Project in Information Engineering: Cell Feature Extraction

Siyue Huang, Siyi Qian, Hang Qin, Qi Xiong, Chenting Zhang

Abstract

In this study, we present a comprehensive framework for the detection, information extraction, and tracking of cells in sequential bright field microscopy images. The methodology encompasses three distinct phases. Initially, we focus on cell localisation by meticulously annotating a cell dataset and employing the YOLOv5 algorithm for precise object detection. After localization, we design a state classifier based on a convolutional neural network (CNN), providing a nuanced analysis of cell states. In the final phase, based on YOLO detection results we design a cell tracking algorithm utilizing the Hungarian algorithm for effective cell association, complemented by the Kalman Filter to ensure robust cell tracking across image sequences. Experimental results show that our approaches can recognize cell states with high accuracy and achieve efficient cell position tracking. Compared with our baseline Baxter Algorithm, our model also has better cell detection and tracking performance.

I. Introduction

Bright field microscopy is a widely used technique to capture magnified images of cells in a simple, cheap, and non-invasive manner. The ability to extract information from cells from a sequence of bright field images has significant applications in both scientific and industrial settings, especially because it provides valuable insights into fundamental cellular processes, disease diagnosis, drug discovery, and cell biology studies [1]. The field of automated cell tracking has contributed extremely valuable tools to life scientists with which to conduct their research [**?**], and it has been a challenging task for over a decade. However, cell tracking based on bright field images is a more difficult task. The bright field images often suffer from low contrast, primarily because most cells are thin and transparent, making it challenging to extract essential information from the images that characterize the cell behaviour. As such, the development of robust and efficient algorithms for tracking and extracting features from cells from a sequence of bright field images is crucial for understanding their dynamic behaviour within the well. In this regard, this project aims to propose advanced image-processing algorithms/techniques to detect, track, and extract essential features of moving cells in time-lapsed sequences of bright field imaging.

In the previous studies [2], two traditional approaches of cell tracking are mainly used, which are the model-based tracking method and tracking by detection. The first method uses parametric models to represent cell boundaries. The model is updated in each frame. It gives a good performance when the image sequences have a high resolution and a high spatial magnification. However, the performance drops once the cells touch or overlap each other. The second method first does cell detection in each image frame and then associates these detections to get the tracks. This method is more popular because it is intuitive. The challenges faced by this method are the uncertain cell correspondence and the errors in cell detection. In this study, we choose the second method for cell tracking. Since the errors in cell detection have a lot of influence on tracking results, a robust cell detection model is needed.

Existing deep learning algorithms have shown remarkable potential in automating cell detection and improving the process of extracting essential cell features in static images [3]. YOLO [4] is a kind of pre-trained detection model that is perfectly suitable for cell detection, and it is usually combined with other tracking methods to perform automated object tracking via visual data. The ability of YOLO for cell tracking in also been proved in the previous research [2]. YOLO shows a great ability to track multiple cells in time-lapse microscopy image sequences containing thousands of images. Deep learning-based methods can not only be used for cell detection but also show good ability in cell cycle state recognition. Convolutional Neuron Network (CNN) based method has been successfully implemented for cancer cell detection and its stages recognition of life cycle [5]. Apart from the cell detection and cycle state recognition method, an efficient tracking model is also essential to achieve our goal of cell tracking. The combination of YOLO and Kalman Filter has shown incredible ability in object tracking [6], [7]. Kalman Filter is an explainable mathematics algorithm for tracking, it does not require training and it is extremely fast compared to the Recursive Neuron Network. Therefore, we consider using Kalman Filter as our tracking algorithm.

In this project, we implement a cell tracking and cycle state estimation method based on YOLO, Kalman Filter, and CNN. We first detect the position of cells from each frame in an image sequence and design a CNN-based network to estimate the cells' life cycle states. Finally, the Kalman Filter is combined with the detection result for cell position tracking. The remaining arrangements for reporting are as follows. The second chapter covers our initial approaches. The third chapter describes the main solution. The results are presented in the fourth chapter, and the last chapter gives a conclusion of this project.

II. Initial Approaches

A. Cell position

1) Baxter Algorithm: The Baxter Algorithm (BA) is a software package for tracking and analysis of cells in microscope images[8]. The software has shown excellent performance in the ISBI Cell Tracking Challenges of 2013, 2014, and 2015. It can handle images produced using either 2D transmission microscopy (e.g. bright field, phase contrast, and differential interference contrast (DIC)) or fluorescence microscopy (e.g. wide field, confocal, and light sheet). The analysis of transmission microscopy is limited to 2D, but 3D stacks of fluorescent images can be processed. In addition to cell tracking, the BA can perform automated analysis of fluorescent histological sections of muscle tissue, and automated analysis of myoblast fusion.

Therefore, the Baxter Algorithm is an ideal tool to label our large training dataset of our model and is also chosen as a baseline algorithm to evaluate our cell tracking algorithm.

2) Labeling task: We have a dataset of 55 time series bright field images, and each folder contains 847 images for 3 minutes of continuous shooting. We plan to use the Baxter Algorithms to label them and generate the training and test set. However, some problems come to us when we implement it.

Firstly, the labels generated by the Baxter Algorithm are unsuitable for our following neural network training job. The main problem of this is that the cell detection is tracked based on the Viterbi algorithm, which sequences detection in a manner that if a cell is not identified in the initial frame, it remains undetected in subsequent images. We tried to label several 3-minute time series by BA and the cell misdetection occurs frequently. This limitation necessitates manual

intervention to ensure comprehensive labelling. Additionally, the output format of labels generated by the Baxter Algorithm is incompatible with the model we want to train. Consequently, there will be more work to reformat the data into a structure that YOLO can interpret for object detection tasks. Considering these problems, we choose to manually label our bright field images instead of labelling them by Baxter Algorithm.

3) YOLOv5: Since our goal is focused on detecting the cells and finding the position of the cells in the image to be able to further track the cells, the YOLO series can perform this work pretty well. Additionally, YOLO models excel in transfer learning and adaptation, making them highly flexible for specialized object detection. Initially, the YOLO model was trained on large COCO datasets such that they learn to recognize various objects. Subsequently, they can be fine-tuned on more specific datasets, tailoring them to particular tasks or environments, the cell class in our case. This process ensures YOLO models maintain accuracy and effectiveness in diverse applications, from general to specialized scenarios.

B. Cell State

1) U-Net: In the previous work, the U-Net, as one type of convolutional network, can be used for biomedical image segmentation. The U-Net architecture is composed of a contracting path, which helps to capture the context, and a symmetric expanding path, which can help achieve precise localization [9]. The cons of using the U-Net is that it can train on the annotated data more efficiently to do the data segmentation, thus having less requirement for the size of training data.

U-Net can play an important role in achieving our goal of cell state estimation. Cells in different states will primarily have different shapes. U-Net can be used to accurately segment cells in an image, or in other words, extract the position and shape of each cell. The data of cell shapes provide detailed information about cells and will help with cell state estimation. However, the algorithm used to achieve cell state estimation from shape data still needs to be developed.

2) CNN Classifier: CNN classifier is a convolutional network model for image classification tasks. The primary goal of the CNN classifier is to divide input images into predefined categories, rather than performing pixel-level image segmentation as implemented in U-Net. The training data will be cell images with their states labelled. The output of the trained CNN classifier should be the state labels of the cell image. Since the cell state estimation task mainly involves classifying the cells without the need for detailed pixel level segmentation and we can get the dataset of single cell images from the last step of Yolov5, the CNN classifier may be a relatively simpler and more efficient option for cell state estimation.

C. Cell Tracking

1) Data Association: Data association is a step that determines the relationship between objects from different frames. It is a key step used to correlate features of the same target at different frames to form a trajectory of the target object. In the cell tracking task, a well-developed data association step will generate the cell position correspondence relationship between two frames with the least mismatching rate, thereby connecting cells in a time series and eliminating the noise influences to the greatest extent.

Two common data association methods are the Nearest Neighbor Algorithm and the Hungarian Algorithm. The Nearest Neighbor Algorithm is a simple data association method that matches

each target with its closest neighbour intuitively and is easy to implement. However, it has environmental limitations. When dealing with large-size targets or dense environment targets, it may be affected by ambiguity and have a higher mismatching rate [10]. The Hungarian Algorithm was first proposed by Harold Kuhn in 1955, solving the problem of minimizing the overall matching cost to determine the best match, based on the concept of bipartite graph matching in graph theory [11]. The Hungarian Algorithm can find globally optimal matches rather than just locally optimal ones.

2) Forecasting Using the Kalman Filter: The Kalman Filter[12] is a mathematical model that can estimate unknown variables with high accuracy, given the previous observations in a time series. One common application of the Kalman Filter is vehicle guidance, navigation, and control. It is also widely used for trajectory prediction.

The advantage of using the Kalman Filter is that it has a good performance on moving object's position estimation especially when the moving state of the object is changing. Since the cells are quite small and the moving pattern of a cell under microscopy is close to random moving, the Kalman Filter is undoubtedly suitable for our case.

In our case, we use the Kalman Filter to estimate the cell position of the next frame, given the current observation and the last estimation.

III. Main Solutions

In this chapter, the main solutions of our project are presented. A YOLOv5 pre-trained model is fine-tuned on our cell image data, A CNN-based cell cycle state classifier is trained and tested based on the YOLO detection results. A Kalman Filter-based cell position tracking algorithm is designed and combined with YOLO.

A. YOLO V5

1) General description: YOLOv5 (You Only Look Once, version 5) is an advanced deep-learning model used for real-time object detection. It's part of the YOLO family of models, which are known for their speed and accuracy in detecting objects in images and videos. Figure 1[4] illustrated the basic idea of YOLOv5. YOLOv5, as well as other models in the YOLO family, first processed the image through an input layer and then sent it to the backbone network for feature extraction. After obtaining the feature maps of different sizes, the selected features were sent to the fusion network to generate the final feature maps of size 80×80, 40×40, 20×20 for detecting small, medium, and large objects in the picture. After sending to the prediction step, the bounding boxes are generated.

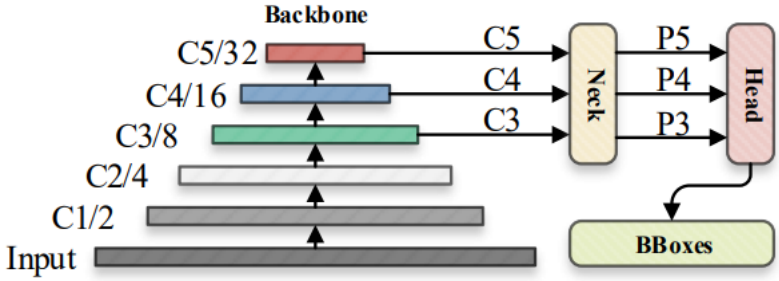

Fig. 1: Basic flowchart of YOLOv5[image source: [4]]

These bounding boxes are defined by five predictions:

- **x**: The x-coordinate of the center of the box relative to the bounds of the grid cell.
- **y**: The y-coordinate of the center of the box relative to the bounds of the grid cell.
- **width**: The width of the box relative to the whole image.
- **height**: The height of the box relative to the whole image.
- **confidence**: The confidence score reflects how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts.

Particularly, the confidence score for each bounding box is computed as the probability that the box contains an object multiplied by the Intersection over Union (IoU) between the predicted box and the ground truth. This score is a measure of the overlap between the predicted bounding box and the ground truth, with a higher score indicating a better prediction.

$$\text{Confidence Score} = \text{P(Object)} \times \text{IoU(pred, truth)}, \tag{1}$$

where P(Object) is the probability that there is an object in the box, and IoU(pred, truth) is the intersection over Union between the predicted box and the ground truth box.

After computing the confidence score, the YOLOv5 implements a process called non-max suppression(NMS) which filters out the bounding boxes with confidence scores below a threshold as well as overlapping boxes that are above one threshold. These parameters will be further discussed in the following validation part.

Additionally, for multi-class prediction, a conditional class probability is defined. It is denoted as $\text{P}(\text{Class}_i|\text{Object})$, which indicates the probabilities of the detected object belonging to a particular class.

The final score for each class is given by:

$$\text{Final Score}_{\text{class}_i} = \text{Confidence Score} \times \text{P}(\text{Class}_i|\text{Object}) \tag{2}$$

2) Generate our labelled cell dataset: In this project, we employed an AI annotation tool Robotflow[13] to do our labelling task. We defined a new class called cell. In total, 912 original images are labelled. We also split the labelled dataset in a proportion of 70%, 20%, and 10% and assigned them as training, testing and validation datasets respectively. Additionally, for the sake of reducing tedious manual labour, we also implemented data augmentation skills, meaning we enlarged our training dataset by flipping, rotating and cropping skills such that the training dataset is now three times the size of the original one.

After generating the dataset, we convert the label to the format specified for the YOLO model and write in a txt file containing the Class ID as well as the bounding box's x center, y center, width and height.

3) Training phase: Since our training dataset has a limited size and we also want to speed up our training process with limited computational resources, we did not train the YOLOv5 model from scratch, instead, we loaded the pre-trained initial weights from yolov5s.pt. And then we fine-tuned our model based on our customed dataset.

Thus, the hyperparameters for training the initial model stay the same. For our cell class training process, we define a few hyperparameters. First of all, we indicate that the images fed into the model have a resolution of 1388 pixels. The training process is set to run in batches of 4. The model is scheduled to undergo 10 training epochs, where an epoch represents one complete cycle through the full dataset, allowing the model to learn from the data iteratively

and refine its ability to detect objects. The dataset and model configurations are drawn from our defined data.yaml file which contains the classes of objects to be detected as well as the paths to the training and validation datasets. It is illustrated in table 1.

Within the YOLOv5 architecture, BCELoss (Binary Cross Entropy Loss) is utilized in YOLOv5 binary classification tasks. In binary classification, the goal is to predict an output that is either 0 or 1. BCELoss is particularly well-suited for problems where the model needs to output a probability belonging to one of two classes.

For a single data point, the BCELoss is defined as:

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \tag{3}$$

where y is the true label which can be either 0 or 1, $\hat{y}$ is the predicted probability between 0 and 1, representing the model's confidence that the input should be classified as class 1. The BCELoss takes the log of the predicted probability if the true label is 1, or the log of one minus the predicted probability if the true label is 0, then applies a negative sign. The loss is minimized when the predicted probability $\hat{y}$ is close to the true label y. Conversely, the high discrepancy between the true label and the predicted probability will result in a large loss which thus penalizes the model.

4) Validation phase: After we trained our model, we validated our model in an entire image sequence which contains 465 images. We adjust two parameters model.conf to be 0.15 and model.iou to be 0.3.

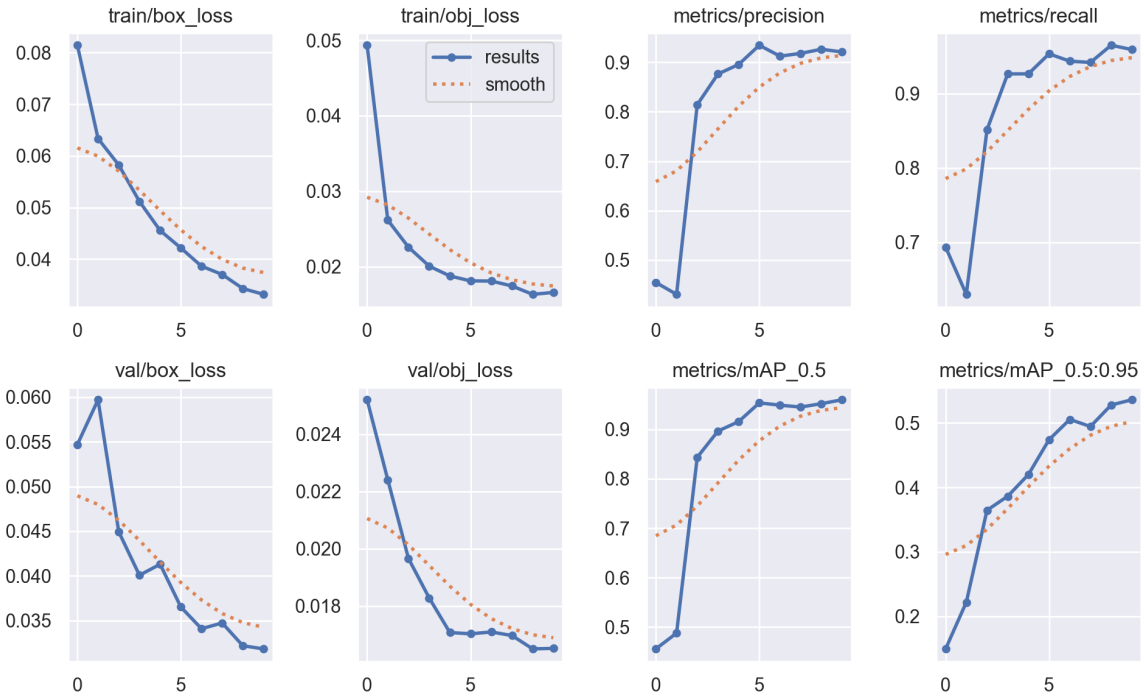

Fig. 2: Training result on our customized cell dataset

- **train/box_loss**: train/box_loss represents the loss associated with the bounding box predictions during training. The figure shows a decreasing tendency which means the model performs well on this indicator.
- **train/obj_loss**: train/obj_loss Shows the object loss during training, which is the loss related to the objectness score that a predicted box actually contains an object. The figure shows a decreasing tendency which means the model performs well on this indicator.
- **metrics/mAP_0.5, metrics/mAP_0.5:0.95**: Mean average precision (mAP) is essential for evaluating object detection models. The mAP compares the ground-truth bounding box to the detected box and returns a score. Metrics/mAP_0.5 shows the mean Average Precision at the Intersection over Union (IoU) threshold of 0.5. The figure shows that the mAP value could improve up to 0.9 within the first few epochs. However, when IoU sets up to 0.95, it is more challenging for the model to meet the more stringent IoU thresholds and in the first 10 epochs, the mAP can only reach 0.5.

Hyperparameter	Value
Learning Rate	0.001
Batch Size	4
Number of Epochs	10
model.conf	0.15
model.iou	0.3

TABLE I: Hyperparameters of our trained YOLO Model

B. CNN

1) General description: CNN is a good choice for recognizing cell states from images by automatically learning hierarchical features directly from the input data. At the core of CNNs are convolutional layers, which employ filters or kernels to scan input images for local patterns. These filters capture spatial hierarchies of features, learning low-level features like edges and textures in early layers and more complex features in deeper layers. This hierarchical feature learning allows CNNs to robustly recognize objects and patterns invariant to translation, rotation, and scale.

CNN architectures typically consist of multiple convolutional layers, followed by pooling layers to reduce spatial dimensions and decrease computational complexity. Fully connected layers at the end of the network combine learned features for final classification. Besides tasks of image classification, CNN is extending to various domains such as object detection, segmentation, and even natural language processing.

2) Training and testing phase: We want to use YOLO detection results, the pixels inside the bounding box, to train the CNN classifier. However, the bounding box is not the same size for different shapes of cells in different states. Most cells in State G are small regular circles, but some cells in State M are hanging around with long tails. So it is necessary to adjust the input data to have the same size by keeping the center and changing the width and height of the bounding box to 90 x 50 pixels. Then these cropped cell images are manually classified as State G, State M, and State S, as shown in figure 3. Three CNN models are trained as cell cycle state classifiers for each state based on our dataset.

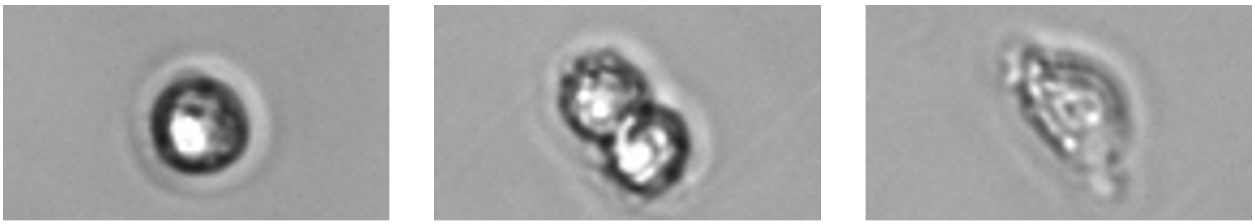

Fig. 3: Examples of cropped cell images on different states

C. Kalman Filter

In this project, the inputs of the Kalman Filter are the current cell position observation and the estimation of the last frame. The output is the estimation of the next frame.

The mathematical model of the Kalman Filter is shown as follows.

Model: Linear Gaussian State Space

$$x_{k+1} = A_k x_k + f_k u_k + w_k, \quad x_0 \sim \pi_0 \tag{4}$$

$$y_k = C_k x_k + g_k u_k + v_k. \tag{5}$$

$w_k \sim \boldsymbol{N}(0, Q_k), v_k \sim \boldsymbol{N}(0, R_k), \pi_0 \sim \boldsymbol{N}(\hat{x_0}, \Sigma_0)$.

Here x_k is the real value at k frame, y_k is the observation at k frame, A_k is the state transition model, u_k is the control input, C_k is the observation model, and w_k, v_k are the noises.

The implementation of the Kalman Filter in our cell tracking algorithm is shown as follows.

Kalman Filter equations: (for conditional mean and covariance):

$$\hat{x}_{k+1|k} = A_k \hat{x}_k, \tag{6}$$

$$\Sigma_{k+1|k} = A_k \Sigma_k A_k^{'} + Q_k \tag{7}$$

$$S_{k+1} = C_{k+1} \Sigma_{k+1|k} C_{k+1}^{'} + R_{k+1}, \tag{8}$$

$$\hat{x}_{k+1} = \hat{x}_{k+1|k} + \Sigma_{k+1|k} C_{k+1}^{'} S_{k+1}^{-1} (y_{k+1} - y_{k+1|k}), \tag{9}$$

$$\Sigma_{k+1} = \Sigma_{k+1|k} - \Sigma_{k+1|k} C_{k+1}^{'} S_{k+1}^{-1} C_{k+1} \Sigma_{k+1|k}. \tag{10}$$

$$\tag{11}$$

Here we use the detection result from YOLOv5 as our observation, thus we do not need to update $y_{k+1|k}$. Since the cells are moving on their own, there is no outside input in our model. Thus $f_k u_k$ is removed. The $\Sigma_{k+1|k} C_{k+1}^{'} S_{k+1}^{-1}$ is the Kalman gain, which is used to decide how much the estimation should be changed given the current measurement. Q_k is the covariance noise matrix. It is a small number added to the real covariance.

D. Tracking Algorithm

In this section, the tracking algorithm is presented. After we obtain cell position from YOLO, we design the following tracking algorithm based on YOLO prediction, Kalman Filter, and Hungarian algorithm. The following pseudo-code shows the whole algorithm for cell tracking.

Algorithm 1 Cell tracking algorithm

An image sequence with K frames, for each frame $f_k, 1 \leq k \leq K$, we observe N cells from YOLO: $y_k^1, y_k^2, ..., y_k^n, ..., y_k^N$, and we have M prediction cells from the previous frame: $x_k^1, x_k^2, ..., x_k^m, ..., x_k^M$

A Kalman Filter KF can calculate the predicted cell position for the next frame, from the current observation and last prediction. $x_{\mathrm{k}+1}^m, \Sigma_{\mathrm{k}+1}^m = \mathrm{KF}(x_{\mathrm{k}}^m, y_{\mathrm{k}}^n, \Sigma_k^m)$

Initialize $x_0^m = (0,0), \Sigma_0^m = I_2$, calculate $x_1^m, \Sigma_1^m = \mathrm{KF}(x_{\mathrm{k}}^m, y_{\mathrm{k}}^n, \Sigma_k^m), k = 1$
  while $k \leq K$ **do**
    $association\_index = \mathrm{Huangarian}(y_k, x_k)$
    $association\_index$ is the matched index pairs for m and n
    $k \leftarrow k + 1$
    if $M \neq N$ **then**
      if $M < N$ **then**
        Some cells are divided, for unassociated y_k^n, find their parents based on nearest neighbour: $m^* = \arg\min_m \|y_k^n - x_k^m\|$
      else
        YOLO missed some cells, for the unassociated x_k^m, just copy themselves as their matched y_k^n
      end if
    end if
    for each associated (m, n) **do**
      Compute predicted position and update covariance matrix,
      $x_{\mathrm{k}+1}^m, \Sigma_{\mathrm{k}+1}^m = \mathrm{KF}(x_{\mathrm{k}}^m, y_{\mathrm{k}}^n, \Sigma_k^m)$, where $(x_{\mathrm{k}}^m, y_{\mathrm{k}}^n)$ is matched.
    end for
  end while
  return all $association\_index$ and all x_{k}^m

For every frame in the time series images, we will have two sets of data, the observed N cells y_k from YOLO and the predicted M cells x_k from the previous frame. There are many reasons why there exists a deviation between the observed N number and the predicted M number. It may be caused by the observation generated by YOLO missing some cells, or some cells just dividing in the frame. That's why we need to combine the x_k and y_k to get the prediction for the next frame.

To make the algorithm more general and improve fault tolerance, we will look into the 3 cases based on the relation between N and M numbers. For the case where $N = M$, we simply assume that the two lists of cells can correspond to each other perfectly and no more additional operation is needed. For the case where $M < N$, we assume that the YOLO does the correct detection then the case indicates that some cells are divided. We will find a parent for the newborn cell using the nearest neighbour matching. For the case where $M > N$, we just consider YOLO missing some cells. To solve this problem, fortunately, we have the predicted cells from the last frame given by the Kalman Filter. Usually, the cells move relatively slowly frame by frame, hence the predicted cell position should be close to the missing observation, which makes it logical to use

predictions from the previous frame to compensate for missing observations. The if statement in the algorithm demonstrates this idea.

After adjusting the data association results, the algorithm repeats the loop calculation for each frame, and the tracking results in obtained.

IV. Results

In this chapter, the results of the studies are presented. The results can be divided into three parts, cell detection results, cell cycle state recognition results, and cell tracking results.

A. Cell Detection

The cell detection results given by our trained YOLOv5 model are the coordinates of the bounding box, where inside the bounding box is the object that YOLO has detected. To show the detection results intuitively, we plot the bounding box directly upon the image data. Moreover, to compare the advantages of YOLO, we compare the detection performance with the Baxter Algorithm. From figure 4, our YOLO model can detect cells that the Baxter Algorithm cannot find, which is of great help to the downstream tracking task.

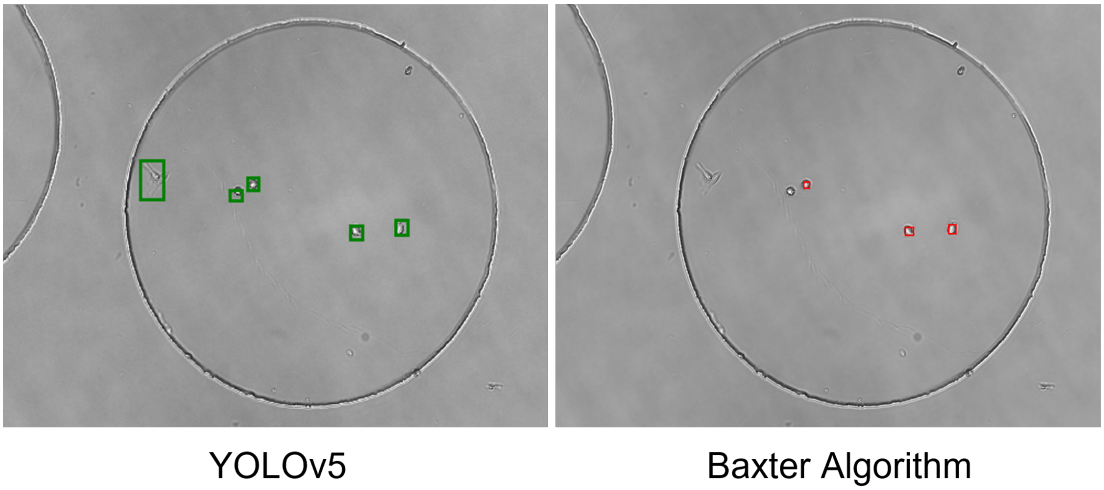

Fig. 4: Cell detection results of YOLO and Baxter Algorithm

B. Cell State

The cell state recognition is a classification task. To show the classification performance of our CNN-based cell state classifier, the confusion matrix on test data is shown in figure 5. As we can see from the figure, the classes are extremely unevenly distributed. Most of the cells are in State G and only a few cells are in State S. However, our CNN-based state classifier still shows great accuracy among the three classes. 16 of the 18 cells in the S state were accurately predicted, and the overall classification accuracy is over 92%.

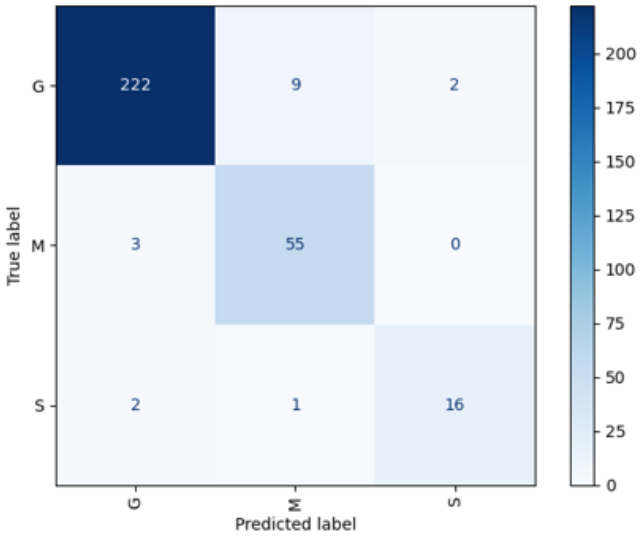

Fig. 5: Confusion matrix for cell state classification

C. Cell Tracking

To evaluate the performance of our cell tracking algorithm, we first collect all the predicted positions from Kalman Filter from every frame, and based on the data association result, calculate the trajectories.

We first wish to assess whether the tracking algorithm works properly. Therefore, we need to assume that the detection algorithm will not go wrong. To achieve this, we perform cell detection on YOLO training data, so that YOLO can achieve near-perfect detection accuracy and we can focus on the performance of the tracking algorithm. The tracking results under this condition are shown in Fig. 6. The different colours represent the trajectories of different cells. We also plot all the cell positions detected by YOLO, and we can see that our tracking algorithm fits the observed data well and obtains continuous and clear cell movement trajectories, which should prove the effectiveness of our tracking algorithm

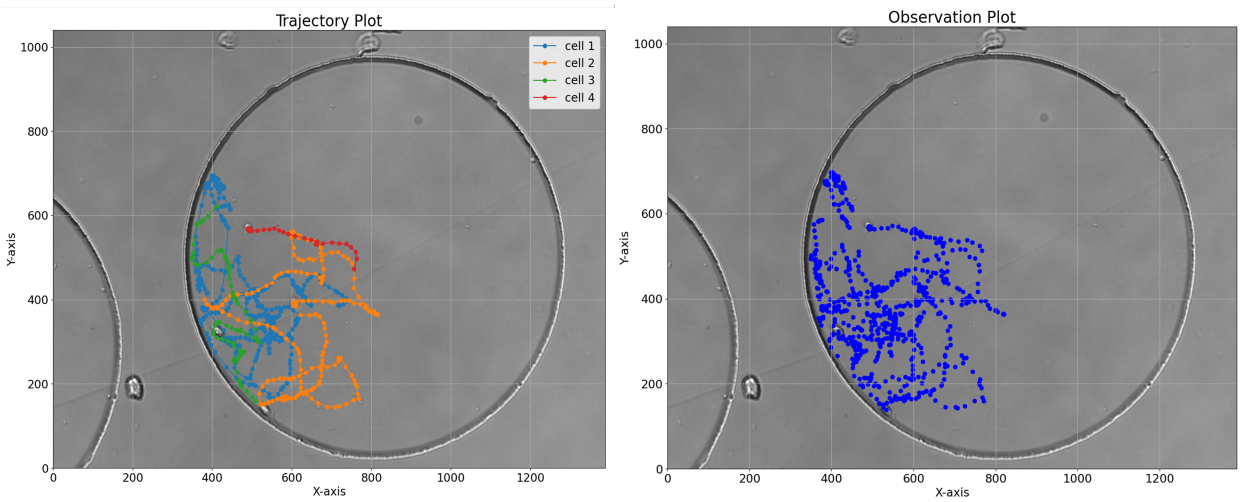

Fig. 6: Tracking trajectories under perfect detection condition

To verify the performance of the whole algorithm, we tested it on another separate cell image sequence, and the result is shown in Fig. 7. There are some jumps between different trajectories, which are caused by errors in the detection algorithm. However, due to the additional fault-tolerant options in our tracking algorithm that take into account YOLO misrecognition, this kind of jumping has been reduced a lot and most of the trajectories are still clear and coherent.

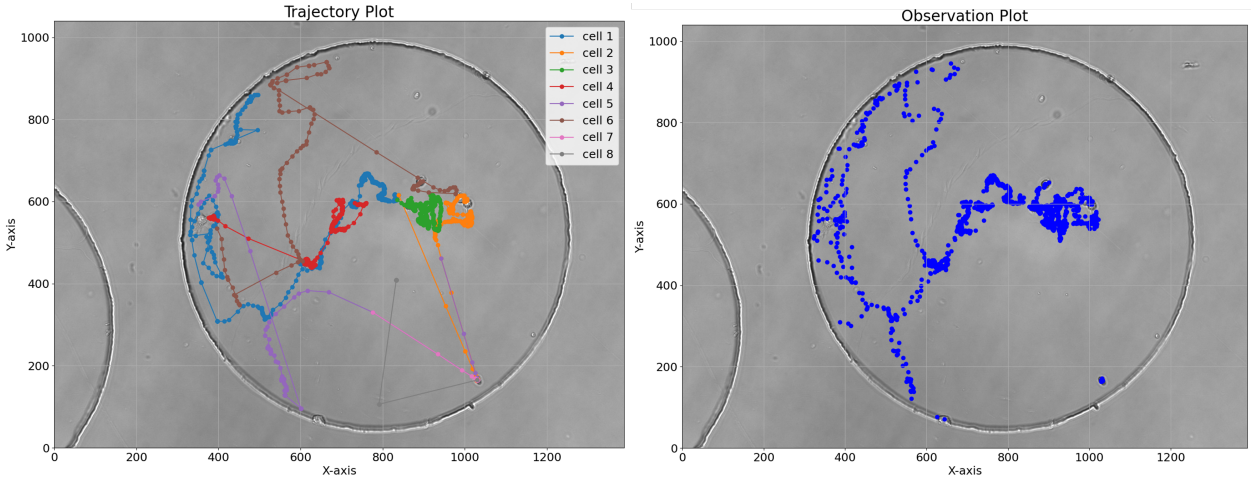

Fig. 7: Tracking trajectories under real testing condition

V. Conclusions

In this study, we implement the design of cell detection and cell tracking in a time sequence bright field microscopy images. We used YOLOv5 for cell detection, CNN for cell state estimation, and the Hungarian algorithm and the Kalman Filter for cell tracking. We show that YOLOv5 can detect the position of cells from bright field images, and with a few layers of CNNs, the state of cells can be predicted accurately. We also show that the combination of YOLO, the Hungarian Algorithm, and the Kalman Filter can track the cells effectively.

Although our approach shows efficient cell tracking and cell information extraction ability, this project is still a basic implementation of the design. Several improvements are possible. Firstly, the detection performance of YOLOv5 is not that stable. Sometimes it loses cells and sometimes it does wrong detection. Thus, more training data is required to improve the accuracy of the YOLOv5 model. Secondly, we implement CNNs to predict the cell state. However, this information is not used when doing cell tracking. The cell state can be used to help with cell tracking. For example, if a cell is in state M, then in the following frames there is a high probability that YOLO detects one more cell than the prediction from the last frame. Finally, the discussion regarding the number of cells detected by YOLO and the number of cells predicted from the previous frame is not sufficient yet. For example, if the number of YOLO-detected cells is equal to the prediction, we always take it as if nothing goes wrong. But in fact, the case can be that YOLO loses one cell while a cell divides in this frame.

All in all, we have achieved effective cell information extraction and cell tracking with little data and little training, which is a clever combination of deep learning, signal processing methods, and algorithm design.

REFERENCES

[1] J. Selinummi, P. Ruusuvuori, I. Podolsky, A. Ozinsky, E. Gold, O. Yli-Harja, A. Aderem, and I. Shmulevich, "Bright field microscopy as an alternative to whole cell fluorescence in automated analysis of macrophage images," *PloS one*, vol. 4, no. 10, p. e7497, 2009.

[2] M. Chen, "Cell tracking in time-lapse microscopy image sequences," in *Computer Vision for Microscopy Image Analysis*, pp. 101–129, Elsevier, 2021.

[3] E. Moen, D. Bannon, T. Kudo, W. Graf, M. Covert, and D. Van Valen, "Deep learning for cellular image analysis," *Nature Methods*, vol. 16, no. 12, pp. 1233–1246, 2019.

[4] H. Liu, F. Sun, J. Gu, and L. Deng, "Sf-yolov5: A lightweight small object detection algorithm based on improved feature fusion mode," *Sensors*, vol. 22, no. 15, p. 5817, 2022.

[5] H. Hu, Q. Guan, S. Chen, Z. Ji, and Y. Lin, "Detection and recognition for life state of cell cancer using two-stage cascade cnns," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 17, no. 3, pp. 887–898, 2017.

[6] M. d. O. Barreiros, D. d. O. Dantas, L. C. d. O. Silva, S. Ribeiro, and A. K. Barros, "Zebrafish tracking using yolov2 and kalman filter," *Scientific reports*, vol. 11, no. 1, p. 3219, 2021.

[7] A. L. R. Siriani, V. Kodaira, S. A. Mehdizadeh, I. de Alencar Nääs, D. J. de Moura, and D. F. Pereira, "Detection and tracking of chickens in low-light images using yolo network and kalman filter," *Neural Computing and Applications*, vol. 34, no. 24, pp. 21987–21997, 2022.

[8] K. Lasma, "Baxter algorithms: Software for segmentation, tracking and analysis of cells in microscope image sequences." https://github.com/klasma/BaxterAlgorithms, 2018.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[10] Y. Bar-Shalom and T. Fortmann, *Tracking and Data Association*. Mathematics in Science and Engineering, Academic Press, 1988.

[11] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[12] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, pp. 35–45, Mar 1960.

[13] Robotflow, "Robotflow – ai driven workflow automation." https://www.robotflow.com, 2023.