Convolutional graph neural networks for tracking yeast cells in segmentations

N. Vadot, F. Massard, G. Brenna, S. Rahi, V. Gligorovski

***Abstract*—In many bioimaging applications, segmentation algorithms are used in order to treat microscopy images. This is nowadays done using deep learning methods, which achieve human-like performance and accelerate research. However, one often overlooked problem is that of tracking cell masks throughout movies. This post-segmentation step is often done using classical bipartite graph matching algorithms. Often manual labor is required to double-check cell masks are tracked correctly throughout the movie. This is especially time-consuming for large cell colonies or long movies, and the performance of classical algorithms degrades quickly as the time difference between images grows. We present a cell mask tracking method based on recent advancements in deep graph convolutional networks [1]. This method achieves better accuracy than classical algorithms, thanks to the context-awareness of the graph structure. Our method (1.) extracts simple geometrical information from the cell masks, (2.) feeds the extracted information into a graph convolutional neural network, (3.) applies linear cost optimizations to the output scores of the neural network, producing (4.) an assignment matrix which directly translates back to cell mask trackings. For this, we train on a dataset of segmented budding yeast images, containing 5 colonies each growing to approximately 110 cells by the end of the 180 frames. Cutting down on manual labor, this method performs with near-perfect accuracy on hard tracking problems, even in situations where cells dissapear from the colony or new buds appear.**

I. Introduction and problem statement

The tracking problem emerges in timelapse microscopy, where images of a cell colonies are taken at a fixed rate (typically around one image every 15 minutes for budding yeast). Segmentation of these images refers to finding the pixelwise masks that cover each cell visible in a given image. Each cell is then given a unique identifier, and tracking translates to matching each cell of the current image to a cell in the next image. This is a particularly challenging task for fast-growing microbes, such a budding yeast. Dependingly on the time interval between each image, the cell colony might have changed significatively : new buds appear, cell geometry changes due to growth or pressure from neighbouring cells, cells get flushed away or move around.

Correct tracking of cells is especially important when addressing a more specific problem, that of lineage determination of budding yeast : for each new bud, the parent cell needs to be determined. This is made difficult in densely packed environments. One solution to this problem is the use of fluorescent markers that label the site of emerging bud on the parent cell. However, it might not always be convenient to use such markers, in which case one can only rely on the microscopy images and the corresponding segmentations. If mistakes are made during tracking of the cell masks, then this directly causes mistakes in the lineage relationships, which utilize the same cell identifiers as the masks.

The classical approach to tracking is to extract some features from the segmentation geometry, and minimize the pairwise euclidean distance in the feature space using the Hungarian algorithm. The problem with this method is that it starts to break down for large colonies, since in that regime many cells have similar geometry or start drifting away from the colony.

One key observation is that cells not only have features (area, radius, eccentricity, etc.) of their own, but also exist in a neighborhood of other cells. In other words, a lot of identifying information is encoded in the *neighbourhood* of each cell, such as the relative position with each neighbour, and the features of neighbouring cells.

A graph is the most intuitive way to store this information, where cells are represented by nodes, and edges link nearest neighbours together. Nodes then store cell features, and edges store relative positions of a cell pair. This graph representation is especially powerful, as it directly encodes locality in the colony, and repeatedly performing graph convolutions lets this information diffuse to neighbouring cells. The recent development of graph neural networks (GNN) [2] and growing open-source codebase [3], [4] makes these methods attractive for solving the problem of tracking cells in a timelapse microscopy.

II. Prior work

Most of the state-of-the-art cell tracking methods rely on conventional algorithms such as bipartite matching [5], [6] and Viterbi [7], [8]. Bipartite matching (also equivalently found under the names of "assignment problem", "linear cost optimization" or "Hungarian algorithm") assigns a cost to each possible pair of cell matching between the current and next image. For instance in YeaZ [6], this cost is derived from the euclidean distance in feature space (center of mass and area) between a pair of cells. The Viterbi algorithm models the change of individual cell features as a Markov process, and maximizes the likelihood of the hidden intermediate cell features resulting in the observed features on the next frame.

However, these traditional approaches have limitations and deep learning methods have proven to have great potential to improve performance. In recent years, various Convolutional Neural Networks (CNN), and recurrent CNN-based approaches have been introduced for tracking cells in microscopy images (e.g. [9], [10], [11], [12], [13]).

While they have proven useful for general cell tracking tasks, currently available tracking tools designed specifically for budding yeast imaging still rely on traditional approaches. This is due to the particularity of budding

yeast, where cells bud frequenty (the cell-cycle time of budding yeast is around 90 minutes), grow fast, move around the colony, and microscopy images are sparse in time. Recent works on segmentation and tracking of budding yeast cells in brightfield microscopy focus rather on developing and training networks for segmentation, than on updating tracking techniques that are still based on conventional algorithms [6], [14], [15].

This paper proposes a novel graph neural network (GNN) approach for tracking yeast cells from segmentations [6], [16]. We take inspiration from tracking algorithms used in particle trajectory reconstruction in detectors [17], where geometrical features of detection events are encoded using a simple Multi-Layer Perceptron (MLP), after which a classifier attributes multiple detection events to the same particle. We also apply ideas from [18], which considers the problem of finding a subgraph in a larger graph. It does so by applying graph convolutions, embedding the nodes in a high-dimensional space, and minimizing the total cost of pairwise euclidean distance to match nodes from the two graphs. We thus reformulate the tracking problem as a graph matching problem, and use a similar idea by maximizing a score to assign nodes between two graphs.

III. Data preprocessing

A. Feature extraction and cell graph generation

The full data preprocessing pipeline is schematized in FIG. 3. The data used here consists of 5 colonies of budding yeast, with microscopy images taken at 5 minute intervals, for a total of 180 images (15 hours of growth time) per colony. On average, the colonies started with 2 cells and had 110 cells at the end.

To generate the ground truth dataset, the movies were segmented and manually tracked using YeaZ [6]. From the segmentation masks, simple geometrical features were extracted from the cells : mask surface area A, radius r and eccentricity e. To do this, an ellipse was fitted to the cell by using PCA analysis on the cell mask coordinates, which can then be used to obtain the semi-major and minor axes a and b respectively. Then we compute $e = \sqrt{1-(\frac{a}{b})^2}$, and the radius of the circle with the same area as the ellipse $r = \sqrt{ab}$. FIG. 1 shows the distribution of the node features. The distributions are strongly peaked around their mean, respectively $\langle A \rangle \approx 21 \pm 14$ µm^2, $\langle r \rangle \approx 2.5 \pm 0.8$ µm and $\langle e \rangle \approx 0.52 \pm 0.14$. Notice the distribution for r has a few outliers corresponding to very large cells, and these outliers are even more visible for A, explained by the scaling law $A \sim r^2$.

To extract nearest neighbour features, we first extracted the contour of each cell mask. Cells are considered neighbours if the minimal distance between their contours is below 1.3 µm (about half a mean cell radius), which is a parameter that can be adjusted. Extracted features are ρ, the center of mass (CM) to CM distance between the neighboring cells, θ, the angle of the CM to CM vector with respect to the horizontal axis, and ℓ, the minimal distance between the contours. FIG. 2 shows

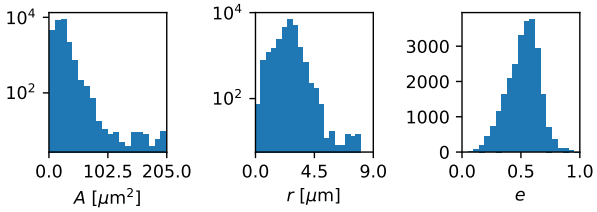

Figure 1: Histogram of node features (20 bins). For area and cell radius, a log scale is used in order to emphasize the outliers (very large cells).

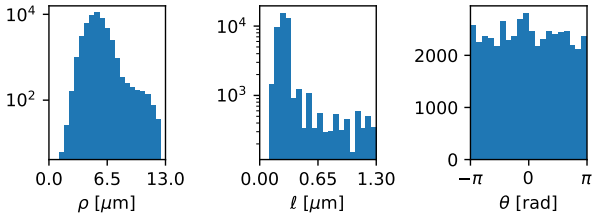

Figure 2: Histogram of edge features (20 bins). For CM to CM and cell contour distance, a log scale is used.

the distribution of these features. ρ shows a distribution peaked around $\langle \rho \rangle \approx 5.4 \pm 1.2$, which is in accordance with the average cell radius of $r \approx 2.5$ µm meaning two average cells would be separated by approximately $2r \approx 5.0$ µm. θ shows a near-uniform distribution, with mean and standard deviation $\langle \theta \rangle \approx 0 \pm 1.8$. Again, this is intuitive, since we don't expect cells to be preferentially aligned in one direction. Note that we allow θ to take values in $[-\pi, \pi]$ instead of $[0, \pi]$, because edges are not stored bidirectionally (to save memory), so the sign of θ encodes directionality of the edge. ℓ is strongly peaked around $\langle \ell \rangle \approx 0.34 \pm 0.22$ µm. Physically, ℓ is not an exact representation since cell edges are supposed to be in contact, and the mean being slightly offset is an artifact from the segmentation process.[1] Nevertheless, ℓ can be a good distance metric between cells, since it is independent of their sizes.

In total, 5 colonies × 180 images = 900 cell graphs are constructed, for a total of 25510 nodes and 48018 edges.

B. Assignment graph generation

Suppose we have generated two graphs $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$ storing extracted features from two segmentations. The goal is to track cells from $\mathcal{G}^{(1)}$ to $\mathcal{G}^{(2)}$, that is to match the nodes $\boldsymbol{v_i^{(1)}}$ of the first graph to nodes $\boldsymbol{v_a^{(2)}}$ of the second graph. An edge $\boldsymbol{e_{i,j}^{(1)}}$ is placed between nodes of the first graph if cells i and j are neighbours, and edges of the second graph $\boldsymbol{e_{a,b}^{(2)}}$ are constructed similarly. We store features obtained in subsection III-A in the respective nodes and edges.

[1] This artifact is a result from the watershed algorithm used to segment the output of YeaZ's CNN. The connectivity parameter is set to a nonzero value, resulting in gaps between neighouring cell masks.

The tracking problem is converted to a binary classification problem by constructing an assignment graph $\mathcal{G}^{(a;x,e)}$, composed of nodes $\boldsymbol{v}_{\boldsymbol{ia}}^{(\boldsymbol{a})}$ and edges $\boldsymbol{e}_{\boldsymbol{ia,jb}}^{(\boldsymbol{a})}$. Nodes $\boldsymbol{v}_{\boldsymbol{ia}}^{(\boldsymbol{a})}$ are obtained by concatenating features from $\boldsymbol{v}_{\boldsymbol{i}}^{(\boldsymbol{1})}$ and $\boldsymbol{v}_{\boldsymbol{a}}^{(\boldsymbol{2})}$. Edges $\boldsymbol{e}_{\boldsymbol{ia,jb}}^{(\boldsymbol{a})}$ of $\mathcal{G}^{(a)}$ link nodes $\boldsymbol{v}_{\boldsymbol{ia}}^{(\boldsymbol{a})}$ and $\boldsymbol{v}_{\boldsymbol{jb}}^{(\boldsymbol{a})}$ if $\boldsymbol{e}_{\boldsymbol{i,j}}^{(\boldsymbol{1})}$ and $\boldsymbol{e}_{\boldsymbol{a,b}}^{(\boldsymbol{2})}$ exist in $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$ respectively (i.e. cells i, j are neighbours, and a, b are neighbours), in which case the corresponding edge features are concatenated.

Finally, we construct the corresponding target graph $\mathcal{G}^{(a;y)}$ in a similar manner, where the node features are 2-dimensional one-hot encoded vectors, indicating whether a node $\boldsymbol{v}_{\boldsymbol{ia}}^{(\boldsymbol{a})}$ corresponds to a correct tracking. In other words, we want to have 1 if the tracking $i \to a$ is correct, and 0 otherwise. Note that at this point, there is no trace left of the cell ids that were used to differentiate cells in the segmentations.

Assignment graphs $\mathcal{G}^{(a;x,e)}$ and $\mathcal{G}^{(a;y)}$ then correspond to the GNN input and training target respectively. To create the dataset used for training, we take one frame of the movie, and generate assignment graphs with each of the (up to) 20 following frames. This is repeated for every frame in every colony.

IV. Methods

A. GNN structure

The GNN structure is summarized in Fig. 4. Node and edge features of $\mathcal{G}^{(a;x,e)}$ are embedded by two MLPs with ReLU activation. The resulting embedded graph is then convolved by multiple DeepGCN layers [1]. One DeepGCN layer consists of a batch normalization layer (LayerNorm), an activation function (ReLU), one dropout layer, one GENConv layer, then finally a residual connection layer. In particular, the GENConv layer constructs messages $\boldsymbol{x_i}'$ from node and edge features $\boldsymbol{x_i}, \boldsymbol{e_{i,j}}$ as

$$\boldsymbol{x_i}' = \text{MLP}(\boldsymbol{x_i}' + \sigma(\{\text{ReLU}(\boldsymbol{x_i} + \boldsymbol{e_{i,j}}) + \epsilon \mid i \in \mathcal{N}(j)\})),$$

where $\mathcal{N}(j)$ denotes indices of the nodes connected to node j, σ is the softmax function, and ϵ is a small learnable constant. The MLP has 2 layers, and preserves the number of channels of the input.

The final linear layer maps the embedded node channels to two channels, and the resulting graph $\mathcal{G}^{(a;\hat{y})}$ is then passed to CrossEntropyLoss along with the ground truth $\mathcal{G}^{(a;y)}$ for backpropagation.

B. Making predictions

Let n_1 and n_2 be the number of cells in the first and second image, respectively. We define an assignment matrix A with components $A_{i,a} = 1$ if cell i in the first image is the same as cell a in the second image, else $A_{i,a} = 0$. For the example in Fig. 3, the assignment matrix is

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

From the predicted graph $\mathcal{G}^{(a;\hat{y})}$, we can construct an assignment matrix by reshaping the node features into a matrix Z of shape (n_1, n_2, 2), where the last dimension stores weights for the classification decision. In other words, $Z_{i,a}$ stores the two classification weights of the tracking $i \to a$. By taking differences along the last axis of Z, we obtain a score matrix S of shape (n_1, n_2). If $S_{i,a} = Z_{i,a,0} - Z_{i,a,1} > 0$, the classification weight for "yes" is larger than for "no", and vice-versa. The magnitude of $S_{i,a}$ can therefore be interpreted as the GNN's confidence of the classification.

To transform S into A, the simplest approach might be to assign $A_{i,a} = 1$ if $S_{i,a} > 0$, and $A_{i,a} = 0$ otherwise. However, this can lead to situations where a cell in the first frame gets tracked to multiple cells of the second frame. Another approach might be to maximize S row-wise, in which case we ignore the fact cells can be flushed away between two frames. Maximizing S column-wise, we run into the problem that multiple cells of the first frame can get tracked to one cell of the second frame. Instead, we want to maximize the total score over all possible matchings, mathematically we find $A^\star$ which verifies

$$A^\star = \arg \max_{A \in \{0,1\}^{n_1 \times n_2}} \sum_{i,a} S_{i,a} A_{i,a}.$$

This is a linear sum assignment, which can be solved using `scipy.optimize.linear_sum_assignment` from the Scipy package [19].

For the following, we denote the predicted assignment matrix by $\hat{A} := A^\star$.

V. Results

This section discusses performance on the models trained with the constructed graphs, as described in section III. All models are trained with the Adam optimizer with varible exponentially decaying learning rate, adjusted manually depending on the batch size used. Unless specified otherwise, all layers use dropout (if it applies) during training with probability parameter 0.1. The random seed is fixed at the start of each training session. Training and validation datasets are shuffled at each epoch, and make up 80% and 20% of the total dataset respectively.

For evaluation, the assignment matrix predicted by the model, $\hat{A}$ is compared to the ground truth A using the following accuracy metric:

$$\text{acc}(\hat{A}, A) = \frac{1}{n_1 n_2} \sum_{i,j} \delta_{A_{i,j}, \hat{A}_{i,j}}.$$

A. MLP baseline

To assess whether the graph structure is actually informative for the tracking task, we train a MLP on the node features $\boldsymbol{X}$ of $G^{(a;x,e)}$. This task can be thought of as taking two cells from two different frames, and asking whether they are the same.

In order to not bias the MLP during training, we need to equilibrate the labels $\boldsymbol{y}$, because the number of nodes in the assignment graph grows as $n_1 n_2$, but the number of assignments where $y = 1$ grows as n_1. Therefore,

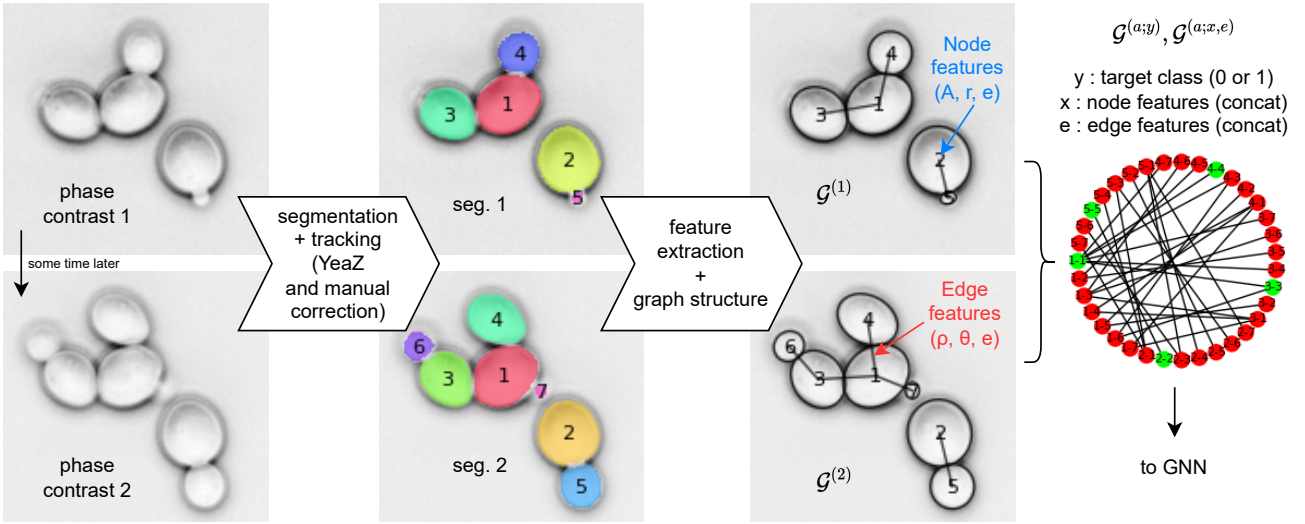

Figure 3: Data preprocessing pipeline. A movie of a budding yeast colony is taken (in phase contrast, as is the case here), then YeaZ [6] is used to segment individual frames, and semi-automatically track cells. Manual corrections to segmentation and tracking are applied as required. From the segmentations, geometric features are extracted, as well as features describing the neighborhood of each cell. This is stored in a graph structure $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$, from which an assignment graph $\mathcal{G}^{(a;x,e)}$ is built. A ground truth assignment graph $\mathcal{G}^{(a;y)}$ is also built. Both assignment graphs are then used by the GNN for training.

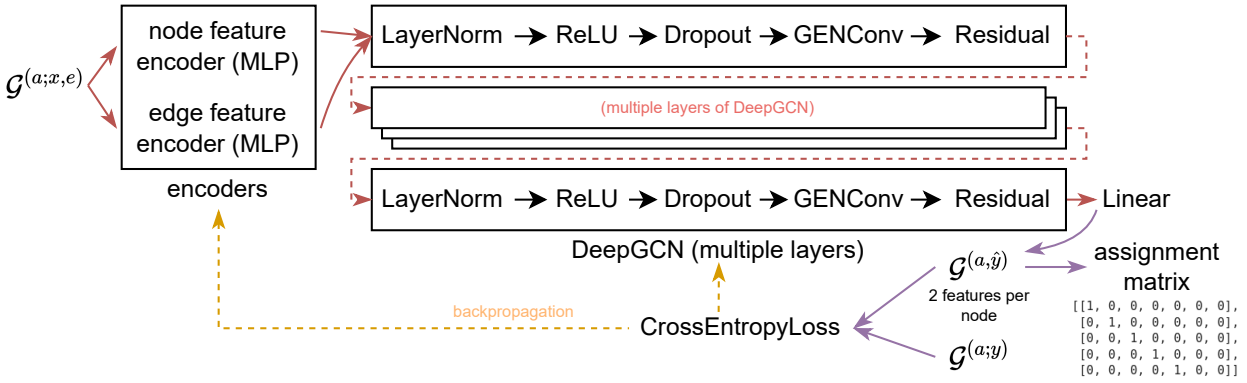

Figure 4: The GNN structure. For a given input graph, the node and edge features are encoded by MLPs. The resulting graph is then convolved multiple times following DeepGCN [1]. A final linear layer maps the encoded node features down to two, and CrossEntropyLoss is computed for backpropagation. For evaluation, the node features of $\mathcal{G}^{(a;\hat{y})}$ can be reduced and reshaped to build an assignment matrix.

labels $y = 1$ are underrepresented in large graphs, and equilibration is done by removing samples where $y = 0$ from the training batch, until both classes are equally represented.

Furthermore, batch normalization is applied before the MLP in order to improve training, as described in [20].

A hyperparameter scan was performed on the number of MLP layers and hidden channels, and found that the best performing MLP obtained an accuracy of approximately 0.88. More layers and hidden channels were found to marginally improve the accuracy, before it started to drop due to overfitting.

B. GNN performance

We perform a scan on the following hyperparameters : N_{enc} number of layers in both encoder MLPs, N_{conv} number of DeepGCN layers, and N_{dim} the number of hidden channels. Other hyperparameters, such as dropout, learning rate, exponential learning rate decay and training batch size were found to not significatively improve final performance, but can decrease training time if correctly chosen (here they were set to 0.1, $\sim 10^{-4}$, $\sim$ 0.98, $\sim$ 32 respectively). For the study, we chose $N_{\text{enc}} \in \{1, 2, 3, 4, 5\}$, $N_{\text{conv}} \in \{1, 2, 3, 4, 5, 8, 11\}$ and $N_{\text{dim}} \in \{30, 60, 90, 120\}$. FIG. 5 plots the results of the hyperparameter scan in parallel coordinates.

Models with less than 0.99 mean accuracy are obtained with few parameters overall, $N_{\text{enc}} \in \{1, 2, 3\}$, $N_{\text{conv}} \in \{1, 2, 3, 4\}$ and $N_{\text{dim}} \in \{30, 60\}$. For better accuracy, it was found most effective to increase the number of encoder layers $N_{\text{enc}} \in \{2, 3\}$, which systematically drove up

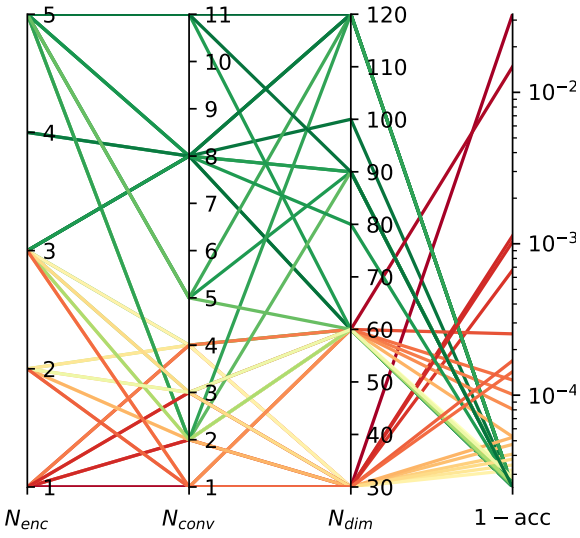

Figure 5: Parallel coordinates plot of the hyperparameter scan. The best performing model uses $N_{\mathrm{enc}} = 5$, $N_{\mathrm{conv}} = 11$, $N_{\mathrm{dim}} = 120$, but a very similar performance can be achieved for less parameters, $N_{\mathrm{enc}} \geq 3$, $N_{\mathrm{conv}} \geq 8$ and $N_{\mathrm{dim}} \geq 80$.

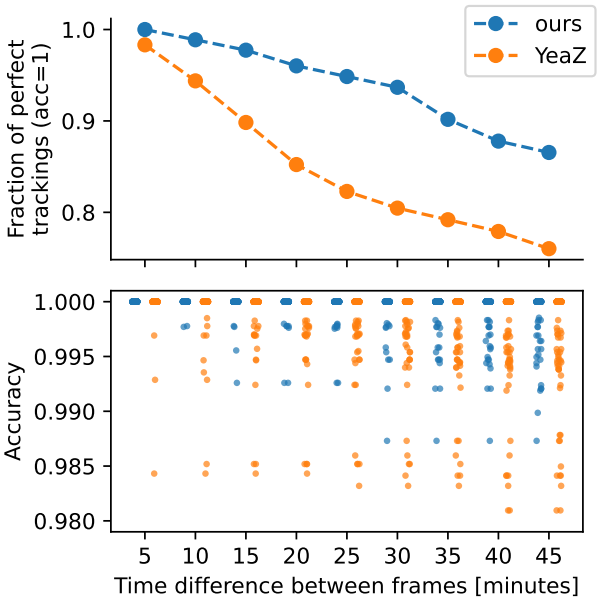

Figure 6: Accuracy strip-plot and fraction of perfect trackings as a function of time difference between frames. A perfect tracking is defined as a prediction containing no tracking mistakes. Note that this plot omits some accuracy outliers around 0.3, originating from YeaZ. At the minimal time difference of 5 minutes, YeaZ makes 3 mistakes (see FIG. 7), whereas our method makes none. The accuracy decays as time between frames increases, and the tracking task becomes harder. YeaZ's accuracy decays much faster than the GNN, showing our method is more stable, as well a performing better.

the mean accuracy up to 0.999. Following this, increasing N_{enc} only marginally improved accuracy, but allowing up to 5 convolution layers gave models obtaining mean accuracy of up to 0.9999. In this regime, the graph neural network is able to capture most of the complexity of the problem, but still struggles with edge cases (for instance, a cell moving a large amount). Further increasing the number of parameters (via N_{conv} or N_{dim}, they seem to compensate each other) results in even better accuracy, where edge cases are now correctly handled. The best models obtained a mean accuracy of 0.99997, and utilized $N_{\mathrm{enc}} \geq 3$, $N_{\mathrm{conv}} \geq 8$ and $N_{\mathrm{dim}} \geq 80$.

Intuitively, the performance gain of increasing N_{enc} eventually caps off, because each cell graph node or edge is decribed with only 3 features, and the MLP can only transform information it is given. Increasing N_{conv} intuitively can be seen as increasing the "communication distance" between cells, meaning each node has a "better view" of the whole graph structure. This of course comes with the cost of requiring more dimensions to store the information in, explaining the need to increase N_{dim}, and how both compensate each other. It is not useful to be able to "see" further in the graph (N_{conv} large) if you cannot "store" the information (N_{dim} small), and vice-versa.

We now compare the performance of the best GNN model ($N_{\mathrm{enc}} = 5$, $N_{\mathrm{conv}} = 11$, $N_{\mathrm{dim}} = 120$) against the built-in tracking system of YeaZ. For this, we use the test dataset on time differences of up to 45 minutes (half the cell cycle period of budding yeast), and compute the accuracy of YeaZ and our tracking method. FIG. 6 shows our method outperforms YeaZ for all time differences, and its accuracy decays more slowly as the frames become increasingly spaced in time. The graph structure therefore provides valuable information for tracking, especially in situations where images are taken infrequently.

VI. CONCLUSION

We have shown that this graph neural network model is able to learn the structure of a yeast cell colony, and is able to reliably track cells from frame to frame. The graph structure containing only simple geometrical information provides a good abstraction of the exact segmentation geometry. It has been shown that the model exceeds YeaZ's integrated tracker performance, using a relatively small training dataset of only 4 colonies growing over 15 hours and imaged every 5 minutes. Using our method, tracking tasks will become easier and significantly less time-consuming, especially for large colonies.

The pre-trained model presented in the results can be downloaded on the github repository[2] as well as a command-line interface, and a Python notebook demonstrating the full pipeline from loading and pre-processing data, to generating graphs and running predictions.

One of the limitations of this model is the need to have segmented the images and verified the segmentations. We

[2] https://github.com/ninivert/bread-tracking

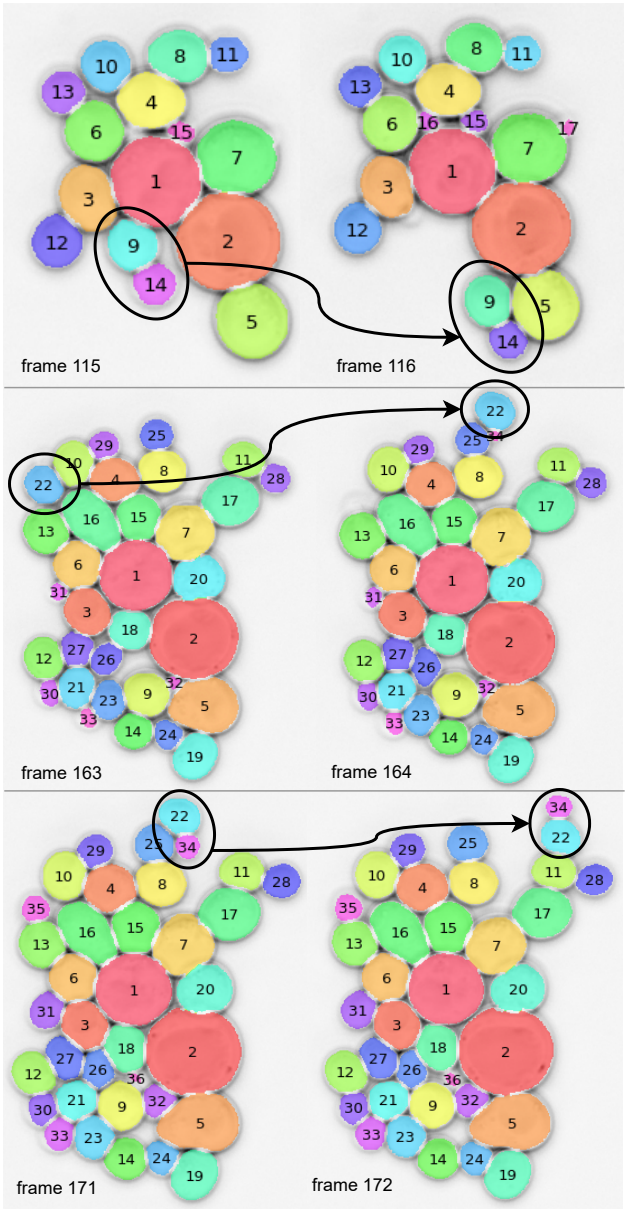

Figure 7: Segmentations from the test dataset highlighting moving cells. Out of the 179 interframes, YeaZ makes 3 tracking mistakes. On frame 115, cells 9 and 14 move together towards cell 5, and YeaZ tracks cell 9 to cell 16 and cell 14 to cell 9. On frame 163, cell 22 moves around the top of the colony, and YeaZ tracks cell 10 to 22, cell 10 to 29, cell 29 to cell 25 and cell 25 to 34. On frame 171, cells 22 and 34 move together around the top while swapping places, and YeaZ tracks cell 22 to 34 and vice-versa.

however argue that thanks to advancements in this domain [6], [21], this has become a trivial task.

The model presented in this report uses only two graphs as an input, and could benefit from having a ”longer history” of the colony in order to make better tracking predictions. Multiple papers [22], [23], [24] address the problem of temporal graphs, and it might be worth investigating to further the research. In summary, using temporal graph networks are a promising approach to constructing lineages from only phase contrast (or bright field) microscopy movies. This would likely have to be coupled to an attention system, for which literature and implementations for graph neural networks already exist [25], [26].

References

[1] G. Li, C. Xiong, A. Thabet, and B. Ghanem, "DeeperGCN: All You Need to Train Deeper GCNs," *arXiv:2006.07739 [cs, stat]*, jun 2020, arXiv: 2006.07739. [Online]. Available: http://arxiv.org/abs/2006.07739

[2] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph Neural Networks: A Review of Methods and Applications," *arXiv:1812.08434 [cs, stat]*, oct 2021, arXiv: 1812.08434. [Online]. Available: http://arxiv.org/abs/1812.08434

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[4] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," *arXiv:1903.02428 [cs, stat]*, apr 2019, arXiv: 1903.02428. [Online]. Available: http://arxiv.org/abs/1903.02428

[5] A. S. Chowdhury, R. Chatterjee, M. Ghosh, and N. Ray, "Cell tracking in video microscopy using bipartite graph matching," pp. 2456–2459, 2010.

[6] N. Dietler, M. Minder, V. Gligorovski, A. M. Economou, D. A. H. L. Joly, A. Sadeghi, C. H. M. Chan, M. Koziński, M. Weigert, A.-F. Bitbol, and S. J. Rahi, "A convolutional neural network segments yeast microscopy images with high accuracy," *Nature Communications*, vol. 11, no. 1, p. 5723, nov 2020. [Online]. Available: https://www.nature.com/articles/s41467-020-19557-4

[7] K. E. Magnusson, J. Jalden, P. M. Gilbert, and H. M. Blau, "Global linking of cell tracks using the viterbi algorithm," *IEEE Transactions on Medical Imaging*, vol. 34, no. 4, p. 911–929, 2015.

[8] D. E. Hernandez, S. W. Chen, E. E. Hunter, E. B. Steager, and V. Kumar, "Cell tracking with deep learning and the viterbi algorithm," pp. 1–6, 2018.

[9] T. He, H. Mao, J. Guo, and Z. yi, "Cell tracking using deep neural networks with multi-task learning," *Image and Vision Computing*, vol. 60, 11 2016.

[10] J.-B. Lugagne, H. Lin, and M. J. Dunlop, "Delta: Automated cell segmentation, tracking, and lineage reconstruction using deep learning," *PLOS Computational Biology*, vol. 16, no. 4, pp. 1–18, 04 2020. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1007673

[11] C. Payer, D. Å tern, M. Feiner, H. Bischof, and M. Urschler, "Segmenting and tracking cell instances with cosine embeddings and recurrent hourglass networks," *Medical Image Analysis*, vol. 57, pp. 106–119, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S136184151930057X

[12] J. Hayashida and R. Bise, "Cell tracking with deep learning for cell detection and motion estimation in low-frame-rate," p. 397–405. [Online]. Available: https://doi.org/10.1007/978-3-030-32239-7_44

[13] E. Moen, E. Borba, G. Miller, M. Schwartz, D. Bannon, N. Koe, I. Camplisson, D. Kyme, C. Pavelchek, T. Price, T. Kudo, E. Pao, W. Graf, and D. Van Valen, "Accurate cell tracking and lineage construction in live-cell imaging experiments with deep learning," *bioRxiv*, 2019. [Online]. Available: https://www.biorxiv.org/content/early/2019/10/14/803205

[14] D. Salem, Y. Li, P. Xi, H. Phenix, M. Cuperlovic-Culf, and M. Kærn, "Yeastnet: Deep-learning-enabled accurate segmentation of budding yeast cells in bright-field microscopy," *Applied Sciences*, vol. 11, no. 6, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/6/2692

[15] H. T. Kruitbosch, Y. Mzayek, S. Omlor, P. Guerra, and A. Milias-Argeitis, "A convolutional neural network for segmentation of yeast cells without manual training annotations," *Bioinformatics*, vol. 38, no. 5, pp. 1427–1433, 12 2021. [Online]. Available: https://doi.org/10.1093/bioinformatics/btab835

[16] C. Stringer, M. Michaelos, and M. Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," *bioRxiv*, 2020. [Online]. Available: https://www.biorxiv.org/content/early/2020/02/03/2020.02.02.931238

[17] G. DeZoort, S. Thais, J. Duarte, V. Razavimaleki, M. Atkinson, I. Ojalvo, M. Neubauer, and P. Elmer, "Charged particle tracking via edge-classifying interaction networks," *arXiv:2103.16701 [hep-ex]*, nov 2021, arXiv: 2103.16701. [Online]. Available: http://arxiv.org/abs/2103.16701

[18] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv:1609.02907 [cs, stat]*, feb 2017, arXiv: 1609.02907. [Online]. Available: http://arxiv.org/abs/1609.02907

[19] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[20] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167 [cs]*, Mar. 2015, arXiv: 1502.03167. [Online]. Available: http://arxiv.org/abs/1502.03167

[21] C. Stringer, M. Michaelos, and M. Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," Tech. Rep., feb 2020, type: article. [Online]. Available: https://www.biorxiv.org/content/10.1101/2020.02.02.931238v1

[22] U. Singer, I. Guy, and K. Radinsky, "Node Embedding over Temporal Graphs," *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 4605–4612, aug 2019, arXiv: 1903.08889. [Online]. Available: http://arxiv.org/abs/1903.08889

[23] W. Jin, M. Qu, X. Jin, and X. Ren, "Recurrent Event Network: Autoregressive Structure Inference over Temporal Knowledge Graphs," *arXiv:1904.05530 [cs, stat]*, oct 2020, arXiv: 1904.05530. [Online]. Available: http://arxiv.org/abs/1904.05530

[24] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal Graph Networks for Deep Learning on Dynamic Graphs," *arXiv:2006.10637 [cs, stat]*, oct 2020, arXiv: 2006.10637. [Online]. Available: http://arxiv.org/abs/2006.10637

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *arXiv:1710.10903 [cs, stat]*, Feb. 2018, arXiv: 1710.10903. [Online]. Available: http://arxiv.org/abs/1710.10903

[26] S. Brody, U. Alon, and E. Yahav, "How Attentive are Graph Attention Networks?" *arXiv:2105.14491 [cs]*, Oct. 2021, arXiv: 2105.14491. [Online]. Available: http://arxiv.org/abs/2105.14491

VII. Appendix

A. Python environment

Training was performed using the following PyTorch ecosystem,

`pytorch-lightning`	`1.7.5`
`torch`	`1.12.1`
`torch-cluster`	`1.6.0`
`torch-geometric`	`2.1.0.post1`
`torch-scatter`	`2.0.9`
`torch-sparse`	`0.6.15`
`torch-spline-conv`	`1.2.1`
`torchmetrics`	`0.10.0`

and data processing using the following Python scientific packages.

`opencv-python`	`4.5.5.64`
`numpy`	`1.21.5`
`scipy`	`1.7.3`
`pandas`	`1.3.5`

Refer to `conda_env.yaml` on the Github repository for full details.

B. Details of the microscope

Images were recorded using a Nikon Ti2-E microscope equipped with a 60x objective and a Hamamatsu Orca-Flash 4.0 camera. The microscope was operated using NIS-Elements software and the objective's axial position was controlled by Nikon Perfect Focus System. Images were taken every 5 min, with 100 ms exposure time for the the brightfield, GFP and mCherry channel.

C. Details of the strain

We used a W303-derived budding yeast strain with the following genotype: MATa ADE2 MYO1::MYO1-GFP::kanMX, HTB2::HTB2-mCherry::HIS5.

D. Details of media and microfluidics chip used for growth

Cells were grown in CellASIC ONIX plate for haploid yeast cells in media controlled by the ONIX2 microfluidics system (Merck, Germany). We used standard synthetic complete medium with 2% (v/v) glucose.