Cell Tracking at Low Frame Rate using Deep Learning and Bayesian Integration

Xiang Zhang
xzhang7@clemson.edu

Follow this and additional works at: https://tigerprints.clemson.edu/all_dissertations

Recommended Citation
Zhang, Xiang, "Cell Tracking at Low Frame Rate using Deep Learning and Bayesian Integration" (2022). All Dissertations. 3051.
https://tigerprints.clemson.edu/all_dissertations/3051

This Dissertation is brought to you for free and open access by the Dissertations at TigerPrints. It has been accepted for inclusion in All Dissertations by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.
CELL TRACKING AT LOW FRAME RATE USING DEEP LEARNING AND BAYESIAN INTEGRATION

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

by
Xiang Zhang
May 2022

Accepted by:
Dr. Federico Iuricich, Dr. Marc R. Birtwistle, Committee Chair
Dr. Victor Zordan
Dr. Brian C. Dean
Abstract

Tracking cells over time is a fundamental task in live-cell imaging, and often requires costly manual analysis if images are not acquired with high enough frame rate. Acquiring high frame rate images, however, can limit the number of conditions explored and cells analyzed, and contribute to photobleaching, which makes fluorophores dimmer and phototoxicity, which affects cell health and renders the resulting data unusable.

Assuming a relatively high frame rate in image acquisition, state-of-the-art cell tracking approaches rely on either spatial proximity or morphological similarity to link cells in consecutive frames. The problem is that, at low frame rate, both approaches fall short since the position and appearance of cells can change significantly.

The goal of this thesis is to improve the robustness of cell tracking at low acquisition rate. To this end, we started by focusing on the first computational problem in cell tracking, which is cell identification. Convolutional Neural Networks (ConvNets) provide a way to accurately detect cells, but the manual annotation needed for training is costly. Thus, our first research question is focused on 1) how to train deep ConvNets for cell identification without manually-annotated cell labels? We proposed an image processing pipeline which uses fluorescent images to generate cell labels for training ConvNets. The experiment results showed that the proposed model can achieve competitive performance (recall 0.89 and precision 0.92) for identifying cells in a completely automatic manner. Then, we focused on the actual cell tracking problem, i.e., how to follow cells in consecutive frames. Inspired by the biologically proven theory that a cell’s morphology suggests its moving direction, we studied 2) if we can design a set of features to represent the cell shape and estimate the cell velocity by regression to predict the cell’s future position. We used hand-crafted geometric features for modeling the shape of cells and the experiments demonstrated that the cell velocity can be estimated using these features. Given that geometric features extracted from image
patches can describe the motion of a cell, we focused on the third research question which is 3) how to integrate cell velocity estimations to improve the cell tracking accuracy at low frame rates? Our proposed approach contains two innovative components. First, we proposed a new deep-learning-based approach to automatically derive cell velocity information from image patches without the need for manually-defined geometric features. Second, we designed a new Bayesian framework which leverages cell position information and cell velocity estimations to track cells. We compared our cell linking method to both state-of-the-art tracking approaches and tracking algorithms implemented in well-established toolboxes for cell analysis. Our approach outperformed existing methods while allowing a 4x reduction in the frame rate.

In addition to the cell tracking project introduced above, the author participated in two projects in Dr. Feng Luo’s lab, which resulted in three publications. The first project titled “Cyberbullying detection based on ConvNets” aimed at detecting cyberbullying content in social networks. The second project titled “Efficient ConvNets design” aimed to investigate design patterns for ConvNets. We introduce these two projects at the end of the thesis.
Dedication

This dissertation is dedicated to my wife Cheng, my daughter Juan Juan, and my family.
Acknowledgments

I would like to thank my advisers for their extraordinary support and patience in debugging my thinking process, teaching me how to ask the right research questions, and improving my writing skills. I would like to thank my advisory committee for guiding my research. I would also thank the support from the School of Computing and Dr. Feng Luo. I am very grateful to my fellows in Birtwistle's lab and the Visual Computing division for your help, training, encouragement, and feedback.
Table of Contents

Title Page ................................................................. i
Abstract ................................................................. ii
Dedication ............................................................... iv
Acknowledgments ......................................................... v
List of Tables .............................................................. viii
List of Figures ............................................................. ix
1 Introduction ............................................................. 1
2 Background .............................................................. 7
  2.1 Image Acquisition .................................................. 7
  2.2 Cell Tracking ....................................................... 10
3 Related Work .......................................................... 12
  3.1 Cell Identification .................................................. 12
  3.2 Cell Linking ......................................................... 18
4 Stain-free Cell Segmentation ........................................... 27
  4.1 U-Net based Segmentation ......................................... 28
  4.2 Experiment ......................................................... 33
  4.3 Conclusion ........................................................ 49
5 Morphological Feature based Velocity Regression ...................... 50
  5.1 Velocity Regression ............................................... 52
  5.2 Experiment ......................................................... 55
  5.3 Conclusion ........................................................ 65
6 Cell Linking based on Migration Estimation ............................. 67
  6.1 Model ............................................................... 68
  6.2 Experiment ......................................................... 74
  6.3 Conclusion ........................................................ 83
7 Conclusion .............................................................. 85
8 Other Projects .......................................................... 88
  8.1 Cyberbullying Detection with a Pronunciation based Convolutional Neural Network 89
  8.2 CrescendoNet: A New Deep Convolutional Neural Network with Ensemble Behavior 101
8.3 Conclusion ........................................................................................................ 114

Appendices ........................................................................................................... 115
  A Velocity Regression Models .......................................................................... 116

Bibliography ........................................................................................................... 119
List of Tables

4.1 Nuclei detection performance comparing with the annotation. ............................. 41
4.2 Cytoplasm segmentation performance comparing with the fluorescent labels. ........ 47

5.1 RMSE of displacement along the major axis. ..................................................... 59
5.2 RMSE of displacement along the minor axis. ..................................................... 59
5.3 RMSE of the absolute displacement. ................................................................. 59
5.4 Correlation of displacement along the major axis. ........................................... 61
5.5 Correlation of displacement along the minor axis. .......................................... 61
5.6 Correlation of absolute displacement. ............................................................. 61

6.1 Method comparison using 12 fields of view (FOV) at 1-hour frame rate. Columns of track fraction and complete track show the corresponding scores in percentage. Row FOV indicates the index of the field of view. ....................................................... 77
6.2 Track fraction score at varying frame rates ....................................................... 79
6.3 Complete track score at varying frame rates ..................................................... 79
6.4 Specifications of four additional datasets. .......................................................... 80
6.5 Complete track score of LAP and the proposed method using four datasets, including eight fields of view, at four different frame-rate scales (1x, 3x, 6x, 12x). .................................................. 81
6.6 Complete track score of LAP and the proposed method with ground truth 12-class direction label using four datasets, including eight fields of view, at four different frame-rate scales (1x, 3x, 6x, 12x). ........................................ 82
6.7 Complete track score of LAP and the proposed method assuming the ground truth direction probability is known to the direction estimator using Myoblast and MUSC datasets, including four fields of view, at four different frame-rate scales (1x, 3x, 6x, 12x). .................................................. 82

8.1 Examples of noisy data. The examples are from Formspring.me website. ......... 90
8.2 Examples of word-to-pronunciation conversion. The examples are from Twitter and Formspring datasets. ................................................................. 94
8.3 Approach comparison on twitter dataset. Results are average of 10-fold cross validation. The accuracy of first four classifiers from [12] is not given. ....................... 98
8.4 Approach comparison on Formspring dataset. Results are average of 5-fold cross-validation. ................................................................. 99
8.5 Class imbalance-tackling on Twitter dataset. .................................................... 99
8.6 Class imbalance-tackling on Formspring dataset. ........................................... 100
8.7 Whole net classification error (%) with CIFAR10/CIFAR100/SVHN. ............... 108
8.8 Subnet classification error (%) with CIFAR10/CIFAR100/SVHN. ..................... 109
9 Neural network structure for velocity regression. .............................................. 116
# List of Figures

1-1 Microscopic image acquisition procedure. .................................................. 2
1-2 In the high frame rate sequence, pairing cells based on spatial proximity suffices to correctly reconstruct cell tracks. A lower frame rate introduces approximation errors leading to incorrect cell tracks. The proposed method uses cell movement predictions to fill the gaps created by low frame rate acquisitions. In a nutshell, our model estimates the probability of where each cell will move and uses this information to compute the link. ................................................................. 3

2-1 Bright-field image and fluorescent image. ................................................... 8
2-2 Fluorescence from stain and protein. ........................................................ 9

4-1 Segmentation and fluorescent labels. ........................................................ 28
4-2 U-Net architecture. ............................................................ 29
4-3 Dropout operation ............................................................ 30
4-4 Convolution illustration ............................................................ 30
4-5 Cell images in different modalities ................................................... 34
4-6 Preprocessing pipeline ............................................................ 35
4-7 Rolling ball algorithm. ............................................................ 36
4-8 Preprocessing pipeline ............................................................ 37
4-9 Comparison of annotations and predicted nuclei ..................................... 39
4-10 Illustration of the precision and recall. ................................................. 40
4-11 Recall change with respect to the density .............................................. 42
4-12 Images with low recall values. ...................................................... 43
4-13 Precision change with respect to the density ........................................ 44
4-14 Images with low precision. .......................................................... 45
4-15 Comparison of performance with and without noise. ................................ 46
4-16 Comparison of segmentation performance of cytoplasm for different densities. 48

5-1 Morphology and motion pattern ........................................................ 51
5-2 Steps in cell spreading ............................................................ 52
5-3 Velocity regression workflow ........................................................ 52
5-4 Ellipse fitting on the cell mask. ...................................................... 54
5-5 Features and labels. ............................................................ 55
5-6 Visualization of annotated tracks .................................................. 56
5-7 Cell Separation ............................................................ 57
5-8 Displacement error of the linear regression. ......................................... 62
5-9 Displacement error of the MLP based regression model. .................... 63
5-10 Velocity regression visualization .................................................. 64
5-11 Examples of under-segmentation. .................................................. 65
5-12 Limitation of the ellipse fitting. ...................................................... 66
6-1 Example of a cell hectically changing speed over time. Changes in speed do not necessarily correspond to changes in shape. ................................................. 69

6-2 The proposed model is composed of three components. The first component classifies cells dividing them into fast and slow cells. The second component estimates cell moving direction using a ConvNet model. Cell direction information and cell speed information are combined by the third component, a Bayesian framework, to produce the final cell linking. The method works sequentially on all time steps $t$ and $t + 1$ until cell tracks are reconstructed in full. ............................................................... 70

6-3 Cell direction prediction is modeled as a classification problem. Each class $k_i$ represents a group of possible directions for the cell. The total number of classes $K$ is a parameter defined during training. ................................................................. 71

6-4 The loss function is a combination of two terms $l_a$ and $l_b$. For a ground-truth moving direction $k_y$, $l_a( |k_y - k|)$ rewards classes based on their angular distance from $k_y$. The bimodal term $l_b( |k_y - k|)$ rewards them in a symmetric manner. The illustration is for providing intuition and the exact definition is given in Equation 6.3. .............. 72

6-5 (a) Given a cell $x_i$ moving towards $y_j$, the estimated position $\hat{x}$ is defined by the angle of the class bisector $u$, and the distance of $y_j$ from $x_i$ (i.e., the speed of $x_i$). (b) The distance rank is computed by sorting all cells $y_i$ according to their distance from $\hat{x}$. $\text{Rank}(x_1) = 1$ because $y_1$ is the closest cell to $\hat{x}$. (c) $\text{Rank}(x_1)$ is 2 because $y_1$ is the second-closest cell to $\hat{x}$. This means that, with $K = 4$ the predicted position of $x_i$ in the next frame (e.g., $\hat{x}$) is closer to the wrong cell $y_2$. (d) By increasing the number of classes to $K = 8$ we obtain $\text{Rank}(x_1) = 1$ which means the new prediction $\hat{x}$ will be closer to the correct cell $y_1$. ............................................................... 75

6-6 Bar chart showing the total number of cells having distance rank 2 when using number of classes $K = 4, 8, 12, 24$. ....................................................................... 76

6-7 Performance change of LAP, TRACKMATE, and the proposed method at frame rates (0.5h to 3h). The mean track fraction score and complete track score at each frame rate are calculated for the three methods using 12 sequences. As the time interval increases, the proposed method outperforms LAP and TRACKMATE by relatively larger margins in terms of mean performance metrics. .................................................. 78

6-8 Lens magnification and the optimal direction class number for four datasets, including eight fields of view, at four different frame-rate scales (1x, 3x, 6x, 12x). ........................................ 83

8-1 The architecture of PCNN ................................................................. 93

8-2 CrescendoNet architecture used in experiments, where scale = 4 and interval = 1. 102

8-3 Path-wise training procedure. ............................................................ 105

8-4 Error rates of subnets with different branch combinations when training with CIFAR10.111

8-5 The relation between the accuracy and resource budget when testing with CIFAR10. 112
Chapter 1

Introduction

Microscopic live-cell imaging is a fundamental tool for understanding biological and chemical phenomena. Quantitative information, such as the longevity or the average velocity of cells, provides key biological insights into cell behavior. To this end, identifying and following individual cells over time-lapse microscopic images is important to cell and molecular biology.

Manual cell tracking can be time-consuming due to the large population involved in a time-lapse image dataset. For example, to track cells, biologists use stains to label cell nuclei, acquire cell images, and manually mark them by image-editing software. In addition, intensive manual analysis of cell events is needed since cells may enter or exit the field of view, divide, die, or move from one frame to the next.

Automatic cell tracking has significantly enhanced tracking efficiency, first by leveraging computer vision techniques [38, 104, 151], and recently by means of machine learning and deep learning approaches [210, 83].

Still, many factors can affect the robustness of a cell tracking algorithm; one of these is the frame rate at which images are acquired. Existing techniques are designed to work at a relatively high acquisition rate (< one image every 5-15 minutes for typical adherent mammalian cells).

However, lower frame rates are beneficial for biologists experimentally. The lower the frame rate, the more images that can be obtained between time points, increasing the number of cells can be analyzed in one experiment [171]. For example, Figure 1-1 shows the steps for acquiring multiple channels of images for an individual region of interest. The plate contains 96 wells, each of which has multiple regions of interest. The imaging time for the whole plate can take an hour or more,
depending on the number of regions of interest per well. Thus, a lower acquisition rate allows imaging more plates and applying more experimental treatments (such as adding different reagents).

Moreover, too frequent imaging is the cause of a number of experimental artifacts and undesirable effects such as photobleaching, which makes fluorophores dimmer, and phototoxicity, which affects cell health [65, 51, 116].

Figure 1-1: Procedure for the cellular image acquisition for a region of interest. The photo shows a cell culture plate containing 96 wells. Inside each well, there are cells attached to the bottom and ready for imaging. For each region of interest in a well, the imaging facility will move the camera to point to the well, locate the region, tune the focus, and then take photos using different channels.

While beneficial for domain scientists, low acquisition rates introduce several challenges for automatic cell tracking since longer time intervals increase uncertainty regarding appearance and position changes. State-of-the-art cell tracking approaches rely on either spatial proximity [104, 151, 205, 206, 7] or morphological similarity [170, 83] to link cells in consecutive frames. At low frame rate, both approaches fall short since a cell’s position and appearance can change significantly (see Figure 1-2).
Figure 1-2: In the high frame rate sequence, pairing cells based on spatial proximity suffices to correctly reconstruct cell tracks. A lower frame rate introduces approximation errors leading to incorrect cell tracks. The proposed method uses cell movement predictions to fill the gaps created by low frame rate acquisitions. In a nutshell, our model estimates the probability of where each cell will move and uses this information to compute the link.

This thesis focuses on improving cell tracking robustness in time-lapse microscopic images at low frame rate. We started with generating a new dataset containing twelve time-lapse videos of human epithelial cells (MCF10A cells) in exponential growth for three days at a 15-minute frame rate at varying confluencies. By dropping images in each sequence, we simulated a lower frame rate. We used this dataset to test our proposed cell tracking approaches.

Given the cell images, the first step of cell tracking at low frame rate is to identify cells. Recent deep learning approaches, i.e., deep neural networks, have enabled accurate cell recognition using microscopic images by leveraging big training data. However, acquiring extensive and annotated cell images is demanding and time-consuming. Thus, we wondered if we can automatically annotate
cell images using computational approaches and fluorescent cell images, in which cells are highlighted. By using fluorescent cell images, we proposed an image processing pipeline to automatically generate cell annotations, which have been used for training a deep learning model for identifying cells. This approach is discussed in Chapter 4.

The next step of cell tracking is to link identified cells. Existing cell linking approaches match cells in consecutive frames by spatial proximity [104, 151, 205, 206, 7] or morphological similarity [170, 83], which, however, become unreliable at low frame rate. We observed that cells change their body shape regularly when moving. Researches also show that the cell movements are connected with their appearance changes [167]. Thus, we were inspired to predict cell motion by its morphology, i.e., shape information. Specifically, we proposed a group of morphological features and a regression model for estimating cell velocity. We evaluated the cell velocity regression performance using the manually annotated cell velocity. The experiment results showed us that we can predict cell velocity with an error smaller than the major-axis length of cell nuclei at low frame rate. In other words, we found that a cell’s morphology suggests its velocity.

However, two major limitations affect the use of morphological features. First, to help extract morphological features, we used fluorescent images of cell cytoplasm, which is the material enclosed by the cell membrane. Acquiring fluorescent cytoplasm images requires an additional microscopic imaging device which can emit and receive light with specific wavelength. Thus, using fluorescent cytoplasm images limits the model applicability.

Second, the hand-crafted features are time-consuming to define and do not generalize well. The proposed features include environment information such as the number of neighbors around the target cells but miss other key environmental features such as the layout of the neighbors.

To eliminate the need of fluorescent cytoplasm images and improve the feature richness related to motion information, we proposed a framework to estimate cell velocity using the bright-field image. First, we predicted moving direction by deep learning model using an input image patch. Then we estimated the probability of a given speed using training data. Finally, given the independently estimated speed and direction probability distribution, we introduced a Bayesian framework to calculate a probabilistic cell association score, indicating the probability of two cells to be linked.

Our approach has been formally tested on our dataset of time-lapse cell images. The computational experiments show that our approach outperforms state-of-the-art cell tracking algorithms [104, 151, 205, 7] at acquisition intervals ranging from 30 minutes to 3 hours for this cell system.
To measure the generality and limitations of our approach, we have tested the proposed framework using four additional public datasets [210, 113] with different cell lines, frame rates, image modality, and number of frames. The experiment results suggest that the proposed method can generalize well to different datasets if the ConvNets-based direction estimator can be effectively trained, which requires enough training samples and model fine-tuning.

Overall, the contributions of the thesis are as follows:

• Firstly, we investigated the feasibility of automatically using fluorescent cell images to generate cell labels for training ConvNets to identify cells in bright-field images by i) building a required cell dataset, ii) designing an image processing pipeline for generating labels, and iii) conducting cell identification experiments. The experiment results show that the proposed model can achieve competitive performance (recall 0.89 and precision 0.92) for identifying cells in a completely automatic manner.

• Secondly, we proposed a group of morphological features and a velocity regression model to predict cell velocity. The experiments show that morphological features extracted from image patches can describe the motion of a cell, which laid a foundation for designing more advanced velocity estimation approach and relevant motion information integration algorithm for improving cell tracking performance at low frame rate.

• Thirdly, we parsed motion prediction into direction and speed components, which can be estimated independently. We proposed a deep learning model with a new loss function based on cell motion behavior to predict cell movement using an input image patch. The velocity estimator is data-driven without assumptions of cell motion models.

• Finally, we designed a Bayesian framework to integrate disparate pieces of information about cell motion in a statistically rigorous manner. The predicted linkage probabilities as association scores are used by linear assignment based cell tracking approaches and benefits the cell tracking performance at low frame rate. The proposed framework can apply to generic object tracking at low frame rates when the velocity magnitude and direction can be learned from the data.

The thesis is organized as follows. Chapter 2 introduces background about data acquisition and cell tracking problem formulation. Chapter 3 discusses the state of the art of cell identification
and tracking approaches. Chapter 4 presents an implementation for pixel-level cell identification, without using manual annotation for training the cell identification model. Chapter 5 introduces a method to predict cell velocity from cell morphology. Chapter 6 proposes a new method for cell linking based on the estimation of cell velocities using a proposed deep learning model. Chapter 7 draws the conclusion and discusses the future work. Chapter 8 discusses published works about two side projects.
Chapter 2

Background

2.1 Image Acquisition

The analysis of cells requires the acquisition of digital images from a microscope. The simplest and most cost-efficient technique is based on bright-field imaging which uses white light to illuminate samples. The images generated show contrast values depending on the light absorbed by the sample. Figure 2-1a shows an example of bright-field image.

The main problem when dealing with automatic cell processing is that cells in bright-field images have low contrast which makes their recognition more challenging. To this end, fluorescent images are preferred. Fluorescent imaging approaches use fluorescence to label molecular mechanisms and structures. The most commonly used cellular components for cell identification via fluorescence are nuclei and cytoplasm which are jelly-like substances, spreading between the cell nucleus and membrane. Figure 2-1b shows an example of a fluorescent image where cell nuclei and cytoplasm are labeled by two different fluorescent light channels.

Fluorescent images are obtained either by means of chemical dyes such as Hoechst stain (see Figure 2-2a), or by means of fluorescent proteins (see Figure 2-2b). In general, fluorescent images acquired by Hoechst staining have a high signal-to-noise ratio but the toxicity of the dye will kill cells within a few hours. Fluorescent-protein images have more noises but do not affect cell life cycle and behavior.

For this project, we use bright-field images for cell recognition and use fluorescent images as the ground truth. When performing cell tracking tasks, we use manually annotated cell tracks as the
To generate long-term cell positions as the ground truth for cell recognition, we choose to generate fluorescent images based on proteins, not affecting the cell life cycle. Specifically, we use *mCherry fluorescent protein* to identify cell nuclei, and *Clover fluorescent protein* to identify cell cytoplasm. Then we manually annotate the fluorescent images to obtain the cell position and trajectory.

Figure 2-1: Bright-field image and fluorescent image. The red and green fluorescent labels indicate nuclei and cytoplasm respectively. Fluorescent labels are commonly used for cell identification and tracking.
We acquired cell images at the 15-minute interval and 20x magnification for 72 hours by GE IN Cell Analyzer 2500 HS. Images were taken from 12 fields of view via three channels, including the bright-field, the mCherry fluorescence for nuclei, and the Clover fluorescence for cytoplasm.

A field of view is a region of cells visible to the camera. At each field of view, the camera takes a photo every 15 minutes to acquire an image sequence including 289 images with the size 2040 by 2040 pixels per channel. For each channel, we have 3,468 images which are subdivided into 12 image sequences by the field of view.

Specifically, cells were cultured in complete sterile filtered (VWR 10040-436) media, consisting of DMEM F12 (Gibco #11330-032) supplemented with 2 mM L-Glutamine (Gibco # 25-005-C1), 20ng/mL EGF (Peprotech AF-100-15), 10ng/ml insulin (Sigma #I-1882), 0.5ng/ml hydrocortisone (Sigma #H- 0888), 100mg/ml cholera toxin (Sigma #C-8052) and 5% horse serum (Invitrogen #16050-122). Cells were passaged with 0.25% trypsin (Gibco #25200056) to maintain sub confluency. Cells were maintained at 37°C, 5% CO2. Cells were seeded in separate wells of a 96 well plate (Corning #3603) at 5000 cells/well for low density and 10000 cells/well for high density and allowed to grow in complete media. Ten fields of view were low density and two were high density.

We manually annotated cell position in terms of the center of mass of the nucleus and
trajectory for each image sequence. Annotations have been created with the open-source platform called ImageJ Fiji [186] and the TrackMate [205] plugin. The annotation also reflects cell behavior including migration, mitosis, and death. Cell migration refers to the cell movement within or through the field of view. Cell mitosis corresponds to a mother cell dividing into two daughter cells.

2.2 Cell Tracking

This section shows the formulation of the cell tracking problem.

Cell tracking consists of two essential tasks including cell identification, i.e., identifying the location of individual cells in a digital image, and cell linking, i.e., associating the observations of the same cells between consecutive images [210].

Cell identification can be seamlessly performed using general object detection approaches or segmentation approaches [210]. Most cell identification approaches train a machine learning or deep learning model to recognize cells in images. Details of these approaches are described in Section 3.1.

After detection, cell $i$ in frame $t$ can be represented as a node $x_i$ with associated features, such as the pixel intensity. Cell linking approaches aim to connect cell nodes into lineage trees under biological constraints.

Cell events are also modeled by the way that nodes are connected or not connected during the cell linking process. Specifically, track initialization, i.e., starting a new track from a node indicates that a cell moves into the field of view. Track termination at a node indicates that a cell moves out of the field of view or dies in the subsequent frame. Track extension caused by migration event by connecting two nodes represents that a cell moves from one node to another. Finally, track split by connecting two subsequent nodes to one previous node denotes a cell division event (also known as mitosis).

Cell tracks can be built by multiple-hypothesis tracking (MHT) [178], which exhaustively searches through all the valid track ensembles to find the best solution. However, MHT is computation-prohibitive even for small-scale tracking problems [104]. Thus, we follow the linear assignment problem (LAP) formulation [104], which is an accurate approximation of MHT but computation-efficient.

LAP formulation tracks cells by two phases: linking cells into track segments and assembly segments into tracks, both modeled as a linear assignment problem.

The linking phase uses cells to build track segments based on the migration events. Intuitively,
a node will be linked to a previous or subsequent node if there exists a migration event otherwise keep unlinked as a single-node track segment. Specifically, cell nodes in consecutive frames are iteratively connected by the linear assignment. The link between cells is one-to-one which means there is either none or one node target cell connected to the source cell. Assume $\mathcal{X}$ and $\mathcal{Y}$ are sets of cells in a pair of consecutive frames. $|\mathcal{X}|$ and $|\mathcal{Y}|$ represent the number of cells in $\mathcal{X}$ and $\mathcal{Y}$, respectively. The assignment is defined by a matrix $\mathbf{L}$ with size $|\mathcal{X}| + |\mathcal{Y}|$, where its binary element $L_{ij}$ represents whether a cell is linked to another one or not linked to any cell. Specifically, $L_{ij} = 1$ means:

- Cell $x_i$ is linked to cell $y_j$, which is a migration event, when $i \leq |\mathcal{X}|$ and $j \leq |\mathcal{Y}|$.
- Cell $x_i$ is not linked to any subsequent cell when $i \leq |\mathcal{X}|$ and $j > |\mathcal{Y}|$.
- Cell $y_j$ is not linked to any previous cell when $i > |\mathcal{X}|$ and $j \leq |\mathcal{Y}|$.

A cost matrix $\mathbf{W}$ defines all possible assignment costs, $w_{i,j}$ for $i, j \in 1, 2, \ldots, |\mathcal{X}| + |\mathcal{Y}|$. The objective is to minimize the objective function

$$
\sum_{i=1}^{(|\mathcal{X}|+|\mathcal{Y}|)} \sum_{j=1}^{(|\mathcal{X}|+|\mathcal{Y}|)} L_{ij} W_{ij}
$$

Each element $W_{ij}$ represents the cost of assignment $L_{ij}$.

The assembly phase combines track segments to complete tracks by capturing events such as cell division and gap-closing. All the linked cell nodes, representing track segments across all the frames, are joined into tracks by the linear assignment. The assignment between segments is still one-to-one since the events that happened to the two segments are exclusive. For example, a source segment can be disconnected from any subsequent segment, representing a track termination caused by cell moving-out or death event. In addition, a target segment can either be connected to a middle node of a previous segment, denoting a cell division event, or to the end node of a source segment, denoting a gap-closing event, but not both. The assignment cost matrix consists of the costs for cell division event, gap-closing event, track initialization, and track termination, which require to be specifically designed. The assignment will generate the final cell tracks.

This research work focuses on cell identification and migration-related cell linking.
Chapter 3

Related Work

In this chapter, we describe state of the art in cell tracking. Generally, the problem of tracking cells in time-lapse images is organized into two sub-tasks cell identification and cell linking. Then, we describe current approaches for identifying cells in microscopy images in Section 3.1 and approaches for linking detected cells in Section 3.2.

3.1 Cell Identification

The purpose of this task is to identify the location of cells given an input microscopy image. The objective is a specialization of object identification, a general problem in computer vision that relates to identifying the location of an object in a digital image. Indeed, all the approaches currently applied to the identification of cells are adaptations of methods developed for object identifications.

3.1.1 Object Identification

In general, object identification involves three steps, including region selection, feature extraction, and classification [233]. A key distinction for methods used for object identification is between conventional methods and approaches based on deep convolutional neural networks.

Conventional object identification approaches use handcrafted features to distinguish different objects in the image. These features include intensity difference of adjacent image regions [138], the histogram of intensity gradients [45], and key points [145], and are used for classification in combination of standard machine learning approaches [36, 66, 60].
The main drawback of designing ad-hoc feature-based approaches is the lack of robustness. That is, the object appearance variation or the illumination can severely affect detection accuracy. Also, low-level features are not sufficiently discriminative and generic [233].

ConvNets-based identification has become the standard de facto for object detection thanks to the benefits brought by deep convolutional neural networks (deep ConvNets) [187, 76, 175, 143, 179, 74]. In contrast with conventional machine learning models, deep ConvNets can learn rich and hierarchical features automatically using an extensive amount of data [199, 88]. Moreover, weights learned during the training phase are easily transferred to different datasets, thus improve robustness and generality [56].

Object identification methods can be further classified by the subject of the final classification step. The classification can apply to regions to achieve object detection or to pixels for image segmentation.

One-stage Detection vs. Two-stage Detection All methods for object detection split the original image in candidate regions where the object is searched. Then, a first classification is done based on the method used for generating such regions.

One-stage methods are based on sliding-window detectors, which perform object classification with fixed regions centered at each pixel in the image. The regions anchored at each pixel in the image can include one or more bounding boxes with different sizes and aspect ratios. Classifiers predict, for each bounding box, the class of the object contained, the object likelihood, and the offset to the predefined box location [187, 175, 143].

The most widely used classification approach is linear support vector machine (SVM) [36] based on handcrafted features such as the histogram of intensity gradients [45]. State-of-the-art approaches are based on ConvNets, which are able to learn rich features and run efficiently using GPU [187, 175, 143].

ConvNets-based classifiers can naturally classify each region separately by learning the features of each region and predicting the likelihood of objects contained. However, the features of the overlapped area of regions are calculated repeatedly. To this end, Sermanet et al. [187] use ConvNets to learn the features of the whole image before the object classification, which makes detection efficient.

Another problem with the one-stage methods is that the predefined bounding boxes may
miss the target object of different sizes and aspect ratios. To this end, regression approaches, usually based on ConvNets, are used to move and re-scale predefined bounding boxes [160, 175]. Redmon et al. [175, 176, 177] incrementally improved the detection accuracy and speed by techniques such as multi-scale and multi-aspect-ratio bounding boxes. In addition, Liu et al. [143, 106, 67, 188, 137] improve the detection accuracy by performing the classification on a different scale of feature vectors generated from different layers of ConvNets.

One-stage approaches based on ConvNets can be very efficient when using GPU. However, input bounding boxes may limit localization accuracy and flexibility. To this end, designing algorithms for accurate object localization before classification have attracted a lot of attention in recent years.

Two-stage methods use region proposal approaches to generate a relatively small number of regions to cover objects before classifying each candidate.

The early region proposal approaches hierarchically cluster pixels into groups by similarity measures, such as distance, color, texture, and shape [8, 34, 208]. Girshick et al. [76] propose R-CNN, which uses a clustering-based method, called the selective search [208] to generate region proposals, then use ConvNets and support vector machine (SVM) for feature extraction and classification, respectively. Specifically, the selective search method first clusters pixels by similarity and then proposes regions covering the clusters. All the regions are warped into the same size and converted into feature vectors by ConvNets. Finally, SVM predicts the existence and class of the object contained.

Later, Girshick et al. [74] propose Fast R-CNN, which improves the region classification speed by using ConvNets based classifiers to eliminate the need for extracting features of each region separately. Ren et al. [179] further propose Faster R-CNN, which uses a ConvNets based model for region proposal, which can learn from the domain-specific data to improve the object localization accuracy. In addition, the region proposal model can use the same feature used by the classifier. Thus, no extra time is required for generating region proposals comparing with the selective search method.

Furthermore, various components based on deep ConvNets are continuously developed to incrementally improve the region proposal quality [73, 223, 140, 44, 86, 25] and the computational efficiency [136, 115].

Overall, the two-stage methods based on end-to-end trainable deep ConvNets have achieved state-of-the-art detection accuracy, although they are slower than one-stage methods.
Pixel Segmentation vs. Instance Segmentation  The last classification is based on the output produced by the identification algorithm.

**Pixel segmentation** approaches assign labels to the pixels of the image without analyzing the actual objects present in the image. The only challenge addressed by these methods is recognizing a robust feature representation for correctly classifying pixels. In practice, pixel segmentation is achieved by defining image classifiers to predict the category of each pixel instead of the entire image [144, 181]. In other words, pixel-wise loss instead of single-label loss is used to train the model.

Long et al. [144] design a segmentation approach based on a Fully Convolutional Network (FCN), capable of handling input images of arbitrary size. Also, FCN can reuse the feature extraction layers and corresponding weights of the pre-trained state-of-the-art ConvNets-based classifiers [189, 199] to save the training time.

Similarly, U-Net [181] is agnostic to input image size. However, U-Net has a specially designed structure for segmenting biomedical images, which usually contain small objects to be precisely localized. We refer the reader to Section 4.1 for a detailed description of U-Net.

**Instance segmentation** approaches go a step further and take care of separating every single object in the image. In general, instance segmentation can be done by applying pixel segmentation followed by object localization or by applying the same steps in the reversed order.

To separate instances from the pixel segmentation, in terms of binary masks, the marker-controlled watershed algorithm [18] is commonly used [10, 101]. The marker-controlled watershed algorithm processes the pixel segmentation as a topological surface, where the pixel value denotes the height. Each marker floods its region until reaching the border of the regions belonging to other markers. Finally, all pixel segmentation are separated to belong to different markers.

Since the watershed algorithm only works with a scalar function, binary masks from the pixel segmentation are commonly processed by distance transform into a distance map [101]. The distance map has the same size as the mask, and each pixel value is replaced by the distance to the closest edge. The watershed algorithm can also be combined with deep ConvNets to build an end-to-end trainable segmentation model to learn from the data [10].

However, the watershed algorithm has limitations in separating objects with overlap or without obvious markers. To this end, object detection is usually performed before pixel segmentation to segment instances [75, 134, 86].

Girshick et al. [75] generate pixel segmentation after object detection. Region proposals are
generated and classified using Selective Search [208] and Support Vector Machine [36]. Then pixel-level classification can be performed using ConvNets. A similar model, namely Mask R-CNN [86], was upgraded by using better ConvNets [88] and sharing feature vectors between the classifier and region proposal networks. PANet [218] further improves Mask R-CNN [142] and adds U-Net [181] structure into the Mask R-CNN pipeline.

In general, instance segmentation simultaneously obtains the object contour and class, which makes it preferable to pixel segmentation. State-of-the-art instance segmentation approaches for biomedical images are based on U-Net [181] and Mask R-CNN [86], which won the first and third places, respectively, in the 2018 Data Science Bowl for nucleus segmentation [26]. U-Net is simpler and more efficient than Mask R-CNN but requires post-processing to separate instance from pixel segmentation.

3.1.2 Methods for Cell Identification

The objective of cell identification generally reduces to the identification of cell nuclei, which are well separable and easier to identify. The cytoplasm can also be a target for identification since it provides information about morphology and biological behavior.

The common approach for cell identification is instance segmentation using cell images (see Section 3.1.1), which can provide the cell contour. The cell images are usually acquired by bright-field or phase-contrast techniques (see Chapter 2). In contrast to the high-quality images used by general object detection, the microscopic images often suffer from heavy noise and uneven-illumination, especially when the cell density is high. Consequently, conventional object detection and instance segmentation approaches based on handcrafted features have difficulties identifying cells. In addition, cell identification accuracy is more important than speed, which makes two-stage approaches preferable. To this end, instance segmentation approaches based on deep ConvNets are widely used for cell identification [212, 92, 220, 107, 206, 93].

Instance segmentation for cells can be done by applying cell localization followed by pixel segmentation or by the same steps in the reversed order.

When performing cell localization followed by pixel segmentation, state-of-the-art object detection, and instance segmentation approaches, including Faster R-CNN [179] and Mask R-CNN [86] naturally apply to cell identification. Akram et al. [5] demonstrate that Faster R-CNN can accurately detect cells when the annotated bounding boxes for training are unavailable. Specifically, they use
the annotated cell markers (a point within the cell region) as centers to create bounding boxes with the average cell size. The created bounding boxes are used as the ground truth for training the detection model. Hu et al. [96] demonstrate that Faster R-CNN is able to detect cells in noisy and high-cell-density images, although the training data size is small. Hollandi [93] and Tsai et al. [206] demonstrate that Mask R-CNN can perform accurate and robust instance segmentation of nuclei and whole cells, respectively.

In terms of pixel segmentation followed by instance separation, deep ConvNets including FCN [144] and U-Net [181] show excellent performance for pixel segmentation. Van et al. [212] demonstrate that the cell instance segmentation based on simple ConvNets outperforms conventional segmentation methods using intensity and edge features. Hernandez [92] show that FCN based cell instance segmentation is robust to heavy illumination variance and occlusion from cells or debris. Falk et al. [56] demonstrate the state-of-the-art performance of U-Net for cell instance segmentation given a variety of cell types and imaging environments.

One limitation of methods based on deep ConvNets is the requirement of massive annotation for training the model. Manual annotation of cells in bright-field images can be time-consuming and requires knowledge for distinguishing cells and debris. Although imaging processing tools such as ImageJ Fiji [186] or CellProfiler [24] can help the annotation process, it can take many hours to annotate a dataset containing hundreds of images.

To obtain annotations quickly and with little to no manual interactions, we can use fluorescent labels (See Chapter 2) created by chemical dyes or fluorescent proteins. Dyes or fluorescent proteins can label the nuclei and cytoplasm in the fluorescent images corresponding to the bright-field images. Researchers are inspired to use fluorescent labels generated annotations to replace the manual annotations [183, 33]. Sadanandan et al. [183] demonstrate that the deep ConvNets-based instance segmentation model trained with annotations automatically generated using the fluorescent labels can perform similarly to the same model trained with manual annotations. Christiansen et al. [33] further show that the deep ConvNets-based regression model trained with fluorescent images can predict various cell labels, such as organelle (e.g., nuclei), cell type (e.g., neural), and cell state (e.g., cell death).

Overall, the main challenge of cell instance segmentation is to identify cells using low-quality microscopic images involving high-density cells. Fortunately, general deep ConvNets based models can provide excellent and state-of-the-art cell instance segmentation performance at the cost of
annotation burden.

3.2 Cell Linking

After identifying cells, the next step is to link the same cell in consecutive images so as to reconstruct the cell track. In this case, methods used for cell linking are a derivation of approaches developed for object linking. Approaches can be based on object detection or model evolution, capable of tracking single as opposed to multiple objects.

3.2.1 Tracking by Detection

Tracking by detection is one of the most common approaches. All objects are identified by object detection or instance segmentation algorithms (See Section 3.1) for each image and associated through the images over time. These approaches vary by aiming to track a single object or multiple objects.

3.2.1.1 Single Object Tracking

Single Object tracking refers to the identification of a single target in a sequence of images. Generally the target is specified by a bounding box at the first image and then tracked in the subsequent images [150, 146, 29].

Correspondence searching approaches are widely used for single object tracking [64, 35, 173, 16]. Specifically, in the first image, a template is created from the target region and used to match the possible target regions in the subsequent images.

The most common challenges for single object tracking are the dynamic appearance of the object, occlusion, clutter of background and foreground, and image degradation [150, 146]. The dynamic appearance refers to the variations of object shape, scale, orientation, and illumination. The background clutter may be caused by the high texture similarity between the target and the background. The foreground clutter may be due to the objects with similar appearance, which happens when tracking people. The low resolution and motion blur are also challenging to solve.

To achieve accurate matching between the target object and candidates, researchers have been developing representative features. Freeman et al. [64] use the region containing the target given in the first image as an object template. The similarity between the template and the target
is calculated by cross-correction, which is defined by point-wise matrix multiplication, using the
gray-scale pixel intensity. However, gray-scale intensity-based features are not stable when the object
shape changes and not capable of distinguishing other objects with a similar appearance. To this
end, techniques for extracting more adaptive and distinctive features are proposed, including using
the color histogram as features [35, 16] and incorporating the object surrounding into features [173].

To avoid handcrafted features, Nam et al. [161] propose a ConvNets-based feature extractor,
called MDNet, for tracking. The model is pre-trained using generic data beforehand. During the
tracking, MDNet is trained again with the target region as positive samples and background region
as negative samples for each image. MDNet and its extensions achieve the best robustness and
accuracy. However, MDNet runs slow since it has to perform online training for each image. By
avoiding the online training, Held et al. [91] propose a ConvNets-based model to perform a bounding
box regression given a target crop and a small search region in the following image, which works at a
speed of 100 images per second. To achieve the bounding box regression, Held et al. train the model
with shifted images where the location offset between the shifted image and the original images is
known.

After the feature extraction, correlation filters are commonly used as an efficient way to
compute the affinity between a target and corresponding search regions. Correlation-filters-based
approaches search for the corresponding location in every image given a target template such as a
cropped image patch containing the object [79, 64]. The searching can be done by using a sliding box
to iterate the whole image or by sampling regions from near neighbors. A similarity function, such
as the sum of element-wise multiplication or the sum of squared difference, is required to compare
the template with the candidate.

Correlation filters [23, 149, 211, 81, 148] have drawn significant interests of the community
since the operation can be achieved by element-wise multiplication between the template and search
region, which makes the similarity computation efficient. Despite the unchanged idea of template
matching, the feature representation of target and the whole image [35, 4, 173, 16, 77], the searching
strategy, i.e., how to traverse all possible regions of interest [48, 47, 57, 219], and the similarity
measurement [94, 163, 148] have been upgraded continuously in decades.

Siamese networks [119, 17, 202, 211] incorporate the feature extractor and correlation
filters into a single fully convolutional neural network, which generates a map of similarity score
between the template and the search image. Siamese networks are efficient as well as discriminative
due to their fully convolutional structure, which makes them capable of real-time object tracking applications [85, 231, 232].

The plain Siamese networks use the triplet loss function, which maximizes the feature distance between the target and the positive samples and minimizes the feature distance between the target and the negative samples. In addition, the cross-correlation calculation between the target and the candidate is implemented by convolutional layers, which favors the real-time applications when GPU is available. As a fully convolutional network, Siamese networks have become popular in the community and have many extensions [84, 128, 135]. The state-of-the-art object tracking approaches use Siamese networks aided by region proposal networks [58, 131, 59, 130], which provides more accurate locations. In addition, Siamese networks are capable of real-time tracking [85, 231, 232], object segmentation [219], 3D object tracking [72], and distraction-robust tracking [235].

3.2.1.2 Multiple Object Tracking

Multiple object tracking approaches consist of object identification (See Section 3.1.1) and data association [29, 150]. Data association refers to linking detections or segmentations of the same objects between consecutive images.

Compared with single object tracking that only needs to search for the best match of the target, multiple-object tracking algorithms extensively rely on the data association. Data association approaches aim to not only establish tracks from the detections but also correct detection errors, such as missing detections and false alarms.

Detections can be associated using consecutive images iteratively or using all the images globally. Association approaches using the consecutive images include global nearest neighbor [37, 21], linear assignment [123], and joint probabilistic data association [63, 228, 12].

The global nearest neighbor algorithm [37] assigns the detection with the highest similarity score to the target in the previous image.

Linear assignment associates detections by finding an optimal bijection between two sets of detections. Each mapping between two sets corresponds to a weight. The goal is to minimize the sum of the weights. Two sets can be padded to have an equal size before solving the optimization problem. When iteratively associating detections in two consecutive images and establish all tracks in parallel, object tracking can be formalized as a linear assignment problem. Linear assignment problems have the optimality given polynomial time. For example, the Hungarian algorithm [123] can
find the optimal solution given a time complexity $O(n^3)$, where $n$ is the size of the set for matching. Specifically, the linear assignment problem represents the weights between two sets by a square matrix. The row index and column index denote the detections in two consecutive images. The entry is a specific distance between two detections. By manipulating the weight matrix, one or more globally optimal selections in each row will be indicated by zeros. Then the solution can be found by selecting $n$ zero entries at different rows and columns as the solution.

*Joint probabilistic data association* [63], instead of selecting the most likely candidate or finding the maximum sum of link weights, assigns objects to achieve minimum mean square error estimate for all targets.

*Global linking methods* use all images for data association to leverage more context information. However, the global association for a long video can be slow or impractical, especially when there are too many objects. To this end, incremental track establishment or computational approximation can be used for reducing the computational complexity.

The most widely-used approaches for multiple frame tracking are multiple hypothesis tracking (MHT) [22, 114] and multiple frame assignments (MFA) [172]. MHT [22] keeps a limited number of possible tracks incrementally till the last image is reached and then searches for the best one. For example, the Viterbi algorithm [62] can be used to incrementally selects possible detections to finally find the most likely track, given all the detections in time-lapse images. Specifically, the nodes and the paths in an acyclic oriented graph represent the detections and possible tracks. MFA [172] uses Lagrangian relaxation, which approximates a difficult problem of constrained optimization by a simpler problem, to simply the multiple dimension assignment problem.

The recent fast advance of deep learning-based algorithms improves the data association by using automatically learned feature and similarity functions.

To correct data association errors caused by occlusion, Sadeghian et al. [184] encode long-term temporal dependencies from object appearance, motion, and interaction via recurrent neural networks (RNN), which is a common structure of neural networks to extract features of series. Similarly, Baser et al. [13] use ConvNets to learn a similarity function based on the appearance and spatial features of objects. Feng et al. [61] use three deep neural networks for different tasks to perform long-term tracking, including object detection, long-term object association, and switch-error-aware classification, respectively. To track the object at the pixel level, Voigtlaender et al. [215] extend the Mask R-CNN model [86], which is a state-of-the-art image segmentation method, with a tracking
network at the end.

### 3.2.2 Tracking by Model Evolution

Tracking by detection by nature is able to handle events of appearance, disappearance, and re-appearance. However, it may fail when the quality of detections degrades due to the occlusion, clutter, and illumination variations.

*Model evolution approaches* improve the tracking robustness by using prior knowledge and historical information of the target to estimate object positions [109, 103, 108, 68, 11, 155, 166]. Prior knowledge is often computed by motion models [19] or contour models [111], defined on the target object.

Motion models represent objects by the center of the mass and predict the future position using the current position and velocity by Newton’s laws of motion. For example, Bewley et al. [19] use a linear model to represent and keep track of the object’s position, aspect ratio, area, and corresponding changing rates.

Contour models represent objects by key points and update the current position of the points in the consecutive images to minimize an objective function, which aims to separate the target from the background. For example, Kass et al. [111] use a spline, i.e., piecewise polynomials functions, to detect the edges, lines, and contours.

The states estimated by pure model evolution may drift over time. For example, without external information about the object’s velocity or position, the iteratively estimated position can drift as time goes on due to the accumulated position error. Thus, dynamic motion models often use stochastic observations such as object detections or segmentation to improve position estimation.

Information fusion techniques can be used to estimate the states of the object from multiple information sources, such as the position predicted by the motion model and observed by instance segmentation [109, 103]. Kalman filter [109] and Particle filter [103] are the most common approaches to fuse information for object tracking by iteratively weighting different information sources for linear and non-linear motion or contour models, respectively. The evolution of the object position or contour can also be modeled and estimated implicitly by recurrent neural networks (RNN) [155] or ConvNets [166].

Overall, model evolution can help to reduce tracking errors caused by missing detections and false alarms, given an accurate motion or contour model of the target.
3.2.3 Methods for Cell Linking

Cell tracking can be modeled as an extension of the multiple object tracking problems with biological events such as cell division. In addition, ConvNets based approaches are preferred for cell feature extraction to the traditional machine learning approaches. The motion models or contour models of cells are likely to help the tracking.

We can categorize cell linking methods into two groups depending on whether the cell identification is based on historical information [210].

Tracking by detection first detect cells in all the images at once, then match the detections to establish tracks [151, 6, 206, 83]. Although deep ConvNets [122, 88] significantly improve cell detection and segmentation [56], there are still many challenges in cell linking.

Firstly, cell association is more challenging than which of the general object. Cells are non-rigid with changing appearance and look similar. For example, the relatively stable shape, area, and color features of rigid objects such as cars and pedestrians are representative and distinctive. However, cells can significantly change shapes as they move between two consecutive images and have no color difference from each other.

Secondly, cells can divide or die, which means the association algorithm itself is not enough. General object association approaches cannot handle appearance or disappearance caused by mitosis or cell death, respectively. Thus, detecting cell behavior requires additional classifiers.

Furthermore, detection errors can propagate into cell behavior classification errors and tracking errors. For example, when the mitosis classifier relies on the cell number change in a small region, it may classify a false alarm caused by debris into a daughter cell and create a new track. On the other hand, the sudden number change inside the image may unlikely happen to general objects, thus will be detected as an error.

Given the cell motion model and historical movements, tracking by model evolution is able to reduce the false detections and missing detections. Tracking by model evolution first initializes a motion model (or a contour model) of the cell in the first image, then updates the contour to match the cell in the subsequent images iteratively [54, 105, 9, 90]. The limitation of model evolution approaches is that it requires prior knowledge of cell movement or contour change.
3.2.3.1 Tracking by Detection

*Tracking by detection methods* [151, 185, 14, 20, 207, 82] perform cell detection or segmentation for all images and then associate detections using consecutive or all images. Automatic cell tracking approaches have evolved from computer vision techniques [38, 236, 54, 31] to new machine learning [210, 110] and deep learning [157] approaches. Specific to cell linking, existing methods are classified in: local and global linking methods [210, 157].

*Local linking methods* establish cell tracks by finding an optimal bijection between two sets of detections in consecutive images [38, 104, 20, 205]. Generally, the optimal bijection is found by solving a linear assignment problem based on a cost matrix.

*Global linking methods* establish cell tracks by working on the entire image set simultaneously. A graph is constructed with nodes, representing cell detections, and edges, representing possible links. Tracks are computed by finding optimal paths that connect nodes originating in the first frame to nodes appearing in the last one [151, 92, 204].

The key problem for both local or global linking approaches is how to define the likelihood of two cells to be linked. While all these approaches use a cost matrix $W$ to represent this information, the key difference is in how this matrix is populated. While cells are commonly represented by their center of mass [210], values of $W$ have been defined based on handcrafted spatial or appearance features [50, 96]. Euclidean distance is widely used for linking cells based on their positions [104]. The cosine distance is used to measure the similarity of appearance features such as, color and shape [96, 170]. When cell segmentation is available, the overlapping area is another popular distance measure between two candidates [53, 14, 69, 147]. Recently, ConvNets have been used to learn feature vectors automatically [170, 82, 83]. Payer et al. [170] use ConvNets to learn pixel-level feature vectors that are then used to segment the image and track cells. Hayashida et al. [83] use ConvNets to learn a motion field describing cell movements in each frame. Intuitively, the motion field is a vector field indicating the future motion of the cells. After the training phase, ConvNets directly produce a similarity measure for all cell detections in two consecutive images.

3.2.3.2 Tracking by Model Evolution

The evolution model for tracking cells can be dynamic motion models or contour models, depending on the cell representation.
Motion models [90, 9] represent cells by the center of the mass and predict the future position using motion states such as the current position and velocity. He et al. [90] use a particle filter based on the motion model to predict candidate bounding boxes in the subsequent image. A convolutional neural network classifies the candidates to find the target cell. Similarly, Arbelle et al. [9] use the Kalman filter to estimate the cell motion to match the target cell to candidate segmentations. Motion models have the advantages of reducing missing cells; therefore, they are more likely to reconstruct the entire cell track. However, the motion model itself lacks a mechanism to detect cell division and death. In addition, the observation of velocity by the position difference between two consecutive images can be unreliable as the time interval increases.

Contour evolution methods [54, 105, 220] start from the cell contour, usually segmented by the manual annotation, in the first one or more images and then update the contour in the subsequent images. These methods are based on the assumption of unambiguous spatiotemporal overlap [210], limiting their application to the short imaging time interval. In addition, ConvNets have taken the place of the contour evolution methods in recent years in terms of cell segmentation.

3.2.3.3 Cell linking at low frame rate

The drawback of all these approaches is assuming a relatively high frame rate in image acquisition. To this end, a few approaches have tackled the challenges of low frame rate acquisitions [31, 82].

The approach by Chen et al. [31] was introduced for cell populations where the position of a cell with respect to the rest of the population is largely invariant. This approach uses the color difference, the distance, and angle between each cell and its neighbors as the features to measure cell similarity and build the cost matrix. This idea, however, does not apply to cells (like the mammalian cells used in our study) that can move freely [210]. Arguably, a majority of relevant cell tracking problems involve cells that can move freely.

While the motion field by Hayashida et al. [83] was recently used for cell tracking at the high frame rate, an older version of this approach was developed specifically for low frame rate tracking [82]. A ConvNet network is used to compute a Cell Motion Field (CMF) by using a pair of consecutive frames. The model uses appearance features of the cells in both frames to estimate their similarity and compute the final link.

Compared with the above approach, our proposed method uses ConvNet to predict cell
movement direction based on a single frame. This relaxes the assumption that cells in consecutive frames should be similar to be paired.
Chapter 4

Stain-free Cell Segmentation

This chapter studies the problem of using automatically-generated nucleus labels to train deep ConvNets for cell segmentation. Once trained, the segmentation model can identify cells using only bright-field images without the need for staining cells for helping identification, which is called stain-free. We investigate approaches to achieve stain-free cell identification, the image processing techniques to generate nucleus labels automatically, and demonstrate the cell segmentation results as a foundation for solving cell tracking tasks.

To perform tracking in bright-field images, we first need to identify cells, which can be done by segmentation. Cell segmentation classifies each pixel in an image as either the target or the background. We can have two types of targets: the nucleus and the cell membrane.

Recent deep learning advances, including image classification, object detection, and segmentation (See Section 3 for details.), enable accurate nucleus and cell segmentation from bright-field images. As a state-of-the-art medical image segmentation model based on recent deep learning developments, U-Net [181], and its variants achieved significant success in cell image segmentation. Since our goal is leveraging fluorescent labels to achieve stain-free cell segmentation instead of designing new deep learning models, we adopt U-Net as our segmentation model.

Trained with bright-field images and associated cell labels, convolutional neural networks can predict the segmentation of nuclei or cells from bright-field images only. For example, Figure 4-1a shows a bright-field image. Figure 4-1b and Figure 4-1c show the corresponding segmentation and original fluorescent labels of nuclei in red and cytoplasm in green respectively. The segmentation is generated by the method described in the current section.
Figure 4-1: Segmentation and fluorescent labels of nuclei and cytoplasm from the bright-field image.

4.1 U-Net based Segmentation

U-Net [181, 56] is a state-of-the-art convolutional neural network specially designed for segmenting the medical image. U-Net can be trained using GPU efficiently since its major computational cost is from convolutional operations, which involve massive matrix multiplication.

U-Net [181, 56] adopts a symmetric encoder-decoder architecture, illustrated in Figure 4-2. The encoder part on the left side consists of four combinations of convolutional layers with a kernel size $3 \times 3$, denoted by blue arrows, followed by a max-pooling layer with a kernel size $2 \times 2$, denoted by the red arrows. Convolutional layers are trained to extract features, denoted by blue boxes following blue arrows. Max pooling layers select the maximal feature value from every $2 \times 2$ feature patch. The encoder encodes the image into a low-dimensional feature vector, which will be reconstructed back to a high-dimensional feature map by the decoder.

The decoder in U-Net is designed symmetrically to the encoder. Instead of using max-pooling layers, the decoder uses up-convolutional layers with the size $2 \times 2$, which works like an interpolation operation for upsampling the feature vectors. The last convolutional layer with the kernel size $1 \times 1$ is used to compress the channel of the feature map from 64 to 2, which represents the target and the background.

U-Net first gradually reduces the size of feature vectors from $572 \times 572$ to $32 \times 32$ and then reconstructs them to the original $388 \times 388$. Such a structure forces the model to learn and keep the most representative features in the feature vector of size $32 \times 32$. In addition, U-Net uses
skip-connections, which means using the output feature vectors from a lower layer as the input of a certain higher but not the subsequent layer. In U-Net, the skip connection is implemented by copy-and-crop operations illustrated by gray arrows in Figure 4-2, to fuse lower-level features, such as cell edges, into higher-level features, such as cell shapes, for reconstructing. Skip-connections keep details of the original images and avoid the gradient vanishing problem [88] during the training since the sub-networks created by skip-connects are shallow and easy to train.

Figure 4-2: U-Net architecture [181]. Blue boxes stand for multi-dimensional feature vectors. The number of channels is marked above the box. The x-y-size is annotated at the lower-left side of the box. White boxes denote copied feature vectors. Source: [181]

To optimize U-Net for our purposes, we follow the implementation proposed by Li [133] to increase the depth of the model in terms of convolutional layers from 18 layers to 22 layers, since deeper networks improve the segmentation performance [88, 97].

Dropout layers [193] are used to randomly set a subset of intermediate feature vectors to zero during the training to reduce the risk for over-fitting while improving the representative feature learning. In addition, images are padded before each convolution to keep information on the image border, which is a simple but widely used technique in ConvNets implementation [88]. Dropout
operations randomly set some neuron outputs to zero, as shown in Figure 4-3. This enhances the segmentation performance and forces the model to make good predictions using fewer features or incomplete information. Dropout layers are only used in training and removed when testing the model.

![Figure 4-3: Dropout operation](image1)

The images or feature vectors may lose information on the border after convolution operations. Figure 4-4 shows an example that a $5 \times 5$ feature vector will have the size $3 \times 3$ after being applied by a convolutional kernel of size $3 \times 3$. To avoid this, we pad the image border with zeros before each convolution operation to preserve more information.

![Figure 4-4: Convolution without padding the feature vector](image2)

Finally, the logistic sigmoid function, as defined in Equation 4.1, is used to convert multi-dimensional feature vectors generated by U-Net into the interval $[0, 1]$, which represents the probability of a pixel as the foreground object. This approach, named logistic regression and its variants, is widely used in supervised classification applications such as image classification. [196, 122, 78, 88, 133].

\[
\text{logistic}(X) = \frac{1}{1 + e^{-(\alpha + wX)}}
\]  

(4.1)
where $X$ is the feature vector and $X_i$ is its components. The weight vector $w$ and the bias term $\alpha$ can be learned during the training. The output of $\text{logistic}(X)$ is within the range of $[0, 1]$, which denotes the probability of a pixel belongs to the foreground, either nuclei or cytoplasm in our experiments.

The details of the extended U-Net structure can be seen in the supplement material.

As defined in Equation 4.2, for each pixel in a bright-field image, U-Net will classify it into cell or background.

$$p = f_{UNet}(X; \theta)$$

(4.2)

where $p$ is the predicted vector of probability that corresponding pixels belong to the target object, either nuclei or cytoplasm. $f_{UNet}(X; \theta)$ denotes the U-Net model where $X$ and $\theta$ are the input feature vector, which is the bright-field image, and all the parameters of the model.

Then we can evaluate the prediction during the training by a loss function. The loss function combines two terms: the binary cross-entropy and the weighted Dice similarity coefficient [27].

Binary cross-entropy (BCE) measures the distance between two distributions, as defined in equation 4.3.

$$BCE(p, q) = \sum_{i=1}^{n} -(q_i \log(p_i) + (1 - q_i) \log(1 - p_i))$$

(4.3)

where $n$ is the number of pixels of each image, $q_i$ is the ground truth label for the pixel $i$, set as 0 for the background, and 1 for the nucleus. And $p_i$ denotes the predicted label correspondingly.

The Dice coefficient measures the similarity between two sets of samples, which is defined in equation 4.4. We use the Dice loss term to train the model to increase the intersection area over the union area between the predicted nuclei (or cytoplasm) and the ground truth. The Dice loss term alleviates the data imbalance problem, where the number of cell pixels is significantly less than the background. It is a penalty in the loss function to weight the correct prediction of the cell pixel more than the background pixel.

$$\text{Dice}(p, q) = \sum_{i=1}^{n} \frac{2p_i \cdot q_i}{p_i + q_i}$$

(4.4)

where $n$ is the number of pixels of each image, $q_i$ is the ground truth label for the pixel $i$, set as 0 for the background, and 1 for the nucleus. And $p_i$ denotes the predicted label of the corresponding pixel.

The loss function is defined in Equation 4.5.

$$\text{Loss}(p, q) = w \cdot BCE(p, q) - \text{Dice}(p, q)$$

(4.5)
where the prediction $\mathbf{p}$ is the output of $f_{UNet}(\mathbf{X}; \theta)$ with $\theta$ as its parameters. $w$ is a weight term as a parameter, we use 0.5 in the experiment after tuning.

As in the original U-Net, we initialize the weights of convolutional kernels from a truncated normal distribution [87] centered on 0 with standard deviation $\sqrt{(2/N)}$, where $N$ is the number of units in the weight tensor.

We use Adam optimizer [118, 70] for U-Net optimization. Adam means adaptive moment estimation. Instead of updating the model weights by gradients from the gradient descent algorithm directly, Adam optimizer updates the parameters using a function of the gradients, an exponentially decaying average of past gradients, and squared gradients as defined in Algorithm 1.

In Algorithm 1, $\mathbf{m}$ is the momentum vector, initialized as all zeros, for smoothing the gradient updates. As we mentioned, $\theta$ are the parameters including the weights of convolutional layers and the logistic regression layer in the U-Net model. From Equation 4.2 and Equation 4.5 we know that the loss $Loss(\mathbf{p}, \mathbf{q})$ is a function of $\theta$. Thus, $\nabla_\theta Loss(\theta)$ is the gradient of the loss function of the parameters. $\beta_1$ is a decay factor, used to calculate an exponential decayed moving average of gradients at each training iteration. In our experiment, we set $\beta_1$ to be 0.9, which is the default of the algorithm. The vector $\mathbf{s}$, initialized as 0, is for scaling the momentum vector $\mathbf{m}$. $\beta_2$ is a decay factor used to calculate an exponential decayed moving average of gradients at each training iteration. In our experiment, we set $\beta_2$ to be 0.9, which is the default of the algorithm. $\hat{\mathbf{m}}$ is scaled momentum vector and $t$ denotes the iteration. The vector $\hat{\mathbf{s}}$ is scaled from $\mathbf{s}$ and $t$ denotes the iteration.

Finally, the parameters $\theta$ can be updated by Line 14 in Algorithm 1. $\eta$ is the learning rate initialized to be 0.00001, which is determined by experiments, $\varepsilon$ is a smoother term to avoid division by zero. $\odot$ means the element-wise division.

We train the model for 200 iterations at most but will stop training early when the loss is not decreasing.
Algorithm 1 Adam Optimization

\[ \varepsilon = 10^{-7} \quad \text{▷ Initialize the term } \varepsilon \text{ to avoid zero division} \]

\[ \eta = 10^{-5} \quad \text{▷ Initialize the learning rate } \eta \]

\[ m = 0 \quad \text{▷ Initialize the moment vector } m \]

\[ s = 0 \quad \text{▷ Initialize the scale vector } s \]

\[ \beta_1 = 0.9 \quad \text{▷ Initialize the first decay factor } \beta_1 \]

\[ \beta_2 = 0.999 \quad \text{▷ Initialize the second decay factor } \beta_2 \]

\[ \text{iter} = 0 \quad \text{▷ Initialize the iteration variable } \text{iter} \]

\[ \text{iter}_{\text{max}} = 200 \quad \text{▷ Initialize the maximum iteration} \]

\[ \text{while } \text{iter} < \text{iter}_{\text{max}} \text{ do} \]

\[ m \leftarrow \beta_1 m - (1 - \beta_1) \nabla_{\theta} \text{Loss}(\theta) \quad \text{▷ Update the moment vector } m \]

\[ s \leftarrow \beta_2 s + (1 - \beta_2) \nabla_{\theta} \text{Loss}(\theta) \otimes \nabla_{\theta} \text{Loss}(\theta) \quad \text{▷ Update the scale vector } s \]

\[ \hat{m} \leftarrow \frac{m}{1 - \beta_1^{\text{iter}}} \quad \text{▷ Scale the moment vector } m \]

\[ \hat{s} \leftarrow \frac{s}{1 - \beta_2^{\text{iter}}} \quad \text{▷ Scale the scale vector } s \]

\[ \theta \leftarrow \theta + \eta \hat{m} \otimes \sqrt{\hat{s} + \varepsilon} \quad \text{▷ Update the parameters of the model.} \]

\[ \text{iter} = \text{iter} + 1 \quad \text{▷ Increase the iteration counter} \]

end while

4.2 Experiment

In this section, we show the results obtained by training a U-Net model (see Section 4.1) in identifying cell nuclei and cytoplasm from bright-field images. The objective of our experiment is that of evaluating the robustness of an automatic deep learning-based method for cell segmentation. In particular, we are interested in evaluating the accuracy of the approach to investigate whether the results meet the requirements of the following tasks.

4.2.1 Data Description

The dataset used in our experiments is composed of 3,468 images acquired with GE IN Cell Analyzer 2500 HS. Images are subdivided across 12 fields of view, with 289 images per field. The image size is 2040 \times 2040 pixels.

Images present a single cell type (i.e., MCF10A), which is a non-tumorigenic epithelial cell.
Images are acquired at 20x objective magnification and present two types of fluorescent proteins. A red fluorescent protein, called mCherry, is used to spot cell nuclei. A green fluorescent protein, called Clover, is used to spot cell cytoplasm.

Figure 4-5 shows an example of the input data used in the remaining of this project. Figure 4-5a shows a bright-field image, which can be thought of as the main “picture” acquired by the microscope. Figure 4-5b shows the fluorescent image corresponding to the mCherry protein. We notice that cell nuclei emit a bright red light. Figure 4-5c shows the fluorescent image corresponding to the Clover protein. We notice that, in this case, the green light is emitted by the cell cytoplasm.

![Figure 4-5: (a) Bright field image, (b) fluorescent image corresponding to mCherry protein, and (c) fluorescent image corresponding to Clover protein](image)

### 4.2.2 Data Preprocessing

Two types of annotations have been used.

All images have been manually annotated with cell position. That is, we annotate the position of each cell nucleus. Annotations have been created with the open-source platform called ImageJ Fiji [186] and the TrackMate [205] plugin.

Automatic annotation is also created by leveraging fluorescent images. As we can see from Figure 4-5, the signal-to-noise ratio of fluorescent images is too weak to be used as a label to identify image features (e.g., nuclei or cytoplasm). To this end, feature masks are created following the steps illustrated in Figure 4-6.
Figure 4-6: Five steps used for creating binary masks from fluorescent (mCherry) images. (a) Fluorescent image. (b) Image after the rolling ball algorithm [196]. (c) Blurred image obtained by means of a Gaussian function. (d) Binary mask obtained with the local threshold algorithm [15]. (e) Binary mask after removing small objects. (f) Green boxes indicate the bounding boxes computed for each object in the final binary mask.

The rolling ball algorithm [196] is used to distinguish the background and foreground and to create an evenly illuminated background. Intuitively, the rolling ball algorithm interprets a gray-scale image as a surface $S$. A new surface $S'$ is created by rolling a ball with radius $r$ under $S$ and by collecting all points reached by the top part of the ball.

Figure 4-7 illustrates an example to obtain the background of an image with uneven illumination.
The image profile before and after the rolling ball algorithm.

Figure 4-7: Illustration of the rolling ball algorithm. We show the gray-scale values along the x-axis for a specific y coordinate. The blue line indicates the original pixel values. The red line represents the background found by the top part of the rolling ball, as illustrated by the gray circle. The orange line indicates the result after the background subtraction. The radius of the ball should be greater than the size of the object in the foreground. For example, we use radius $r = 150$ for removing the background of fluorescent images of nuclei.

The ball radius is a user-defined parameter. Figure 4-6b shows a fluorescent (mCherry) image after background subtraction operated by the rolling ball algorithm with radius $r = 150$. In the second step, images are blurred by using a Gaussian function with 4 pixels as the standard deviation to further remove noise (see Figure 4-6c). Next, binary masks are created by local thresholding [15]. This approach uses a local window, a user-defined contrast threshold and a local threshold. The local threshold is set as the local middle-gray value (i.e., the mean of the minimum and maximum grey values in the local window). The pixel is defined as background if the pixel value is less than the middle-gray (otherwise the pixel is defined as foreground). Figure 4-6d shows the result of local thresholding.

Masks are processed to identify independent components, and masks are only retained with
composed by a number of pixels between 100 and 1000. Figure 4-6e shows the image used in the running example after filtering masks based on their size. Finally, bounding boxes are computed for each obtained mask (see Figure 4-6f).

![Images](image1.png)  
(a) Fluorescent image of cytoplasm.  
(b) De-noised image.  
(c) Background subtraction.  
(d) Binary masks of cytoplasm.  
(e) Binary masks of nuclei.  
(f) Overlay.

Figure 4-8: Three steps used for creating binary masks from fluorescent (Clover) images. (a) Fluorescent image. (b) Blurred image obtained by means of a Gaussian function. (c) Image after the rolling ball algorithm [196]. (d) Binary mask obtained with Huang’s method [99]. (e) Binary mask of cell nuclei removing small objects. (f) Green regions indicate the mask computed for cytoplasm, and orange regions indicate the mask computed for the corresponding nuclei mask.

Fluorescent images of cytoplasm undergo a similar pre-processing (see Figure 4-8) with a few differences. Images are blurred before applying the rolling ball algorithm to avoid high-intensity pixels to break the cytoplasm of a single cell in multiple objects. To generate binary masks, local thresholding is not ideal since it works well in practice only for regular shapes. We use Huang’s method [99] instead. Huang’s method searches for an optimal threshold that can minimize the distance between the original image and the binary mask. The distance is defined as the sum of differences between the gray value of a pixel $p$ and the mean of gray values of all pixels belonging to
the class of $p$. Finally, to avoid potential holes in the mask, the mask identifying nuclei is overlaid to the mask identifying cytoplasm.

### 4.2.3 Segmenting cell nuclei and cytoplasm

The U-Net model described in Section 4.1 has been used to automatically identify cell nuclei and cytoplasm from bright-field images. The input is the sole bright-field image (see Figure 4-5a), the output is a binary mask indicating either the cell nuclei present in the image or the cytoplasm.

To train the U-Net model, we use masks automatically generated with the pre-processing pipeline discussion in Section 4.2.2.

Images contained in the 12 fields are randomly split into four groups for cross-validation. For each run, the U-Net model is trained with 2601 images from 9 fields and tested with 867 images from the remaining 3 fields. We use different evaluations for nuclei and cytoplasm.

For cell nuclei, we perform an object-level evaluation by means of the manually annotated nuclei positions. Figure 4-9 shows the list of possible outcomes of our object-level validation. We say that a cell nucleus is correctly predicted (i.e., true positive) if the annotated nuclei position falls inside a predicted mask (Figure 4-9(a)). Predicted masks without matched annotations are false positives. This can be due to a nucleus being recognized as two components (Figure 4-9(b)), due to debris or other morphological artifacts in the original image (Figure 4-9(c)), or due to errors in the annotations (Figure 4-9(d)). False negatives are annotated nuclei with no overlapping masks (Figure 4-9(e) and Figure 4-9(f)). Sometimes, this is caused by two nuclei with a single (i.e., connected) predicted mask (Figure 4-9(g)). Particularly challenging is when the annotated nucleus position falls outside of a predicted mask as this counts as both false positive and false negative (Figure 4-9(h)).
Figure 4-9: Comparison of annotations and predicted nuclei for a group of cells. Red crosses indicate the position of an annotated nucleus. Predicted masks are color coded according to a categorical color map (i.e., different nuclei masks are depicted with different colors). (a) A true positive corresponds to an annotated nuclei falling inside a predicted mask. False positives are created by (b) split masks, (c) debris, or (d) mistakes in the annotations. False negatives may be created by (e-f) nuclei not recognized by the model, or when two distinct nuclei are predicted as one. (h) The annotate nucleus position falls outside the predicted mask causing the error to be counted as both false positive and false negative.

Two different metrics are used to run the object-level validation of the results, namely precision and recall. Figure 4-10 shows a graphical representation of how precision and recall are defined.
Precision is the ratio of correctly predicted nuclei to the total predicted nuclei, i.e.,

\[
Precision = \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false positives}}
\]  

(4.6)

where \# indicates the number of true or false positives. High precision relates to the low false positive rate. Recall is the ratio of correctly predicted nuclei to the total of nuclei, i.e.,

\[
Recall = \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false negatives}}
\]  

(4.7)

The question that the recall answers is: Of all the nuclei, how many did we label?

A third metric is the F1-score which is the weighted average of precision and recall, i.e.,

\[
F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}
\]  

(4.8)

F1 is useful because it takes into account both false positives and false negatives.

Results obtained from four-fold cross-validation are reported in Table 4.1. The last row indicates average scores for recall, precision, and F1 scores.

On average, the U-Net model achieves 88.9\% Recall, which means about 89\% of the cell nuclei are correctly detected. The precision is 92.0\%, which means only one-tenth of predicted
nuclei are false positives. Notice that this does not take into account possible errors present in the annotations.

F1 score is 90.4% on average, which provides an average measurement of recall and precision, which indicates a good overall accuracy of the model.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Recall (%)</th>
<th>Precision (%)</th>
<th>F1 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation 1</td>
<td>86.9</td>
<td>94.6</td>
<td>90.6</td>
</tr>
<tr>
<td>Validation 2</td>
<td>92.8</td>
<td>92.6</td>
<td>92.7</td>
</tr>
<tr>
<td>Validation 3</td>
<td>88.2</td>
<td>91.4</td>
<td>89.7</td>
</tr>
<tr>
<td>Validation 4</td>
<td>87.8</td>
<td>89.3</td>
<td>88.5</td>
</tr>
<tr>
<td>Average</td>
<td><strong>88.9</strong></td>
<td><strong>92.0</strong></td>
<td><strong>90.4</strong></td>
</tr>
</tbody>
</table>

To investigate the main causes affecting the performance of the U-Net model, we have analyzed the vary of recall and precision based on the number of cells in each image, which has been reported as problematic for cell segmentation [210].

Figure 4-11 shows the relationship between recall and the number of cells for each image in the entire dataset. Each run of cross-validation tests three fields of data until all the data are tested.
Figure 4-11: Relationship between recall and number of cells. Each marker corresponds to an image in our test set. Markers of the same type (i.e., same color and same shape) correspond to images in the same field of view.

Overall, Recall is limitedly affected by the increase in the number of cells. However, we can notice that Recall is particularly unstable when the number of cells is low. This is expected since, in these cases, small errors impact more severely on recall. Examples of images with low Recall are illustrated in Figure 4-12. The lower the recall, the more challenging it is to establish the complete track. However, a small number of false-negative detections in the middle of the track can be fixed using its historical and future information at the tracking phase.
Figure 4-12: Images with low recall values due to a limited number of cells. Red crosses indicate annotated cell nuclei. Predicted masks are color coded according to a categorical color map (i.e., different masks have different colors). Red circles indicate false negatives, blue circles indicate false positives. TP, FN, and FP denotes the number of true-positive, false-negative, and false-positive, respectively. (a) Only four cells are missed but the low number of cells (i.e., ten), leads to 60% recall. (b) False positives and false negatives are mainly introduced by annotations not aligning with the predicted masks. (c) Most false negatives are caused by cells touching each other or undergoing mitosis.

Exploring the effect of cells on precision is expected to provide information about the false positives. Figure 4-14 shows similar relationships as for recall. While the instability of precision appears again on images with a low cell number (see Figure 4-14), we also notice a field (i.e., A_07fld04) where precision and number of cells have a negative correlation.
Figure 4-13: Relationship between precision and number of cells. Each marker corresponds to an image in our test set. Markers of the same type (i.e., same color and same shape) correspond to images in the same field of view.

Figure 4-15 shows a comparison between two images, one from the field A_02fld09 (Figure 4-15(a)), and the other from the field A_07fld04 (Figure 4-15(b)). The two images have similar densities but contain a different amount of debris or contamination. The presence of noise clearly impacts the number of false positives, which affect precision. The lower the precision, the more challenging it is to correctly link cells belonging to the same track. The tracking algorithm requires to avoid linking cells to false-positive detections. In addition, denoising techniques are needed when there is significant noise affecting precision.
Figure 4-14: Images with low precision due to a limited number of cells. Red crosses indicate annotated cell nuclei. Predicted masks are color coded according to a categorical color map (i.e., different masks have different colors). Red circles indicate false negatives, blue circles indicate false positives. TP, FN, and FP denotes the number of true-positive, false-negative, and false-positive, respectively. (a) Five false positives indicate errors in the annotation. These severely affect precision since the total number of cells is eleven. (b) In addition to three false positives on the border of the image, two predicted masks fail to cover the annotated nuclei, and another two predicted masks on the cell contacted with others match no annotations.
Figure 4-15: Comparison of performance obtained with (a) a clean image, and (b) an image with debris. Red crosses indicate annotated cell nuclei. Predicted masks are color coded according to a categorical color map (i.e., different masks have different colors). Red circles indicate false negatives, blue circles indicate false positives. TP, FN, and FP denotes the number of true-positive, false-negative, and false-positive, respectively. (a) Despite the high number of cells only a few number of false positives appear. (b) Debris or contamination affect severely the total number of false positives.

For the cell cytoplasm, performing an object-level evaluation is unfeasible. Predicting distinct cytoplasm masks for each cell is still considered an open problem [28].

To this end, we use a pixel-level evaluation of the predicted cytoplasm masks using the masks automatically generated from the fluorescent labels as ground truth. We use two metrics, namely Pearson correlation defined in Equation 4.9 and Jaccard score defined in Equation 4.10.

Pearson correlation measures the linear correlation between two variables. It assumes values between 1 and -1 where, 1 indicates positive linear correlation and -1 indicate negative linear correlation. In the case of images, Pearson’s correlation measures the correlation of pixel-by-pixel intensities. This is defined as follows:

$$\text{Correlation}(p, q) = \frac{\sum_{i=1}^{n} (p_i - \bar{p})(q_i - \bar{q})}{\sqrt{\sum_{i=1}^{n} (p_i - \bar{p})^2} \sqrt{\sum_{i=1}^{n} (q_i - \bar{q})^2}} \quad (4.9)$$

where $n$ is the number of pixels, $p_i$ and $q_i$ are predicted label value and ground truth label value.
of the pixel $i$. $\bar{p}$ and $\bar{q}$ are mean of pixel labels for two images. The ground truth label $q$ can be 0 representing the background (otherwise 1 representing the object). The predicted label $p$ is within the range of $[0, 1]$.

The Jaccard Index, also known as Intersection over Union, is defined as the size of the intersection divide by the size of the union of the sample sets. When working with images, the Jaccard index is computed pixelwise as follows:

$$Jaccard(p, q) = \frac{\sum_{i=1}^{n} \text{Threshold}(p_i)q_i}{\sum_{i=1}^{n} (\text{Threshold}(p_i) + q_i - \text{Threshold}(p_i)q_i)}$$

(4.10)

where $n$ is the number of pixels, $p_i$ and $q_i$ are predicted label values and ground truth label values of the pixel $i$. $\text{Threshold}$ means thresholding the component of the prediction vector into 1 if the component is greater than 0.5, otherwise 0. We use binary masks generated from the fluorescent images as the ground truth. The ground truth label $q$ can be 0 representing the background (otherwise 1 representing the object).

Table 4.2 shows the results obtained after four-fold cross-validation. Overall, performance is satisfying also thanks to the high signal-to-noise ratio. This is confirmed by the visual exploration of the results.

Table 4.2: Cytoplasm segmentation performance comparing with the fluorescent labels.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Correlation</th>
<th>Jaccard (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation 1</td>
<td>0.883</td>
<td>80.6</td>
</tr>
<tr>
<td>Validation 2</td>
<td>0.887</td>
<td>81.0</td>
</tr>
<tr>
<td>Validation 3</td>
<td>0.906</td>
<td>82.6</td>
</tr>
<tr>
<td>Validation 4</td>
<td>0.842</td>
<td>78.1</td>
</tr>
<tr>
<td>Average</td>
<td><strong>0.880</strong></td>
<td><strong>80.6</strong></td>
</tr>
</tbody>
</table>

Figure 4-16 shows the segmentation results of cytoplasm for low values of the Jaccard index. Overall, predicted masks show satisfying visual accuracy, and most pixels occupied by images in the bright field data are correctly labeled.
Figure 4-16: Comparison of segmentation performance of cytoplasm for different densities. Red masks indicate fluorescent labels while green masks indicate predicted masks. The orange color, caused by merging red and green colors shows the overlap between two masks. The cytoplasm segmentation tightly overlaps the fluorescent labels. (a-c) Fluorescent labels overlaid on the bright-field images. (d-f) Predicted masks overlaid on the bright-field images. (h-i) Fluorescent labels overlaid on the predicted masks.
4.3 Conclusion

We propose an approach based on the U-Net model to segment cells from bright-field images. We call the segmentation approach “stain-free” since it enables cell detection without the need for staining cells by fluorescent dyes. To avoid manual annotation of massive images to train the U-Net model, we propose a pipeline to pre-process fluorescent labels of nuclei and cytoplasm as the label for training. To evaluate the segmentation performance, we build a dataset containing 3,468 bright-field images and corresponding fluorescent images from twelve fields of view. We perform four-fold cross-validation using all the images. We train the U-Net model with nine fields of data and test with the rest three alternatively. We use recall, precision, and F1 score to measure the performance of nuclei segmentation. Since the separated cytoplasm ground truth is infeasible to obtain, we use the Pearson correlation and Jaccard index to measure the performance of cytoplasm segmentation. We achieve an average recall 88.9% and precision 92.0% for nuclei segmentation. Also, we achieve an average Correlation 0.88 and Jaccard index 80.6%. We further analyze the relationship between cell density and segmentation performance. The analysis shows that the segmentation performance is robust to the density change but sensitive to the heavy noise on images from a field of view.

The experiment result shows that we can achieve good segmentation performance, which is robust to cell density change. On average, most cells can be correctly detected, thus providing a good foundation for tracking.

In terms of using fluorescent labels for training the segmentation model, we notice some limitations. Some fluorescent labels can be too weak to be recognized, which makes the pre-processing methods very challenging. Furthermore, the same pre-processing pipeline may not work for different fluorescent labels. For example, the pipeline for creating binary masks from fluorescent images of nuclei is different from what we use for cytoplasm.

In conclusion, we demonstrated the feasibility of stain-free cell identification using a segmentation model trained with automatically-generated nucleus labels. In the following chapters, we will focus on the cell linking problem at low frame rate.
Chapter 5

Morphological Feature based
Velocity Regression

After cell identification, the next step is to link cells into tracks. If we know the current velocity of a cell, we can predict its future position. When the position prediction is accurate enough, linking cells by the proximity between the predicted position and future cell candidates become achievable. Thus, we are inspired to design an approach for estimating cell velocity. We start from the observation of cell movement patterns using our data and the investigation of theories behind the observed patterns.

We observe that the cell morphology suggests its motion pattern, especially when there is no contact between cells. Figure 5-1 illustrates cell morphological states with corresponding moving directions by cartoons. These samples suggest that cells tend to move along the axis of their elongation when there are no close neighbors.

In addition, biological studies [197] show that the direction of cell movement is consistent with its lamellipodium (the protein actin projection on the leading edge of the cell) and tail, as shown in Figure 5-2. Cell first extends its lamellipodium and adheres it to the front ground. Then it contracts its back and de-adhere to its old attaching place. This process is reflected in cell morphology so that we may predict the movement from its morphological features.

Recently, Nishimoto et al. [167] proposed a ConvNets based approach to classify the cropped images of moving cells into one of four directions, which provides a simple way of extracting motion
information from cell images. However, the moving cells require to be manually cropped, and the motion inference is only limited to four directions. Thus, we are motivated to design a more general and automatic approach enabling velocity estimation.

Figure 5-1: Morphology and motion pattern. Cartoons illustrate the cell shapes and moving directions (by the black arrow). The color-coded trajectories in the bright-field images indicate the future cell movement. Cross marks the cell position at each frame.

This chapter studies the problem of velocity prediction using morphological features of cells. Inspired by the observation and supporting theories that cell morphology suggests its motion information. Section 5.1 proposes a set of morphological features and two models for cell velocity regression. Section 5.2 describes data preprocessing techniques and experiment results. Finally, Section 5.3 draws the conclusion and introduces the further improvement to accomplish in the following chapter.
5.1 Velocity Regression

Inspired by our observation and related studies [197], we investigated an approach to predict the cell velocity from its morphological features.

Figure 5-3 shows the workflow of the velocity regression. We first obtain the cell masks from binary segmentation, then fit the ellipse on cell masks to extract morphological features, and finally perform a velocity regression.
To separate cells from the binary mask, we apply the marker-controlled watershed algorithm [18], which is commonly used for the instance segmentation when clear markers are available for each region to separate. The marker-controlled watershed algorithm processes the cell mask as a topological surface, where the pixel value denotes the height. Each marker floods its region until reaching the border or the regions belonging to other markers. Finally, all cell masks are separated to belong to different markers. Here we use the predicted nuclei as markers. Finally, we can obtain the features from the individual cell and nucleus mask as we described in Section 5.1.2.

5.1.1 Ellipse Fitting

We start by computing morphological features from the cell masks and then predict the direction and the displacement. Although the manually designed features may underperform the features learned by deep learning approaches, morphological features provide more capability for showing the intuitive connection between features and the prediction. In addition, deep learning approaches can be used to further improve the performance after the traditional approaches are proved effective. We use ellipses for modeling the shape of cell segmentation due to its simplicity and representative capability for cell elongation.

We tested three ways to fit an ellipse on a cell mask, as shown in Figure 5-4. For the first method, we use a convex hull to bound the mask, then fit the contour of the convex hull using an ellipse. For the second, we use a minimum-area rectangle to bound the mask, then find an ellipse enclosed by the rectangle. For the last, we use a convex hull to bound the mask, then choose the pair of the most distant points as the endpoints of the major axis. The length of the minor axis is obtained by averaging the distances over all other vertices to the major axis.

Based on the sampled experiment results, we observe that the ellipse fitting based on the longest axis of the convex hull is more robust to noises (irregular shapes). Thus we use it for morphological feature extraction.
5.1.2 Morphological Feature

To predict the cell velocity for each detection, we compute 21 features related to time, cell density, and shape. We consider time as a feature related to cell movement since cells tend to become inactive as time increases. In addition, the environment becomes more crowded, which limits cell activities. Figure 5-5 shows the rest of the features.
Figure 5-5: Features and labels. Segmentation features include areas and the vector defined by mass centers of the cell and the nucleus, \( \mathbf{u}_{cn} \), of cell and nucleus masks. The area changes when the cell moves or divides. For example, cells shrink into a small circle when they undergo mitosis, showing almost no movement. Density features count the number of neighbor cells within a range and the number of cells in the current image. Intuitively, the environment density may limit the ability of cell movement. The vector \( \mathbf{u}_{cn} \) is used to capture the position of the nucleus relative to the cytoplasm during the cell movement. Ellipse fitting features, derived from the ellipses fitted on cell and nucleus masks, indicate the direction along which the cell elongates and the degree of the elongation. These features include the eccentricity, the unit orientation vector, the length of major and minor axes of ellipses fitted on the cell and the nucleus. Two vectors \( \mathbf{u}_{ne} \) and \( \mathbf{u}_{ec} \) are also included. The vector \( \mathbf{u}_{ne} \) is defined by the mass center of the nucleus mask and the center of the ellipse fitted on the cytoplasm mask. The vector \( \mathbf{u}_{ec} \) is defined by the center of the ellipse fitted on the cell and the mass center of the cell mask. The regression targets, as known as labels, are defined as the displacement \( \delta a \) and \( \delta b \) along major and minor axes of the fitted ellipse during the imaging interval, also denoting the velocity. Given the orientation of the ellipse, the displacement can be converted into vertical and horizontal directions.

Our proposed regression models and loss functions are described in details in Appendix A.

5.2 Experiment

We design the experiments to test the performance of the velocity prediction using proposed regression models and morphological features. In detail, we compare the proposed neural network model with a linear regression model to find out whether the relationship between the features and targets is linear. In addition, we test the two regression models using the feature sets with and without morphological features to show the impact of morphological features. We demonstrate the evaluation results quantitatively and visually in Section 5.2.3.
5.2.1 Data Description

We perform the regression experiment using the MCF10A bright-field images, as we described in Section 4.2.1. The data are acquired in three days and manually annotated with cell positions and tracks. Figure 5-6 shows examples of annotated tracks, predicted segmentation, and corresponding fluorescent labels.

![Figure 5-6: Visualization of annotated tracks.](image)

(a) Annotated tracks.
(b) Segmentation.
(c) Fluorescent image.

Figure 5-6: Visualization of annotated tracks. (a) The annotated tracks are color-coded on a bright-field image (i.e. different colors indicate different tracks). The bright-field image is acquired at the first timestamp of all the tracks visualized. The tracks include cross markers and line segments between them. The markers and line segments indicate the future 40 cell positions and the movement between two consecutive frames. (b) The predicted cell segmentation for extracting features. Green and red regions indicate the predicted cytoplasm and nuclei, respectively. Due to the effect of color combination, the nuclei show orange. (c) The fluorescent image corresponding to the bright-field images. Green and red regions indicate fluorescent labels of cytoplasm and nuclei.

We split the data for training and testing for two requirements. Firstly, we want to make sure the size ratio between the test set and the training set to be about at least one-tenth, which is a rule of thumb for machine learning tasks. Secondly, we keep the cells belonging to the same tracks in either the training or testing set. In detail, we use all the bright-field images from five fields of view and the corresponding annotations to build a dataset for the velocity regression experiment. According to the annotation, there are about 43,947 cells belonging to 540 tracks. We randomly select 30 tracks (six tracks for each field of view), which contains about 3,905 cells as the test set. We use the rest 40,042 cells as the training set.
5.2.2 Data Preprocessing

For all cells in our dataset created in Section 5.2.1, we need to create feature vectors associated with them. We start from the separation of cell segmentation. Figure 5-7 shows examples of the input bright-field image, the binary segmentation, and the separated segmentation.

The segmentation is firstly converted into a binary mask by thresholding. The pixel value denotes the probability that the current point belongs to a cell or nucleus. Since the contrast between the foreground and the background is high, we use 0.5 as the threshold.

Then we remove nuclei and cell (cytoplasm) masks with their area smaller than 400 and 4000 pixels, respectively. After the area-based filtering, the small artifacts can be eliminated.

In addition, the erosion operation is applied to cell masks to reduce the connected area between contacted cells. In detail, we check every $3 \times 3$ patch in the image. If all pixels are equal to 1, we assign the center pixel to be 1. Otherwise, we assign it to be 0. We perform the erosion operation for two iterations. Then we separate cell masks using the method described in Section 5.1.

![Figure 5-7: Cell separation from the binary segmentation. (a) Bright-field image. (b) Green and red regions indicate the predicted cytoplasm and nuclei, respectively. Due to the effect of color combination, the nuclei show orange. (c) Color-coded masks indicate individual cells (i.e. different colors denote different cells).](image)

Each sample is represented by a 21-dimension feature vector associated with two labels (see Section 5.1.2 for details), which are displacements along the major and minor axes of ellipses fitted on cells. We count the number of cells within the 500-pixel distance of the target cell as its local density feature. In addition to testing the velocity regression performance using the complete feature set, we select a subset of features to understand the impact of the ellipse fitting. The subset of
features includes time, the nucleus area, the cell area, the vector defined by the centroids of the cell (cytoplasm) mask and the nucleus mask, the local density, and the global density.

5.2.3 Regression Performance

We test the regression models with the complete feature set and its subset under the same setting. We name the complete feature set by “Morphology” and its subset by “Baseline” merely for convenience. We scrambled the morphological features for each sample to create a new feature set named “Shuffle”, which works as another control group for the morphological features.

We use the root mean squared error (RMSE) and the Pearson correlation coefficient, as described in Section 4.2.3, to evaluate the regression performance. The two input variables to the correlation function are the ground truth displacement and predicted displacement. The absolute displacement is defined by the root sum squared of the displacements along the major axis and the minor axis. The best performance result in each column is in bold.

Tables 5.1, 5.2, and 5.3 show the performance in terms of RMSE of proposed regression models using three sets of features. RMSE evaluates an average of the distance between the predicted cell position and the ground truth. The smaller RMSE, the better. We show the errors of the predicted displacement along with the major and the minor axes and the total displacement. We will understand the results from three perspectives, including the impact of models, features, and time intervals.

In addition, to have an intuitive understanding of the predicted error of displacement prediction compared to the nuclei size, we calculate the length distribution of the major and minor axes of the ellipses fitted on the nuclei of our dataset. The mean values of major and minor axes are about 60 and 25 pixels, respectively. Considering that the cell size is much larger than its nucleus, the error of predicted displacement is small if it is comparable to the nucleus size.

Overall, Tables 5.1, 5.2, and 5.3 show that all error values of displacements are relatively small compared to the nuclei size, which can be covered by a circle with the diameter of 60 pixels. The proposed Multilayer Perceptron (MLP) using the complete feature set achieves the lowest error values. It also shows robustness to the increment of the time interval in all three tables.

In terms of the features, ellipse-related features improve the MLP prediction of the displacement along the major axis in Table 5.1, as well as the displacement magnitude in Table 5.3. In addition, it shows no harm to the prediction of the displacement along the minor axis in Table 5.2.
Table 5.1: RMSE of displacement along the major axis.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Model</th>
<th>15 minutes</th>
<th>1 hour</th>
<th>2 hours</th>
<th>4 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Linear Regression</td>
<td>12.09</td>
<td>38.92</td>
<td>68.67</td>
<td>112.83</td>
</tr>
<tr>
<td>Baseline</td>
<td>Multilayer Perceptron</td>
<td>12.00</td>
<td>38.30</td>
<td>67.28</td>
<td>111.29</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Linear Regression</td>
<td>12.18</td>
<td>39.32</td>
<td>69.49</td>
<td>113.93</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Multilayer Perceptron</td>
<td>12.18</td>
<td>39.34</td>
<td>69.49</td>
<td>113.92</td>
</tr>
<tr>
<td>Morphology</td>
<td>Linear Regression</td>
<td>11.97</td>
<td>38.49</td>
<td>68.04</td>
<td>112.18</td>
</tr>
<tr>
<td>Morphology</td>
<td>Multilayer Perceptron</td>
<td>11.52</td>
<td>36.39</td>
<td>64.64</td>
<td>108.80</td>
</tr>
</tbody>
</table>

Table 5.2: RMSE of displacement along the minor axis.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Model</th>
<th>15 minutes</th>
<th>1 hour</th>
<th>2 hours</th>
<th>4 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Linear Regression</td>
<td>4.65</td>
<td>13.27</td>
<td>23.47</td>
<td>43.95</td>
</tr>
<tr>
<td>Baseline</td>
<td>Multilayer Perceptron</td>
<td>4.65</td>
<td>13.22</td>
<td>23.47</td>
<td>43.77</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Linear Regression</td>
<td>4.67</td>
<td>13.33</td>
<td>23.64</td>
<td>44.09</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Multilayer Perceptron</td>
<td>4.67</td>
<td>13.34</td>
<td>23.62</td>
<td>44.12</td>
</tr>
<tr>
<td>Morphology</td>
<td>Linear Regression</td>
<td>4.65</td>
<td>13.28</td>
<td>23.49</td>
<td>43.97</td>
</tr>
<tr>
<td>Morphology</td>
<td>Multilayer Perceptron</td>
<td>4.61</td>
<td>13.14</td>
<td>23.32</td>
<td>43.68</td>
</tr>
</tbody>
</table>

Tables 5.1 and 5.3 show that MLE using morphological features outperforms MLE using scrambled features, especially when the time interval is long.

As the time interval increases, the error from all combinations also increases. However, the average velocity error decreases, which is an advantage for predicting cell position in the long time interval.

Given the fact that the error achieved by the linear regression model is already small compared to the nuclei size, the proposed MLP outperforms the linear regression model regardless of the feature set and the time interval.

Table 5.1 and Table 5.2 show obvious differences between the error of predicted displacement along the major and minor axes. Since all the experiment settings except the direction are the same, such a difference may suggest cell movement patterns along with the fitted major and minor axes are different.

Table 5.3: RMSE of the absolute displacement.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Model</th>
<th>15 minutes</th>
<th>1 hour</th>
<th>2 hours</th>
<th>4 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Linear Regression</td>
<td>12.47</td>
<td>39.45</td>
<td>69.55</td>
<td>115.87</td>
</tr>
<tr>
<td>Baseline</td>
<td>Multilayer Perceptron</td>
<td>11.54</td>
<td>35.83</td>
<td>63.29</td>
<td>106.23</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Linear Regression</td>
<td>12.86</td>
<td>40.92</td>
<td>72.10</td>
<td>120.23</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Multilayer Perceptron</td>
<td>12.89</td>
<td>41.09</td>
<td>72.32</td>
<td>120.92</td>
</tr>
<tr>
<td>Morphology</td>
<td>Linear Regression</td>
<td>12.34</td>
<td>39.05</td>
<td>68.97</td>
<td>115.12</td>
</tr>
<tr>
<td>Morphology</td>
<td>Multilayer Perceptron</td>
<td>10.99</td>
<td>34.15</td>
<td>60.21</td>
<td>100.06</td>
</tr>
</tbody>
</table>
Tables 5.4, 5.5, and 5.6 show the performance in terms of the Pearson correlation of proposed regression models using three sets of features. The correlation score indicates the linear association between the predicted displacement and the ground truth. The range of the correlation score is within $[-1, 1]$. The scores 1 or $-1$ mean that the prediction and the ground truth have a completely positive or negative linear relationship, respectively. The closer to 1, the better the correlation score.

Tables 5.4, 5.5, and 5.6 show the correlation between the ground truth displacement and the predicted displacement. We will investigate the results from three perspectives, including the impact of models, features, and time intervals.

Overall, Tables 5.4, 5.5, and 5.6 show that most combinations of unscrambled feature sets and models are able to achieve weak or moderate correlation scores between the prediction and the ground truth. Similar to the results in terms of RMSE, MLP combined with the complete feature set achieves the best performance in terms of the correlation for all types of displacements and time intervals but with more considerable margins. Specifically, the prediction from MLP using baseline and morphological features shows moderate correlations to the ground truth as shown in Table 5.4 and Table 5.6 and weak correlations as shown in Table 5.5 for different time intervals. The prediction of models using scrambled morphological features shows no correlations in all experiments.

When using MLP, features from the fitted ellipses show a significant improvement and a slight improvement to the displacement prediction along the major axis in Table 5.4 and the minor axis in Table 5.5, respectively. Considering that a slight improvement is seen for absolute displacement prediction in Table 5.6, which may suggest that MLP without morphological features may overfit the data to achieve a high mean squared error but ending up with inaccurate velocity direction.

In terms of the robustness of the regression to the time interval changes, the stable correlations with respect to the change of time intervals also support that the prediction captures the direction of movement.

For comparing the regression models, the proposed MLP outperforms the linear regression model, especially in terms of the displacement along the major axis and the absolute displacement. Thus, the relationship between the proposed features and the target displacement seems to be non-linear.

Similar to Tables 5.1 and 5.2, Tables 5.4 and 5.5 show obvious differences between the correlation of predicted displacement along the major and minor axes. The correlation differences between the two tables may suggest that the cell moves along the major axis more actively than the
minor axis, which supports our hypothesis.

Table 5.4: Correlation of displacement along the major axis.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Model</th>
<th>15 minutes</th>
<th>1 hour</th>
<th>2 hours</th>
<th>4 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Linear Regression</td>
<td>0.12</td>
<td>0.15</td>
<td>0.16</td>
<td>0.14</td>
</tr>
<tr>
<td>Baseline</td>
<td>Multilayer Perceptron</td>
<td>0.18</td>
<td>0.23</td>
<td>0.25</td>
<td>0.22</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Linear Regression</td>
<td>-0.02</td>
<td>0.03</td>
<td>-0.00</td>
<td>-0.00</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Multilayer Perceptron</td>
<td>-0.01</td>
<td>0.00</td>
<td>-0.01</td>
<td>-0.02</td>
</tr>
<tr>
<td>Morphology</td>
<td>Linear Regression</td>
<td>0.20</td>
<td>0.22</td>
<td>0.22</td>
<td>0.18</td>
</tr>
<tr>
<td>Morphology</td>
<td>Multilayer Perceptron</td>
<td><strong>0.33</strong></td>
<td><strong>0.38</strong></td>
<td><strong>0.37</strong></td>
<td><strong>0.31</strong></td>
</tr>
</tbody>
</table>

Table 5.5: Correlation of displacement along the minor axis.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Model</th>
<th>15 minutes</th>
<th>1 hour</th>
<th>2 hours</th>
<th>4 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Linear Regression</td>
<td>0.07</td>
<td>0.10</td>
<td>0.11</td>
<td>0.08</td>
</tr>
<tr>
<td>Baseline</td>
<td>Multilayer Perceptron</td>
<td>0.08</td>
<td>0.13</td>
<td>0.14</td>
<td>0.12</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Linear Regression</td>
<td>0.00</td>
<td>0.03</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Multilayer Perceptron</td>
<td>-0.03</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Morphology</td>
<td>Linear Regression</td>
<td>0.07</td>
<td>0.09</td>
<td>0.10</td>
<td>0.07</td>
</tr>
<tr>
<td>Morphology</td>
<td>Multilayer Perceptron</td>
<td><strong>0.15</strong></td>
<td><strong>0.17</strong></td>
<td><strong>0.16</strong></td>
<td><strong>0.16</strong></td>
</tr>
</tbody>
</table>

Table 5.6: Correlation of absolute displacement.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Model</th>
<th>15 minutes</th>
<th>1 hour</th>
<th>2 hours</th>
<th>4 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Linear Regression</td>
<td>0.00</td>
<td>-0.01</td>
<td>-0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>Baseline</td>
<td>Multilayer Perceptron</td>
<td><strong>0.27</strong></td>
<td>0.28</td>
<td>0.30</td>
<td>0.27</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Linear Regression</td>
<td>0.04</td>
<td>0.03</td>
<td>0.03</td>
<td>-0.03</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Multilayer Perceptron</td>
<td>0.08</td>
<td>0.00</td>
<td>0.02</td>
<td>-0.01</td>
</tr>
<tr>
<td>Morphology</td>
<td>Linear Regression</td>
<td>-0.03</td>
<td>-0.05</td>
<td>-0.05</td>
<td>-0.03</td>
</tr>
<tr>
<td>Morphology</td>
<td>Multilayer Perceptron</td>
<td>0.25</td>
<td><strong>0.29</strong></td>
<td><strong>0.32</strong></td>
<td><strong>0.35</strong></td>
</tr>
</tbody>
</table>

Figure 5-8 shows the relationship between the displacement error of the linear regression model and the ground truth displacement. The result comes from the baseline, morphological, and
shuffled features for the time intervals of 0.25 hours, 1 hour, and 2 hours. For all time intervals, the linear regression model using morphological features slightly outperforms the baseline features and the shuffled features, which is the worst.

Similarly, Figure 5-9 shows the relationship between the displacement error of the MLP model and the ground truth displacement given different features and time intervals. For all time intervals, the linear regression model using morphological features outperforms the baseline features and the shuffled features, which is the worst. Comparing Figure 5-8a, 5-8b, 5-8c with Figure 5-9a, 5-9b, 5-9c respectively, shuffle features lead to the same worst performance. In addition, MLP outperforms the linear regression model when using morphological features, which is expected.

![Figure 5-8](image)

**Figure 5-8:** The relationship between displacement error of the linear regression and the ground truth displacement. We define the displacement error as the distance between the predicted position and the ground truth in pixel. Blue, orange, and green dots indicate the result of the baseline, morphological, and shuffled features, respectively. The color-coded lines fit dots by the linear regression. The smaller the angle between the fitted line and the x axis, the more accurate the prediction. The fitted line of the ideal prediction should be on the x axis.
Figure 5-9: The relationship between displacement error of MLP based regression and the ground truth displacement. We define the displacement error as the distance between the predicted position and the ground truth in pixel. Blue, orange, and green dots indicate the result of the baseline, morphological, and shuffled features, respectively. The color-coded lines fit dots by the linear regression. The smaller the angle between the fitted line and the x axis, the more accurate the prediction. The fitted line of the ideal prediction should be on the x axis.

Figure 5-10 shows an example of the displacement prediction by MLP using the three different feature sets. We show the future ten positions of the ground truth and the prediction at four different time intervals, 15 minutes, 1 hour, 2 hours, and 4 hours respectively. Figure 5-10a to 5-10d show the regression results using the baseline feature set while Figure 5-10e to 5-10h are related to the complete feature set and Figure 5-10i to 5-10l correspond to the shuffle feature set.

Figures 5-10e to 5-10h show that the displacement prediction errors of MLP using the complete feature set are small for all the time intervals with respect to the cell size. In addition, the predicted positions in red are located close to the ground truth direction. Although the displacement accuracy of the prediction decreases as the time interval increases, the predicted direction is relatively accurate.

In contrast, Figures 5-10a to 5-10d show that the MLP model using baseline features tends to predict the next position to be very close to the current location, which causes the error to be in proportion to the ground truth displacement. In addition, the predicted positions are located around the elongated axis of the cell, although the direction may be the opposite to the movement. It suggests that the vector defined by the centroids of the cell and nucleus may contain the information of the elongation.

Furthermore, Figures 5-10a to 5-10d show that the MLP model using shuffled features simply predicts that the future position is almost at the current position. Consequently, the displacement
error is approximately equal to the actual displacement. Overall, Figure 5-10 shows that the morphological feature helps capture the relationship between cell shapes and velocity.

![Figure 5-10: Velocity regression results of MLP using different feature sets. The ground truth tracks are indicated by the cyan lines with the green dot, annotated by “SRC”, as its origin and cyan dots as the position at subsequent frames. The red dot indicates the predicted position at the next frame based on the ground truth position. Error circles in yellow indicate the ground truth positions corresponding to the predicted positions, where the radius denotes the predicted error. In other words, the smaller the error circle, the more accurate the prediction. The next ground truth position and predicted position are annotated by “True” in cyan and “Pred” in red. (a)-(d) Regression result of the MLP model using the baseline feature set. (e)-(h) Regression results of the MLP model using the whole feature set. (i)-(l) Regression results of the MLP model using shuffled features.](image-url)
5.3 Conclusion

The experiment results show that morphological features can improve the velocity estimation accuracy. However, there are limitations in extracting lamellipodium patterns and environment features. Also, the need for fluorescent cytoplasm images comprises the generality of the proposed approach.

Firstly, the cells are under-segmented since we use cytoplasm labels to train the segmentation model. Figure 5-11 shows three examples of under-segmentation. The under-segmentation loses the information about the lamellipodium, which indicates the moving direction of migrating cells. In addition, unlike the bright-field images, the fluorescent cytoplasm images, which demands advanced imaging facility, may be inaccessible to some research labs. Thus, we need to extract features using the whole-cell region of bright-field images.

![Figure 5-11: Examples of under-segmentation.](image)

Secondly, the ellipse fitting misses the skewness information of the segmentation. Figure 5-12 shows an example that one ellipse can fit two cell masks in opposite directions. We need richer features representing the direction of the lamellipodium as well as the skewness.
Thirdly, the current features include limited environment information. We simply use the number of cells with a range of the current cell and the total number of cells in the image as the environment feature. However, the cell on the front has much more impact than the cell on the side. Thus, the new feature should contain positional information of all neighbors of the target cell.

To eliminate the above limitation of manually designed features and the need for fluorescent cytoplasm images, we propose a deep-learning-based velocity estimator, which can automatically learn features and estimate velocity using bright-field images. We introduce the proposed method and associated experiment results in Chapter 6.
Longitudinally monitoring the behavior of single cells by live-cell imaging is a fundamental tool in cell and molecular biology. Such experiments necessitate cell tracking, which may involve intensive manual analysis. From one frame to the next, cells may enter, exit the field of view, divide, die, or move. Automatic cell tracking has significantly enhanced tracking efficiency, first by leveraging computer vision techniques [38, 104, 151], and recently by means of machine learning and deep learning approaches [210, 83].

Still, many factors can affect the robustness of a cell tracking algorithm; one of these is the frame rate at which images are acquired. Existing techniques are designed to work at a relatively high acquisition rate (< one image every 5-15 minutes for typical adherent mammalian cells). However, lower frame rates are beneficial for biologists experimentally.

The lower the frame rate, the more images that can be obtained between time points, increasing the population of cells can be analyzed in one experiment [171]. Moreover, too frequent imaging is the cause of a number of experimental artifacts and undesirable effects such as photobleaching, which makes cells become dimmer and phototoxicity, which affects cell health [65, 51, 116].

The main tradeoff is that current cell tracking algorithms have much higher uncertainty using low frame rate images. State-of-the-art cell tracking approaches rely on either spatial proximity [104, 151, 205, 206, 7] or morphological similarity [170, 83] to link cells in consecutive frames. At low
frame rate, both approaches fall short since a cell’s position and appearance can change significantly.

To tackle the problem of tracking cells using low frame rate data we introduce a new deep learning framework with multiple novel components. Although several events contribute to overall cell tracks, we specifically focus on links due to cell motion, which make up a large fraction of any cell track.

The innovative components of our framework are: (i) parsing of motion prediction into direction and speed components, (ii) only predicting motion for fast cells, (iii) a new deep learning model that predicts cell movement direction based on an input image patch, (iv) assumption-free, data-driven prediction of motion speed probabilities, and (v) a Bayesian framework to integrate disparate pieces of information about cell motion in a statistically rigorous manner.

Our approach has been formally tested on twelve time-lapse videos of human epithelial cells (MCF10A cells) in exponential growth for three days at a 15-minute frame rate at varying confluencies. By dropping images in each sequence, we simulate a lower frame rate. The computational experiments show that our approach outperforms state-of-the-art cell tracking algorithms [104, 151, 205, 7] at acquisition intervals ranging from 30 minutes to 3 hours for this cell system.

This chapter is organized as follows. Section 6.1 proposes a deep learning model for cell moving direction estimation and a Bayesian framework for cell linking. Section 6.2 describes experiment results. Finally, Section 6.3 draws the conclusion.

### 6.1 Model

The proposed model tracks cells in low frame rate videos by predicting the future cell positions. A key observation for predicting cell movements is that the shape of a cell can describe its moving direction but not its speed [167]. An example is shown in Figure 6-1 where a cell is observed moving at varying speed while its shape remains roughly the same. This underlines the importance of modeling (and predicting) speed and moving direction independently.

Our model works iteratively on two consecutive images moving forward through time, and is therefore a local method. It links cells between these two images in three main steps (see Figure 6-2).

The first step separates fast and slow cells. Intuitively, slow cells are the ones that do not change their position even after an extended period of time, and, for this reason, do not require advanced techniques to be tracked (see Section 6.1). Moreover, predicting motion of non-moving
cells is error-prone and deteriorates performance.

The second step predicts the moving direction of fast cells. This uses a ConvNets-based approach working on the image patch (see Section 6.1).

The third step integrates direction information with speed information using a Bayesian approach to enable final linking of fast cells. The speed information do not make assumptions on the cell motion (e.g., Brownian motion [151], linear motion [205]), but rather provides a data-driven estimation. This last step is described in Section 6.1.

**Fast and Slow Cell Classification** To separate slow cells from fast cells, and apply different strategies to prediction, we use a linear assignment approach based on cell distances. Every cell is characterized by the centroid position of its nucleus. For each interval in the video sequence, we compute a cost matrix $\mathcal{W}$ such that

$$W_{ij} = d(x_i, y_j)$$  \hspace{1cm} (6.1)

where $d(x_i, y_j)$ indicates the Euclidean distance between cells $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Then, we use the Hungarian method [123] to compute the matrix $\mathcal{L}$ based on $\mathcal{W}$.

With the hypothesis that slow cells have short frame-to-frame distances, we use a distance threshold $d_{thr}$ on $\mathcal{LW}$ to classify slow cells. Namely, for every link $\mathcal{L}_{ij} = 1$ with associated cost

---

Figure 6-1: Example of a cell hectically changing speed over time. Changes in speed do not necessarily correspond to changes in shape.
Figure 6-2: The proposed model is composed of three components. The first component classifies cells dividing them into fast and slow cells. The second component estimates cell moving direction using a ConvNet model. Cell direction information and cell speed information are combined by the third component, a Bayesian framework, to produce the final cell linking. The method works sequentially on all time steps $t$ and $t + 1$ until cell tracks are reconstructed in full.

$\forall ij < d_{th},$, the cell $x_i$ is classified as slow cell and linked to $y_j$ (i.e., $L_{ij}$ is accepted as the final link). Fast cells are processed further as described next.

**Cell Direction Estimation** While slow cells are still easily linked by spatial proximity, fast cells require a more advanced approach. The next step for processing fast cells is cell direction estimation.

The idea is to learn a cell’s moving direction from an image patch $I_i$ centered at the cell’s nucleus centroid. We model the estimation of a cell direction as a classification problem where possible moving directions are binned using $K$ classes (see Figure 6-3). This idea has been recently investigated by Nishimoto et al. [167] using ConvNets to estimate the quadrant (i.e., $K = 4$) where a cell will move, based on a manually cropped patch.

Our model introduces several novelties. First, we use $K$ as an input parameter that can be tuned during the training phase. This is because four classes do not provide enough precision to allow for accurate cell tracking (see Section 6.2).

Second, for each cell $x_i$, we automatically select the patch $I_i$ by cropping a $300 \times 300$ pixel image centered at the centroid of $x_i$’s nucleus. This allows for automatic patch creation with no manual interactions. Then, we define our model as,

$$z = g(I_i)$$ (6.2)
where, $I_i$ denotes the image patch, $g$ denotes the EfficientNet-B3 model used in our experiments, and $z$ is a vector of size $K$ representing the output of last layer of the network. Decimal probabilities for each possible moving direction (e.g., classes in $z$) are generated by adding a softmax layer as last layer of the network.

The last novelty introduced by our approach is about the loss function. A categorical loss would not capture the angular error of the prediction. On the other hand, an angular loss encourages the ConvNet to minimize the angular distance between the class prediction (e.g., $k$) and the ground truth (e.g., $k_y$), which ignores the tendency of many cell types to move along a major axis bidirectionally. Thus, in addition to the angular loss term, we propose a bimodal loss term to constrain cell movement along their direction of major elongation. The bimodal loss accounts for the fact that, in the future, ConvNets may process the image of a cell displaying the same shape but moving in the opposite direction. The new loss function is defined as follows,

$$L(z) = -\sum_{k=1}^{K} (e^{-l_a(|k_y-k|)} + e^{-l_b(|k_y-k|)}) \log(z_k)$$  \hspace{1cm} (6.3)

where $z$ is a cell direction prediction formulated on $K$ direction classes, $e^{-l_a(|k_y-k|)}$ is the angular loss term, and $e^{-l_b(|k_y-k|)}$ is the bimodal loss term. The distances $l_a$ and $l_b$ are defined as follows,

$$l_a(h) = \begin{cases} h & \text{for } h < K/2 \\ K - h & \text{for } h \geq K/2 \end{cases}$$  \hspace{1cm} (6.4)
Figure 6-4: The loss function is a combination of two terms \( l_a \) and \( l_b \). For a ground-truth moving direction \( k_y \), \( l_a(|k_y - k|) \) rewards classes based on their angular distance from \( k_y \). The bimodal term \( l_b(|k_y - k|) \) rewards them in a symmetric manner. The illustration is for providing intuition and the exact definition is given in Equation 6.3.

\[
l_b(h) = \begin{cases} 
  h & \text{for } h \leq K/4 \\
  K/2 - h & \text{for } K/4 < h \leq K/2 \\
  h - K/2 & \text{for } K/2 < h \leq 3K/4 \\
  K - h & \text{for } 3K/4 < h < K 
\end{cases} \quad (6.5)
\]

Intuitively, the angular loss term favors classes of \( K \) close to the true direction, the bimodal loss term favors classes symmetric to the true direction (see Figure 6-4).

**Bayesian Integration of Cell Direction and Speed Information** The last step of our framework combines speed information of cells in \( \mathcal{X} \) and \( \mathcal{Y} \), and all moving directions estimated by the model \( g \), to predict the final links. To combine this information we use a Bayesian approach \[162\] to refine the prior probability of \( x_i \) and \( y_j \) to be linked (\( P(L_{ij}) \)), based on new evidence (\( E \)) from our data.

Observations used by existing Bayesian approaches for high frame rate cell tracking include cell pixel density distribution \[190, 49\], or historical cell positions \[174, 209\]. The commonality of these approaches is that they assume gradual changes in cell features. Our approach do not put assumptions on a cell speed or moving direction to address sudden changes due to the low frame rate setting.

Specifically, the objective of our framework is to estimate

\[
P(L_{ij}|E) = \frac{P(L_{ij})P(E|L_{ij})}{P(E)} \quad (6.6)
\]
for any possible pair of cells in $X$ and $Y$.

The prior probability to form a link between $x_i$ and $y_j$ is $\frac{1}{||Y||}$, where $|| \cdot ||$ indicates the cardinality of a set (i.e., the number of candidate cells).

$P(E|L_{ij})$ is obtained by combining independent posterior information regarding moving direction ($E_k$) and the speed ($E_d$) of $x_i$. Since cell moving direction and cell speed are independent,

$$P(E|L_{ij}) = P(E_k|L_{ij}) P(E_d|L_{ij})$$  \hspace{1cm}  (6.7)

The probability, $P(E_k|L_{ij})$ describes the conditional probability that a cell $x_i$ moves in the direction of $y_j$. This is estimated by the model described in Section 6.1 as

$$P(E_k|L_{ij}) = \sigma(g(I_i))_k$$  \hspace{1cm}  (6.8)

where $k$ is the direction class of $x_i$ containing $y_j$ and $\sigma$ is the softmax normalization.

The conditional probability, $P(E_d|L_{ij})$, represents the probability that cell $x_i$ moves distance $d_{ij}$. This is estimated from the training data as the number of true links between cells at distance $d_{ij}$ divided by the total number of true links.

Finally, we compute the probability of $E$ as

$$P(E) = P(E|L_{ij})P(L_{ij}) + P(E|\overline{L_{ij}})(1 - P(L_{ij}))$$  \hspace{1cm}  (6.9)

where $\overline{L_{ij}}$ represents the event that two cells are not linked. Similarly to Equation 6.7, this probability has two components that incorporate cell moving direction estimation $P(E_k|\overline{L_{ij}})$, and cell speed estimation $P(E_d|\overline{L_{ij}})$.

$P(E_k|\overline{L_{ij}})$ is computed by conditional probability based on the probability that $x_i$ will move in the direction of all possible cells in $Y$ but $y_j$, namely,

$$P(E_k|\overline{L_{ij}}) = \frac{P(E_k, \overline{L_{ij}})}{P(\overline{L_{ij}})}$$  \hspace{1cm}  (6.10)

The total probability formula gives

$$P(E_k|\overline{L_{ij}}) = \frac{\sum_{n=1}^{||Y||} P(E_k|L_{in})P(L_{in})}{1 - P(L_{ij})}$$  \hspace{1cm}  (6.11)
Each posterior probability in Equation 6.11 is estimated by the model described in Section 6.1.

\[ P(E_s|\overline{L}_{ij}) \] is computed from the training data dividing the number of non-linked cell pairs with distance \( d_{ij} \) by the total number of non-linked cell pairs.

Using the posterior probabilities we populate a cost matrix \( W \) for the fast cells as

\[ W_{ij} = -\log(P(L_{ij}|E)) \]

and we obtain the final links for fast cells (e.g., \( L \)) by linear assignment based on the Hungarian method [123] to optimize the objective function

\[ \sum_{x_i \in X} \sum_{y_j \in Y} L_{ij} W_{ij} \]

where \( W \) is a cost matrix. Each element \( W_{ij} \) represents the cost of linking cell \( x_i \) to \( y_j \).

### 6.2 Experiment

In this section, we compare the proposed approach with existing state-of-the-art and tool-box cell tracking approaches using 12 time-lapse cell image datasets, described in details in Section 2.1. The objective is to evaluate the tracking performance of the proposed approach at low frame rate. In addition, we are interested in the tracking robustness given increasingly longer frame intervals. We measure the tracking performance by how much track fraction and how many complete tracks can be reconstructed, which have both algorithmic and biological significance and are described in more detail below.

#### 6.2.1 Metrics

We measured the tracking performance using the track fraction and the complete track score used in three editions of the Cell Tracking Challenge [210]. The track fraction score focuses on the lengths of the reconstructed tracks. For a ground-truth cell track with the number of links (length) equal to \( \rho \), the track fraction computes the maximum number of correct, consecutive, links in the reconstructed track \( \overline{\rho}_{max} \). Then, the **track fraction score** is the average of \( \overline{\rho}_{max} / \rho \) computed on all tracks of a field of view. Complete track score [210] measures the number of completely correct tracks,
in proportion to the total number of tracks reconstructed. Since in our evaluation we hard-code each track start and end, the complete track score is the number of correct tracks normalized by the number of tracks in the ground truth.

6.2.2 Parameters setting

Slow/fast cell classification. To set the distance threshold for the slow cell classifier we used the reasoning that slow cells will have highly overlapping nuclei and thus have high probability to be the same. Based on our dataset the average nuclei radius (defined by the half of its major axis length) is 30 pixels, so we set $d_{thr} = 30$. This results in an average linking precision of 99.9%, where the precision is the ratio of the number of true links to all predicted links. Slow cell links are immediately set, and remaining cells are treated as fast with links to be determined.

Cell direction prediction. To train the ConvNet model in predicting cell movement directions we need to define the number of classes $K$ for the classification problem. To determine a suitable value for $K$, we introduced a new measure called distance rank. Figure 6-5(a) illustrates how the distance rank is computed for a true link between cells $x_i$ and $y_j$, and a given number of slices (e.g., $K = 4$). We compute the predicted movement for $x_i$ as $\tilde{x} = x_i + u \cdot v$, where $u$ indicates the bisector of the slice containing $y_j$ (i.e., estimated moving direction), and $v$
Figure 6-6: Bar chart showing the total number of cells having distance rank 2 when using number of classes \(K = 4, 8, 12, 24\).

represents the ground truth speed of \(x_i\). Then, the distance rank counts the number of cells in \(Y\) which are closer to \(\hat{x}\) than \(y_j\). This is done by sorting the cells in \(Y\), and checking the position of \(y_j\) in such order. For example, \(x_1\) in Figure 6-5(b) has distance rank 1 because its ground truth next position \(y_1\) is the closest cell to its predicted position \(\hat{x}\). In Figure 6-5(c) \(x_1\) has distance rank 2 since \(y_1\) occupies the second position in the order. Intuitively, the distance rank answers the following question: “Assuming we estimate the correct direction and speed for \(x_i\), how close does the prediction \(\hat{x}\) get to the ground truth position \(y_i\)?” This is relevant information because, as illustrated in Figure 6-5(c), a small value of \(K\) could prevent correct links, even with perfect direction and speed estimations. Tuning \(K\) can solve this issue (see Figure 6-5(d)).

We used the distance rank during training to identify a lower bound for the parameter \(K\). Figure 6-6 shows the cells with distance rank 2 identified on the training set when using a variable number of classes. The number of cells with distance rank 2 drops considerably for \(K > 8\) and becomes 0 for values above \(K = 24\). Then, we narrowed the parameter selection to \(K = \{12, 24, 36\}\) and we evaluated the model’s accuracy with the complete track score metric. After reaching its maximum for \(K = 12\) (mean score 0.959), it starts dropping to 0.957 for \(K = 24\), and 0.954 for \(K = 36\) respectively. This result was consistent in all test runs, so we use \(K = 12\) in all our experiments. It is instructive to recall that previous approaches using \(K = 4\) would similarly have much higher cell track error.

**Model training.** The EfficientNet B3 [201] model is implemented using TensorFlow [3]. Experiments are performed using 11 image sequences for training and one sequence for testing. One-tenth of the training set is used for validation to select the model checkpoint with the minimum loss for testing. We use Adam optimizer [118] with initial learning rate 0.0001 and the default
Table 6.1: Method comparison using 12 fields of view (FOV) at 1-hour frame rate. Columns of track fraction and complete track show the corresponding scores in percentage. Row FOV indicates the index of the field of view.

<table>
<thead>
<tr>
<th>Method</th>
<th>Track fraction</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOV</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>PROPOSED</td>
<td>99.6</td>
<td>97.3</td>
<td>99.4</td>
<td>100</td>
<td>98.3</td>
<td>100</td>
<td>98.2</td>
<td>100</td>
<td>98.5</td>
<td>93.1</td>
<td>100</td>
<td>97.9</td>
</tr>
<tr>
<td>LAP</td>
<td>98.4</td>
<td>96.1</td>
<td>99.2</td>
<td>100</td>
<td>98</td>
<td>100</td>
<td>98.6</td>
<td>100</td>
<td>96.9</td>
<td>90.7</td>
<td>100</td>
<td>95.4</td>
</tr>
<tr>
<td>TRACKPY_B</td>
<td>98.4</td>
<td>96.1</td>
<td>97.8</td>
<td>100</td>
<td>97.1</td>
<td>100</td>
<td>96.7</td>
<td>100</td>
<td>96.8</td>
<td>88.1</td>
<td>100</td>
<td>94.8</td>
</tr>
<tr>
<td>TRACKPY_V</td>
<td>96.8</td>
<td>93.5</td>
<td>96.1</td>
<td>100</td>
<td>94.6</td>
<td>98.5</td>
<td>94.2</td>
<td>98.9</td>
<td>95.7</td>
<td>82.2</td>
<td>92.6</td>
<td>94.3</td>
</tr>
<tr>
<td>TRACKMATE</td>
<td>99.7</td>
<td>96</td>
<td>98</td>
<td>100</td>
<td>97.1</td>
<td>100</td>
<td>96.8</td>
<td>94.6</td>
<td>97.9</td>
<td>92.5</td>
<td>99</td>
<td>94.2</td>
</tr>
<tr>
<td>VITERBI</td>
<td>75.9</td>
<td>60.3</td>
<td>59.4</td>
<td>63.2</td>
<td>65</td>
<td>70.9</td>
<td>58.9</td>
<td>69.9</td>
<td>66</td>
<td>43.9</td>
<td>70.9</td>
<td>55.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Complete track</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FOV</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>PROPOSED</td>
<td>98</td>
<td>96.3</td>
<td>98.3</td>
<td>100</td>
<td>92.8</td>
<td>100</td>
<td>94</td>
<td>100</td>
<td>93.9</td>
<td>82.6</td>
<td>100</td>
<td>95</td>
</tr>
<tr>
<td>LAP</td>
<td>95.9</td>
<td>93.9</td>
<td>96.1</td>
<td>100</td>
<td>91.3</td>
<td>100</td>
<td>94.6</td>
<td>100</td>
<td>92.1</td>
<td>77.4</td>
<td>100</td>
<td>91.2</td>
</tr>
<tr>
<td>TRACKPY_B</td>
<td>95.9</td>
<td>93.9</td>
<td>92.3</td>
<td>100</td>
<td>84.2</td>
<td>100</td>
<td>91.4</td>
<td>100</td>
<td>91.2</td>
<td>65.8</td>
<td>100</td>
<td>84.8</td>
</tr>
<tr>
<td>TRACKPY_V</td>
<td>93.4</td>
<td>86.1</td>
<td>86.7</td>
<td>100</td>
<td>78</td>
<td>95.8</td>
<td>83.8</td>
<td>94.3</td>
<td>90.4</td>
<td>46.1</td>
<td>89.1</td>
<td>77.1</td>
</tr>
<tr>
<td>TRACKMATE</td>
<td>98</td>
<td>92</td>
<td>93</td>
<td>100</td>
<td>88.9</td>
<td>100</td>
<td>88.1</td>
<td>98.6</td>
<td>94.7</td>
<td>80.8</td>
<td>96.6</td>
<td>87.9</td>
</tr>
<tr>
<td>VITERBI</td>
<td>43.9</td>
<td>31.9</td>
<td>26.1</td>
<td>31.2</td>
<td>28.4</td>
<td>41.3</td>
<td>27.2</td>
<td>35.9</td>
<td>36.5</td>
<td>14.1</td>
<td>44.1</td>
<td>20</td>
</tr>
</tbody>
</table>

parameters to schedule the learning rate. We train the estimator for 100 epochs but stop it early when validation loss does not decrease for 5 epochs.

6.2.3 Comparisons

We compare our framework to five existing approaches for cell tracking. The first method (LAP) [104] is a simple yet effective approach widely used in the biochemical community [24, 205]. This model uses the Hungarian method [123] to compute the matrix $L$ using the cost matrix from Equation 6.1 and the objective function from Equation 6.13. The second and third approaches are both implemented in Trackpy [7]. The second model (TRACKPY_B) [38] uses a Brownian motion model to calculate the probability of each link. The third model (TRACKPY_V), provided by the Trackpy Toolbox [7], estimates the cost of linking cells $x_i$ and $y_i$ by assuming constant velocity for $x_i$. Specifically, given the cell position $x_i$ and its velocity $v$ at the previous time step, the model updates the cell position as $\tilde{x} = x_i + v$ and defines the cost as $W_{ij} = d(\tilde{x}, y_j)$. The fourth model (TRACKMATE) is provided by TrackMate [205]. This approach is similar to (TRACKPY_V) but this time the velocity of a cell is obtained dynamically by means of a Kalman filter [109] and a linear motion model. The fifth model (VITERBI) [151] is a global linking approach. Differently from the other models, this method computes tracks globally (i.e., considering all the time frames at once). Final cell tracks are selected by means of the Viterbi algorithm [62]. This model ranked first in the recently published cell tracking challenge [210].
For all methods we isolate the linking problem by hard-coding all cell events that do not include cell motion (entry, division, exit and death). Cells dying and cells exiting the frame are considered the end of a track. Mitosis events and cells entering the frame are considered the beginning of a new track. This way we can focus our evaluation specifically on cell linking accuracy due to cell motion.

**Results.** Table 6.1 shows the tracking performance at one-hour frame rate for all methods. The proposed method has the best average performance for both track fraction and complete track score. Noticeably, LAP is the second-best scoring approach on average after the proposed model, indicating that, despite its simplicity, it remains somewhat effective even when frame rate is low. TRACKPY.B slightly underperforms LAP. TRACKPY.B uses an adaptive search range to select cell candidates that helps reduce the computational complexity at the cost of accuracy. TRACKMATE scores best for two fields of view, which indicates that the predictions of a Kalman filter can provide accurate results for specific moving patterns but not for general low frame rate tracking. As opposed, the constant cell velocity information used by TRACKPY.V seems to directly impact the tracking performance on all fields of view. Finally, VITERBI suffers most from the low frame rate compared to other methods. This may be caused by its global objective function, which prioritizes the construction of longer tracks at the cost of local linking errors.

![Figure 6-7: Performance change of LAP, TRACKMATE, and the proposed method at frame rates (0.5h to 3h). The mean track fraction score and complete track score at each frame rate are calculated for the three methods using 12 sequences. As the time interval increases, the proposed method outperforms LAP and TRACKMATE by relatively larger margins in terms of mean performance metrics.](image)

**Varying the frame rate.** Since TRACKMATE and LAP have scored best in two specific
Table 6.2: Track fraction score at varying frame rates

<table>
<thead>
<tr>
<th>Method</th>
<th>FOV</th>
<th>0.5h</th>
<th>1h</th>
<th>1.5h</th>
<th>2h</th>
<th>2.5h</th>
<th>3h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td>1</td>
<td>100.0</td>
<td>99.6</td>
<td>99.6</td>
<td>99.5</td>
<td>98.2</td>
<td>98.4</td>
</tr>
<tr>
<td>Trackmate</td>
<td>100.0</td>
<td>99.7</td>
<td>96.8</td>
<td>95.3</td>
<td>93.8</td>
<td>92.1</td>
<td></td>
</tr>
<tr>
<td>Proposed</td>
<td>7</td>
<td>99.7</td>
<td>98.2</td>
<td>96.8</td>
<td>94.7</td>
<td>91.1</td>
<td>91.1</td>
</tr>
<tr>
<td>LAP</td>
<td>100.0</td>
<td>98.6</td>
<td>96.2</td>
<td>94.4</td>
<td>89.0</td>
<td>89.5</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: Complete track score at varying frame rates

<table>
<thead>
<tr>
<th>Method</th>
<th>FOV</th>
<th>0.5h</th>
<th>1h</th>
<th>1.5h</th>
<th>2h</th>
<th>2.5h</th>
<th>3h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td>7</td>
<td>98.8</td>
<td>94.0</td>
<td>88.6</td>
<td>87.8</td>
<td>75.9</td>
<td>79.0</td>
</tr>
<tr>
<td>LAP</td>
<td>100.0</td>
<td>94.6</td>
<td>89.8</td>
<td>86.0</td>
<td>76.6</td>
<td>79.0</td>
<td></td>
</tr>
<tr>
<td>Proposed</td>
<td>9</td>
<td>93.9</td>
<td>93.9</td>
<td>89.2</td>
<td>77.1</td>
<td>78.7</td>
<td>80.4</td>
</tr>
<tr>
<td>Trackmate</td>
<td>93.9</td>
<td>94.7</td>
<td>84.7</td>
<td>82.6</td>
<td>80.6</td>
<td>71.0</td>
<td></td>
</tr>
</tbody>
</table>

fields of view, we analyzed their performance at the varying frame rate. Specifically, we tested the three approaches at decreasing frame rates ranging from 30 minutes to 3 hours. Figure 6-7 shows the average scores achieved by the three approaches. The proposed method outperforms LAP and TRACKMATE by larger margins as the frame rate decreases indicating the improved scalability of the proposed approach.

Table 6.2 compares the proposed approach with TRACKMATE and LAP using the field of view they respectively performed best at 1 hour frame rate for track fraction. As the time interval increases, the proposed approach catches up to the compared methods and finally outperforms them. Table 6.3 shows a similar trend, but for complete track score. The tracking performance evaluation using the specific field of view also demonstrates the improved scalability of the proposed approach, which is consistent with the aggregated results.

6.2.4 Model Generality and Limitation

To understand the generality and limitations of the proposed method, we performed additional experiments using four public datasets with different cell lines, frame rates, image modality, and number of frames.

The specifications of four datasets are described in Table 6.4. The Myoblast dataset was a subset of the data released by Ker et al. [113] for studying dynamic behavior of cells. The Myoblast dataset was fully annotated by [83] for investigating deep learning based cell tracking algorithms. Other three datasets, named PSC, U373, and MUSC are used in Cell Tracking Challenge (CTC) [210]. So far, CTC [210] provides ten 2D time-lapse datasets. We limited our analysis to three datasets.
which contain bright-field and phase contrast images.

Table 6.4: Specifications of four additional datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Cell line</th>
<th>Image modality</th>
<th>Rate</th>
<th>Frame #</th>
<th>Duration</th>
<th>Mag.</th>
<th>FOV1 Track #</th>
<th>FOV2 Track #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Myoblast</td>
<td>Myoblast</td>
<td>phase contrast</td>
<td>5 min</td>
<td>100</td>
<td>8 h</td>
<td>5x</td>
<td>271</td>
<td>305</td>
</tr>
<tr>
<td>PSC</td>
<td>Pancreatic stem</td>
<td>phase contrast</td>
<td>10 min</td>
<td>300</td>
<td>50 h</td>
<td>4x</td>
<td>1370</td>
<td>1025</td>
</tr>
<tr>
<td>U373</td>
<td>Human brain tumor</td>
<td>phase contrast</td>
<td>15 min</td>
<td>115</td>
<td>28.5 h</td>
<td>20x</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>MUSC</td>
<td>Mouse muscle stem</td>
<td>bright field</td>
<td>5 min</td>
<td>1370</td>
<td>146 h</td>
<td>10x</td>
<td>71</td>
<td>48</td>
</tr>
</tbody>
</table>

We want to know the model generality with the velocity estimator pre-tuned on MCF10A data on these four datasets.

To this end, we kept all the parameter setting used for testing MCF10A dataset except updating the image patch according to the lens magnification and cell lines of the four datasets if images are not scaled. The greater the lens magnification, the larger the cell shown in the image. Different cell lines may also have different cell sizes. The image patch is selected to cover the target cell region in the image. Image patches are too small or large can miss cell morphological features or involve irrelevant information. Cells in these four datasets show smaller than the ones in MCF10A datasets, so we reduce the image patch accordingly. The image patch sizes we used are 120 × 120 for U373, 32 × 32 for PSC, 160 × 160 for MUSC, and 100 × 100 for Myoblast.

Another cell-line-dependent parameter in our model is the nucleus size. We measured the nucleus size for the four datasets given the fact that no corresponding fluorescent images of nuclei are available for accurately capture the nucleus regions. Accordingly, we use 5 pixels as the distance threshold for Myoblast and MUSC datasets and 10 pixels for PSC and U373 datasets.

By dropping images in each sequence, we simulated lower frame rates. Starting from the base frame rate, i.e., 1x time interval, we generated frames at lower rates by sampling frames at 3x, 6x, and 12x time intervals. We use the frame-rate scale instead of actual time interval between sampled frames to define how slow the frame rate is since the exact frame rate is relative to datasets.

We used LAP as the baseline method since it performs the second-best on average following the proposed method in the linking experiments using MCF10A dataset. Table 6.5 shows the complete track score comparison between LAP and the proposed method using time-lapse cell images of eight fields of views from the four datasets. The proposed method performs on a par with LAP using datasets PSC and U373 while underperforming LAP using datasets Myoblast and MUSC.

Moreover, for the datasets, Myoblast and MUSC, where the proposed method underperformed LAP especially at low frame rate, we want to understand what factors limit the performance and
Table 6.5: Complete track score of LAP and the proposed method using four datasets, including eight fields of view, at four different frame-rate scales (1x, 3x, 6x, 12x).

<table>
<thead>
<tr>
<th>Method</th>
<th>Scale</th>
<th>psc_1</th>
<th>psc_2</th>
<th>u373_1</th>
<th>u373_2</th>
<th>myo_1</th>
<th>myo_2</th>
<th>musc_1</th>
<th>musc_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAP</td>
<td>1</td>
<td>0.986</td>
<td>0.998</td>
<td>1</td>
<td>1</td>
<td>0.987</td>
<td>0.993</td>
<td>0.746</td>
<td>0.646</td>
</tr>
<tr>
<td>Proposed</td>
<td>1</td>
<td>0.988</td>
<td>0.998</td>
<td>1</td>
<td>1</td>
<td>0.977</td>
<td>0.993</td>
<td>0.563</td>
<td>0.208</td>
</tr>
<tr>
<td>LAP</td>
<td>3</td>
<td>0.981</td>
<td>0.981</td>
<td>0.833</td>
<td>0.957</td>
<td>0.974</td>
<td>0.578</td>
<td>0.458</td>
<td></td>
</tr>
<tr>
<td>Proposed</td>
<td>3</td>
<td>0.981</td>
<td>0.981</td>
<td>0.833</td>
<td>0.951</td>
<td>0.974</td>
<td>0.359</td>
<td>0.104</td>
<td></td>
</tr>
<tr>
<td>LAP</td>
<td>6</td>
<td>0.945</td>
<td>0.951</td>
<td>1</td>
<td>0.906</td>
<td>0.924</td>
<td>0.483</td>
<td>0.292</td>
<td></td>
</tr>
<tr>
<td>Proposed</td>
<td>6</td>
<td>0.945</td>
<td>0.951</td>
<td>1</td>
<td>0.845</td>
<td>0.917</td>
<td>0.317</td>
<td>0.125</td>
<td></td>
</tr>
<tr>
<td>LAP</td>
<td>12</td>
<td>0.905</td>
<td>0.926</td>
<td>0.625</td>
<td>0.773</td>
<td>0.825</td>
<td>0.509</td>
<td>0.261</td>
<td></td>
</tr>
<tr>
<td>Proposed</td>
<td>12</td>
<td>0.905</td>
<td>0.926</td>
<td>0.625</td>
<td>0.698</td>
<td>0.844</td>
<td>0.509</td>
<td>0.217</td>
<td></td>
</tr>
</tbody>
</table>

thus clarify the requirement for applying the proposed tracking method to new datasets.

We started by investigating the impact of the ConvNets-based direction estimator to the tracking performance. Before tuning the number of direction classes which is 12, we replaced the direction estimated by ConvNets with the ground truth direction to find out whether the underperformance is caused by under-fitting, i.e., ConvNets is not trained well due to the limited data. Table 6.6 shows the complete track score comparison between LAP and the proposed method, named GT CNN K12, with the assumption that the ConvNets-based direction estimator predicts ground truth direction label within 12 direction classes.

The proposed method, GT CNN K12, outperformed LAP using datasets PSC and Myoblast, performed on a par with LAP using U373, and underperformed LAP using MUSC, respectively. On one hand, a better direction estimator improved the linking performance using PSC and Myoblast datasets, compared with the results in Table 6.5. On the other hand, the proposed method underperformed LAP using MUSC dataset, which could be caused by the number of classes used by the direction estimator. Intuitively, the smaller the cell region in the image, the higher the number of direction classes is required to distinguish cells located towards the close orientation.

We wanted to further find out whether the direction class number affected tracking performance of the proposed method. We calculated the optimal class number for the dataset from each field of view by using distance rank measure. Figure 6-8 shows the optimal class number and corresponding lens magnification of the four datasets at decreasing frame-rate scales. Figure 6-8 (a)-(d) correspond to frame-rate scales 1x, 3x, 6x, and 12x respectively. In our previous experiments, we used 12 direction classes. Figure 6-8 (a) shows 12 direction classes can satisfy all datasets except MUSC, which explains the worse performance on MUSC, although we used ground truth direction.
Table 6.6: Complete track score of LAP and the proposed method with ground truth 12-class direction label using four datasets, including eight fields of view, at four different frame-rate scales (1x, 3x, 6x, 12x).

<table>
<thead>
<tr>
<th>Method</th>
<th>Scale</th>
<th>psc_1</th>
<th>psc_2</th>
<th>u373_1</th>
<th>u373_2</th>
<th>myo_1</th>
<th>myo_2</th>
<th>musc_1</th>
<th>musc_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAP</td>
<td>1</td>
<td>0.986</td>
<td>0.998</td>
<td>1</td>
<td>1</td>
<td>0.987</td>
<td>0.993</td>
<td>0.746</td>
<td>0.646</td>
</tr>
<tr>
<td>GT CNN K12</td>
<td></td>
<td>0.987</td>
<td>0.998</td>
<td>1</td>
<td>1</td>
<td>0.993</td>
<td>0.993</td>
<td>0.366</td>
<td>0.25</td>
</tr>
<tr>
<td>LAP</td>
<td>3</td>
<td>0.981</td>
<td>0.981</td>
<td>1</td>
<td>0.833</td>
<td>0.957</td>
<td>0.974</td>
<td>0.578</td>
<td>0.458</td>
</tr>
<tr>
<td>GT CNN K12</td>
<td></td>
<td>0.981</td>
<td>0.981</td>
<td>1</td>
<td>0.833</td>
<td>0.974</td>
<td>0.985</td>
<td>0.281</td>
<td>0.146</td>
</tr>
<tr>
<td>LAP</td>
<td>6</td>
<td>0.945</td>
<td>0.951</td>
<td>1</td>
<td>1</td>
<td>0.906</td>
<td>0.924</td>
<td>0.483</td>
<td>0.292</td>
</tr>
<tr>
<td>GT CNN K12</td>
<td></td>
<td>0.946</td>
<td>0.951</td>
<td>1</td>
<td>1</td>
<td>0.98</td>
<td>0.985</td>
<td>0.183</td>
<td>0.146</td>
</tr>
<tr>
<td>LAP</td>
<td>12</td>
<td>0.905</td>
<td>0.926</td>
<td>1</td>
<td>1</td>
<td>0.773</td>
<td>0.825</td>
<td>0.509</td>
<td>0.261</td>
</tr>
<tr>
<td>GT CNN K12</td>
<td></td>
<td>0.912</td>
<td>0.929</td>
<td>0.625</td>
<td>1</td>
<td>0.932</td>
<td>0.962</td>
<td>0.281</td>
<td>0.196</td>
</tr>
</tbody>
</table>

Starting from Figure 6-8 (b) to Figure 6-8 (d), the required direction class numbers increase for all datasets, which can be a reason why the performance of the proposed method dropped as the frame-rate scale decreases as shown in Table 6.6. Figure 6-8 (d) shows that the optimal direction class number of MUSC data becomes 72 which is much higher than what we used in previous experiments. Thus, the tracking performance is likely limited by the class number used by direction estimation. In addition, from all results shown in Figure 6-8, it is worth noticing that to the magnification show no obvious relationship with the optimal class number.

Table 6.7: Complete track score of LAP and the proposed method assuming the ground truth direction probability is known to the direction estimator using Myoblast and MUSC datasets, including four fields of view, at four different frame-rate scales (1x, 3x, 6x, 12x).

<table>
<thead>
<tr>
<th>Method</th>
<th>Scale</th>
<th>myo_1</th>
<th>myo_2</th>
<th>musc_1</th>
<th>musc_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAP</td>
<td>1</td>
<td>0.987</td>
<td>0.993</td>
<td>0.746</td>
<td>0.646</td>
</tr>
<tr>
<td>Ideal CNN</td>
<td></td>
<td>0.993</td>
<td>0.993</td>
<td>0.901</td>
<td>0.875</td>
</tr>
<tr>
<td>LAP</td>
<td>3</td>
<td>0.957</td>
<td>0.974</td>
<td>0.578</td>
<td>0.458</td>
</tr>
<tr>
<td>Ideal CNN</td>
<td></td>
<td>0.974</td>
<td>0.985</td>
<td>0.969</td>
<td>0.854</td>
</tr>
<tr>
<td>LAP</td>
<td>6</td>
<td>0.906</td>
<td>0.924</td>
<td>0.483</td>
<td>0.292</td>
</tr>
<tr>
<td>Ideal CNN</td>
<td></td>
<td>0.98</td>
<td>0.985</td>
<td>0.967</td>
<td>0.833</td>
</tr>
<tr>
<td>LAP</td>
<td>12</td>
<td>0.773</td>
<td>0.825</td>
<td>0.509</td>
<td>0.261</td>
</tr>
<tr>
<td>Ideal CNN</td>
<td></td>
<td>0.939</td>
<td>0.962</td>
<td>0.965</td>
<td>0.957</td>
</tr>
</tbody>
</table>

Table 6.7 shows the complete track score of the proposed method with ground truth direction label and optimal direction class number, named Ideal CNN, using time-lapse cell images of Myoblast and MUSC datasets. We still used LAP as a reference. We used class number 36 and 72 for Myoblast dataset and MUSC dataset respectively. Ideal CNN outperformed LAP by a large margin, especially at low frame rates. Compared with Table 6.6, increasing direction class number effectively improves the tracking performance. The results verified the benefits of proposed Bayesian integration method.
to tracking performance at low frame rate.

In conclusion, the proposed method generalized well to different datasets if the ConvNets-based direction estimator can be effectively trained, which requires enough training samples and model fine-tuning.

6.3 Conclusion

In this chapter, we showed that tracking cells at low frame rate is improved by using a deep learning approach to estimate moving direction and a Bayesian method to further predict the future position. Experiment results using additional public datasets validated the effectiveness of the proposed Bayesian integration approach and showed the importance of fine-tuning ConvNets-based
direction estimator with enough data. The novelty of our method is separating the object movement prediction into the direction and distance/speed, which both can be probabilistic and therefore integrated with the Bayesian formula, and the separation of linkage for slow and fast cells. The proposed position prediction framework can generally apply to generic object tracking at low frame rates when the velocity magnitude and direction can be learned from the data.
Chapter 7

Conclusion

Automatic cell tracking approaches using microscopic images have been developed for many years. State-of-the-art approaches identify cells by deep learning models and link them using information from all images to establish tracks. However, these studies have not focused on tracking cells at low acquisition rate, which is beneficial to high-throughput cell analysis.

Our proposed work extends the state-of-the-art cell tracking approaches to tracking cells in time-lapse microscopic images at low acquisition rate by estimating cell velocity by deep learning model and integrating predicted future cell position using Bayesian framework.

Cell tracking starts from cell identification. To avoid the need for manual annotation for training deep-learning-based cell identification models, we investigated the feasibility of using fluorescent cell images to generate cell labels for training. We built our own cell dataset, designed an automatic pipeline for generating cell labels, and performed experiments for testing the cell identification accuracy of the model trained with automatically-generated annotation. Our experiment results show that the proposed approach can achieve competitive performance (recall 0.89 and precision 0.92) for identifying cells in a completely automatic manner. We think the main factor limiting the cell identification accuracy is the low contrast of the fluorescent labels used for training the ConvNets-based model. To acquire long-term cell images, we chose fluorescent protein instead of fluorescent dyes, which can kill cells gradually, to mark cells. However, the fluorescent protein is much dimmer than the fluorescence expressed by dyes. Thus, using the latter to generate cell annotations for training is likely to lead to better results. Once the model has been trained, it can identify cells in long-term bright-field cell images for further tracking purposes.
Based on the cell identification results, we used hand-crafted morphological features for modeling the shape of cells and the experiments demonstrate that the cell velocity can be estimated using these features. Given that morphological features extracted from image patches can describe the motion of a cell, we focused on integrating cell velocity estimations to improve the cell tracking accuracy at low frame rates. We upgraded the velocity estimator by a new deep-learning-based approach to automatically derive cell velocity information from image patches without the need for hand-crafted features. We separated the cell movement prediction into the direction and distance/speed, which both can be probabilistic and therefore integrated with the Bayesian formula. In addition, we linked slow and fast cell separately to avoid the need for estimating direction for slow cells, which is highly uncertain. The loss function was also deliberately designed for capturing the cell motion behavior.

Finally, we designed a new Bayesian framework which leverages cell position information and cell velocity predictions to estimate a probabilistic association score for cell pairs in consecutive frames. The association scores can be seamlessly used by linear assignment algorithms for cell linking at low frame rate.

Our tracking framework is data-driven thus free of assumptions of cell motion models. We compared our cell linking method to both state-of-the-art tracking approaches and tracking algorithms implemented in well-established toolboxes for cell analysis. Our approach outperforms existing methods while allowing a 4x reduction in the frame rate. The tracking experiment results using four public datasets show that our approach can generalize to new datasets if the ConvNets-based direction estimator is well-trained, which requires sufficient training data and fine-tuned parameters. In addition, the proposed position prediction framework can apply to generic object tracking at low frame rate when the velocity magnitude and direction can be learned from the data.

The proposed framework focuses on cell linking at low frame rate, however, other events characterize the life of a cell. While predicting the appearance (e.g., mitosis, cell moving in the frame) [141, 234, 132, 152], or disappearance of new cells (e.g., death, cell moving out of the frame) [30, 124, 156] are both extensively researched problems in the high frame rate, these are still open problems in the low frame rate setting. For example, novel deep learning has been developed recognizing cell mitosis [132]. In the high frame rate setting, the model can exploit the large number of images depicting all the transitions of cell division events to refine the prediction. However, when frames are dropped, the transitions will be more sparsely sampled or may be missed completely. Given that these are challenging problems to solve, our proposed framework has the potential for
incorporating additional predictions. Specifically, thanks to the Bayesian framework our model could include additional probabilities estimating for each cell, closeness to the boundary of the image (for cell entering and exiting the frame), and probabilities of mitosis and death events. This will require investigating what is a practical upper bound for low frame rate when considering these events. While our evaluation suggested that our current model scales better than other methods even when the frame rate drops considerably (one image every 3 hours). The same frame rate could be unusable when other cell events are included (e.g., cell death). Regardless, as mentioned above, advances in predicting any of these other important events could be integrated using the proposed Bayesian foundation so long as such predictions are probabilistic.

In summary, tracking cells at low frame rate is improved by using a deep learning approach to estimate moving direction and a Bayesian method to further predict the future position. On one hand, deep ConvNets with big data enable learning of cell motion behavior to avoid assumption of motion models, which is a foundation of position prediction but challenging to define for different cells and environment. On the other hand, Bayesian framework provides a statistically rigorous way to integrate more observations beneficial to cell tracking at low frame rate. The future work will focus on the remaining open problems caused by low frame rate, in order to build a whole cell tracking pipeline.
Chapter 8

Other Projects

This chapter introduces two research projects in which the author participated before the current research work.

The first project titled “Cyberbullying detection based on ConvNets” aims at detecting cyberbullying content in social networks. Cyberbullying can have a deep and long-lasting impact on its victims, who are often adolescents. Accurately detecting cyberbullying helps prevent it. However, the noise and errors in social media posts and messages make detecting cyberbullying very challenging. Inspired by the observation that the pronunciation of misspelled words in informal online conversations is often unchanged, we used the phoneme codes of the text as the features for a convolutional neural network. This procedure corrects spelling errors while preserving the pronunciation, thereby alleviating the problem of noise and bullying data sparsity. In Section 8.1, we introduce the proposed pronunciation-based convolutional neural network (PCNN) and the related experiments for evaluating its performance.

The second project titled “Efficient ConvNets design” aimed to investigate design patterns for ConvNets. ConvNets are essential to the deep learning-based segmentation models. However, designing high-performance ConvNets is still empirical. In addition, training Deep ConvNets requires heavy computational resources. In Section 8.2, we introduce the proposed design pattern for deep ConvNets which enables defining the structure of ConvNets using only three parameters.
8.1 Cyberbullying Detection with a Pronunciation based Convolutional Neural Network

8.1.1 Introduction

The rise of social media has significantly influenced our lives. However, this puts adolescents at risk of becoming victims of online misconduct, especially cyberbullying. Cyberbullying refers to an aggressive, intentional act conducted by either a group or an individual in cyberspace using information and communication technologies (e.g., email, mobile phone, and social networks) repeatedly or over time against victims who cannot easily defend themselves [55]. According to a 2015 report from the Cyberbullying Research Center, about one-third of the high school students from random samples have experienced cyberbullying [168].

Unlike in traditional bullying, techniques and forms used by cyberbullies change rapidly and are more harmful and harder to detect [169]. For example, it is easy to anonymously spread rumors about people online, and there is a low risk of being caught. Thus, it is necessary to detect cyberbullying in order to protect adolescents.

Unlike video and image-based methods, text-based cyberbullying is the most commonly form used by perpetrators. Moreover, other forms are usually combined with bullying text. Thus, we focus on detecting textual cyberbullying in this study. Textual cyberbullying detection methods can be divided into two categories: keywords-based and artificial intelligence (AI) based [52].

The simplest way is the keyword method, which uses keywords to search for sensitive content within a text. Although the idea is straightforward, this method can still obtain high precision score by using informative query terms and leveraging the internet searching [129, 120]. However, keywords themselves are far from representative of all cyberbullying content. Thus, keywords based approaches have difficulty achieving high recall, a more important metric than precision and accuracy for cyberbullying detection. This is because it is better to detect more cyberbullying posts, even if there are false positives, than to have a high precision but only find a fraction of the cyberbullying posts [180]. In addition, accuracy is not a useful metric in this context because the classifier can easily achieve relatively high accuracy of 93% by predicting all the samples as negatives (non-bullying) but have zero recall.

The AI method is more complex. The three core components of AI, representation, inference,
and learning, create three corresponding research directions for cyberbullying detection methods [52].

To be specific, most methods are based on supervised machine learning classifiers [180, 95, 39, 41, 222, 40, 43, 158, 42, 100, 71, 112, 153, 192]. Our cyberbullying detection methods also belong to the machine learning based approach.

The characteristics of posts and messages with bullying content make the detection of cyberbullying very challenging. First, as shown in Table 8.1, these texts have many words with incorrect spellings and symbols. However, we observed that those misspelled words are often informative. In addition, many sentences made up of symbols contain bullying.

Table 8.1: Examples of noisy data. The examples are from Formspring.me website.

<table>
<thead>
<tr>
<th>Examples</th>
<th>Noise Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;w@nN@ l!qqH+ y0 d!(k 0N f!r3 N d3nN sM0k3 !t w!+ m@ v@q!n@&quot;</td>
<td>Symbols</td>
</tr>
<tr>
<td>&quot;wHy yUhH w0N+ fU(k m3 !N d@ @$$ h0l3 ??&quot;</td>
<td>Symbols</td>
</tr>
<tr>
<td>&quot;lol yew on sum otha shxt nd not even dressed in all black&quot;</td>
<td>Intended typos</td>
</tr>
<tr>
<td>&quot;im sur3 sh3 d0nt want y0u&quot;</td>
<td>Numbers</td>
</tr>
<tr>
<td>&quot;iloveyourpenis&quot;</td>
<td>Concatenation</td>
</tr>
</tbody>
</table>

Second, the distribution of the classes within the data is imbalanced and the proportion of bullying content varies from different websites. For example, approximately 17% of the messages in the samples from Formspring.me, a question and answer-based social network, contain bullying content [180]. The ratio dropped to approximately 5% after we parsed the messages into individual sentences on our analysis of the same data.

The motivation of our work is the need for a practical, robust, and universal cyberbullying detection classifier with high performance. In this study, we propose to use the pronunciation of words within the texts as the features for a convolutional neural network (CNN), a classifier that has shown high performance on natural language processing, e.g., sentiment analysis [117]. The pronunciation conversion corrects spelling errors that did not change the original pronunciation of the word by mapping each word to phonetic code, which also reduces the size of the feature space. Our new pronunciation based convolutional neural network (PCNN) can alleviate the noise in social media text and improve classification performance. In order to overcome the class imbalance problem, we adopted three techniques: threshold-moving (TM) and cost function adjusting (CFA), and a hybrid solution (TM CFA) [46]. We tested our model on the Twitter dataset used in [112] and the Formspring dataset used in [180] to have a clear comparison between our approach and other works.
8.1.2 Related Work

Kontostathis et al. [120] analyzed cyberbullying corpora using the bag-of-words model to find the most common used terms by cyberbullies and used them to create queries capable of reaching a precision of 91.25% on average. Lempa et al. [129] developed an Android application, embedded with two methods, to implement the cyberbullying detection. One method is built on a brute force search algorithm search for sensitive words and phrases within the text. The other method extracts words and phrases as seed words and detects cyberbullying online with keyword categorization and relevance matching. The top precision of both methods reaches 89% and 91%, respectively [129].

Regarding classifier design, researchers have tested various classifiers, including Naïve Bayes, C4.5 decision tree, Random forests, and SVM with different kernels on corpora collected from popular social networks, such as Twitter and YouTube [180, 222]. Reynolds et al. [180] found that both the C4.5 decision tree and 3-nearest neighbor classifiers can reach a recall of 78.5% on the text-based dataset collected from Formspring.me, a question and answer based social network. Bullying posts (positives) were duplicated ten times to compensate for the imbalance within the data. However, this oversampling method is unreliable since it exaggerates the occurrence rate of the positive samples [180]. On the Twitter dataset, Xu et al. [222] showed that SVM with a linear kernel using unigrams and bigrams as features can achieve a recall of 79% and a precision of 76%. Other works are focused on ensemble methods such as cooperative and hybrid classifiers [40, 42, 153]. Dadvar et al. [40, 42] introduced two approaches to combine machine learning methods and expert systems. The different combinations depending on which classifier’s output is used as the input of the other. An accuracy metric called the area under the curve (AUC) was used to evaluate their approach. The hybrid system is made up of expert system and Naïve Bayes classifier, achieving their highest AUC score of 0.76. Mangaonkar et al. [153] evaluated 15 cooperative classifier combinations, including heterogeneous, homogeneous, and selective cooperation with different parallelisms. These ensemble classifiers are extremely complex and tuning the hyperparameters is difficult.

For feature selection, various textual content-based features, such as the basic bag-of-words and advanced sentiment prediction, were used as the input to classifiers [158, 71, 112]. Kasture took advantage of a psychometric feature analysis tool called Linguistic Inquiry and Word Count (LIWC) used as a feature extractor. These features were used to train a variety of classifiers and the best performance reached 96.3% recall and 98.4% precision on Random Forests using 10-fold
cross validation on the Twitter dataset [112]. To detect the cyberbullying and cyberstalking in
e-mails and messages, Ghasem et al. [71] selected the 500 most informative words as the feature
vector and achieved an F1 score of approximately 95% on SVM and a neural network classifier.
Nahar et al. [158] introduced a weighted TFIDF feature extractor and used LIBSVM with a linear
kernel to detect cyberbullying content in three social networks: Kongregate, Slashdot, and Myspace.
The experiment results show that their feature design significantly improved the performance of
the baseline LIBSVM. For example, the recall jumped from 25% to 98% on the Myspace dataset.
However, oversampling was used to handle the imbalance problem, which is not a useful method in
real-world implementations [158].

To further improve performance, many researchers implemented context-based features
such as user profile information and online duration [95, 39, 41, 40, 43, 100, 192]. Dadvar et al.
[39, 41, 40, 43] investigated incorporating user information as features to improve the performance.
They established a comprehensive, context-based feature set covering age, behavior, cross-platform
information, and activity history. These features were first tested by SVM and then used to build a
hybrid detection approach including an expert system as mentioned above. Patterns of social network
structures involving user behavior were used to detect and analyze cyberbullying [95, 100, 192].
Features like the number of friends, relation centrality, and bullying propagation were investigated and
used to aid the detection. Their research results show that human relationships and action dynamics
within social network structures can be taken into account to improve the results of cyberbullying
detection and prediction. However, this type of context information is usually unavailable due to
privacy protections. Thus, an effective and robust cyberbullying detection method should be able to
perform well without this information.
8.1.3 Method Description

8.1.3.1 Data Collection

Due to the relatively limited amount of research in cyberbullying detection, there are no well-established benchmark datasets to test various approaches. The datasets used in previous publications differ significantly in size, source, and format. We decided to use the two datasets used in [112] and [180] for the following reasons.

The Twitter dataset was used since Twitter is a popular platform and the dataset has been recently created and analyzed [112]. However, the dataset contains only 1313 messages, and the bullying content proportion, approximately 38.8%, is significantly higher than it would be under realistic conditions.

Another dataset, collected from the social network Formspring.me and used in [180], was chosen to give an additional evaluation of our approach. 13,000 messages were collected and then labeled by a web service called Amazon Mechanical Turk, where three workers each voted on whether or not a document contains bullying content. Thus, every message has a corresponding number of votes from the workers. Approximately 6.6% of the messages were labeled as bullying posts by at least two workers. We parsed the messages from the original dataset into sentences and relabeled the messages containing at least one vote. This resulted in 23,243 sentences in which 1,623, or approximately 7%, are labeled as bullying messages.
8.1.3.2 Data Preprocessing and Word-to-Pronunciation Conversion

The Twitter dataset had already been preprocessed by removing usernames, hashtags, and hyperlinks from the Tweets [112]. The Tweets were then converted into plain text by replacing accented characters and removing non-alphanumeric tokens, excluding the apostrophe. After preprocessing, the maximum length of any Tweet was 33 words, and only approximately 15% of the words of the dataset cannot be found in a dictionary.

We performed similar preprocessing on the Formspring dataset. Specifically, we removed irrelevant words like hyperlinks, user indicators (“Q:” and “A.”) and non-alphanumeric tokens. Then, a term-compression operation was performed to ensure that there are no more than two consecutive occurrences of any character in a word. For example, “cooll,” “bitchhh,” and “guesss” become “cooll,” “bitchh,” and “guess”. This simple technique helps the pronunciation conversion procedure to group misspelled words with the same meaning and pronunciation together with the corrected word.

After preprocessing the datasets, we created the phonetic representation of each word using eSpeak, an open-source speech synthesizer software [1]. This conversion was based on pronunciation rules and a dictionary lookup list, both of which can be manually modified to better suit the purpose and research context.

The phoneme strings can be encoded using ASCII code or the International Phonetic Alphabet (IPA), which uses characters from the Latin alphabet. Here, we used ASCII code to remain consistent with the original plain text. Some examples of the conversion are in Table 8.2.

The word-to-pronunciation conversion can map some misspelled words to the pronunciation of the

Table 8.2: Examples of word-to-pronunciation conversion. The examples are from Twitter and Formspring datasets.

<table>
<thead>
<tr>
<th>Word and phrases</th>
<th>Phoneme Code</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>“fuck, fuc, fuk”</td>
<td>f'Vk</td>
<td>Positive</td>
</tr>
<tr>
<td>“fuckk”</td>
<td>f'Vkk</td>
<td>Neutral</td>
</tr>
<tr>
<td>“shitt, shit”</td>
<td>S'It</td>
<td>Positive</td>
</tr>
<tr>
<td>“suck, suk, suc”</td>
<td>s'Vk</td>
<td>Positive</td>
</tr>
<tr>
<td>“dik, dic, dick”</td>
<td>d'Ik</td>
<td>Positive</td>
</tr>
<tr>
<td>“guesss, guess”</td>
<td>g‘Es</td>
<td>Positive</td>
</tr>
<tr>
<td>“bitchh, bitch, bich”</td>
<td>b’Rs</td>
<td>Positive</td>
</tr>
<tr>
<td>“cool, cooll”</td>
<td>k’u:l</td>
<td>Positive</td>
</tr>
<tr>
<td>“cum, come”</td>
<td>k’Vm</td>
<td>Negative</td>
</tr>
</tbody>
</table>
corrected word. We observed that, especially in bullying posts, perpetrators tended to use slang or intentionally misspell words when insulting others. For example, five out of six words in the phrase “did u now tht ur ugli” are typos. However, the pronunciation of these misspelled insults usually remains unchanged. This means that the word-to-pronunciation conversion can generate the same phonetic code for “ugli” and “ugly,” effectively “correcting” the misspelled word.

On the other hand, this conversion can create noise by mapping a bad word such as “cum” and a normal word such as “come” to the same phonetic representation. However, we believe the benefits of this procedure outweigh the costs since it correctly maps words much more often than it creates noise. Furthermore, surrounding context words could be used to distinguish words like “cum” and “come.”

After preprocessing the data and applying the word-to-pronunciation conversion, the phonetic representation of each word was converted to a randomly initialized 300-dimensional vector. Then, a zero vector with the same dimension was used to pad each sentence so that they are all the same length. Finally, each sentence was projected to a matrix of the same size.

8.1.3.3 CNN and PCNN

Convolutional neural networks (CNN), originally created for image processing, have performed very well in natural language processing (NLP), especially in sentiment analysis and question classification [117, 126, 224]. Inspired by their powerful feature representation capability, flexible structure, and high efficiency for training using a GPU, we adopted CNN as the baseline classifier. To have a clear performance comparison between PCNN and the baseline CNN, we used the same model architecture in [117]. As shown in the PCNN architecture diagram, only one layer of convolution and max-pooling was used with three different filter sizes. The sizes of the three convolutional filters were chosen to be 1, 2, and 3, slightly differing from the filters in [117]. The filter sizes were chosen based on how many consecutive words were necessary to detect bullying content. The convolutional operation on m consecutive words is given in:

$$ h_i = f(w_c x_{i:i+m} + b_c) $$

Here, $x_{i:i+m}$, $h_i$, $w_c$, $b_c$, and f are the embedding matrix of m words, the feature value generated by the operation, the weight and bias of the corresponding convolutional filter, and the activation function, respectively.
A max-pooling operation was applied to all the features from one convolutional filter. Then, the features were concatenated into $h$, a feature vector with dimensions equal to the number of filters applied. A softmax layer with dropout was applied to the output of the pooling layer to predict the class probability, $P$, as follows:

$$P(Y = i \mid X, \theta) = \text{softmax}_i (w_s h + b_s)$$

(8.2)

Here $X$, $h$, $Y$, $w_s$, $b_s$, $i$, and are the input embedding matrix, feature vector from the convolutional and pooling layer, class prediction, weights of the penultimate layer, corresponding bias, class number, and parameter set, respectively.

We used two separate CNN to establish a baseline. Three hundred dimensional word-embedding based on Google’s word-to-vector was used to create the feature set for the first baseline CNN, which we named CNN Pre-trained. Randomly generated vectors were used to create the feature set for the second baseline CNN, which we named CNN Random. For PCNN, the phoneme codes were randomly initialized into vectors for the feature set. All the embedding for CNN and PCNN was updated during the training process based on the stochastic gradient descent [227].

Our method and the structure of PCNN are shown in Figure 8-1.

8.1.3.4 Techniques for Handling Class Imbalance

Unlike the movie reviews used in sentiment analysis, the class distribution is imbalanced for most cyberbullying related datasets. For example, only about 6.6% messages in Formspring dataset were labeled as bullying by two voters. The class imbalance within the dataset creates two problems.

First, the small percentage of positive samples makes it difficult to detect them, especially when the dataset is small. Furthermore, the lack of sufficient samples for unique instances of bullying makes it nearly impossible for classifiers to recognize them.

Second, the most commonly used cost functions of CNN, the one we used is the negative log likelihood (NLL), were designed to only improve the accuracy rather than recall or precision. Consequently, CNN models have a bias towards the dominating class.

There are three methods for dealing with the class imbalance problem: oversampling or undersampling the dataset, modifying the classifier to be cost sensitive, and training on only one-class
However, the sampling technique will change the proportion of the classes, causing the data to no longer represent realistic conditions, and the one-class classification method is only useful for detecting anomalies and outliers in the dataset.

Since these two methods are not useful for most datasets, we chose to modify the classifier to be cost sensitive. There are three ways to implement this: threshold-moving, cost function adjusting, and a hybrid solution. Threshold-moving replaces Bayes estimation with maximum likelihood estimation in order to compensate for the great difference between the prior probabilities of the two classes. The prediction using this method is calculated by dividing the prediction of the classes by their prior probabilities, \( P(Y = i) \). It is implemented as follows:

\[
Y_{\text{predict}} = \arg \max_i (P(Y = i \mid X, \theta) / P(Y = i))
\]  

(8.3)

Re-normalization could be added, but it does not affect the prediction result.

Cost function adjusting aims to modify the cost function for the stochastic gradient descent training so that each element of the minority class can cause more parameter optimization than each element of the majority class does to compensate for the class imbalance. The new cost function is given as follows:

\[
\text{cost} = -\left(1/Kn\right) \cdot \sum_{i=1}^{n} \left( \log(P(Y = i \mid X, \theta)) / P(Y = i) \right)
\]  

(8.4)

\( P(Y = i | X, \theta) \) is the predicted probability of class \( i \) when the input embedding matrix is \( X \) and the model parameter set is \( \theta \). \( P(Y = i) \) is the prior probability of class \( i \), \( n \) is the size of the mini-batch, and \( K \) is the number of classes.

### 8.1.4 Experiment Result

We used recall, precision, F1 score, and accuracy as metrics to evaluate the performance of our models. All of these metrics are based on the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). The formulas for calculating the metrics are as follows:

\[
\text{Recall} = \frac{TP}{TP + FN}
\]  

(8.5)

\[
\text{Precision} = \frac{TP}{TP + FP}
\]  

(8.6)
\[ F1 = 2 \times \text{Recall} \times \text{Precision} / (\text{Recall} + \text{Precision}) \]  

(8.7)

We used 5-fold cross-validation for the Formspring dataset. However, we used ten-fold cross-validation on the Twitter data in order to remain consistent with [112]. Two baseline CNN classifiers and PCNN were tested and compared with the other classifiers used in [112]. The code was written in Python and the neural network components were based on the Theano package and Kim’s work [117, 203].

### 8.1.4.1 Comparison of Classification Performance

Table 8.3 shows the results of our methods on the Twitter dataset compared to previous work based on LIWC features in [112]. It shows that PCNN outperforms all models on all metrics in the original paper and is slightly better than CNN with randomly generated word embedding. In addition, PCNN and CNN Random performed better than CNN with pre-trained Google word vectors. This may be because the corpus used for pre-training was not specific to cyberbullying detection.

Table 8.3: Approach comparison on twitter dataset. Results are average of 10-fold cross validation. The accuracy of first four classifiers from [12] is not given.

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Forest</td>
<td>0.984</td>
<td>0.963</td>
<td>-</td>
<td>0.973</td>
</tr>
<tr>
<td>SVM</td>
<td>0.986</td>
<td>0.912</td>
<td>-</td>
<td>0.948</td>
</tr>
<tr>
<td>Multilayer Perceptron</td>
<td>0.951</td>
<td>0.939</td>
<td>-</td>
<td>0.945</td>
</tr>
<tr>
<td>J 48 Decision Tree</td>
<td>0.947</td>
<td>0.941</td>
<td>-</td>
<td>0.944</td>
</tr>
<tr>
<td>CNN Pre-trained</td>
<td>0.973</td>
<td>0.937</td>
<td>0.974</td>
<td>0.955</td>
</tr>
<tr>
<td>CNN Random</td>
<td>0.994</td>
<td>0.962</td>
<td>0.988</td>
<td>0.978</td>
</tr>
<tr>
<td>PCNN</td>
<td>0.991</td>
<td>0.970</td>
<td>0.989</td>
<td>0.980</td>
</tr>
</tbody>
</table>

Table 8.4 summarizes the results of the three approaches evaluated on the Formspring dataset. PCNN outperformed the two baseline CNN models in all metrics, demonstrating the benefit of using the word-to-pronunciation conversion. Despite the excellent results on the Twitter dataset, the overall classification performance on the Formspring data was much lower. This may be due to the severe noise and class imbalance in the Formspring dataset. For example, approximately 55% of the words in the Formspring dataset vocabulary cannot be found in the dictionary while only 15% of the words in the Twitter dataset is misspelled.
Table 8.4: Approach comparison on Formspring dataset. Results are average of 5-fold cross-validation.

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN Pre-trained</td>
<td>0.728</td>
<td>0.364</td>
<td>0.964</td>
<td>0.485</td>
</tr>
<tr>
<td>CNN Random</td>
<td>0.728</td>
<td>0.429</td>
<td>0.966</td>
<td>0.540</td>
</tr>
<tr>
<td>PCNN</td>
<td>0.740</td>
<td>0.453</td>
<td>0.968</td>
<td>0.562</td>
</tr>
</tbody>
</table>

8.1.4.2 Comparison of Techniques for Handling Class Imbalance

Table 8.5 shows the results of the three CNN models on the Twitter dataset using the different class imbalance handling techniques: threshold-moving, cost function adjusting, and a hybrid solution. These techniques slightly improved recall, and the combination of TM and CFA performed the best out of the three. Furthermore, TM CFA PCNN can improve the overall performance and outperforms the two baseline CNN models. However, the improvement is insignificant since the degree of class imbalance in the Twitter dataset is low and the recall is already very high. Thus, these techniques need to be evaluated on a noisier and more imbalanced dataset. Table 8.6 gives the corresponding results on the Formspring dataset. It shows that all three techniques enhanced the recall at the cost of precision and even accuracy. Among them, CFA improved the overall classification performance the most, increasing recall and F1 score without hurting accuracy. Moreover, PCNN obtained the highest recall and F1 score than others when using cost function adjusting. The results on both datasets show that cost function adjusting is an effective technique to handle datasets with class imbalance. In addition, the word-to-pronunciation conversion contributes to the increase the recall without other losses.

Table 8.5: Class imbalance-tackling on Twitter dataset.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>TM</td>
<td>CNN Pre-trained</td>
<td>0.910</td>
<td>0.943</td>
<td>0.956</td>
<td>0.926</td>
</tr>
<tr>
<td></td>
<td>CNN Random</td>
<td>0.984</td>
<td>0.961</td>
<td>0.985</td>
<td>0.972</td>
</tr>
<tr>
<td></td>
<td>PCNN</td>
<td>0.989</td>
<td>0.972</td>
<td>0.989</td>
<td>0.980</td>
</tr>
<tr>
<td>CFA</td>
<td>CNN Pre-trained</td>
<td>0.954</td>
<td>0.946</td>
<td>0.971</td>
<td>0.956</td>
</tr>
<tr>
<td></td>
<td>CNN Random</td>
<td>0.992</td>
<td>0.960</td>
<td>0.986</td>
<td>0.976</td>
</tr>
<tr>
<td></td>
<td>CFA PCNN</td>
<td>0.991</td>
<td>0.972</td>
<td>0.990</td>
<td>0.981</td>
</tr>
<tr>
<td>TM CFA</td>
<td>CNN Pre-trained</td>
<td>0.919</td>
<td>0.949</td>
<td>0.960</td>
<td>0.934</td>
</tr>
<tr>
<td></td>
<td>CNN Random</td>
<td>0.986</td>
<td>0.965</td>
<td>0.986</td>
<td>0.975</td>
</tr>
<tr>
<td></td>
<td>PCNN</td>
<td>0.991</td>
<td>0.975</td>
<td>0.990</td>
<td>0.983</td>
</tr>
</tbody>
</table>
Table 8.6: Class imbalance-tackling on Formspring dataset.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>TM</td>
<td>CNN Pre-trained</td>
<td>0.328</td>
<td>0.602</td>
<td>0.923</td>
<td>0.425</td>
</tr>
<tr>
<td>TM</td>
<td>CNN Random</td>
<td>0.280</td>
<td>0.694</td>
<td>0.894</td>
<td>0.399</td>
</tr>
<tr>
<td>TM</td>
<td>PCNN</td>
<td>0.305</td>
<td>0.717</td>
<td>0.902</td>
<td>0.428</td>
</tr>
<tr>
<td>CFA</td>
<td>CNN Pre-trained</td>
<td>0.440</td>
<td>0.529</td>
<td>0.947</td>
<td>0.480</td>
</tr>
<tr>
<td>CFA</td>
<td>CNN Random</td>
<td>0.562</td>
<td>0.558</td>
<td>0.960</td>
<td>0.560</td>
</tr>
<tr>
<td>CFA</td>
<td>PCNN</td>
<td>0.540</td>
<td>0.606</td>
<td>0.958</td>
<td>0.571</td>
</tr>
<tr>
<td>TM CFA</td>
<td>CNN Pre-trained</td>
<td>0.168</td>
<td>0.733</td>
<td>0.818</td>
<td>0.273</td>
</tr>
<tr>
<td>TM CFA</td>
<td>CNN Random</td>
<td>0.203</td>
<td>0.778</td>
<td>0.846</td>
<td>0.322</td>
</tr>
<tr>
<td>TM CFA</td>
<td>PCNN</td>
<td>0.254</td>
<td>0.787</td>
<td>0.881</td>
<td>0.384</td>
</tr>
</tbody>
</table>
8.2 CrescendoNet: A New Deep Convolutional Neural Network with Ensemble Behavior

8.2.1 Introduction

Deep convolutional neural networks (CNNs) have significantly improved the performance of image classification [122, 88, 199]. However, training a CNN also becomes increasingly difficult with the network deepening. One of important research efforts to overcome this difficulty is to develop new neural network architectures [97, 125]. Recently, the residual network [88] and its variant [97] have used residual connections among layers to train very deep CNN. The residual connections promote the feature reuse, help the gradient flow, and reduce the need for massive parameters. The ResNet [88] and DenseNet [97] achieved state-of-the-art accuracy on benchmark datasets. Alternatively, FractalNet [125] expanded the convolutional layers in a fractal form to generate deep CNNs. Without residual connections [88] and manually deep supervision [127], FractalNet achieved high performance on image classification based on network structural design only.

Many studies tried to understand reasons behind the representation view of deep CNNs. Veit et al. [213] showed that residual network could be seen as an ensemble of relatively shallow effective paths. However, Greff et al. [80] argued that ensembles of shallow networks cannot explain the experiment results of lesioning, layer dropout, and layer reshuffling on ResNet. They proposed that residual connections have led to unrolled iterative estimation in ResNet. Meanwhile, Larsson et al. [125] speculated that the high performance of FractalNet was due to the unrolled iterative estimation of features of the longest path using features of shorter paths. Although unrolled iterative estimation model can explain many experimental results, it is unclear how it helps improve the classification performance of ResNet and FractalNet. On the other hand, the ensemble model can explain the performance improvement easily.

In this work, we propose CrescendoNet, a new deep convolutional neural network with ensemble behavior. Same as other deep CNNs, CrescendoNet uses stacking simple building blocks, called Crescendo blocks (Figure 8-2). Each Crescendo block comprises a set of independent feed-forward paths with increased numbers of convolution and batch-norm layers [102]. We only use the identical size, $3 \times 3$, for all convolutional filters in the entire network. Despite its simplicity, CrescendoNet shows competitive performance on benchmark CIFAR10, CIFAR100, and SVHN.
CrescendoNet architecture used in experiments, where *scale* = 4 and *interval* = 1.

CrescendoNet does not include residual connections. The high performance of CrescendoNet also comes entirely from its network structural design. Unlike the FractalNet, in which the numbers of convolutional layers and associated parameters increase exponentially, the numbers of convolutional layers and parameters in Crescendo blocks increase linearly.

CrescendoNet shows clear ensemble behavior (Section 8.2.3.4). In CrescendoNet, although the longer paths have better performances than shorter paths, the combination of paths with different length have even better performance. A set of paths outperform its subsets, which is different from FractalNet, in which the longest path alone achieves the similar performance as the entire network does, far better than other paths do. The implicit ensemble behavior enables CrescendoNet to have an anytime classification property, which means the classifier can always perform an acceptable prediction given limited time budget. For example, an instance of CrescendoNet can achieve 95.19% prediction accuracy with CIFAR10 while a subset of its branches can reach 91.31% accuracy using only one-fourth of computational cost. Thus, CrescendoNet has potential application for real-time and safety-critical inference problems, like perception for self-driving vehicles.

Furthermore, the independence between paths in CrescendoNet allows us to introduce a new path-wise training procedure, in which paths in each building block are trained independently and sequentially. The path-wise process can reduce the memory needed for training. Specifically, we can
reduce the amortized memory used for computing gradients and for storing gradients to about one fourth when using momentum algorithms.

We summarize our contribution as follows:

- We propose the Crescendo block with linearly increased numbers of convolutional and batch-norm layers. The CrescendoNet generated by stacking Crescendo blocks further shows that the high performance of deep CNNs can be achieved without explicit residual learning.

- Through our analysis and experiments, we discovered an emergent behavior which is significantly different from which of FractalNet. The entire CrescendoNet outperforms any subset of it can provide an insight of improving the model performance by increasing the number of paths by a pattern. We also demonstrated the anytime classification property of CrescendoNet. The classifier can achieve good prediction accuracy and improve smoothly as the time budget increases.

- We introduce a path-wise training approach for CrescendoNet, which can lower the memory requirements without significant loss of accuracy given sufficient data.

8.2.2 CrescendoNet

8.2.2.1 Architecture Design

Crescendo Block The Crescendo block is built by two layers, the convolution layer with the activation function and the following batch normalization layer [102]. The convolutional layers have the identical size, $3 \times 3$. The Conv-Activation-BatchNorm unit $f_1$, defined in the Eq.8.8 is the base branch of the Crescendo block. We use ReLU [159] as the activation function to avoid the problem of vanishing gradients.

$$f_1(z) = \text{batchnorm}(\text{activation}(\text{conv}(z)))$$ (8.8)

The variable $z$ denotes the input feature maps. We use two hyper-parameters, the scale $S$ and the interval $I$ to define the structure of the Crescendo block $H_S$. The interval $I$ specifies the depth difference between every two adjacent branches and the scale $S$ sets the number of branches per
block. The structure of the \( n^{th} \) branch is defined by the following equation:

\[
f_n(z) = f^n_{\text{I}}(z)
\]  

(8.9)

where the superscript \( n\text{I} \) is the number of recursion time of the function \( f_1 \). The structure of Crescendo block \( H_S \) can be obtained below:

\[
H_S(z) = f_1(z) \oplus f_2(z) \oplus \ldots f_S(z)
\]

(8.10)

where \( \oplus \) denotes an element-wise averaging operation. Note that the feature maps from each path are averaged element-wise, leaving the width of the channel unchanged. A Crescendo block with \( S = 4 \) and \( I = 1 \) is shown in Figure 8-2.

The structure of Crescendo block is designed for exploiting more feature expressiveness. The different depths of parallel paths lead to various receptive fields and therefore generate features in different abstract levels. Also, such an incremental and parallel form explicitly supports the ensemble effects, which shows excellent characteristics for efficient training and anytime classification. We will explain and demonstrate this in the following sections.

**CrescendoNet Architecture** The main body of CrescendoNet is composed of stacked Crescendo blocks with max-pooling layers between adjacent blocks (Figure 8-2). Following the main body, like most deep CNNs, we use two fully connected layers and a soft-max layer as the classifier. In all experiments, the two fully connected layers have 384 hidden units and 192 hidden units respectively. The overall structure of CrescendoNet is simple, and we only need to tune the Crescendo block to modify the entire network.

**8.2.2.2 Path-wise training**

To reduce the memory consumption during training CrescendoNet, we propose a path-wise training procedure, leveraging the independent multi-path structure of our model. We denote stacked Conv-Activation-BatchNorm layers in one Crescendo block as one path. We train each path individually, from the shortest to the longest repetitively. When we are training one path, we freeze the parameters of other paths. In other words, these frozen layers only provide learned features to support the training. Figure 8-3 illustrates the procedure of path-wise training within
Figure 8-3: Path-wise training procedure.

A CrescendoNet block containing four paths. The path-wise training method has two advantages. First, it significantly reduces the memory requirements for convolutional layers, which constitutes the primary memory cost for training CNNs. For example, the upper bound of the memory required for computation and storage of gradients using momentum stochastic gradient descent algorithms can be reduced to about 40% for a Crescendo block with four paths where interval = 1. Second, path-wise training works well with various optimizers and regularizations. Even dropout and drop-path apply to the model during the training process.

8.2.2.3 Regularization

Dropout [193] and drop-connect [216], which randomly set a selected subset of activations or weights to zero respectively, are effective regularization techniques for deep neural networks. Their variant, drop-path [125], shows further performance improvement by dropping paths when training FractalNet.

We use both dropout and drop-path for regularizing the Crescendo block. We drop the branches in each block with a predefined probability. For example, given drop-path rate, $p = 0.3$, the expectation of the number of dropped branches is 1.2 for a Crescendo block with four branches. For the fully connected layers, we use L2 norm of their weights as an additional term to the loss.

8.2.3 Experiments
8.2.3.1 Datasets

We evaluate our models with three benchmark datasets: CIFAR10, CIFAR100 [121], and Street View House Numbers (SVHN) [165]. CIFAR10 and CIFAR100 each have 50,000 training images and 10,000 test images, belonging to 10 and 100 classes respectively. All the images are in an RGB format with the size of $32 \times 32$-pixel. SVHN are color images, with the same size of $32 \times 32$-pixel, containing 604,388 and 26,032 images for training and testing respectively. Note that these digits are cropped from a series of numbers. Thus, there may be more than one digit in an image, but only the one in the center is used as the label. For data augmentation, we use a widely adopted scheme [139, 125, 97, 98, 195, 191, 88]. We first pad images with 4 zero pixels on each side, then crop padded images to $32 \times 32$-pixel randomly and horizontally flipping with a 50% probability. We preprocess each image in all three datasets by subtracting off the mean and dividing the variance of the pixels.

8.2.3.2 Training

We use Mini-batch gradient descent to train all our models. We implement our models using TensorFlow distributed computation framework [2] and ran them on NVidia P100 GPU. We also optimize our models by adaptive momentum estimation (Adam) optimization [118] and Nesterov Momentum optimization [164] respectively. For Adam optimization, we set the learning rate hyper-parameter to 0.001 and let Adam adaptively tune the learning rate during the training. We choose the momentum decay hyper-parameter $\beta_1 = 0.9$ and $\beta_2 = 0.999$. And we set the smoothing term $\epsilon = 10^{-8}$. This configuration is the default setting for the AdamOptimizer class in TensorFlow. For Nesterov Momentum optimization, we set the hyper-parameter $momentum = 0.9$. We decay the learning rate from 0.1 to 0.01 after 512 epochs for CIFAR and from 0.05 to 0.005, then to 0.0005, after 42 epochs and 63 epochs respectively for SVHN. We use truncated normal distribution for parameter initialization. The standard deviation of hyper-parameters is 0.05 for convolutional weights and 0.04 for fully connected layer weights. For all datasets, we use the batch size of 128 on each training replica. For the whole net training, we run 700 epochs on CIFAR and 70 epochs on SVHN. For the path-wise training, we run 1400 epochs on CIFAR and 100 epochs on SVHN.

Using a CrescendoNet model with three blocks each contains four branches as illustrated in Figure 8-2, we investigate the following preliminary aspects: the model performance under different
block widths, the ensemble effect, and the path-wise training performance. We study the Crescendo block with three different width configurations: using an equal width globally, an equal width within the block, and increasing width. All the three configurations have the same fully connected layers. For the first one, we set the number of feature maps to 128 for all the convolutional layers. For the second, the numbers of feature maps are (128, 256, 512) for convolutional layers in each block. For the last, we gradually increase the feature maps for each branch in three blocks to (128, 256, 512) correspondingly. For example, the number of feature maps for the second and fourth branches in the second block is (192, 256) and (160, 192, 224, 256). The following equation defines the exact number of maps for each layer:

$$n_{maps} = n_{inmaps} + i_{layer} \frac{n_{outmaps} - n_{inmaps}}{n_{layers}}$$

(8.11)

where $n_{maps}$ denotes the number of feature maps for a layer, $n_{inmaps}$ and $n_{outmaps}$ are number of input and output maps respectively, $n_{layers}$ is the number of layers in the block, and $i_{layer}$ is the index of the layer in the branch, starting from one.

To inspect the ensemble behavior of CrescendoNet, we compare the performance of models with and without drop-path technique and subnets composed of different combinations of branches in each block. For the simplicity, we denote the branch combination as a set $P$ containing the index of the branch. For example, $P = \{1, 3\}$ means the blocks in the subnet only contains the first and third branches. The same notation is used in Table 8.8 and Figure 8-4.

### 8.2.3.3 Results of the whole net

Table 8.7 gives a comparison among CrescendoNet and other representative models on CIFAR and SVHN benchmark datasets. For five datasets, CrescendoNet with only 15 layers outperforms almost all networks without residual connections, plus original ResNet and ResNet with Stochastic Depth. For CIFAR10 and CIFAR100 without data augmentation, CrescendoNet also performs better than all the given models except DenseNet with bottleneck layers and compression (DenseNet-BC) with 250 layers. However, CrescendoNet’s error rate 1.76% matches the 1.74% error rate of given DenseNet-BC, on SVHN dataset which has rich data for each class. Comparing with FractalNet, another outstanding model without residual connection, CrescendoNet has a more straightforward structure, fewer parameters, but higher accuracies.
Table 8.7: Whole net classification error (%) with CIFAR10/CIFAR100/SVHN.

<table>
<thead>
<tr>
<th>Method</th>
<th>Depth</th>
<th>Params</th>
<th>C10</th>
<th>C10+</th>
<th>C100</th>
<th>C100+</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network in Network</td>
<td>-</td>
<td>-</td>
<td>10.41</td>
<td>8.81</td>
<td>35.68</td>
<td>-</td>
<td>2.35</td>
</tr>
<tr>
<td>All-CNN</td>
<td>-</td>
<td>-</td>
<td>9.08</td>
<td>7.25</td>
<td>-</td>
<td>33.71</td>
<td>-</td>
</tr>
<tr>
<td>Deeply Supervised Net</td>
<td>-</td>
<td>-</td>
<td>9.69</td>
<td>7.97</td>
<td>-</td>
<td>34.57</td>
<td>1.92</td>
</tr>
<tr>
<td>Highway Network</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7.72</td>
<td>-</td>
<td>32.39</td>
<td>-</td>
</tr>
<tr>
<td>FractalNet (dropout+drop-path)</td>
<td>21</td>
<td>38.6M</td>
<td>7.33</td>
<td>4.60</td>
<td>28.20</td>
<td>23.73</td>
<td>1.87</td>
</tr>
<tr>
<td>ResNet</td>
<td>110</td>
<td>1.7M</td>
<td>13.63</td>
<td>6.41</td>
<td>44.74</td>
<td>27.22</td>
<td>2.01</td>
</tr>
<tr>
<td>Stochastic Depth</td>
<td>110</td>
<td>1.7M</td>
<td>11.66</td>
<td>5.23</td>
<td>37.80</td>
<td>24.58</td>
<td>1.75</td>
</tr>
<tr>
<td>Wide ResNet</td>
<td>16</td>
<td>11.0M</td>
<td>-</td>
<td>4.81</td>
<td>-</td>
<td>22.07</td>
<td>-</td>
</tr>
<tr>
<td>with Dropout</td>
<td>28</td>
<td>36.5M</td>
<td>-</td>
<td>4.17</td>
<td>-</td>
<td>20.50</td>
<td>-</td>
</tr>
<tr>
<td>ResNet (pre-activation)</td>
<td>164</td>
<td>1.7M</td>
<td>-</td>
<td>5.46</td>
<td>-</td>
<td>24.33</td>
<td>-</td>
</tr>
<tr>
<td>DenseNet (k = 12)</td>
<td>40</td>
<td>1.0M</td>
<td>7.00</td>
<td>5.24</td>
<td>27.55</td>
<td>24.42</td>
<td>1.79</td>
</tr>
<tr>
<td>DenseNet-BC (k = 24)</td>
<td>250</td>
<td>15.3M</td>
<td>5.19</td>
<td>3.62</td>
<td>19.64</td>
<td>17.60</td>
<td>1.74</td>
</tr>
</tbody>
</table>

| CrescendoNet Nesterov         |       |        |         |         |         |         |      |
| (128, 128, 128)               | 15    | 4.1M   | 7.26    | 5.53    | 29.83   | 25.09   | 1.90 |
| (128, 256, 512)-W             | 15    | 18.3M  | 7.08    | 5.20    | 27.48   | 23.57   | 1.90 |
| (128, 256, 512)               | 15    | 27.7M  | **6.81**| 5.03    | **26.39**| 22.97   | 1.78 |
| without drop-path             | 15    | 27.7M  | 8.80    | 6.42    | 29.14   | 23.94   | 2.04 |

| CrescendoNet Adam             |       |        |         |         |         |         |      |
| (128, 128, 128)               | 15    | 4.1M   | 7.26    | 5.20    | 33.04   | 25.76   | **1.73**|
| (128, 256, 512)               | 15    | 27.7M  | **6.90**| 4.81    | 30.00   | 24.67   | 1.76 |
| without drop-path             | 15    | 27.7M  | 9.20    | 6.90    | 33.50   | 26.35   | 1.79 |
| path-wise training            | 15    | 27.7M  | 8.93    | 6.90    | 34.88   | 29.95   | 1.95 |

We highlight the top three accuracies in each column with the bold font. The three numbers in the parentheses denote the number of output feature maps of each block. The plus sign (+) denotes the data augmentation. The sign (-W) means that the feature maps of layers in each branch increase as explained in the model configuration section. The compared models include: Network in Network [195], ALL-CNN [191], Deeply Supervised Net [127], Highway Network [195], FractalNet [125], ResNet [88], ResNet with Stochastic Depth [98], Wide ResNet [225], and DenseNet [97].

The lower rows in Table 8.7 compare the performance of our model given different configuration. In three different widths, the performance simultaneously grows with the number of feature maps. In other words, there is no over-fitting when we increase the capacity of CrescendoNet in an appropriate scope. Thus, CrescendoNet demonstrates a potential to further improve its performance by scaling up. Also, the drop-path technique shows its benefits to our models on all the datasets, just as it does to FractalNet.

Another impressive result from Table 8.7 is the performance comparison between Adam and Nesterov Momentum optimization methods. Comparing with Nesterov Momentum method, Adam
Table 8.8: Subnet classification error (%) with CIFAR10/CIFAR100/SVHN.

<table>
<thead>
<tr>
<th>Branches</th>
<th>Depth</th>
<th>C10</th>
<th>C10+</th>
<th>C100</th>
<th>C100+</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,2,3,4}</td>
<td>15</td>
<td>6.90</td>
<td>4.81</td>
<td>30.00</td>
<td>24.67</td>
<td>1.76</td>
</tr>
<tr>
<td>{2,3,4}</td>
<td>15</td>
<td>6.91</td>
<td>4.93</td>
<td>29.90</td>
<td>24.92</td>
<td>1.87</td>
</tr>
<tr>
<td>{1,2,4}</td>
<td>15</td>
<td>7.61</td>
<td>5.59</td>
<td>32.25</td>
<td>26.65</td>
<td>1.94</td>
</tr>
<tr>
<td>{1,2,3}</td>
<td>12</td>
<td>7.94</td>
<td>6.00</td>
<td>31.86</td>
<td>27.18</td>
<td>2.02</td>
</tr>
<tr>
<td>{3,4}</td>
<td>15</td>
<td>7.54</td>
<td>5.31</td>
<td>31.61</td>
<td>26.29</td>
<td>1.97</td>
</tr>
<tr>
<td>{2,4}</td>
<td>15</td>
<td>7.73</td>
<td>5.56</td>
<td>32.60</td>
<td>27.09</td>
<td>2.01</td>
</tr>
<tr>
<td>{2,3}</td>
<td>12</td>
<td>8.03</td>
<td>5.85</td>
<td>32.08</td>
<td>28.24</td>
<td>2.04</td>
</tr>
<tr>
<td>{1,4}</td>
<td>15</td>
<td>8.66</td>
<td>6.38</td>
<td>35.81</td>
<td>29.74</td>
<td>2.05</td>
</tr>
<tr>
<td>{1,2}</td>
<td>9</td>
<td>10.58</td>
<td>8.69</td>
<td>37.03</td>
<td>34.08</td>
<td>2.75</td>
</tr>
<tr>
<td>{4}</td>
<td>15</td>
<td>10.69</td>
<td>7.96</td>
<td>38.66</td>
<td>33.71</td>
<td>2.53</td>
</tr>
<tr>
<td>{3}</td>
<td>12</td>
<td>11.31</td>
<td>8.27</td>
<td>38.26</td>
<td>34.70</td>
<td>2.43</td>
</tr>
<tr>
<td>{2}</td>
<td>9</td>
<td>12.13</td>
<td>10.14</td>
<td>40.32</td>
<td>37.05</td>
<td>2.78</td>
</tr>
<tr>
<td>{1}</td>
<td>6</td>
<td>28.60</td>
<td>30.31</td>
<td>70.51</td>
<td>73.41</td>
<td>8.74</td>
</tr>
</tbody>
</table>

The numbers in the curly brackets denote the branches used in each block.

Table 8.8 provides a performance comparison among different path combinations of CrescendoNet, trained by Adam optimization, with block-wise width (128, 256, 512). The results show the ensemble behavior of our model. Specifically, the more paths contained in the network, the better the performance. And the whole net outperforms any single path network with a large margin. For example, the entire net and the net based on the longest path show the inference error rate of 6.90% and 10.69% respectively, for CIFAR10 without data augmentation. This implicit ensemble behavior differentiates CrescendoNet from FractalNet, which shows a student-teacher effect. Specifically, the longest path in FractalNet can achieve a similar or even lower error rate compared to the whole net. To investigate the dynamic behavior of subnets, we test the error rate changes of subnets during the
training. We use Adam to train the CrescendoNet with the structure shown in Figure 8-2 on CIFAR10 for 450 epochs. Figure 8-4 illustrates the behavior of different path combinations during the training. It shows that the inference accuracy of the whole net grows simultaneously with all the subnets, which demonstrates the ensemble effect. Second, for any single path network, the performance grows with the depth. Like FractalNet, CrescendoNet also shows this behavior of the anytime classifier. However, the depth of paths in CrescendoNet increases linearly instead of exponentially, which enables a more smooth relationship between the time budget and the performance. Figure 8-5 uses an instance of CrescendoNet (128, 256, 512) to show the number of parameters, the computational cost (FLOPS), and the accuracy of different subnets. On average, the accuracy increases simultaneously and smoothly with more cost. With the anytime classification property, we could use the small subnetworks to give a rough but quick inference, then use larger subnetworks to achieve better accuracy. The anytime classifier is useful for time-critical applications, like perception tasks for self-driving vehicles. Figure 8-5 shows two slight accuracy drops after increasing the cost, e.g., from the path set \( \{1, 2, 4\} \) to \( \{2, 4\} \) and from the path set \( \{1, 2, 3\} \) to \( \{2, 3\} \). The possible reason is that the subnet, i.e., path \( \{1\} \) is under-trained and the same condition happens to other paths in the training process.

### 8.2.4 Related Work

Conventional deep CNNs, such as AlexNet [122] and VGG-19 [189], directly stacked the convolutional layers. However, the vanishing gradient problem makes it difficult to train and tune very deep CNN of conventional structures. Recently, stacking small convolutional blocks has become an important method to build deep CNNs. Introducing new building blocks becomes the key to improve the performance of deep CNN. Lin et al. [139] first introduced the NetworkInNetwork module which is a micro neural network using a multiple layer perceptron (MLP) for local modeling. Then, they piled the micro neural networks into a deep macro neural network.

Szegedy et al. [199] introduced a new building block called Inception, based on which they built GoogLeNet. Each Inception block has four branches of shallow CNNs, building by convolutional kernels with size \( 1 \times 1 \), \( 3 \times 3 \), \( 5 \times 5 \), and max-pooling with kernel size \( 3 \times 3 \). Such a multiple-branch scheme is used to extract diversified features while reducing the need for tuning the convolutional sizes. The main body of GoogLeNet has nine Inception blocks stacked each other. Stacking multiple-branch blocks can create an exponential combination of feed-forward paths. Such a structure combined
with the dropout technique can show an implicit ensemble effect [213, 193]. GoogLeNet was further improved with new blocks to more powerful models, such as Xception [32] and Inception-v4 [200]. To improve the scalability of GoogLeNet, Szegedy et al. [200] used convolution factorization and label-smoothing regularization in Inception-v4. In addition, Chollet [32] explicitly defined a depth-wise separable convolution module replacing Inception module.

Recently, Larsson et al. [125] introduced FractalNet built by stacked Fractal blocks, which are the combination of identical convolutional layers in a fractal expansion fashion. FractalNet showed that it is possible to train very deep neural network through the network architecture design. FractalNet also achieved deep supervision and student-teacher learning by the fractal architecture. However, the fractal expansion form increases the number of convolution layers and associated parameters exponentially. For example, the original FractalNet model with 21 layers has 38.6 million parameters, while a ResNet of depth 1001 with similar accuracy has only 10.2 million parameters [97]. Thus, the exponential expansion reduced the scalability of FractalNet.

Another successful idea in network architecture design is the use of skip-connections [88, 89, 97, 225, 221]. ResNet [88] used the identity mapping to short-connect stacked convolutional layers,
which allows the data to pass from a layer to its subsequent layers. With the identity mapping, it is possible to train a 1000-layer convolutional neural network. Huang et al. [97] recently proposed DenseNet with extremely residual connections. They connected each layer in the Dense block to every subsequent layer. DenseNet achieved the best performance on benchmark datasets so far. On the other hand, Highway networks [194] used skip-connections to adaptively infuse the input and output of traditional stacked neural network layers. Highway networks have helped to achieve high performance in language modeling and translation. Zagoruyko et al. [226] proposed DiracNets which implicitly use skip-connections to train very deep neural networks. The idea is to use Dirac weight parameterization when training the model. Due to the generality of Dirac weight parameterization, it can apply to many neural network architectures.

It is worth mentioning that the baseline CrescendoNet architecture (Figure 8-2), looks similar to one instance of Deeply fused nets (DFN) [217]. However, CrescendoNet and DFN are different in terms of the design pattern. DFN manually designs a few standalone feed-forward networks of different layers and then fuses the segments from different networks to build a single net. Note that DFN uses one fully connected layer for each branch, the base model has seven individual fully
connected layers. In contrast, CrescendoNet generates the whole architecture by the expansion rule, and its base model has only two sequent fully connected layers before the soft-max layer. Also, for CIFAR10 and CIFAR100 datasets with widely-used data augmentation scheme, CrescendoNet with 15 layers outperforms DFN with 50 layers by a large margin, which further demonstrates the difference between two models.

8.2.5 Discussion

CNN has shown excellent performance on image recognition tasks. However, it is still challenging to tune, modify, and design a CNN. We propose CrescendoNet, which has a simple convolutional neural network architecture without residual connections [88]. Crescendo block uses convolutional layers with same size $3 \times 3$ and joins feature maps from each branch by the averaging operation. The number of convolutional layers grows linearly in CrescendoNet while exponentially in FractalNet [125]. This leads to a significant reduction of computational complexity.

Even with much fewer layers and a more straightforward structure, CrescendoNet matches the performance of the original and most of the variants of ResNet on CIFAR10 and CIFAR100 classification tasks. Like FractalNet [125], we use dropout and drop-path as regularization mechanisms, which can train CrescendoNet to be an anytime classifier, namely, CrescendoNet can perform inference with any combination of the branches according to the latency requirements. Our experiments also demonstrated that CrescendoNet synergized well with Adam optimization, especially when the training data is sufficient. In other words, we can avoid scheduling the learning rate which is usually performed empirically for training existing CNN architectures.

CrescendoNet shows a different behavior from FractalNet in experiments with CIFAR10/100 and SVHN. In FractalNet [125], the longest path alone achieves the similar performance as the entire network, far better than other paths, which shows the student-teacher effect. The whole FractalNet except the longest path acts as a scaffold for the training and becomes dispensable later. On the other hand, CrescendoNet shows that the entire network significantly outperforms any set of it. This fact sheds light on exploring the mechanism which can improve the performance of deep CNNs by increasing the number of paths. Also, the implicit ensemble behavior of CrescendoNet enables its anytime classification property, which is useful for real-time and safety-critical classification tasks. Our future works may focus on extending the model to more complicated computer vision tasks, including object detection and segmentation.
8.3 Conclusion

In this chapter, we introduced two projects in Section 8.1 and Section 8.2 respectively.

For the first project, we evaluated the performance of our models using two cyberbullying datasets collected from Twitter and Formspring.me. The results of our experiment show that PCNN can achieve improved recall and precision compared to baseline convolutional neural networks. Based on PCNN [229], we then developed a mobile cyberbullying defense system called MCDefender [214], that can effectively detect and prevent cyberbullying in mobile social networks.

For the second project, we demonstrated that the proposed ConvNets [230] can achieve competitive performance with respect to state-of-the-art approaches while at a lower computational cost and using a simpler structure.

Acknowledgement

Thanks to Dr. Feng Luo, who supervised the author from 2016 to 2018 in the School of Computing of Clemson University. The two research projects in which the author participated in Dr. Luo’s lab have resulted in three publications [229, 214, 230].
Appendices
Appendix A  Velocity Regression Models

We propose a neural network, also known as Multilayer Perceptron (MLP), for velocity prediction given the features above. For comparison, we also test a linear regression model using the same feature sets.

The structure of the neural network model is described in Table 9.

Table 9: Neural network structure for velocity regression.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Layer type</th>
<th>Features out</th>
<th>Kernel size</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Input</td>
<td>21</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B1</td>
<td>Batch Normalization</td>
<td>21</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>F1</td>
<td>Fully Connected</td>
<td>30</td>
<td>30</td>
<td>ReLU</td>
</tr>
<tr>
<td>D1</td>
<td>Dropout</td>
<td>30</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B2</td>
<td>Batch Normalization</td>
<td>30</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>F2</td>
<td>Fully Connected</td>
<td>100</td>
<td>100</td>
<td>ReLU</td>
</tr>
<tr>
<td>D2</td>
<td>Dropout</td>
<td>100</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B3</td>
<td>Batch Normalization</td>
<td>100</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>F3</td>
<td>Fully Connected</td>
<td>100</td>
<td>100</td>
<td>ReLU</td>
</tr>
<tr>
<td>Out</td>
<td>Fully Connected</td>
<td>2</td>
<td>2</td>
<td>None</td>
</tr>
</tbody>
</table>

The input layer denotes the feature vector. The value 21 in the first row of Table 9 denotes the number of features. When we use a subset of the features, we need to change this value accordingly. The numbers in the features-out column denote the size of output features for the layer.

The body of the network includes two combinations of the batch normalization, the fully connected layer, and the dropout layer (see Section 4.1 for more details). Then another batch normalization and two fully connected layers are used. All the fully connected layers except the last one are followed by Rectifier Linear Unit (ReLU) [159] as the activation function.

Batch normalization [102] shifts the mean and rescales the standard deviation of feature vectors in each batch. It helps to avoid gradient vanishing and exploding problems during the training. The feature vectors are first normalized by Equation 12, Equation 13, and Equation 14.
\[ \mu_B = \frac{1}{n_B} \sum_{i=1}^{n_B} X^{(i)} \]  

(12)

where \( X^{(i)} \) is the feature vector \( i \) in the batch. \( \mu_B \) is the vector of the mean of input for the entire batch. \( n_B \) is the number of samples in each batch.

\[ \sigma^2_B = \frac{1}{n_B} \sum_{i=1}^{n_B} (X^{(i)} - \mu_B)^2 \]  

(13)

\( \sigma_B \) is the vector of the standard deviation of the feature vector for the entire batch.

\[ \hat{X}^{(i)} = \frac{X^{(i)} - \mu_B}{\sqrt{\sigma^2_B}} \]  

(14)

where \( \hat{X}^{(i)} \) is the zero-mean and normalized feature vector \( i \) in the batch.

\[ Z^{(i)} = \gamma \otimes \hat{X}^{(i)} + \beta \]  

(15)

where \( \gamma \) is a trainable scale parameter, and \( \beta \) is a trainable shift parameter. The operation \( \otimes \) denotes element-wise multiplication. The notation \( Z^{(i)} \) is the output vector of the batch normalization layer.

The fully connected layer in our model consists of a number of Perceptrons [182] using the Rectifier Linear Unit (ReLU) [159] as the activation function. Each operation is defined by an affine transformation followed by an activation function.

The Rectifier Linear Unit (ReLU) is defined by Equation 16.

\[ ReLU(u) = \max(0, u) \]  

(16)

where \( u \) is an input value.

The Perceptron can be defined by Equation 17.

\[ z = ReLU(w \cdot X + b) \]  

(17)

where \( w \) is a trainable weight vector, \( b \) is a trainable bias. \( X \) is the input feature vector. \( z \) is the output of the Perceptron. Multiple Perceptrons generate a feature vector with corresponding
dimensions, which is the output of a fully connected layer. The number of Perceptrons for fully
connected layers is specified in the kernel size column in Table 9.

We use the mean squared error (MSE) as a term in the loss functions for both regression
models. We use $L2$ regularization in the loss function to avoid over-fitting for the linear model, as
shown in Equation 19. In addition, we use the dropout layers with 20% drop rate [193], described in
Section 4.1, for the neural network model. We use the Adam optimizer [118] to train the Multilayer
Perceptron by the similar way we described in Section 4.1.

The loss function for the Multilayer Perceptron (MLP) is defined by Equation 18.

$$\text{Loss}_{\text{MLP}} = \|y - f_{\text{MLP}}(X; \theta)\|^2_2$$

(18)

where $y$ is the ground truth, i.e., the velocity from the annotation. $f_{\text{MLP}}(X; \theta)$ is the prediction
from the Multilayer Perceptron model with $X$ as the input feature vector and $\theta$ as the parameters of
the model. The operation $\| \cdot \|^2_2$ means the $L2$ norm.

The loss function for the linear regression model (LR) is defined in Equation 19.

$$\text{Loss}_{\text{LR}} = \|y - wX\|^2_2 + \|w\|^2_2$$

(19)

where $y$ is the ground truth, i.e., the velocity from the annotation. $X$ is the feature vector and $w$ is
the weights of the linear regression model.
Bibliography


123


