutional layer, batch normalization [8], and leaky ReLU. The same network is used both for 2D and 3D datasets. All C-LSTM kernels are of size 5×5 and all convolutional layers use kernel size 3×3. The feature depths in the encoder and decoder paths is set to be (128; 256; 512; 1024) and (1024; 512; 256; 128), respectively. All maxpool layers use kernel size 2×2 without overlap. The last convolutional layer uses kernel size 1×1 with depth 3 for the soft segmentation layer S_t and depth 1 for the marker layer, M_t, followed by a softmax or a sigmoid layer, respectively, to produce the final probabilities (see Figure 1).

1.2.3. Training Regime

We trained the networks on each dataset separately for 200K iterations (300K for the 3D datasets) using the ADAM optimizer [9] with learning rate of 0:0001. The unroll length parameter was set to $\tau = 4$ and the batch size was set to five sequences. The weights w were set to be 0:15; 0:25 and 0:6 for background, foreground and cell boundary, respectively (Section 1.1.8). We trained the networks using the gold truth annotations as our segmentation ground truth for the following datasets: Fluo-C2DL-Huh7, Fluo-C2DL-MSC, Fluo-C3DH-A549, Fluo-C3DH-H157, Fluo-C3DL-MDA231, Fluo-N3DH-CE, Fluo-N3DH-CHO, Fluo-C3DH-A549-SIM , Fluo-N3DH-SIM+ . We trained the networks using the silver truth annotations as our segmentation ground truth for the following datasets: BF-C2DL-HSC, BF-C2DL-MuSC, DIC-C2DH-HeLa, Fluo-N2DH-GOWT1, Fluo-N2DL-HeLa, PhC-C2DH-U373, PhC-C2DL-PSC.

2. GRAPH NEURAL NETWORK FOR CELL TRACKING IN MICROSCOPY VIDEOS

2.1. Method

Here, we present the method's building blocks. A visualization of the method outline is presented in Fig. 2. The reader is also referred to [10] for further details about our method.

2.1.1. *Graph Formulation*

We use a direct, acyclic graph to model cell-to-cell associations in microscopy sequences. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ define a graph represented by its vertices (nodes) $\mathcal{V}$ and edges (links) $\mathcal{E}$. Let $M = \sum_{t=1}^{T} K_t$ represent all cell instances in the entire frame sequence. A graph representation of cells and their associations is composed of $|\mathcal{V}| = M$ vertices, where each node $\nu_i \in \mathcal{V}$, $i = 1, \ldots, M$ represents a single cell instance $c_{t=\tau}^{k}$. For convenience, we can set $i = \sum_{t=1}^{\tau-1} K_t + k$.

An edge $e_{i,j} \in \mathcal{E}$ represents a potential association between a pair of vertices ν_i, ν_j, representing cell instances in consecutive frames. To reduce the number of edges, we connect a pair of cells only if their spatial Euclidean distance is within a neighborhood region, which is calculated based on the training set.

We address cell tracking as an edge classification problem. The desired output are labeled sets of edges defined by an edge function $Y\colon \mathcal{E} \rightarrow \{0, 1\}$. Let ν_i, ν_j represent cell instances denoted by c_t^k and c_{t+1}^l, respectively.

$$Y(e_{i,j}) = y_{i,j} = \begin{cases} 1, & \text{if } \psi(c_t^k) = \psi(c_{t+1}^l) \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

Accurate prediction of $Y(e_{i,j})$ for the complete set of graph edges provides the entire cell lineage associated with the observed microscopy sequence. A cell trajectory $\mathcal{T}_n$ can be either defined by a sequence of cell instances represented by the graph's vertices $\{\nu_{i_1}, \nu_{i_2}, \ldots \nu_{i_n}\}$, or by a sequence of edges $\mathbf{e}_n = \{e_{i_1 i_2}, \ldots, e_{i_{n-1} i_n}\}$, that connect cell instances in consecutive frames, where $\{e_{ij} \in \mathbf{e}_n \mid Y(e_{ij}) = 1\}$.

We assume that a non-dividing cell instance has at most a single successor while a cell that undergoes mitosis may have two successors and even more (in rare occurrences). Ideally, if two (or more) different nodes in a frame $\nu_j \neq \nu_{j'}$ are connected to the same node ν_i in a previous frame, i.e.; $Y(e_{i,j}) = Y(e_{i,j'}) = 1$, then we can assume that $P(\psi(c_{t+1}^l)) = P(\psi(c_{t+1}^{l'})) = \psi(c_t^k)$ and detect a mitosis event. In practice, often the visual features of daughter cells differ from those of their parent and an additional process is required to identify and validate parent-daughter relations. The complete representation of the proposed graph is defined by the following attribute matrices: i) A node feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d_{\mathcal{V}}}$ with $d_{\mathcal{V}}$ features per node. ii) A graph connectivity matrix $\mathbf{E} \in \mathbb{N}^{2 \times |\mathcal{E}|}$ which represents all possible linked cell indices from source to target. iii) An edge feature matrix $\mathbf{Z} \in \mathbb{R}^{|\mathcal{E}| \times d_{\mathcal{E}}}$, where each row in $\mathbf{Z}$ consists of $d_{\mathcal{E}}$ features of an edge e_{ij} in the graph. We aim to predict $\hat{\mathbf{Y}} \in \mathbb{R}_{[0,1]}^{|\mathcal{E}| \times 1}$ that represents the probability to represent an actual cell association. Next, we present the initial graph embedding.

2.1.2. *Feature Extraction*

The success of cell tracking algorithms depends on correct associations of instances of the same biological cells. We consider instances of the same cell as members of the same class. Altogether, we have N mutually exclusive classes.

Deep Metric Learning (DML) Features. We use *deep metric learning* to learn cell feature embeddings that allow us to assemble instances of the same biological cells and distinguish between different ones. For this purpose, we use the cell segmentation maps or marker annotations to crop each frame into sub-images of all cell instances. Following [11] we use a *hard mining* strategy and a *multi-similarity loss* function to train a ResNet network [12] to predict such embeddings. Specifically, we generate batches of cell sub-images, where each is composed of m same-class instances from κ classes. Since the cell's appearance gradually changes during the sequence we perform the *m-per-class* sampling [13] using temporally adjacent frames.

Spatio-temporal Features. While the learned feature vectors serve to distinguish between cell instances based on their visual appearance alone they do not account for temporal and global spatial features that also characterize the cells. These include the coordinates of the cell center, its frame number, and intensity statistics (minimum, maximum, and average). In the case where we have an instance segmentation mask, the cell's area, the minor and major axes of a bounding ellipse, and bounding-box coordinates are also considered. We denote by $\mathbf{V}_{\text{ST}} \in \mathbb{R}^{|\mathcal{V}| \times d_{\text{ST}}}$ the spatio-temporal (ST) feature matrix, which is composed of the d_{ST}-dimensional feature vectors of all nodes.

Initial Edge and Node Features. The complete feature matrix of all nodes in the graph includes both the learned and the spatio-temporal features and is denoted by $\mathbf{V}_{\mathcal{V}} \in \mathbb{R}^{|\mathcal{V}| \times d_{\mathcal{V}}}$, where, $d_{\mathcal{V}} = d_{\text{ST}} + d_{\text{DML}}$. It is generated by a concatenation of $\mathbf{V}_{\text{DML}}$ and $\mathbf{V}_{\text{ST}}$. Having $\mathbf{V}_{\mathcal{V}}$ we construct the initial edge feature matrix $\mathbf{V}_{\mathcal{E}}$ using the *distance* & *similarity* operation defined in Eq. 17. Since $\mathbf{V}_{\text{DML}}$ and $\mathbf{V}_{\text{ST}}$ are from different sources and are in different scales, we homogenize them and reduce the complete feature vector dimension via mapping by multi-layer perceptron (MLP) networks. These MLPs are connected to the proposed GNN (see Section 2.1.3) in an end-to-end manner. We denote the initial node feature vector of a vertex ν_i by $\mathbf{x}_i^{(0)}$, where $\mathbf{x}_i^{(0)}$ is the i-th row in $\mathbf{X}^{(0)}$.

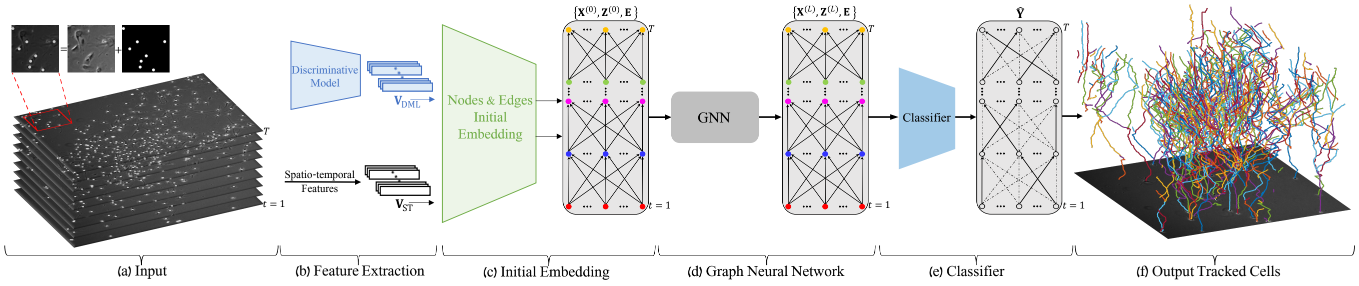


Fig. 2. **An outline of the proposed cell tracking framework.** (a) The input is composed of a live cell microscopy sequence of length T and the corresponding sequence of label maps. (b) Each cell instance in the sequence is represented by a feature vector which includes DML and spatio-temporal features. (c) The entire microscopy sequence is encoded as a direct graph where the cell instances are represented by its nodes and their associations are represented by the graph edges. Each node and edge in the graph has its own embedded feature vector. (d) These feature vectors are encoded and updated using Graph Neural Network (GNN). The GNN (which is illustrated in Fig. 3(a)) is composed of L message passing blocks which enable an update of edge and node features by their L-th order neighbors (i.e., cell instances which are up to L frames apart). (e) The GNN's edge feature output is the input for an edge classifier network which classifies the edges into active (solid lines) and non-active (dashed lines). During training, the predicted classification $\hat{\mathbf{Y}}$ is compared to the GT classification $\mathbf{Y}$ for the loss computation. Since all the framework components are connected in an end-to-end manner the loss backpropogates throughout the entire network. (f) At inference time, cell tracks are constructed by concatenating sequences of active edges that connect cells in consecutive frames

2.1.3. *Graph Neural Network*

The core of the proposed cell tracking framework is the graph neural network (GNN) illustrated in Fig. 3(a). Exploiting the GNN model and the message passing paradigm allows us to simultaneously trace entire cells tracks rather than locally associate cell instances in a frame-by-frame manner. One of our main contributions is the graph message passing block presented in Fig. 3(b) called the Edge-oriented Pathfinder Message Passing Neural Network (EP-MPNN). Specifically, we extend the MPNN block presented in [14] by introducing an edge encoder, thus enabling an interplay between the edge and node feature update simultaneously with an edge-attention mechanism.

The GNN is composed of L EP-MPNN blocks where L determines the message passing extent. In other words, the associations of cell instances in consecutive frames are affected by the respective connections along the sequence up to L frames away. The input to the $l+1$-th EP-MPNN block (which is, in fact, the output of the l-th EP-MPNN block) is composed of the updated node and edge features, denoted by $\mathbf{X}^{(l)} = \{x_i^{(l)}\}_{\nu_i \in \mathcal{V}}$ and $\mathbf{Z}^{(l)} = \{z_{ij}^{(l)}\}_{e_{ij} \in \mathcal{E}}$, respectively, where $l = 0, \ldots, L$. The nodes are updated using the pathfinder discovery network convolution (PDN-Conv) [14] which is one of the two EP-MPNN components. The other component is the *edge encoder* (illustrated in Fig. 3(c)) which is trained to embed the edge features based on the node features. We introduce the edge encoder in the following.

Node Feature Update. The features of each node ν_i are updated based on the weights of the incoming edges. These weights are learned using an attention mechanism. Let $f_{\text{edge}}^{\text{PDN}}: \mathbb{R}^{d_\mathcal{E}} \rightarrow \mathbb{R}$ define a function implemented by an MLP that is trained to output scalars which represent the weights of the edges, given their current features. Let $f_{\text{node}}^{\text{PDN}}: \mathbb{R}^{d_\mathcal{V}} \rightarrow \mathbb{R}^{d_\mathcal{V}}$ define a vector function implemented by an MLP that is trained to output updated feature vectors given the current ones. The updated feature vector of a node ν_i is obtained by a weighted sum of its own and its neighboring nodes, as follows:

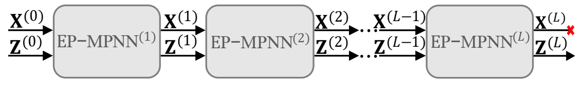
$$\mathbf{x}_i^{(l)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \overbrace{f_{\text{edge},l}^{\text{PDN}}(\mathbf{z}_{j,i}^{(l)})}^{\omega_{ji}^{l}} \overbrace{f_{\text{node},l}^{\text{PDN}}(\mathbf{x}_j^{(l-1)})}^{\tilde{\mathbf{x}}_j^{(l-1)}}, \qquad (16)$$

where $\mathcal{N}(i)$ denotes the neighbors of ν_i; i.e., all the nodes ν_j for which there exist $e_{j,i} \in \mathcal{E}$. Note that $f_{\text{edge},l}^{\text{PDN}}$ and $f_{\text{node},l}^{\text{PDN}}$ are trained separately for each block. Eq. 16 can be interpreted as attention through edges, where, $\omega_{ji}^{l} = f_{edge,l}^{\text{PDN}}(\mathbf{z}_{j,i}^{(l)})$ is the predicted attention parameter of an edge $e_{j,i}$ ($\omega_{ii}^{l} = 1$) and $\tilde{\mathbf{x}}_j^{(l-1)} = f_{\text{node},l}^{\text{PDN}}(\mathbf{x}_j^{(l-1)})$ is the *mapped* feature vector of a node ν_j in the l-th EP-MPNN.

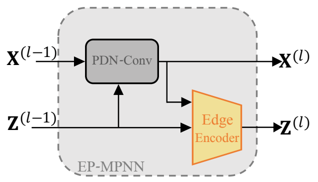
Edge Feature Update. The main contribution of the proposed GNN framework is a mechanism for edge feature update that enhances the message passing process. Unlike the GNN presented in [14] here, the edge and node features are alternately updated. We denote by ***D-S*** a function that returns the distance & similarity vector of two feature vectors of connected nodes as follows:

$$\boldsymbol{D\text{-}S}(\mathbf{v}_i, \mathbf{v}_j) = \left[|v_i^1 - v_j^1|, \ldots, |v_i^{d_\mathcal{V}} - v_j^{d_\mathcal{V}}|, \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\| \|\mathbf{v}_j\|}\right] \quad (17)$$

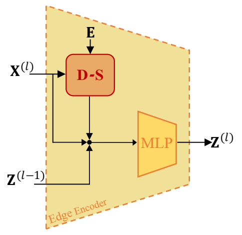
which is a concatenation of the absolute values of the differences between corresponding elements in $\mathbf{v}_i$ and $\mathbf{v}_j$ and their



(a) GNN



(b) EP-MPNN



(c) Edge Encoder

Fig. 3. **(a) A Graph Neural Network (GNN).** The GNN is composed of L EP-MPNN blocks where L determines the message passing extent. The l-th EP-MPNN block updates the nodes and edge features, i.e., $\mathbf{X}^{(l-1)} \rightarrow \mathbf{X}^{(l)}$ and $\mathbf{Z}^{(l-1)} \rightarrow \mathbf{Z}^{(l)}$. **(b) Edge-oriented Pathfinder - Message Passing Network (EP-MPNN) Layer**. The basic layer in the graph neural network step comprises a PDN-Conv and an edge encoder. The PDN-Conv updates the node feature vectors based on their current values and an edge attention model. **(c) Edge encoder.** Updates the edge feature vectors. Its ***D-S*** block calculates the distance and similarity between the feature vectors of each pair of nodes by an edge. The output of the ***D-S*** block along with the nodes and the current edge features compose the input of an MLP which outputs the new feature vectors of the edges. The concatenation (denoted by '$\otimes$') of the ***D-S*** block's output along with the current node and edge features formulated in Eq. 18, is the input for an MLP which is trained to learn a new edges representation

cosine similarity $\frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\|\|\mathbf{v}_j\|}$. In the l-th block $\mathbf{v}_i = \mathbf{x}_i^{(l)}$ and $\mathbf{v}_j = \mathbf{x}_j^{(l)}$. In the initial phase, when the input to the first GNN block is formed, $\mathbf{v}_i$ and $\mathbf{v}_j$ are the i-th and the j-th rows in $\mathbf{V}_{\mathcal{V}}$, respectively. The construction of the initial feature matrix $\mathbf{V}_{\mathcal{V}}$ is described in Section 2.1.2.

The edge update function $f^{\text{EE}}_{\text{edge},l}$ (implemented as an MLP) returns the updated features of each edge $e_{i,j}$ given the concatenation of the current edge features, the updated feature vectors of the nodes it connects, and the output of the ***D-S*** block applied to these nodes. Formally,

$$\mathbf{z}_{ij}^{(l)} = f^{\text{EE}}_{\text{edge},l}([\mathbf{z}_{ij}^{(l-1)}, \mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}, \boldsymbol{D\text{-}S}(\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)})]) \tag{18}$$

2.1.4. *Classifier and Training*

The output edge feature vectors are the inputs to the edge classifier network. The classifier is an MLP with three linear layers each is followed by a ReLU activation, when a Sigmoid function is applied to the output layer. The output is a vector $\hat{\mathbf{Y}} \in \mathbb{R}_{[0,1]}^{|\mathcal{E}|\times 1}$ that represents the probability for each edge to be active $(= 1)$ or not $(= 0)$. We use the ground-truth (GT) edge activation vector $\mathbf{Y}$ to train the model. Since most of the edges in the graph dataset are not active (i.e., do not link nodes), our data are highly imbalanced. Therefore, we use a weighted cross-entropy loss function with adaptive weights which are determined by the average number of neighbors in a batch, i.e., $\left(\frac{1}{|\mathcal{N}|}, \frac{|\mathcal{N}|-1}{|\mathcal{N}|}\right)$. Note that since the size of the neighborhood region remains fixed throughout the sequence, the number of neighbors increases as the frames become denser.

2.1.5. *Cell Tracking Inference*

The output of the proposed deep learning framework is a probability matrix which identifies active edges. It is used together with the connection matrix $\mathbf{E}$ to construct candidates for cell trajectories as described in Section 2.1.1. Specifically, predictions of cell tracks are obtained at the inference phase in the form of a directed graph with soft edge weights (the output of a sigmoid). The edges in the graph represent only outgoing, potential associations between consecutive frames. The soft weights (association probabilities) allow us to partition the graph edges into active and non-active. Considering the outgoing/incoming edges of a specific node, there could be one of the following outcomes: 1) All outgoing/incoming edges are non-active - which may indicate end/beginning of a track. 2) Only one outgoing edge is active. 3) Two or more outgoing edges are active which may indicate mitosis (cell division). 4) More than a single incoming edge is active - i.e., when different cell instances are associated to the same cell instance. Above 99% of incoming/outgoing edges conflicts are avoided thanks to the proposed training scheme. We note that the network is implicitly trained to prefer bijection cell associations thanks to the attention-based mechanism of the GNN blocks and the weighted loss function (see Section 2.1.4). Nevertheless, to ensure one-to-two mapping at most (case 3), in case that the association probabilities of more than two outgoing edges are higher than 0.5 (extremely rare events) only the top-2 are considered as active. In addition, to ensure injective mapping (case 4), only the incoming edge with the highest association probability (as long as it is higher than 0.5) is considered active. This obviously feasible ad-hoc strategy ensures a single path to each cell.

2.1.6. *Mitosis Detection*

Since daughter cells usually have different visual features than their parent, it is not very frequent for a node to have two active outgoing edges. In most mitotic events the parent track terminates whereas two new tracks initiate. To associate pairs of daughter cells to their parents we consider the detected tracks of all cells. We then look for triplets of trajectories $(\mathcal{T}_k, \mathcal{T}_l, \mathcal{T}_m)$ where $k, l, m \in \{1, \ldots, N\}$ such that $t^k_{\text{init}} = t^l_{\text{init}} = t^m_{\text{fin}} + 1$. If the spatial coordinates of $c^k_{t_{\text{init}}}, c^l_{t_{\text{init}}}$ and $c^m_{t_{\text{fin}}}$ are within the same neighborhood region, then we set $P(k) = P(l) = m$.

2.2. Implementation Details

2.2.1. *Graph Neural Network*

We implemented the proposed graph neural network (GNN) model using the Pytorch Geometric library [15]. We train our framework with graphs based on microscopy sub-sequences of 10 frames while for the inference we use the entire sequence to construct the input graph. The prediction of all edges (a classification into 'active' and 'non-active' edges) is performed simultaneously.

The spatio-temporal features are normalized by *min-max scaling* for each graph, while the deep metric learning features are not pre-processed. To accommodate the high number of cell instances within a frame and to reduce the computational complexity, cell instances in consecutive frames are connected by edges only if their spatial Euclidean distance is smaller than a predefined threshold that is determined by the cells' *neighborhood region*. The neighborhood region $\mathcal{N}_R$ is defined based on the size of the cells' bounding box $size_{BB}$ and the rate of the cells' movement $size_{move}$. Formally, $\mathcal{N}_R = \alpha \cdot \max(\max_i(size_{BB_i}), \max_j(size_{move_j}))$. The maximization is applied to each axis separately. The hyper-parameter α is set to 2 or 4, depending on the sequence's density. For the graph neural network, we set the number of layers $L = 6$ to perform six message-passing steps, enabling information propagation between cell instances that are 6 frames apart. We set the dimension $d_{\mathcal{V}}$ of the node feature matrix to 32, where $d_{\mathcal{E}} = 64$ for the edge feature matrix. The *Adam* optimizer [16] is used with a learning rate of $1e-3$ and a weight decay of $1e-5$.

2.2.2. *Deep Metric Learning*

We use Pytorch metric learning library [17] to train ResNet18 [12] followed by multi-layer perceptron (MLP). The final embedding is $L2$ normalized and $d_{\text{DML}} = 128$. The training is done using batches with a size of 32. Batches are constructed by *m-per-class* sampler, which first randomly samples κ classes, and then randomly samples m images for each of the κ classes. Since the cell's appearance gradually changes during the sequence we perform the *m-per-class* sampling [13] using temporally adjacent frames. We set $\kappa = 8$ and $m = 4$. The ResNet18 and MLP models are optimized using two separated *Adam* optimizers [16] for each model with learning rates of $1e-5$ and $1e-4$, respectively. We also use weight decay of $1e-4$. We use the cell segmentation maps or marker annotations to crop each frame into sub-images of all cell instances. We constructed the datasets used for DML training by assigning to each cell instance the index of its biological cell. In case the cell segmentation maps (rather than marker annotations) are available we exploit them to filter out the background via pixel-wise multiplication and extract features such as cell size and intensities.

3. REFERENCES

[1] A. Arbelle and T. Riklin Raviv, "Microscopy cell segmentation via convolutional lstm networks," pp. 1008–1012, 2019.

[2] A. Arbelle and T. Riklin Raviv, "Microscopy cell segmentation via adversarial neural networks," *IEEE ISBI*, pp. 645–648, 2018.

[3] Filip Lux and Petr Matula, "Dic image segmentation of dense cell populations by combining deep learning and watershed," in *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, 2019, pp. 236–239.

[4] J A Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[5] H. Wang D.-Y. Yeung W.-K. Wong S. Xingjian, Z. Chen and W. c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, 2015, pp. 802–810.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[7] R. J. Williams and J. Peng, """an efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural computation*, vol. 2, no. 4, pp. 490–501, 1990.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015, pp. 448–456.

[9] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," 2017.

[10] T. Ben-Haim and T. Riklin Raviv, "Graph neural network for cell tracking in microscopy videos," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[11] X. Wang, X. Han, W. Huang, D. Dong, and M. R Scott, "Multi-similarity loss with general pair weighting for deep metric learning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5022–5030.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[13] K. Musgrave, S. Belongie, and S.-N. Lim, "A metric learning reality check," in *European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 681–699.

[14] B. Rozemberczki, P. Englert, A. Kapoor, M. Blais, and B. Perozzi, "Pathfinder discovery networks for neural message passing," in *Proceedings of the Web Conference*, 2021, pp. 2547–2558.

[15] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[16] D. P Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[17] K. Musgrave, S. Belongie, and S.-N. Lim, "PyTorch metric learning," 2020.