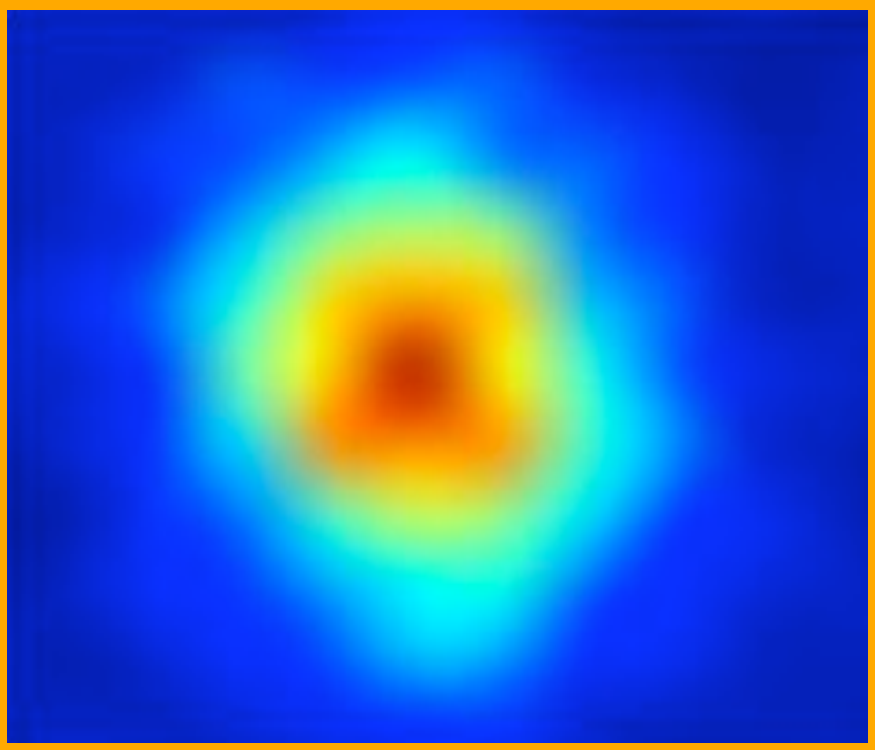
Intelligent Acquisition and Learning of Fluorescence Microscope Data Models



Charles Jackson

bimagicLab
Center for Bioimage Informatics
Dept. of Biomedical Engineering
Carnegie Mellon University

Intelligent Acquisition and Learning of Fluorescence Microscope Data Models

Charles Jackson

Advisor: Prof. Jelena Kovačević

Department of Biomedical Engineering
Carnegie Mellon University, Pittsburgh, PA 15213

Thesis Manuscript

Submitted in partial fulfillment of the requirements towards the Ph.D. degree awarded by the
Department of Biomedical Engineering, Carnegie Inst. of Tech., Carnegie Mellon University.

Thesis Committee Members

Jelena Kovačević (Advisor)
Carnegie Mellon University

Frederick Lanni
Carnegie Mellon University

Robert F. Murphy
Carnegie Mellon University

Gustavo K. Rohde
Carnegie Mellon University

To my beloved parents,
for instilling in me the confidence,
to pursue my interests and dreams.

Abstract

This thesis presents a new acquisition framework that models fluorescence microscope data during acquisition, and uses these learned models to intelligently guide future acquisitions. This framework results in significant time savings, as well as in reducing the photobleaching and phototoxicity incurred during acquisition.

Fluorescence microscopy is a popular tool for live-cell imaging, and in recent years, there has been an explosion in the amount of data acquired with this technique. Visual inspection of this data is time-consuming and not reproducible, motivating the goal of automated image analysis. Furthermore, we would ideally like to acquire all types of cells under all conditions, but standard acquisition methods are too time-consuming to achieve this feat. This work proposes to address these problems with a new acquisition framework that builds models of the data while it is being acquired, and uses these models to carry out intelligent acquisition. The goal is to reduce total acquisition time by identifying and acquiring only the data that is necessary for building the model, as well as to acquire in a way that reduces photobleaching and phototoxicity—two fundamental limitations associated with fluorescence microscopy.

We evaluate the framework experimentally on synthetic and real data. First, we present a possible method to build models of a single object within a cell, of multiple objects in a cell, and of a population of cells. Then, we present intelligent acquisition algorithms to determine where to acquire in a cell, when to acquire in a cell, when to stop acquiring from a cell, and how many cells to acquire from a population. We show that the combination of model building and intelligent acquisition results in time savings, reduced photobleaching, and reduced phototoxicity, without loss of accuracy.

Acknowledgments

This thesis required much more than four years of study to complete. It could not exist without the support, technical assistance, and friendship, of countless others. These few pages of acknowledgement do not even begin to do those people justice.

I would like to express the deepest appreciation for my advisor, Jelena Kovačević. Her outstanding support has been the foundation of a fun and rewarding four years. When I first came to Pittsburgh, we had only exchanged one phone call, but I quickly became aware of my good fortune in an advisor, and I cannot imagine doing a Ph.D. without her. Her flexible approach gave me the freedom to explore different approaches and ideas, and to discover the research path best suited to me. Her help ranged from technical assistance to critical paper reviews (not least, this thesis!) to general life and career advice. In addition, I will always have fond memories of our lab dinners, group meetings, insightful discussions, and of course, 'Dance Dance Revolution' at the CBI holiday parties!

I would like to thank my committee members for the big time investment that they have been willing to make. Robert Murphy first proposed the idea for intelligent acquisition of fluorescence microscope images. He provided the data set used in this work, and this research draws heavily on the prior work of his group. As a co-author on all of my publications, his editing and feedback have been invaluable. I would also like to thank him for his help in my application to the Machine Learning department, and finally, to acknowledge his lab party as my first-ever karaoke performance!

Gustavo Rohde brought a new and insightful perspective to every discussion, continually pushing this work to a higher level. His office door was always open, and he was always been willing to bounce around ideas.

Frederick Lanni, too, was ever-ready to help. His extensive knowledge—

biological and otherwise—never failed to expand my horizons, and his infectious enthusiasm invigorates any discussion.

I would like to thank Estelle Glory, a postdoc in CBI, for all of her patient explanations and brainstorming sessions, as well as for being instrumental in finding and processing the data set used in this work. It was a pleasure to have her as a neighbor, and I will miss our discussions.

I would like to thank my fellow teammates from CBI, both past and present, for their company and valuable input. In particular, I wish to acknowledge my predecessors Amina Chebira and Gowri Srinivasa, for showing me the light at the end of the tunnel. They welcomed me from the day I arrived and remained great friends throughout. My sadness at seeing these two depart helped spur me to finish myself. Also, I want to give Ramu Bhagavatula the credit for the formatting of the figures in this thesis.

Many others in Pittsburgh have contributed to this work by virtue of their friendship, helping to keep me sane during the more trying times: Richard Pelikan, with our weekly squash games; the climbing gang—Stefanie Hassel, Phil Tilman, Nel de Jong, and others—for our weekly rock-climbing; Steve Sun, Paul Glass, Pablo Hennings, and Justin Newberg, for the chances to muse about life and graduate school over beers at the Sharp Edge; Mike Crawford, for having an accent that reminds me of home; the many others, whom I am afraid to list for fear of leaving one out, but whose friendship means so much.

An extra special thanks to Sally Hollister, who has been a steady source of support, enouragement, and fun, and has made Pittsburgh a much brighter place. She is probably as happy as I am that this thesis is finally finished, and that I can be a normal person again.

To friends from afar, Nick Butcher and William Connor, my life would not be the same without your friendship.

Most of all, I wish to thank my parents and my family, for always being there, and for the support, encouragement, and opportunities that they have given me. It goes without saying that I would not have made it here without them, and I will always owe them a debt of gratitude.

Contents

List of Figures

List of Tables

1

Thesis Contributions and Outline

Fluorescence microscopy is a popular tool for live-cell imaging. In recent years, as the trend in biology has moved more and more towards high-throughput applications, there has been an explosion in the amount of data being acquired and analyzed with this technique. The first bottleneck is visual inspection; it is time-consuming, subjective, and not reproducible, motivating the goal of automated analysis. A second bottleneck is the acquisition process itself. Ideally, we would like to acquire all types of cells under all conditions, repeating multiple times to assess the variance in the resulting data. However, we are limited by the prohibitive time required for this feat, as well as photobleaching and phototoxicity in the acquisition process.

To address these issues, we build automated models of fluorescence microscope data. These models, which may be application-specific, can describe either a single cell or a whole class of cells. Their goal is to capture all information required for the end-application, thus removing the need for visual inspection. What makes our proposal unique is that we do not build these models as a post-processing step. Instead, we build them while we are acquiring the data. This allows us to use the models to guide future acquisitions, in a process which we call intelligent acquisition. Our framework is summarized in Fig. 1.1.

When we are building models of a single cell, intelligent acquisition is helpful in two ways: First, we can automatically determine when to stop acquiring the cell. With naive acquisition, we cannot

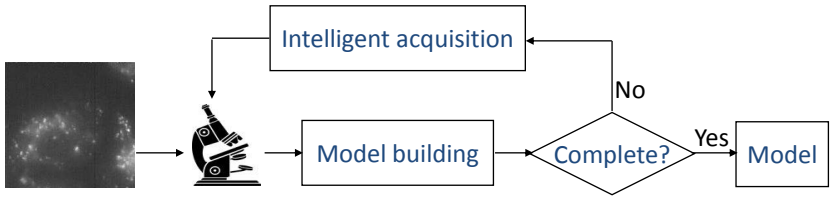


Figure 1.1: A diagram of the proposed research. The model-building module constructs a model from the microscope data, and the intelligent acquisition module determines which acquisitions to make to efficiently improve on this model.

usually determine in advance how many frames to acquire, and thus we risk stopping too early and not gaining sufficient information, or acquiring more frames than necessary and wasting time. Second, intelligent acquisition can optimize how we acquire. It can choose where to acquire, when to acquire, at which resolution to acquire, and even the intensity of the excitation light and the length of each exposure. This capability is useful when we consider two fundamental limitations of fluorescence microscopy: photobleaching and phototoxicity. In fluorescence microscopy, images are acquired by shining excitation light onto the cell to activate fluorescence. However, this excitation light damages both the fluorescent signal (photobleaching) and the cell itself (phototoxicity). Such effects limit the duration over which we can view the cellular process. Hence, there is strong motivation to acquire in a way that maximizes information gain relative to photodamage, resulting in the most accurate model possible when the fluorescent signal or cellular function are destroyed.

When we wish to build models of a whole class of cells, the role of intelligent acquisition takes on even greater importance. In addition to optimizing the acquisition of each individual cell, we must now determine how many cells to acquire from that class. We wish to acquire enough cells to characterize the class well, but we do not want to waste time by acquiring more than we have to. In addition, we must determine how many frames to acquire from a cell before moving onto the next cell. Generally speaking, if we determine that a cell is similar to previous cells of that class, we want to stop acquiring and move onto a different cell that contains new information.

All of these intelligent acquisition algorithms hinge around the

ability to build models of what we are acquiring. It is the recognition that the end goal of acquisition is a model rather than an image that lets us choose which data is most relevant to our task. Furthermore, the models give an estimate of the future state of a cell, which is critical to the intelligent acquisition module's ability to choose how to acquire.

Hence, the overall goal of this work is to:

Develop a new acquisition framework that models fluorescence microscope data during acquisition and uses these learned models to intelligently guide future acquisitions.

We evaluate this framework on both a synthetic data set and a real data set. For the synthetic data set, we look at the simplified case where we have a single object moving in a cell. We present a method which, given a reasonable model, learns the model parameters of this object. We then validate our method by comparing the output to the ground truth model used to construct the data set. For real data, we develop a method to model a cell containing multiple objects. Additionally, we describe how to model a whole class of cells. We validate these models in two ways: first, by predicting the locations and types of objects in a frame prior to acquiring that frame; second, by classifying a cell into its appropriate class.

We then describe a set of intelligent acquisition algorithms to learn model parameters efficiently. For cell models, these intelligent acquisition algorithms dictate where to acquire, when to acquire, and when to stop acquiring. For class models, the algorithms dictate when to stop acquiring new cells from that class, and when to stop acquiring each individual class. We test our methods by acquiring a cell or class intelligently and measuring the accuracy of our resulting model. We then acquire the same amount of data from that cell or class using a naive algorithm, and show that the accuracy of the resulting model is lower than if we had acquired intelligently.

In this thesis, *model building* refers to estimating the parameters for a given class of models, which will typically depend on the end-application. To demonstrate and evaluate our acquisition framework, we define a class of models and develop an assoicated model-building procedure. We then show how intelligent acquisition algorithms can be used to build such models more efficiently. Although the specific

algorithms are tied to the class of models being used, the general principles behind them are applicable to a wide range of modeling scenarios.

1.1 Thesis Contributions

1. **Framework.** We present a new acquisition framework for the intelligent acquisition and learning of fluorescence microscope data sets, and demonstrate its applicability on synthetic and real data.

2. **Model Building.** We present example model-building methods for both a single cell and a class of cells. We validate their accuracy on synthetic and real data. As part of this, we apply the models to classification of 3T3 cells, and demonstrate a slightly higher accuracy than previous results on this data set.

3. **Intelligent Acquisition.** We describe a set of algorithms for intelligent acquisition, and show that we can build more accurate models when data is acquired intelligently rather than naively. The specific operations we cover include:

    - Where to acquire in a frame
    - When to acquire frames
    - When to stop acquiring frames
    - How many cells to acquire

1.2 Thesis Outline

The thesis is divided into three main parts. In the first part, we provide the background necessary both for understanding this work, as well as placing it in a larger context. In the second part, we explain our model-building procedure, along with methods to validate its accuracy both on synthetic and real data. The third and final part presents our intelligent acquisition algorithms and validates them experimentally. Parts II and III are original work, with the exception of the object detection and feature extraction methods in Section 5.2. We detail the outline of this thesis as follows.

- Part I consists of Chapters 2 and 3, and presents all the necessary background for our work as well as an overview of related methods. Chapter 2 overviews fluorescence microscopy, the mode of acquisition at which this work is targeted. This includes a discussion of photobleaching and phototoxicity, which are two of the drawbacks of fluorescence microscopy that helped motivate this work. Chapter 3 provides background on model estimation and particle filters, both essential for our model-building methods. This is followed by an overview of classification and some common algorithms, as well as a discussion of active learning. Finally, we survey related work both in model building and intelligent acquisition.

- Part II consists of Chapters 4 and 5, and describes the model-building algorithms that we use to evaluate our proposed acquisition framework. Chapter 4 explains how we build models for the case of a single object moving in a cell. The presented method can be used for a wide range of motion models, and we validate it on synthetic data. Chapter 5 explains how we build models for the case of multiple objects moving in a cell, and validates this on real data. In addition to predicting the locations and types of objects in future frames, we also show how these models can be used to classify, and compare the results with another classification method.

- Part III consists of Chapters 6 through 9. Each chapter covers a different aspect of intelligent acquisition. Chapter 6 discusses which pixels to acquire in each frame. Chapter 7 discusses when to acquire each frame. Chapter 8 discusses when to stop acquiring frames. Chapter 9 discusses how many cells to acquire. We validate the intelligent acquisition algorithms from Chapters 6 and 7 on synthetic data, those from Chapter 8 on both synthetic and real data, and those from Chapter 9 on real data.

Part I

Background

2

Fluorescence Microscopy

Fluorescence microscopy is a rapidly expanding microscopy technique, and is especially popular for live cell imaging. Because cellular components are colorless, they are difficult to view unless they are stained. In fluorescence microscopy, subcellular components of interest are labeled with fluorescent components called fluorophores. These fluorophores absorb excitation light at a certain wavelength, and subsequently emit light at a longer wavelength. By filtering the emission light from the excitation light prior to detection, the locations of the fluorophores and the components of interest are revealed.

An understanding of fluorescence microscopy is important for this thesis. In this chapter, we provide details on the physics of fluorescence, fluorescent probes, types of microscopes, and the associated problems of photobleaching and phototoxicity.

2.1 Physics of Fluorescence

A fluorophore is a functional group in a molecule that absorbs excitation light at a specific frequency and subsequently emits light at a lower (also specific) frequency. As shown in Fig. 2.1, excitation takes places when a photon of a specific energy collides with an electron in an atom and excites it to a higher energy level. The energy of this incoming photon must be close to the required energy for this electron transition. If the photon is of higher energy, then it is usually converted into vibrational and rotational energy instead. If the photon is of lower energy, it has insufficient energy to cause the transition. Photon energy is proportional to the frequency of the light,

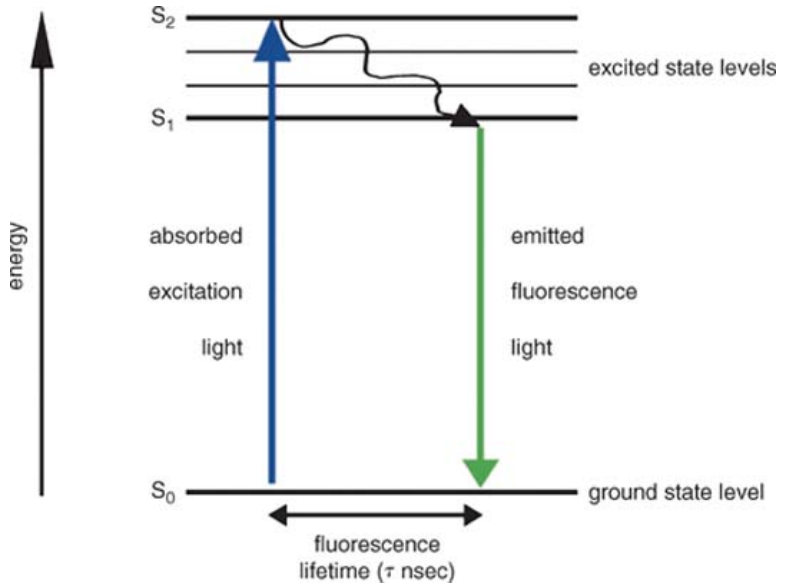


Figure 2.1: Excitation and emission of a fluorophore. An incoming photon collides with an electron and promotes it to a higher energy state. This electron subsequently relaxes back to its ground state, causing the emission of a photon. Some vibrational energy is lost, thus the emitted photon is of lower energy [http://media.wiley.com/CurrentProtocols/CY/cy1210/cy1210-fig-0001-1-full.jpg].

and the excitation light must be in the appropriate frequency range, typically in the blue or ultraviolet part of the spectrum.

Fluorescence occurs when the excited electron relaxes back to its ground state and emits a photon. Because vibrational energy is lost during this process, the emitted photon is of a lower energy than the photon that was originally absorbed, and hence the emitted light is of lower frequency than the excitation light. This shift in frequency is called Stoke's shift; it allows the emitted light to be separated from the excitation light. Such a separation is important because the excitation light may be up to a million times brighter than the emitted light. Hence, it is desirable to have the Stoke's shift be as large as possible. The exact Stoke's shift, as well as the frequencies of the excitation and emitted light, depend on the type of fluorophore.

The time lag between photon absorption and its subsequent emis-

sion is extremely small (on the order of nanoseconds). The sensitivity is such that as few as 50 fluorophores per cubic micrometer can be detected [44]. We can increase fluorescence, and thus improve contrast, by increasing the intensity of the excitation light. However, this may result in *saturation*, where all the fluorophores within the focal volume become excited. We can also increase fluorescence by lengthening the exposure time, but, as we discuss in Section 2.4, this could apply too much excitation light to the specimen.

We now discuss ways of introducing fluorescence into a specimen, thereby allowing fluorescence microscopy to be used.

2.2 Fluorescent Probes

Although some specimens exhibit *autofluorescence* (fluorescence without added fluorophores), techniques to artificially introduce fluorophores into a sample have greatly increased the value of fluorescence microscopy. When we refer to *labeling* components of interest, we mean the introduction of fluorophores specifically configured to attach to these components. One way to achieve target-specific labeling is through *immunoflourescence*. Here, a protein of interest is labeled by a fluorophore attached to an antibody of that protein. Unfortunately, this method requires that the cells are first fixed (killed) and permeabilized, meaning that immunofluorescence cannot be used for live cell imaging.

For live-cell imaging, we can use green fluorescent protein (GFP). This protein was originally isolated from the jellyfish *Aequorea victoria*, which fluoresces green when exposed to blue light. Its advantage is that it is not species specific, and so can be configured, from the gene level, to bind to almost any target protein with no observed effect on cell growth or function.

Jarvik *et al.* introduced Central Dogma (CD)-tagging [29], a method that can randomly label and study proteins in living cells. This method inserts a specially designed DNA sequence (the CD cassette), into genomic DNA. If the insertion occurs in the intron of an active gene and in the correct orientation, the protein function of that gene will be annotated by that of the CD cassette. By putting GFP in the CD cassette, the tagged proteins become fluorescent. Hence, CD-tagging can be used to create a data set in which different proteins are fluorescently labeled. We use such a data set in Chapter

5.

2.3 Types of Microscopes

We show a generic fluorescence microscope in Fig. 2.2. Its main task is to shine excitation light on the specimen to excite the fluorophores, and then to stop this excitation light from reaching the image capture device. The key component to achieve this is the dichromatic mirror, which is designed to efficiently reflect light above a certain cutoff frequency, while transmitting light below that frequency. The mirror is chosen such that the cutoff frequency lies between the frequencies of the excitation light and the emission light. Accordingly, excitation light passes through the excitation filter (which ensures that only a specific frequency hits the mirror) and this light is then reflected downwards onto the specimen. The emission light that is then released by the fluorophores in the specimen is able to pass through the dichromatic mirror (because it is below the cutoff frequency), but any scattered excitation light is reflected away. In practice, some of the excitation light still passes through, and the barrier filter is needed to block this remaining excitation light. The fluorescence emission can then be projected onto an image capture device.

In a *widefield microscope*, the whole image plane is excited all at once, and all the emission from that plane is collected simultaneously (or viewed by eye). The problem with widefield microscopy, especially for thick specimens, is that fluorescence from image planes that are out of focus can interfere with the primary fluorescence. Although image quality can be improved using *deconvolution*, a more direct method to reduce out-of-focus fluorescence is to use a confocal microscope.

In a laser-scanning confocal microscope (LCSM), images are acquired line-by-line, pixel-by-pixel. This is done by moving a single-point laser beam across the sample (or, alternatively, moving the sample), exciting one pixel at a time. As shown in Fig. 2.3, out-of-focus fluorescence is eliminated by the use of a small pinhole that ensures only fluorescence from the focal plane reaches the detector. As a result, an LCSM provides better resolution than a widefield microscope, but acquisition speed is slower. Because each pixel is excited individually, we can no longer view the resulting image by eye, and thus we must capture the pixel value with an image capture

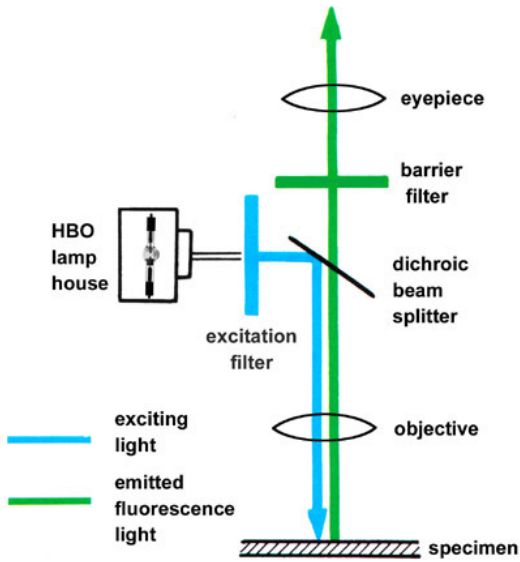


Figure 2.2: Generic fluorescence microscope. The excitation light is reflected by the dichromatic mirror onto the specimen. The emitted light is of a lower frequency and able to pass through the mirror, thus separating it from scattered excitation light [http://web.uvic.ca/ail/techniques/epi-fluorescence.html].

device such as a photomultpilier tube.

An alternative confocal microscope is the spinning disk confocal microscope. This acquires multiple pixels simultaneously using a spinning disk that contains multiple sets of pinholes and a rotating array of lenses that focuses the excitation light on the corresponding pixels. It has a much faster acquisition time than the LSCM, but spatial resolution is lower.

For all of these microscopes, we obtain 3D images by adjusting the plane of focus, and thus acquiring a set of 2D images at different heights. Because height is usually denoted by the z-dimension, a 2D image at a particular height is often called a *z-slice*, and a set of such z-slices (a 3D image) is often called a *z-stack*.

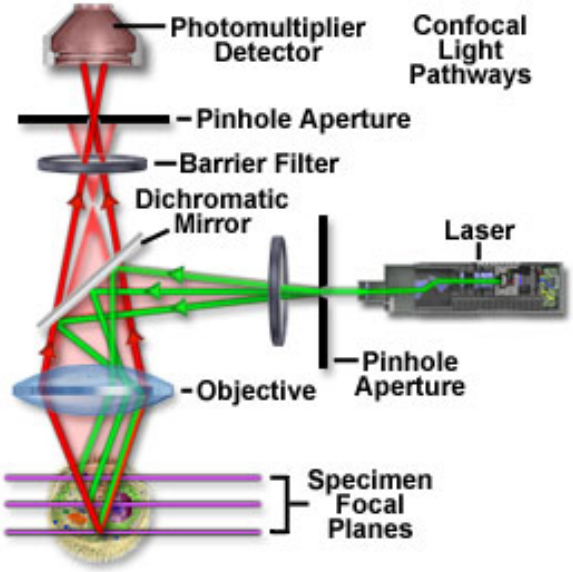


Figure 2.3: A confocal microscope schematic. The small pinhole aperture ensures that only fluorescence from the focal plane reaches the detector [http://www.olympusconfocal.com/theory/index.html].

2.4 Photobleaching and Phototoxicity

Photobleaching and phototoxicity can occur whenever a fluorophore is subjected to excitation light. Normally, fluorophores are excited from the ground state to an excited singlet state, and then relax back to the ground state. However, there is a certain probability that they will instead move from an excited singlet state to an excited triplet state. While in this state, they can react with molecular oxygen and undergo an irreversible covalent modification that destroys their ability to continue fluorescing. As a result, the fluorescent signal fades over time, as shown in Fig. 2.4. Some techniques, such as fluorescence recovery after photobleaching, exploit photobleaching to gain useful information. However, in general, photobleaching limits the duration over which we can acquire.

Phototoxicity also occurs when a fluorophore in the excited triplet state reacts with molecular oxygen. This reaction releases a free radical, which can damage proteins, nucleic acids, and other cellular components [30]. Phototoxicity may only be obvious when it is suffi-

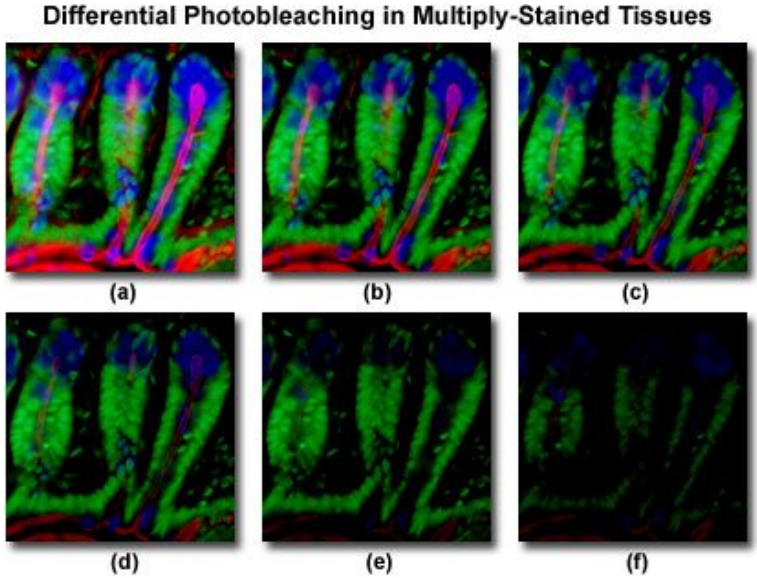


Figure 2.4: Photobleaching. This shows pictures of the same specimen at 2-minute intervals. The photobleaching is clearly visible during this time [http://micro.magnet.fsu.edu/primer/java/fluorescence/photo bleaching/].

cient to cause cell death, but prior to this, there may be more subtle effects on cellular function.

Both photobleaching and phototoxicity can be reduced by decreasing the amount of excitation light applied to the specimen, whether by reducing the intensity of that light, or by reducing the duration of exposure. Unfortunately, this also reduces the strength of our fluorescent signal. A primary motivation of this work is to gain as much information as possible from each application of excitation light, thereby ensuring that we learn as much as possible before photobleaching and phototoxicity force us to cease acquisition.

3

Model Building and Intelligent Acquisition

In this chapter, we provide an introduction to some basic model estimation techniques, along with the more advanced technique of particle filtering (used heavily in Chapter 4). We then provide an overview of classification, as one of the primary validation methods in this thesis, as well as an introduction to active learning, the topic in the machine learning literature that most overlaps with the concept of intelligent acquisition. Finally, we present related work both for model building and for intelligent acquisition.

3.1 Model Estimation

In this section, we discuss how to estimate a model given some observed data. We assume that we know the form of the model, but that we must estimate its parameters. In particular, we discuss two common methods for model estimation: the maximum likelihood estimator and Bayes estimators. We then discuss information measures that help evaluate the quality of our estimation.

3.1.1 Maximum Likelihood Estimator

The maximum likelihood estimator chooses the model parameters that make the observed data more likely than they would be with any other choice of model parameters. If $\mathbf{X} = \{x_1, ..., x_N\}$ is the observed data, then $\ell(\mathbf{m})$, the likelihood of a vector of model parameters $\mathbf{m}$,

is defined by $\ell(\mathbf{m}) = P(\mathbf{X}|\mathbf{m})$. For example, suppose our model is a Bernoulli distribution, which means that a random variable x_i takes the value 1 with probability m, and 0 with probability $(1 - m)$ (in this instance, m is a scalar rather than a vector, because the model has only one parameter). The likelihood of m is then given by:

$$\ell(m) = m^{S_N}(1 - m)^{N - S_N}, \tag{3.1}$$

where:

$$S_N = \sum_{i=1}^{N} x_i. \tag{3.2}$$

Our goal is to find m such that this likelihood is maximized.

Instead of maximizing the likelihood directly, it is often more convenient to maximize the log-likelihood. The logarithm is a monotonic transformation, and so the maximum of the log-likelihood occurs at the same point as that of the likelihood. Taking the logarithm of each side in (3.1), we get:

$$\log(\ell(m)) = S_N \log(m) + (N - S_N) \log(1 - m) \tag{3.3}$$

To maximize this expression, we must set its derivative to zero. Differentiating each side, we get:

$$\frac{\partial \log(\ell(m))}{\partial m} = \frac{S_N}{m} - \frac{N - S_N}{1 - m}, \tag{3.4}$$

which is equal to zero at the maximum likelihood value, $\hat{m}$:

$$\hat{m} = \frac{S_N}{N}. \tag{3.5}$$

Hence, the maximum likelihood estimator for a Bernoulli distribution is simply the proportion of observations that are 1.

3.1.2 Bayes Estimators

Bayesian inference is an approach to model estimation that differs from maximum likelihood estimation in that it also considers the prior distribution on the model parameters, $\pi(\mathbf{m})$. The prior distribution reflects our belief about $\mathbf{m}$ *before* we have observed any data, whereas our belief *after* observing data is described by the posterior distribution. The relationship between the prior probability, the

posterior probability, and the likelihood function, can be expressed with Bayes' theorem:

$$p(\mathbf{m}|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{m})\pi(\mathbf{m})}{p(\mathbf{X})}, \tag{3.6}$$

where $p(\mathbf{m}|\mathbf{X})$ is the posterior probability of $\mathbf{m}$, $p(\mathbf{X}|\mathbf{m})$ is equivalent to the likelihood $\ell(\mathbf{m})$ from Section 3.1.1, and the marginal probability of $\mathbf{X}$, $p(\mathbf{X})$, acts as a normalizing constant.

A Bayes estimator for $\mathbf{m}$ tries to minimize the posterior expected value of a cost function (also known as the loss function, or the Bayes risk). This cost function, $C(\mathbf{m}, \hat{\mathbf{m}})$, associates a cost with estimating the true parameter vector $\mathbf{m}$ with the estimated parameter vector $\hat{\mathbf{m}}$. The Bayes estimator then seeks to minimize $E(C(\mathbf{m}, \hat{\mathbf{m}}))$, where the expectation is taken over the posterior distribution of $\mathbf{m}$. A common choice of cost function is the mean square error, $C(\mathbf{m}, \hat{\mathbf{m}}) = (\mathbf{m}-\hat{\mathbf{m}})(\mathbf{m}-\hat{\mathbf{m}})^{\mathrm{T}}$, in which case the best estimator of $\mathbf{m}$ is the mean of the posterior distribution.

Taking the example of a Bernoulli distribution from Section 3.1.1 and assuming a uniform prior (m is equally likely to occur anywhere between 0 and 1), this estimator gives:

$$\hat{m} = \frac{S_N + 1}{N + 2}. \tag{3.7}$$

We can see that this lies between the maximum likelihood estimate (S_N/N) and the prior mean (0.5), and that it moves closer to the maximum likelihood estimate as the number of observations increases. It is a general property that Bayesian and maximum likelihood results converge with a large enough data set. In this case, the convergence happens quickly because we chose a *weak prior*—one which did not favor any choice of m over any other. However, the convergence would be slower had we chosen a *strong prior*—one that significantly favored some choices of m over others.

3.1.3 Information Measures

We now discuss two information measures that help evaluate the quality of model estimation: entropy and the Kullback-Leibler divergence.

Entropy

Entropy is a measure of the uncertainty associated with a random variable. If $p(x)$ is the distribution of a discrete random variable x, then the entropy $H(x)$ is defined as:

$$H(x) = -\sum_x p(x) \log p(x). \tag{3.8}$$

We can see that this is the negative of the expected value of $\log p(x)$, and thus it gives an indication of how well we can predict future values of x if we know the true distribution $p(x)$. Such predictions are hardest when all events are equally probable, and thus the highest possible entropy (highest uncertainty) is obtained with a uniform probability distribution.

If x is a continuous random variable, we can use the *differential entropy*, defined as:

$$H(x) = -\int_x p(x) \log p(x). \tag{3.9}$$

Entropy is an important concept in model estimation because it provides an upper bound on how well we can predict future observations even when the true distribution of those observations are known. Specifically, it is the negative of the expected log-probability of future observations.

Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence is an assymetric measure of the difference between two probability distributions $p(x)$ and $q(x)$, defined as:

$$KL(p||q) = -\int p(x) \log q(x) dx - (-\int p(x) \log p(x) dx) \tag{3.10}$$

$$= -\int p(x) \log(\frac{q(x)}{p(x)}) dx. \tag{3.11}$$

The KL-divergence equals zero if and only if $p(x) = q(x)$.

If x is governed by the true probability distribution $p(x)$, the KL-divergence measures the cost of predicting x using $q(x)$ instead of $p(x)$. Specifically, it measures the resulting decrease in the expected log-probability of those predictions.

3.2 Particle Filters

Having introduced the general concepts of model estimation, we now discuss a specific simulation-based technique: particle filters. These filters, also known as sequential Monte Carlo methods [17], require neither linearity nor Gaussian noise, and their performance often exceeds that of the extended Kalman filter (a nonlinear version of the Kalman filter [20]). In this section we review the state-space approach upon which particle filters are based, and then outline the estimation process.

3.2.1 State-Space Approach

By using the state-space approach, we assume that our system has a true underlying state, and that our observations give some interpretation of this state [48]. The state-space equations in their most general form are given by:

$$\mathbf{x}_{t+1} = g(t, \mathbf{x}_t, \boldsymbol{\nu}_t), \tag{3.12}$$

$$\mathbf{z}_t = h(t, \mathbf{x}_t, \mathbf{w}_t), \tag{3.13}$$

where $\mathbf{x}_t$ is the state vector—the set of state variables that can represent the entire state of the system at time t. For example, if we were modeling objects within a cell, then the state vector could include the positions, shapes, and motion models of these objects. The vector $\mathbf{z}_t$ refers to the observed vector. The first equation, (3.12), is the state-update equation, with the function g describing how the state of the system evolves with time. This includes a noise term $\boldsymbol{\nu}_t$ to reflect that the model of the state evolution will not be exact, and that the actual state will differ from the predicted state. In the measurement equation, (3.13), h relates the system state to the observed measurement at time t, with $\mathbf{w}_t$ allowing for measurement noise.

The reader may be more familiar with the linear forms of these equations:

$$\mathbf{x}_{t+1} = \mathbf{G}_t\mathbf{x}_t + \boldsymbol{\nu}_t, \tag{3.14}$$

$$\mathbf{z}_t = \mathbf{H}_t\mathbf{x}_t + \mathbf{w}_t, \tag{3.15}$$

which can be used when the dynamics of the system are known and linear.

3.2.2 Estimation Process

The basic approach in particle filtering is to represent the posterior distribution of the state vector using a set of sample states known as particles. We begin by generating N particles drawn from the prior distribution of the state space. At each time step, we propagate the particles forward according to the state update equation, (3.12). As observations are made, those particles consistent with the observations are given high weights, and inconsistent particles are given low weights. To avoid a concentration of particles in low-likelihood regions, a resampling procedure then duplicates those particles with high weights and eliminates those with low weights. There are many ways of doing this [34, 3, 39], and in this work we use sampling-importance-resampling [20]. When an observation is made, we assign each particle a weight according to the posterior probability of the particle given the observation. We then resample the particles with probabilities proportional to their weights—that is, we draw N particles with replacement from the current particle set.

The main limitation of particle filters is that high accuracy requires high computation time. This is especially true when the problem is high-dimensional or when there are unknown static parameters in the state space. Because the latter situation occurs frequently in our work, we draw special attention to an enhancement given by Gordon *et al.* [20], known as roughening, which treats these unknown static parameters as though they were dynamic, by adding a small amount of random noise to them at each iteration of the algorithm. The random noise makes some particles move closer to the truth, and others further away. Those that move closer are more likely to agree with future observations, and thus more likely to be duplicated. Hence, the particles eventually converge on the true parameter values.

Particle filters are attractive because they make few assumptions about the dynamics of the state space model and they are easy to implement even for complex models [1]. We do not need to derive equations to calculate the model likelihood for any given observation. Instead, we only need to simulate multiple instances of particles following different models, and the likelihood function reveals itself.

3.3 Classification

The goal of classification is to take an item and assign it to a particular category. For example, we may wish to assign an image of a biological cell to the category 'cancerous cell' or to the category 'non-cancerous cell'. The first stage of a classification system is to somehow represent the input item as set of numbers, which we call the *input vector*. For the image of a biological cell, the input vector would be the set of pixel values that make up that image. The output categories are called *classes*. Hence, a classification system can be viewed as a black box that takes an input vector and outputs a class.

The term *classification* implies that there are a finite number of discrete classes. If instead we wanted to output a continuous variable, then the task would be *regression*. If we did not have a set of target values, but wanted to group together similar input vectors, the task would be *clustering*.

3.3.1 Overview of Classification Systems

A classification system tries to learn a function that maps an input vector to an output class. The form of this function depends on the particular classifier being used (see Section 3.3.2) and its parameters are learned during a *training phase*. This training phase requires a *training set*, which is a set of examples for which both the input vector and the true class are known. Typically, the classifier tries to learn a function that gives a high accuracy on this training set.

It is often impractical for a classifier to operate on the input vector directly because it may be very large. For example, if the input vector contains all pixels of a 1024×1024 image, then it has over a million numbers. To deal with this, most classification systems have a stage called *feature extraction*, in which a much smaller set of numerical features are extracted from the input vector. For the case of a cellular image, examples of such numerical features could be the size or eccentricity of the cell. More such examples can be found in Section 3.5.1. Fig. 3.1 gives a block diagram of a typical classification system.

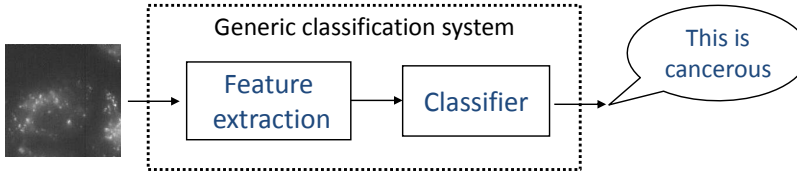


Figure 3.1: A generic classification system.

3.3.2 Classifiers

There are many different ways to learn a mapping from a feature vector to a classification decision. Some associate a probability with each possible class label, whereas others are decision machines and make hard choices. We outline two examples of possible classifiers—Bayes classifiers and support vector machines. Additional classifiers include decision trees [40], k-nearest neighbor classifiers [14], logistic regression [24], and neural networks [22].

Bayes Classifiers

Bayes classifiers assume that the feature vector $x_1, ..., x_n$ captures all probabilistic values associated with the problem. If there are k possible classes $C_1, ..., C_k$, then we can find the probability that x comes from class C_i using Bayes' theorem:

$$p(C_i|x_1, ..., x_n) = \frac{p(C_i)p(x_1, ..., x_n|C_i)}{p(x_1, ..., x_n)} \tag{3.16}$$

We find this probability for all classes $C_1, ..., C_k$, and then choose the class of highest probability as our classification decision. Because the denominator in (3.16) is the same for all classes, we can remove it from the calculations.

The training phase consists of learning the values of $p(C_i)$ and $p(x_1, ..., x_n|C_i)$ for each class C_i. The values of $p(C_i)$ are typically (1) known *a priori*, (2) assumed to be identical, or (3) learned based on the frequency of each class in the training set. If the feature vector is discrete, we can also learn the joint probability $p(x_1, ..., x_n|C_i)$ based on the frequency with which that feature vector occurs for each class in the training set. If the feature vector is continuous then we can

model the joint probability using a multivariate Gaussian, or some other distribution.

In most real-world situations, the number of features n is large and the training set is small. This makes it difficult to accurately learn $p(x_1, ..., x_n|C_i)$. In such cases we can make the naive conditional independence assumption that:

$$p(x_1, ..., x_n|C_i) = p(x_1|C_i)p(x_2|C_i)...p(x_3|C_i). \tag{3.17}$$

The resulting classifier is called a *naive Bayes classifier* [37]. Even in cases where the conditional independence assumption is clearly invalid, the results from this classifier are often surprisingly accurate [49].

Support Vector Machines

A support vector machine (SVM) [13, 7] is fundamentally a two-class classifier. It views the training data as points in an n-dimensional space and then looks for a hyperplane that separates the points from each of the two classes. The n-dimensional space could be the feature space, or it could be a transformation of the feature space. There may be many different hyperplanes that would separate the points from the two classes. In this case, an SVM chooses the hyperplane that maximizes the margin, where the margin is defined to be the smallest distance between the hyperplane and any of the training points.

In some cases, there may be no hyperplane that can separate the points in the feature space. There are two ways to deal with this: First, the feature space can be transformed to a different (usually higher-dimensional) space where linear separation is possible. Second, we can introduce the idea of a soft margin, which allows some training set data points to be misclassified, but penalizes misclassification based on how far from the margin the misclassified data point lies.

There are several methods for extending a support vector machine to be a k-class classifier (where $k > 2$). A common method is to construct k separate SVMs, each of which separate one class from all the remaining classes. Another method is to construct $k(k-1)/2$ SVMs on all possible pairs of classes. Hsu & Lin give a review of these and other methods [25].

3.3.3 Validation of Classification Systems

To evaluate a classification system, we take the full set of data for which we know the true class labels. We then split this data into two sets—the training set, and the testing set. We train the classification system using all the examples in the training set, and then we test the trained classifier on the examples in the testing set. Because we also know the true class labels of the testing set, we can measure the classification accuracy as the proportion of classification decisions that agree with those true labels.

Suppose that we had 100 examples for which we knew the true labels. We could use examples 1-50 as the training set, and examples 51-100 as the testing set. We can gain an even more accurate estimate of classification accuracy if we then train with examples 51-100, and test on examples 1-50. In this way, every example gets tested on once, and the final classification accuracy is based on the testing results of all examples 1-100. This scheme is called *two-fold cross-validation*, because the validation process is performed two times.

The problem with this scheme is that we are validating our classification system with a classifier that is only trained with 50 examples, whereas the final classification system will be trained with 100 examples. We can improve this by using *ten-fold cross-validation*. Here, we would first train with examples 1-90, and test on examples 91-100. We would then train with examples (1-80, 91-100) and test on examples (81-90), and then train with (1-70, 81-100) and test on (71-80), and so forth. By rotating the testing set through all the samples, we are still testing on every example exactly once, but the classifier is being trained with 90 examples instead of 50.

In the limit, this process leads to the *leave-one-out cross-validation*. Here, we train with all but one example, and then test on that one example. We rotate the testing example through the entire data set. In the case above, where we had 100 examples, leave-one-out cross-validation would be synonymous with 100-fold cross-validation. The only problem with such a scheme is that it involves training a classifier 100 times, which may be impractical if the training process is slow.

3.4 Active Learning

The problem of intelligent acquisition comes under the framework of active learning, which refers to any form of learning in which the learning program has some control over the inputs on which it trains [12]. Assuming that data acquisitions are expensive, the goal is to request the data that is most informative. This is typically framed in the context of classification, where we have a set of input vectors for training but do not know the class labels of these vectors. Assuming that a class label can be acquired at a cost, the goal is to choose which input vectors to request labels for such that an accurate classifier can be trained at the lowest cost.

Perhaps the best known examples of active learning frameworks are *uncertainty sampling* and *query by committee*, which we now outline. Settles [42] provides a more detailed survey of these and other active learning frameworks.

3.4.1 Uncertainty Sampling

Uncertainty sampling [31] requests class labels for training instances whose class the learner is least certain about. This method requires that we can estimate the certainty of our classifications. One problem with this approach is that a single classifier is not necessarily representative of all the classifiers that are consistent with the labeled data. Nonetheless, uncertainty sampling has been shown to greatly reduce the number of instances that an expert need label.

3.4.2 Query by Committee

Query by committee (QbC) [43] takes all classifiers that are consistent with previously labeled instances, and then requests labels for training instances that cause the most disagreement among these classifiers. This is generally better than uncertainty sampling: all classifiers might be equally uncertain about a particular class label, and thus attaining that label (as uncertainty sampling would do) may not be helpful. However, the QbC approach of attaining a label that causes disagreement among classifiers is guaranteed to eliminate some potential candidates, thus drawing us closer to the true classifier.

3.5 Related Work

3.5.1 Subcellular Protein Location Pattern Analysis

The field of proteomics is the study of proteins and their role and functions in cellular mechanisms. Location proteomics is a recent subfield of proteomics that focuses on the location of proteins within cells. Murphy *et al.* have pioneered the use of automated and objective methods for interpreting protein subcellular location images [5, 6, 19]. Because we rely on aspects of this work heavily, we outline their methods for two specific tasks: recognition of protein subcellular location images and recognition of subcellular objects.

Recognition of Protein Subcellular Location Images

We described general classification systems in Section 3.3. Murphy *et al.* have designed such a system for recognizing subcellular patterns in a number of cell types. These systems use a set of numerical features called subcellular location features, which are designed to be insensitive to the position, rotation, and total intensity of a cell image. Static features have been developed for 2D and 3D images, and temporal features for 2D time series and 3D time series.

Systems have been built to recognize all major protein subcellular location patterns in both two-dimensional and three-dimensional HeLa cell images with high accuracy (over 95% and 98%, respectively)[5, 6, 35, 10]. Of special interest to this thesis is a similar system applied to a set of 3T3 cell time series images, because we also attempt to classify this same data set (described in Section 5.1). We now outline the features used, and then summarize the classification results on the 3T3 data set.

Subcellular Location Features The major features used are:

- Morphological features [6, 36, 47]. These features are mainly based on the characteristics of objects within a cell, where an object is defined as a group of connected pixels that are above some threshold. Examples include the number of objects per cell, average number of pixels per object, and average distance of objects to the center of fluorescence. These can be defined both in 2D and in 3D.

- Edge features [6, 11]. These include features such as the fraction of pixels distributed along edges, the homogeneity with which edges are aligned, and the total fluorescence of the edge pixels. Edges are places in the image with discontinuities in the intensity (where the intensity changes sharply). For a 3D image, the total set of edges consists of the edges from each 2D slice of that 3D image.

- Haralick texture features [21, 6, 36, 11]. These features summarize the relative frequency with which one gray level appears adjacent to another. For 2D images, adjacency can occur in four directions (horizontal, vertical, and two diagonal directions). In 3D images, adjacencies can be found in 13 directions.

- Other static features include geometric features [6], DNA features [6], Zernike moment features [5], skeleton features [36], Daubechies-4 wavelet features [28], and Gabor texture features [28].

- Temporal texture features [26]. These are the same as the Haralick texture features, except that they summarize the relative frequency with which one gray level appears adjacent to another in *time*. These can be defined both for 2D time series and for 3D time series.

- Normal flow features [26]. The normal flow field is based on the gradient of pixel intensity in time. Several statistics can be calculated on this field.

- Fourier transform features [26]. By considering each pixel at the same position over time as a time series signal, we can calculate a Fourier transform of this signal. Fourier transform features are statistics based on the first four coefficients of this transform.

Results for 3T3 Time Series A classification system using these features was applied to 3T3 time series by Hu *et al.* [27, 26]. The system used a support vector machine as the classifier (see Section 3.3.2). Although the data set contains 3D time series, initially classification was done with only the middle slice from each time point, using 2D static features and 2D temporal features as described above.

This gave an accuracy of 83.9%. Interestingly, when the entire 3D image at each time point was used and classified with 3D static and temporal features, the accuracy actually dropped to 81.6%. When only static features were used to classify, the accuracy was 71.1% for 2D images, and 77.3% for 3D images. These results are summarized in Table 3.1.

Input	**Features**	**Classification Accuracy**
2D images	static	71.1%
3D images	static	77.3%
2D images	static and temporal	83.9%
3D images	static and temporal	81.6%

Table 3.1: Classification accuracy for different configurations

Recognition of Subcellular Object Types

Whereas the previous section looked at classification on a cell level, Murphy *et al.* also studied the object level. Here, the goal is to take subcellular objects and to group them into types. Because there are no ground truth labels for the objects, and the number of types is unknown, a clustering approach is chosen rather than a classification approach.

Object Detection The first stage is object detection. This aims to find sets of connected pixels in 3D which display a higher intensity level than their local environment. Because the background intensity is not uniform, a multi-threshold approach is used. First, the most common pixel value is found from those pixels whose intensity is below the mean intensity of the image. This value is subtracted from every pixel. Second, the pixels are smoothed with a Gaussian filter. Third, an object is defined as the biggest set of 3D-connected pixels that contains only one maximum intensity.

Object Features Having determined the locations of the objects, and the voxels assigned to them, the next step is to calculate static features on each of the objects. The list of features are shown in Table

3.2 [46, 50]. Many of these features are adapted from the features mentioned in the previous section, except that they are applied on each object individually.

Index	Feature Description
3D-SOF1.1	Number of voxels in the object
3D-SOF1.2	Distance between object center of fluorescence and DNA center of fluorescence
3D-SOF1.3	Fraction of object voxels overlapping with DNA
3D-SOF1.4	A measure of the eccentricity of the object
3D-SOF1.5	Euler number of the object
3D-SOF1.6	A measure of the roundness of the object
3D-SOF1.7	The length of the object's skeleton by homotopic thinning
3D-SOF1.8	The ratio of skeleton length to the area of the convex hull of the skeleton
3D-SOF1.9	The fraction of object voxels contained within the skeleton
3D-SOF1.10	The fraction of object fluorescence contained within the skeleton
3D-SOF1.11	The ratio of the number of branch points in the skeleton to the length of skeleton

Table 3.2: List of static subcellular object features (SOF) defined to describe objects in 3D [46].

Object Types Finally, the objects are grouped into types [46, 50]. An object type is defined as a group of objects with similar characteristics that form a cluster in the feature space. These clusters are learned by applying k-means clustering to all the objects. The optimal number of clusters, k is determined by minimizing the Akaike Information Criterion (AIC), which specifies a tradeoff between complexity of the model (number of clusters) and goodness of the model (compactness of the clusters).

3.5.2 Efficient Acquisition in Fluorescence Microscopy

Efficient acquisition for fluorescence microscopy is a recent problem. Related work on efficient acquisition for fluorescence microscopy was done by Merryman & Kovačević [33]. They presented an algorithm to reduce the number of pixels acquired in a 2D or 3D image when

using a laser scanning confocal microscope, with the end application being recognition of proteins based on their subcellular location [5, 6, 19, 10]. The goal was to reduce the time spent acquiring low fluorescence regions, which presumably contain little useful information. The algorithm begins by scanning the field at low resolution. Each scanned value is examined, and if found to be significant, the area around it is scanned at a higher resolution. The process is repeated iteratively. The limitation of this technique is that it cannot adapt to specifically seek out information required for the end application, nor can it use knowledge of the cellular dynamics.

Hoebe *et al.* introduced controlled light-exposure microscopy [23], which can use a different exposure time for each pixel. If no significant fluorescent signal is detected at a pixel, the exposure time for that pixel is reduced. Similarly, if the signal is very strong, the exposure time will also be reduced because the signal-to-noise ratio will still be high. As a result, this method allows images to be acquired faster and with less overall light exposure, thus reducing photobleaching and phototoxicity. This technique could be included within our proposed framework.

3.5.3 Efficient Acquisition in Magnetic Resonance Imaging

Work on efficient acquisition is also found in the field of magnetic resonance imaging (MRI). For example, Liang & Lauterbur [32] present a method to efficiently acquire a time series of images by observing that the high-resolution image morphology does not generally change from one image to another. Then, using a generalized series model, they eliminate the repeated encodings of this stationary information in the conventional Fourier methods. An alternative approach uses a singular value decomposition of the first (base) image to design excitation sequences that efficiently acquire the data in subsequent images [38]. A more comprehensive overview of efficient acquisition in MRI is given by Tsao *et al.* [45].

3.5.4 Compressed Sensing

Compressed sensing [8, 9, 16, 2] is a technique that allows sub-Nyquist image acquisition provided that (1) the signal is sparse in some target basis, and (2) we can acquire in a basis that is mu-

tually incoherent with this target basis. A popular choice of incoherent basis is to acquire random projections of the signal. Only a small number of such projections are acquired, and so the task of converting these into the target basis is an underdetermined set of linear equations. Typically such equations are solved by minimizing the L_2 norm, however, the compressed sensing approach exploits the sparsity property and instead minimizes the L_1 norm. This can be expressed as a linear program, for which efficient solutions exist (whereas, say, the L_0 norm would give even sparser solutions but is computationally infeasible for large data sets).

We investigated using compressed sensing for our application by acquiring random linear combinations of pixels instead of individual pixels. However, to acquire such projections while holding photobleaching and acquisition time constant, we were forced to dramatically reduce the exposure time of each acquisition. We found that the resulting increase in noise offset the benefits.

Part II

Model Building

The goal of this part is to construct a model by using the raw data acquired from the microscope along with any prior knowledge. We assume that the class of models is known, and thus model building refers to estimating the parameter values. Our models are based on the dynamics of objects between frames. We split this part into two chapters: Chapter 4 discusses model building for the case of a single object moving between frames. This chapter is based entirely on synthetic data, providing us with a ground truth to validate our model-building procedure. Chapter 5 discusses model building for the case of multiple objects moving between frames. This chapter is based entirely on real data, providing us with a real-world experimental validation of our methods.

There are two types of models we consider: (1) Cell models are based solely on a single cell. (2) Class models are based on all cells from a particular class. Our discussion of single-object modeling focuses only on the first type—cell models—but for multiple-object modeling we consider both cell and class models.

4

Single Object

This chapter discusses model building for the case of a single object. All models are cell models, meaning that they are based solely on a single cell (rather than on a group of cells). Experiments in this chapter are performed on synthetic data. This allows us to assess the potential of our methods objectively, because we have accurate ground truth.

The synthetic data is generated using specific *motion models*, described in Section 4.1, and this choice of motion models is known by the model-building module. However, because model building is done using particle filters (see Section 3.2), our methods should work equally well for most other choices of motion models.

When designing the model-building module, we must consider the form of the input data. As we describe in Part III, our intelligent acquisition methods include acquiring only a subset of pixels in a frame, as well as varying the frame rate. Therefore, the model-building method must be able to model such data. We also make some simplifications regarding the acquisition and detection processes: First, we assume perfect object detection provided that the appropriate region of the image is acquired. Second, we assume that all pixels in a frame are acquired at precisely the same instant. We discuss these assumptions further in Section 4.4.

We begin this chapter by providing more details on the synthetic data set used for our single-object experiments. Then, in Section 4.2 we describe how to build the model, and in Section 4.3 how to validate that the model is learned correctly.

4.1 Data Set: Generated Synthetic Tracks

Each cell in our synthetic data set contains a single object. We assume that this object is a point source, meaning that it only occupies one pixel, and that it moves between frames according to its *motion model*. There are two motion models that we consider, which subcellular objects are commonly observed to follow: random walk (RW) and constant velocity (CV).

Random Walk Motion Model In the RW model, objects move in a random direction between frames. The position of an object in a frame depends only on its previous position, and thus velocity and acceleration of the object are not conserved. Its position at time t, $\mathbf{y}_t$, is simply its previous position, $\mathbf{y}_{t-1}$, perturbed by the displacement $\mathbf{d}_t$ of additive Gaussian noise of mean $\mathbf{0}$ and covariance $\mathbf{\Sigma}$, $N(\mathbf{0}, \mathbf{\Sigma})$. The parameters of the covariance matrix $\mathbf{\Sigma}$ are often known as rate parameters because they govern the rate at which the objects move in each dimension. The RW model is thus described by:

$$\mathbf{y}_t = \mathbf{y}_{t-1} + \mathbf{d}_t. \tag{4.1}$$

Constant Velocity Motion Model In the CV model, objects move with a constant velocity $\boldsymbol{\mu}$ between frames. Once again, we have displacement $\mathbf{d}_t$ as additive Gaussian noise, which determines the extent to which velocity is conserved. The CV model is governed by:

$$\mathbf{y}_t = \mathbf{y}_{t-1} + \boldsymbol{\mu} + \mathbf{d}_t. \tag{4.2}$$

We can see from this equation that the RW model is a special case of the CV model, with $\boldsymbol{\mu} = \mathbf{0}$. Therefore, both of these models can be characterized by $(\boldsymbol{\mu}, \mathbf{\Sigma})$, where $\boldsymbol{\mu}$ is a three-dimensional vector. Note that a general symmetric covariance matrix $\mathbf{\Sigma}$ has $D(D+1)/2$ independent parameters [4], where D is the dimension, and thus for $D = 3$ we have 6 independent parameters. Combining this with the 3 parameters of $\boldsymbol{\mu}$, the model has 9 independent parameters in total.

When considering motion models, we must also consider the boundary conditions. The behavior at the boundary has no significant effect on our experiments, and so for simplicity we assume that an object cannot move outside the image boundary and that it bounces back upon hitting this boundary.

Therefore, to generate a synthetic track, we randomly choose a motion model and corresponding parameters, drawn from some prior distribution. We assume that this prior distribution is also known to the model-building module. Our prior distribution gives equal probability to RW and CV models, and equal probability to all parameter values within these models, with an upper bound. For the CV velocity parameters, this upper bound was 2 pixels/frame in each dimension. Because $\mathbf{\Sigma}$ must be positive semi-definite to be a valid covariance matrix, we initially chose variances for (x, y, z) in the range $(0, 10)$, and created a diagonal matrix from these values. We then randomly rotated it in (x, y, z) to create a general covariance matrix. The range of parameter values was based on observations of the 3T3 data set that we use in Chapter 5. However, the model-building methods are not sensitive to this range. Finally, we initialize the position of the object to a random pixel in the image, and begin propagating it between frames according to its motion model.

4.2 Building the Model

In this scenario, there is just a single point-source object, and so the only goal of model building is to learn the motion model that describes that object. We do this using particle filters, outlined in Section 3.2. Particle filters are a useful choice because they make few assumptions about the model, and are easy to implement even for complex models [1]. We do not need to derive equations to calculate the model likelihood for any given observation. Instead, we only need to simulate multiple instances of particles following different models, and the likelihood function will reveal itself.

For the synthetic data set described, learning the motion model just consists of learning the parameters $(\boldsymbol{\mu}, \mathbf{\Sigma})$. It may seem that particle filters are unnecessarily complicated for such a task. A simpler method would be to record the displacement of the object in each frame, and then use the sample mean and covariance of these displacements as our model parameters. Even with the small error introduced by pixelation, this closed-form method is fast and accurate. However, we use particle filters for two reasons: First, the closed-form method cannot handle the case where the intelligent acquisition module chooses to acquire only a subset of pixels in a frame, or to skip a frame entirely. Second, we want a method that can, with

little or no modification, be used for a wide range of motion models other than those outlined in Section 4.1.

To use particle filters, we must first define the state vector. This consists of the position, $\mathbf{y}_t$, as well as the motion model parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. These last two variables are static parameters and the goal of the model building is to learn them. We initialize the particle filter by generating N particles from the state space. Each particle is assigned values for $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, using the same prior distribution that was used to create the synthetic data. We also assign initial positions to each of the particles. Rather than assign these positions to random parts of the image, we instead assume that we begin acquisition by acquiring a complete frame and that we observe the pixel of the object's true position. Thus, the initial positions of the particles can be set to random positions within the actual initial pixel of the object.

Following an observation of the object, we use the following simple weighting procedure: If a particle is at the same pixel as the observed object, we keep it; if a particle is at a different pixel, we eliminate it (assign it zero weight). If we acquire a set of pixels and do not observe the object, then we eliminate the particles in the acquired set of pixels, and retain all other particles. In either case, the surviving particles all have equal weight, and so the resampling procedure just involves duplicating these particles to keep the overall number at N. We could improve efficiency and reduce degeneracy by giving fractional weight to particles that are in neighboring pixels of the observed object; we leave such considerations for future work.

Using this procedure, the likelihood of any model is given by the distribution of the models of the surviving particles. As N approaches infinity, this distribution converges to the true likelihood function. Furthermore, as the number of observations increases, the likelihood function converges to the object's true model.

When acquisition is complete, we can take the models of all the surviving particles and use them to make a point estimate of the final model. If the parameters of the model of particle i are $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ respectively, we choose the parameters of the final model as follows:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{\mu}_i \tag{4.3}$$

$$\boldsymbol{\Sigma}^2 = \frac{1}{N^2} \sum_{i=1}^{N} \boldsymbol{\Sigma}_i{}^2 \tag{4.4}$$

These parameters represent the Gaussian distribution that best approximates the mean of the distributions of each surviving particle's model.

4.3 Validating the Model

We now seek to validate the effectiveness of our model-building procedure. A good way to do this is to see how well the final model can predict future frames. We do this by finding the log-probability with which those future frames occur under the final model, or equivalently, the log-likelihood of the final model given those future frames. The goal is to choose the model that maximizes this log-likelihood.

When we consider real data in Chapter 5 we will measure this log-likelihood based on actual observations of future frames. However, with synthetic data we know the true underlying model, and so we can compute the expected log-likelihood in closed form. If the true model follows some distribution $p(x, y, z)$, and our estimate of the model follows $q(x, y, z)$, then the expected log-likelihood is given by:

$$\log(l) = \int_{x,y,z} p(x, y, z) \log(q(x, y, z)). \tag{4.5}$$

Note that this quantity is maximized when $p = q$ (remembering that p and q are probability distributions, and thus each sum to one). Also note that the expected log-likelihood is related to the KL divergence $K(p||q)$ of q from p, and the entropy E_p of p, as shown (see Section 3.1.3):

$$\log(l) = -E_p - K(p||q). \tag{4.6}$$

This means that maximizing $\log(l)$ is equivalent to minimizing $K(p||q)$.

The KL-divergence between two Gaussian distributions, $K(m_t||m_e)$, and the entropy of a Gaussian distribution, E_{m_t}, are as follows [15]:

$$\begin{aligned} K(m_t||m_e) &= \frac{1}{2}[\log(\frac{|\mathbf{\Sigma}_e|}{|\mathbf{\Sigma}_t|}) + (\boldsymbol{\mu}_t - \boldsymbol{\mu}_e)^T \mathbf{\Sigma}_e{}^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu}_e) \\ &\quad + \mathrm{tr}(\mathbf{\Sigma}_e{}^{-1}\mathbf{\Sigma}_t) - 3] \\ E_{m_t} &= \frac{1}{2}(3 + 3\log(2\pi) + \log(|\Sigma_t|) \end{aligned}$$

Combining these two facts with (4.6), we conclude that the log-likelihood of the true model $m_t = N(\mu_t, \Sigma_t)$ given the estimated

model $m_e = N(\mu_e, {\Sigma_e}^2)$ is given by:

$$log(l)_{t,e} = -\frac{1}{2}[\ln(|\boldsymbol{\Sigma}_e|) + \mathrm{tr}({\boldsymbol{\Sigma}_e}^{-1}\boldsymbol{\Sigma}_t) + (\boldsymbol{\mu}_e - \boldsymbol{\mu}_t)^{\mathrm{T}}{\boldsymbol{\Sigma}_e}^{-1}(\boldsymbol{\mu}_e - \boldsymbol{\mu}_t) + 3\ln(2\pi)].$$

We can then use $log(l)_{t,e}$ to measure how well our model is predicting future frames, with the upper bound being given by the entropy E_{m_t}.

In Fig. 4.1, we simulate an object moving between frames as described in Section 4.1. We then use our model-building method to learn a model of this object. For this experiment, we assume that we acquire every pixel in the frame, but in Chapter 6 we show examples where we only acquire a subset of pixels in each frame. The solid red line shows the log-likelihood increasing as the model is learned. The dotted black line shows the entropy of the true model, which represents the maximum log-likelihood achievable. We can see that the log-likelihood converges to this entropy, thereby validating our model-building procedure.

4.4 Discussion

Because particle filters are very general, this method works for a wide range of motion models. The same is true of the corresponding intelligent acquisition algorithms that we introduce in Part III. For this reason, we have not focused on finding motion models that exactly mimic real object motion, but rather on finding a model-building method that has broad applicability. We also did not consider computation time extensively, but will pay more attention to it in Chapter 5 when working with real data. Typically, however, we used around one million particles, which resulted in a processing time of about a second between frames (Intel Core Duo 2.2GHz processor with 1.96GB of memory).

We can make several extensions to the model building. We can expand the state space to include static features such as the size of the object, and then similarly expand the model to include changes in these static features. We can also account for imperfect detection. For example, when we acquire a set of pixels and do not find the object in those pixels, the current implementation eliminates all the corresponding particles. Instead, it could simply reduce the weight of these particles to reflect that the object may still lie within the set of pixels but not have been detected.

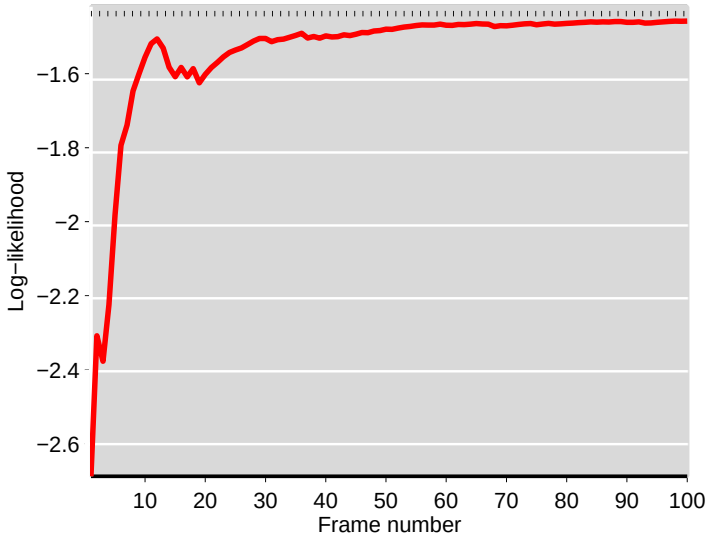


Figure 4.1: Model building (single object, synthetic data). The solid red line shows the log-likelihood of the estimated model at every frame, given the true model. The dotted black line shows the entropy of the true model, which is the maximum log-likelihood achievable. We can see that the model estimate improves throughout the duration of acquisition (although not monotonically), converging on the entropy.

Although the pixels in a frame are all acquired at slightly different times, the model building is mainly affected by the time between two successive acquisitions of an object. Assuming that objects do not move far between frames, this time should be close to the time between frames, and hence the assumption that all pixels in a frame are acquired at the same instant should not greatly affect the model accuracy. Nevertheless, an extension to the model building would be to more accurately account for the exact time at which a pixel is acquired.

Perhaps the biggest limitation with the method described in this chapter is that it is restricted to the case of a single object. We could extend to multiple objects by using a separate particle filter for each object, but then we would have to find which objects in a frame t match to which objects in the subsequent frame $t + 1$.

This is easy under some conditions, such as when the objects can be distinguished by their static features, when the motion of the objects can be precisely predicted, or when the objects are sufficiently far apart and do not move far between frames. This last scenario is applicable when we only wish to model a small number of objects: If only modeling a few objects, we do not need to acquire many pixels in each frame. This in turn would allow for high temporal resolution, which means objects do not move too far between frames, thereby allowing us to find the object matches with high accuracy.

In situations where the object matches are not clear, we can use the distribution of particles to determine the probability that an object in frame t matches to a given object in frame $t+1$. We determine this with the proportion of particles from the object in t that move to the object in $t+1$. We can then use the the Hungarian algorithm [18] (also known as the Kuhn-Munkres algorithm or Munkres assignment algorithm) to find the most likely combination of matches between all objects in t and $t+1$. An example of this is shown in Table 4.1. The problem with this algorithm is that it makes hard matches, and the motion modeling is very sensitive to a single mistake. Alternatively, methods such as the multiple hypothesis tracker [41] try to keep track of all possible match combinations, which works better, but the number of possible combinations grows exponentially and so eventually hard matches must still be made.

We have had the most success with a different approach, which is to have only one particle filter, and after each frame to retain particles in the same pixel as *any* object. As a result, object matches do not need to be found. This method does not output the motion model of each individual object in the cell, but rather the distribution of motion models, which is often all that is needed. We can improve the resampling procedure further by considering combinations of particles that can occur together: In Table 4.1 we showed the combination of matches with the highest joint probability. However, other combinations of matches also have a given joint probability, and so the marginal likelihood of an individual match can be computed as the sum of the joint probabilities of each combination in which that individual match occurs. Therefore, if we have a particle that moved from a particular object in frame t to a particular object in frame $t+1$, we can weight this particle according to the marginal likelihood of the corresponding match.

Despite these solutions to the multiple-object problem, real data

still poses many challenges. These include appearing and disappearing objects, changing shapes and sizes of objects, objects that change motion models, and global cell motion. Although the proposed extensions to multiple objects work well in simple simulations, and may work for certain data sets, more work is needed before they can be used for the 3T3 data set described in Chapter 5. For this reason, and for other reasons that we describe shortly, we did not use a particle filter-based method for multiple-object model building.

	1	2	3	4
1	0.1011	0.2595	0.3325	**0.3070**
2	0.0349	0.4208	**0.3865**	0.1578
3	**0.2804**	0.1272	0.2230	0.3694
4	0.1095	**0.4194**	0.3145	0.1566

Table 4.1: Sample probabilities for matching 4 objects in frame t with 4 objects in frame $t+1$. Each row corresponds to frame t, and each column corresponds to frame $t+1$. The bolded numbers represent the combination of matches with the highest joint probability (0.014).

5

Multiple Objects

This chapter discusses our method for building models of cells containing multiple objects. This method is tested exclusively on real data. Although Section 4.4 described ways to extend the particle filter method to work with multiple objects, we do not continue this approach. There are two main reasons for this: First, real data presents us with challenges such as changing sizes and shapes of objects, as well as the appearing and disappearing of objects, and more work is needed to enhance the particle filter method to adequately meet these challenges. This is especially true of the data set that we use in this chapter, where objects can change size and shape dramatically, and can be difficult to track even by eye. Second, we began this work with the main focus of deciding where and when to acquire in cells. The particle filter method is very useful there because it is difficult to build models in closed-form when only a subset of pixels are acquired in each frame, or when the time between successive frames changes. Increasingly, however, the focus of our work shifted towards determining when to stop acquiring a cell, and in the case of class models, determining how many cells to acquire. For this, we assumed that all pixels in a frame were acquired, and that the frame rate was constant. This removed one of the main motivations for using particle filters. Hence, instead of extending the particle filter method, this chapter focuses on faster and simpler closed-form methods for multiple-object model building.

An important aspect of model building is the ability to validate the accuracy of the constructed models. We want to show that the accuracy of cell models improves as more frames are acquired, and

that the accuracy of class models improves as more cells are acquired. With synthetic data (Chapter 4), we measured accuracy with the log-likelihood of the true underlying model given the estimated model. With real data, we do not know the true underlying model; instead, we can use the log-likelihood of future observations (which will follow the true underlying model). Alternatively, we can validate the models by attempting to classify a cell model into its correct class model. We use both of these methods as validation of our model building (and later our intelligent acquisition) algorithms.

This chapter begins by describing the data set used for our experiments in multiple-object scenarios. Then, in Section 5.2, we explain how objects are detected in this data set and how static features are computed on those objects. Section 5.3 discusses how to build cell models and validates this process using the log-likelihood method. Section 5.4 discusses how to build class models and also validates this process with the log-likelihood method. Finally, we discuss our alternative method of choice for validation—classification—in Section 5.5.

5.1 Data Set: 3T3 Time Series

We perform our experiments on a collection of 304 3D time series of GFP-tagged proteins in NIH 3T3 cells over 12 different cell lines, with a different protein labeled in each cell line. The proteins were tagged with the CD-tagging protocol described in Section 2.2. Proteins included in this study are located among 7 major organelles. We define each cell line/protein as a *class*.

Each time point of the time series contains a single channel stack of 15 z-slices, 1280×1024 pixels each. The x,y-resolution is 0.11μm, and the distance between pixels in the z-direction is 0.5μm. There is a 45s interval between frames. Further acquisition parameters are described by Hu *et al.* [27]. Table 5.1 summarizes the gene/protein labeled for each cell line, along with the subcellular location where the protein is expressed, the number of time series for each cell line, and the number of frames in each time series. The number of frames varies depending on the amount of photobleaching incurred during acquisition.

Figures 5.1 and 5.2 each show two successive frames from different Cav cells. Although we have a 3D image for each frame, only a 2D

Gene	Protein	Location	# cells	# frames
Dia1	Cytochrome b-5 - reductase	cytoplasm	20	19-24
Anxa5	Annexin A5	nucleus, cytoplasm	18	11-25
Sdpr	Serum deprivation response protein	vesicles, cytoplasm	23	20-31
Adfp	Adipose differentiation - related protein	vesicles	51	21-27
Timm23	ADP-ATP translocase 23	mitochondria	40	16-22
Atp5a1	ATP synthase	mitochondria	20	21
Hspa9a	Mitochondrial stress-70 protein	mitochondria	24	9-46
Glut1	Glucose transporter 1	plasma membrane	17	13-82
Cav	Caveolin	plasma membrane	16	11-49
Tctex1	t-complex testis expressed 1	cytoskeleton	30	13-36
Actn4	Actinin, alpha 4	cytoskeleton	29	9-25
Cald1	Caldesmon 1	cytoskeleton	16	12-34

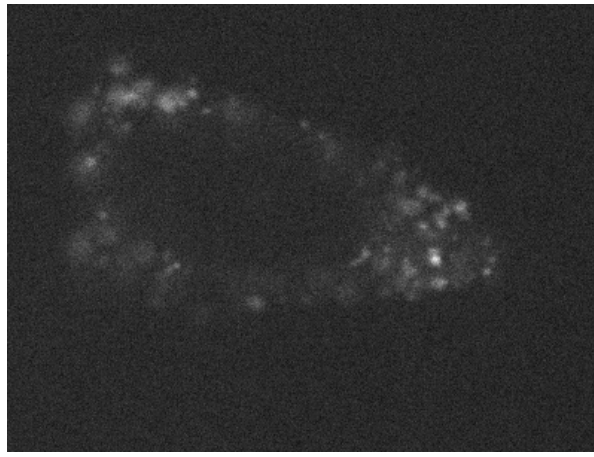
Table 5.1: Overview of the experimental dataset composed of 12 cell lines [26].

cross-section is shown in these figures.

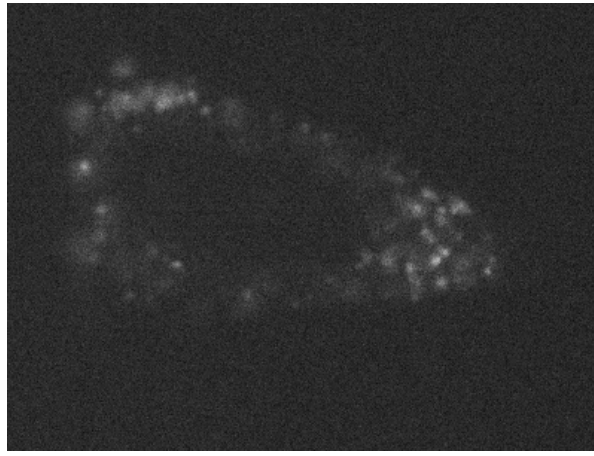
5.2 Object Detection and Feature Extraction

Our model-building method is based only on the objects present within a cell. Therefore, the first step is to detect those objects and compute their static features. For this, we follow the prior work summarized in Section 3.5.1, detecting objects using the multi-threshold approach described in that section. Because we do not have access to a DNA channel, we calculate only 7 of the 11 static features listed in Table 3.2. This subset is shown in Table 5.2.

As was also done in the prior work, we reduce the static features into a single object type. The method is to take all objects across all frames and time series, and use the batch k-means algorithm

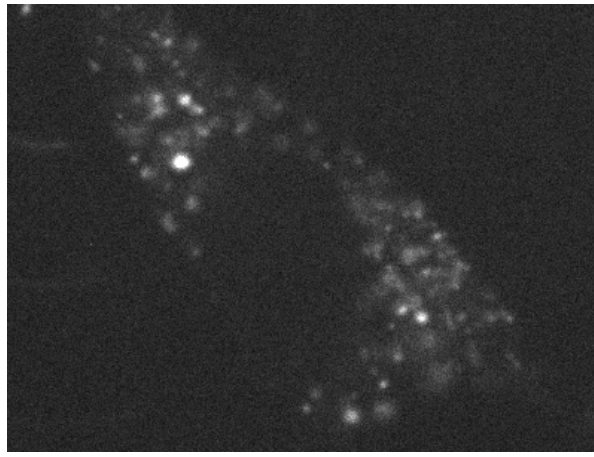


(a)

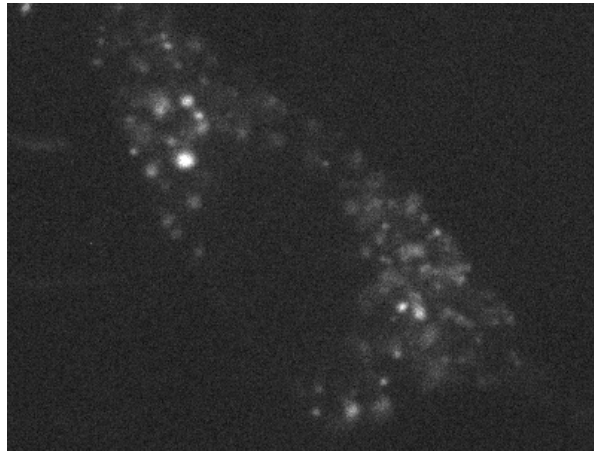


(b)

Figure 5.1: Two successive frames from a Cav time series. Although we have a 3D image for each frame, only a 2D cross-section is shown in these figures.



(a)



(b)

Figure 5.2: Two successive frames from a Cav time series. Although we have a 3D image for each frame, only a 2D cross-section is shown in these figures.

to cluster them based on their static features. Prior to running this algorithm, we normalize the features to unit standard deviation. The object's type is determined by the cluster into which it falls. This algorithm requires determining k, the number of object types. In the aforementioned work, the Akaike information criterion was used to choose the optimal number of types. However, in our work, we choose k to maximize either the classification accuracy or the log-likelihood, as we will discuss.

Index	Feature Description
3D-SOF1.1	Number of voxels in the object
3D-SOF1.4	A measure of the eccentricity of the object
3D-SOF1.6	A measure of the roundness of the object
3D-SOF1.7	The length of the object's skeleton by homotopic thinning
3D-SOF1.8	The ratio of skeleton length to the area of the convex hull of the skeleton
3D-SOF1.9	The fraction of object pixels contained within the skeleton
3D-SOF1.11	The ratio of the number of branch points in the skeleton to the length of skeleton

Table 5.2: List of the 7 static subcellular object features (SOF) used in this work. The complete set of 11 features is shown in Table 3.2 [46].

5.3 Cell Models

5.3.1 Modeling Displacement

Our method for multiple objects does not try to model every object separately. Instead, the model expresses some kind of average behavior for an object (although it allows different object types to show different behaviors). We start our discussion by ignoring the static features of objects, and describe a model that considers only the locations of objects in each frame (or more specifically, the locations of the centroids of the objects). Given an object's location in frame t, this initial model represents the probability of finding an object in any nearby location in frame $t+1$. We assume shift invariance: that

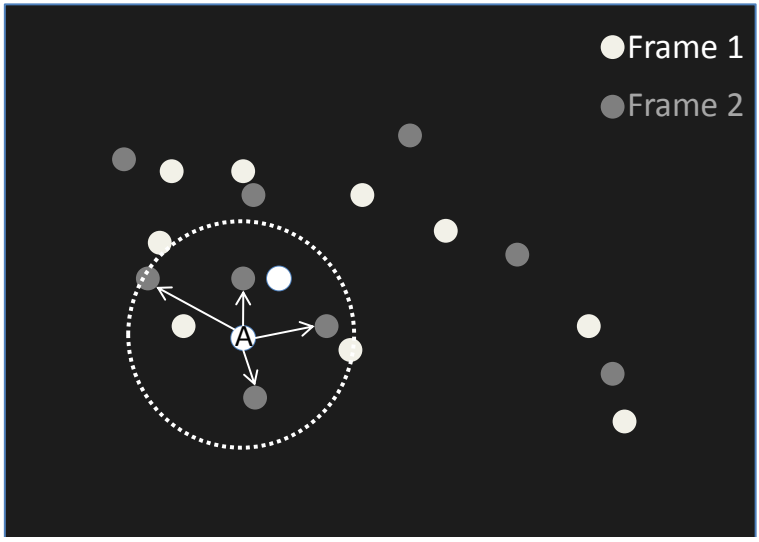


Figure 5.3: Finding nearby objects. The white circles represent objects in frame 1, and the gray circles represent objects in frame 2. The arrows indicate the nearby objects for object A within a radius of d_{max}.

is, the probability of a given displacement is independent of the object's location in the image. Hence, the model only has to associate a probability with each displacement.

Building the Model

To build this model, we first choose some maximum distance, d_{max}. We then iterate through the centroids of every object in frame 1, looking in frame 2 for *nearby objects*, which we define as objects whose centroids are within a distance of d_{max}. An example is shown in Fig. 5.3. For each nearby object found, we record the displacement (rounding to the nearest pixel), or we record that no nearby objects were found. We do not restrict an object in frame $t+1$ from being a nearby object to two or more objects in frame t. However, the nature of the data set and the choice of d_{max} were such that this event only occurred about 15% of the time. The event of an object in t having no nearby objects in $t+1$ also occurred about 15% of the time.

We repeat the above process for objects in frame 2 (looking for

nearby objects in frame 3) and so forth for every frame. Let $N_{x,y,z}$ be the total number of times that a nearby object was observed at displacement (x, y, z), $N_\emptyset$ be the total number of times that an object had no nearby objects, and N be the total number of objects observed across all frames. We then define the model, m, as follows:

$$m = (m_{x,y,z}, m_\emptyset), \tag{5.1}$$

where:

$$m_{x,y,z} = \frac{N_{x,y,z} + 1}{N + 2}, \tag{5.2a}$$

$$m_\emptyset = \frac{N_\emptyset + 1}{N + 2}. \tag{5.2b}$$

The model tries to capture the proportion of objects that have a nearby object at displacement (x, y, z), or that have no nearby objects. These equations show the Bayes estimates of those proportions, where the cost function is mean square error, and a derivation was given in Section 3.1.2. The 1 in the numerator and the 2 in the denominator come from the assumption of a uniform prior.

Fig. 5.4 shows the x,y-intensity distribution of $m_{x,y,z}$ for $z = 0$. This model was built from a Cav cell of 23 frames, with approximately 300 objects in each frame. In this particular example, we can see that an object in $t + 1$ is typically found within 2 pixels of an object in t, with motion in the y-direction being more probable than motion in the x-direction.

Validating the Model

As we discussed at the beginning of the chapter, we need some way to measure the accuracy of our model. With real data, we use two different methods of validation: The first is classification, and we discuss this in Section 5.5; the second, which we discuss here, is to use the model to predict some cell behavior, and to then test the accuracy of that prediction.

An intuitive way to do this would be to build a model on frames $1, ..., t$, and then make predictions for the displacement of objects from t to $t + 1$. We could then look at frame $t + 1$ and test the accuracy of those predictions. The problem with this method is that the cell suffers from photobleaching as t increases, and so the number of objects in each frame decreases, and the proportion of each object

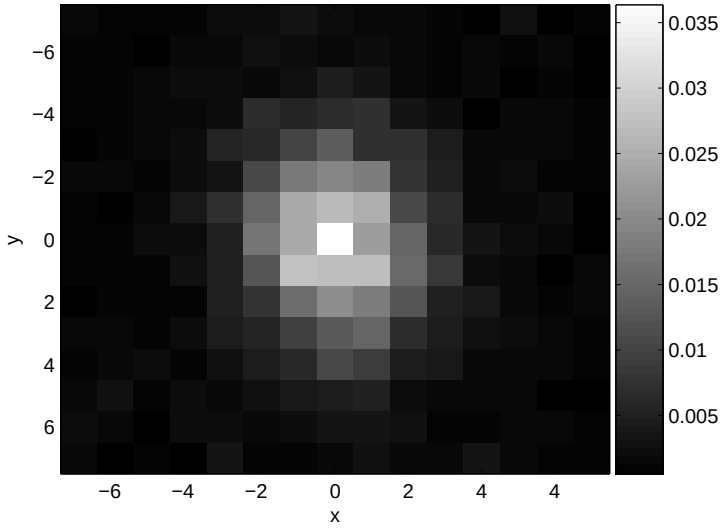


Figure 5.4: Displacement model. This image shows the xy-intensity distribution of $m_{x,y,z}$ for $z = 0$ for a Cav cell of 23 frames. Objects in $t + 1$ are typically found within 2 pixels of an object in t. Motion in the y-direction is more probable than motion in the x-direction.

type also changes. As a result of this, we found that frames always get more predictable as t increases, even if we do not change our model. Hence, this method does not demonstrate whether or not the model is improving with time.

Instead, we do something slightly different. We set aside the first two frames as testing frames, and use the remainder (frame 3 onwards) as training frames. Using the model built from the training frames, we try to predict the displacements of objects from our first testing frame to the second. We show that the more training frames used to build the model, the better the prediction of displacements.

To measure the prediction accuracy, we count the number of displacements of (x, y, z) between the testing frames and label this count as $N'_{x,y,z}$. We set $N'_{\emptyset}$ to the number of objects in the first testing frame that had no nearby objects in the second. We then measure

the likelihood of our model given these observations using:

$$l = {m_\emptyset}^{N'_\emptyset}(1 - m_\emptyset)^{(N' - N'_\emptyset)} \prod_{(x,y,z)} {m_{x,y,z}}^{N'_{x,y,z}}(1 - m_{x,y,z})^{N' - N'_{x,y,z}}, \tag{5.3}$$

where there are N' objects in the first testing frame. This expression ignores any dependencies between the model parameters and thus may not be a good estimator of the likelihood. However, as we see in the next experiment and also when we build a classifier in Section 5.5, it still provides a good measure of the model accuracy.

We use this method of validation on an Adfp cell. In Fig. 5.5, the dashed blue line shows how the log-likelihood, $\log(l)$, increases when the model is built from more frames. We can see that it increases significantly with the first few frames, but subsequently levels off, even decreasing at times. The law of diminishing returns explains why the model accuracy increases more slowly as more frames are acquired—subsequent frames provide duplicate information, and thus the marginal return of each additional frame is lower. However, this does not explain why the model accuracy decreases at times. There are two explanations for these decreases: First, the time series exhibit photobleaching. For the cell used in Fig. 5.5, photobleaching causes the number of objects detected per frame to drop from 2310 to 1197 over the duration of acquisition. This could change the distribution of observed displacements because brighter objects are more likely to still be detected in the last frame and may move differently from the dimmer objects. Second, we assumed that the model is constant, but it could actually change with time. Because our testing frames were taken from the beginning of the time series, the early training frames may be more helpful, whereas the later training frames may be governed by a slightly different model.

To investigate whether photobleaching was a major reason for the observed decreases in model accuracy in Fig. 5.5, we compared time series with extensive photobleaching to those with little photobleaching. In general, we found that for time series with little photobleaching, the model accuracy continued to increase throughout the duration of acquisition. The dashed blue line in Fig. 5.6 shows an example of this. In contrast, for time series with extensive photobleaching, the model accuracy decreased significantly towards the end, supporting the hypothesis that photobleaching causes apparent changes in the model. We also tried reversing the order of our

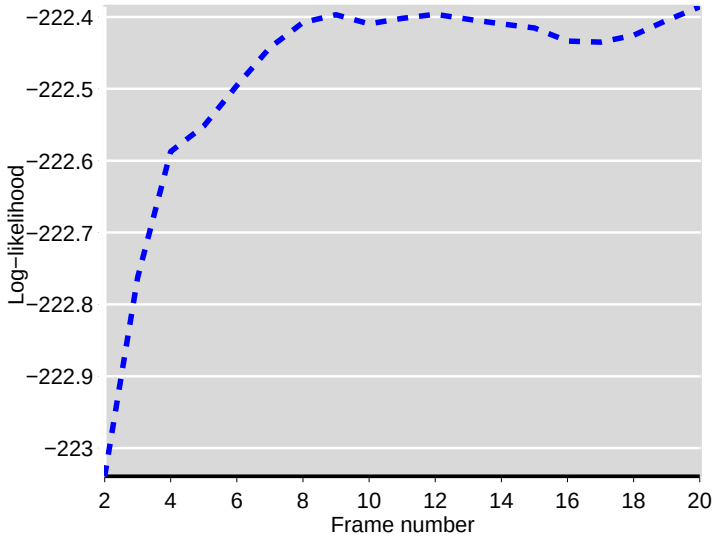


Figure 5.5: Model building (multiple objects, real data). The y-axis shows the model accuracy, estimated by the log-likelihood of the model given the observed displacements in the testing frames. The x-axis shows the number of frames used to build the model. We can see that the model accuracy increases, and then levels off, as more frames are acquired.

training frames, building the model from the last frame and working backwards. The model accuracy for this experiment is given by the dotted black line in Fig. 5.6. We can see that the model is still learned quickly, but is slower than when the training frames were in forward order. This may indicate that the model changes slightly over the acquisition period, or may just be because of minor photobleaching.

5.3.2 Modeling Distance Instead of Displacement

Until now, we have been modeling the displacement between an object in t and a nearby object in $t + 1$. However, a useful simplification is to focus solely on the distance between the objects, and to ignore the direction. Hence, (x, y, z) is replaced with d, where $d = \sqrt{x^2 + y^2 + z^2}$. This reduction in dimensionality speeds up

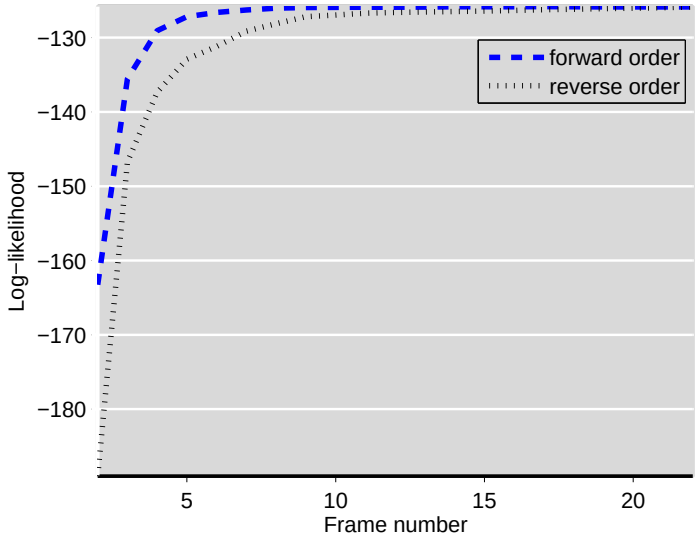


Figure 5.6: Model building (multiple objects, real data). The y-axis shows the model accuracy, estimated by the log-likelihood of the model given the observed displacements in the testing frames. The x-axis shows the number of frames used to build the model. The dashed blue line shows the case when the training frames are added in forward order. The dotted black line shows the case when the training frames are added in reverse order. The rate of increase follows a similar trend, although adding frames in reverse order is initially slower because the model is being learned on frames that are farther in time from the testing frames.

model learning. More importantly, using distance instead of displacement means that the end model is rotation invariant. This is useful when we want to compare cell models, or to combine them to form a class model. Therefore, we use this simplification exclusively in such situations.

5.3.3 Modeling Object Types

We now enhance our model by considering static features of the objects. Specifically, we consider each object's type. Initially, we look at how knowledge of an object's type helps us predict the locations

of its nearby objects more accurately. Then, we look at predicting both the locations and the types of those nearby objects.

Building the Model

To incorporate object types into our displacement model, we build a separate model for each object type. Hence, we record the number of times that each displacement (x, y, z) is observed for each object type λ, and we denote this as $N_{\lambda,x,y,z}$. Then, $N_{\lambda,\emptyset}$ is the number of times in which an object of type λ in frame t had no nearby objects in frame $t+1$, and N_λ is the total number of objects of type λ in the time series. The revised model can now be estimated as follows (compare this to (5.1)-(5.2)):

$$m = (m_\lambda, m_{\lambda,x,y,z}, m_{\lambda,\emptyset}), \tag{5.4}$$

where:

$$m_\lambda = \frac{N_\lambda + 1}{N + 2} \tag{5.5a}$$

$$m_{\lambda,x,y,z} = \frac{N_{\lambda,x,y,z} + 1}{N_\lambda + 2}, \tag{5.5b}$$

$$m_{\lambda,\emptyset} = \frac{N_{\lambda,\emptyset} + 1}{N_\lambda + 2}. \tag{5.5c}$$

As explained earlier, these are the Bayes estimates where the cost function is mean square error (see Section 3.1.2), and under the assumption of a uniform prior. We note that this model m also includes m_λ, which models the proportion of each type in the time series.

We can expand on this model even further by predicting not just the locations of nearby objects in the subsequent frame, but also their types. To do this, we assume that the displacement and the type transition are conditionally independent given the initial type: that is, we assume $p(\lambda_{t+1}, x, y, z|\lambda_t) = p(\lambda_{t+1}|\lambda_t)p(x, y, z|\lambda_t)$ where λ_t is the object type in frame t, λ_{t+1} is the type of the nearby object in frame $t+1$, and (x, y, z) is the displacement between the two objects. The conditional independence assumption is solely to reduce the dimensionality of the resulting model.

We have already described how to build the displacement model for a given type, and this remains unchanged. The only addition is to include a model of the type transition. For this, we iterate through every combination of types (λ, λ'), and count the number of objects

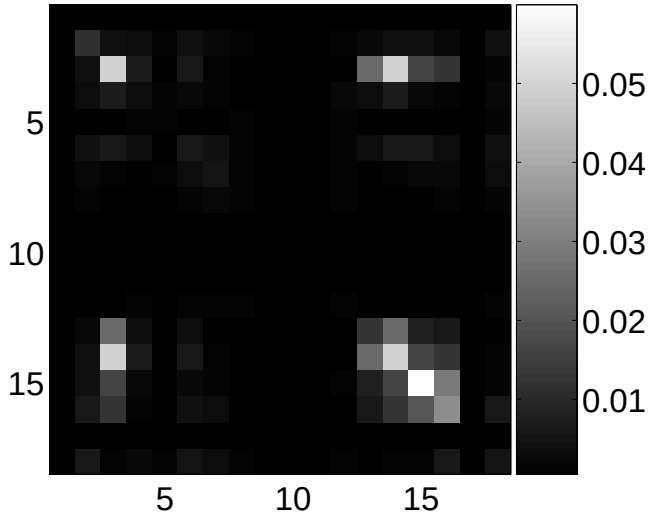


Figure 5.7: Type transition model. This image shows $m_{\lambda,\lambda'}$ for a Tctex1 cell. Only 13 of the 18 types are found in this cell, and $m_{\lambda,\lambda'}$ tends to be highest when $\lambda = \lambda'$ because objects tend to remain the same type between frames.

of type λ that have nearby objects of type λ' in the subsequent frame. The total number of such type transitions is denoted by $N_{\lambda,\lambda'}$. We then add the component $m_{\lambda,\lambda'}$ to our model, where $m_{\lambda,\lambda'}$ is given by:

$$m_{\lambda,\lambda'} = \frac{N_{\lambda,\lambda'} + 1}{N_\lambda + 2}. \tag{5.6}$$

Fig. 5.7 shows the values of $m_{\lambda,\lambda'}$ for a Tctex1 cell. Although we were using 18 types, only 13 of those types are found in this cell. We can see that $m_{\lambda,\lambda'}$ tends to be highest when $\lambda = \lambda'$, because objects tend to remain the same type between frames. However, in this example, we can also see that $m_{\lambda,\lambda'}$ is high for $(\lambda = 3, \lambda' = 14)$. This indicates either that objects easily switch between these two types, or that objects of these types are found close together in space.

To summarize, the final cell model consists of four components, $(m_\lambda, m_{\lambda,x,y,z}, m_{\lambda,\emptyset}, m_{\lambda,\lambda'})$:

1. The first component, m_λ, is a k-by-1 vector representing the proportion of objects of type λ.

2. The second component, $m_{\lambda,x,y,z}$, is a k-by-n-by-n-by-n tensor representing the proportion of objects of type λ that have a nearby object at a displacement (x, y, z) in the subsequent frame (where d_{max} spans n pixels).

3. The third component, $m_{\lambda,\emptyset}$, is a k-by-1 vector representing the proportion of objects of type λ that have no nearby objects in the subsequent frame.

4. The fourth component, $m_{\lambda,\lambda'}$, is a k-by-k matrix representing the proportion of objects of type λ that have a nearby object of type λ' in the subsequent frame.

Validating the Model

In Section 5.3.1, we built a model using a training set (consisting of frame 3 onwards), and then measured the log-likelihood of this model given the observed displacements in a testing set (consisting of frames 1 and 2). We now repeat this experiment, but show that we can get an even higher log-likelihood when we use knowledge of an object's type. We thus replace $m_{x,y,z}$ and $m_\emptyset$ with $m_{\lambda,x,y,z}$ and $m_{\lambda,\emptyset}$. We performed this experiment on the same Adfp cell as was used for Fig. 5.5, and plot the new results in Fig. 5.8. The dashed blue line (model accuracy without object type information) is the same in both figures. The solid red line shows the model accuracy with object type information. We can see that initially the accuracy is lower, and this can be explained by the increased number of parameters that must be learned. However, as more frames are included, the accuracy of the model with object type information surpasses the accuracy of the model without object type information.

As mentioned earlier, we need to choose k, the number of object types. We can do this by testing many values of k and choosing the one which maximizes the log-likelihood of the model given the testing frames. For the above experiment, Fig. 5.8, this happened at $k = 4$, which is the value used to make the solid red line in that figure. In subsequent sections, k is much higher (usually 18 or so), because we are grouping objects across all cells and classes, instead of across a single cell.

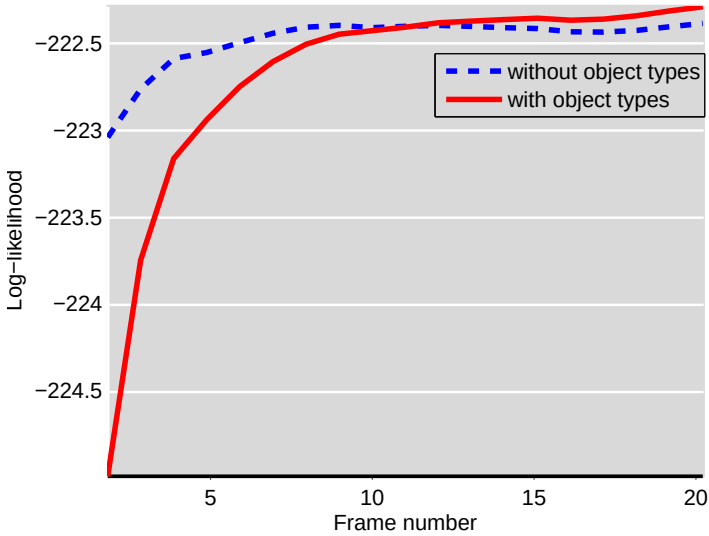


Figure 5.8: Model building with object types (multiple objects, real data). The y-axis shows model accuracy, measured by the log-likelihood of the model given the observed displacements in the testing frames. The x-axis shows the number of frames used to build the model. The dashed blue line is the case without object type information (as was shown in Fig. 5.5), whereas the solid red line is the case with object type information. Although the model accuracy is initially lower with object type information due to the extra parameters that must be learned, this accuracy eventually surpasses that of the model built without object type information.

Before concluding our discussion of cell models, we run a validation experiment on the complete model, $(m_\lambda, m_{\lambda,x,y,z}, m_{\lambda,\emptyset}, m_{\lambda,\lambda'})$. Once again, we set aside the first two frames as testing frames, and measure the likelihood of our model given the observed objects in testing frame 1 and the corresponding nearby objects in testing frame 2. The difference from the previous experiments is that we now consider the types of the nearby objects in frame 2—effectively measuring the ability of our model to predict the types of nearby objects as well as their displacement. We plot the resulting log-likelihood in Fig. 5.9, and as before we see the model accuracy steadily increasing

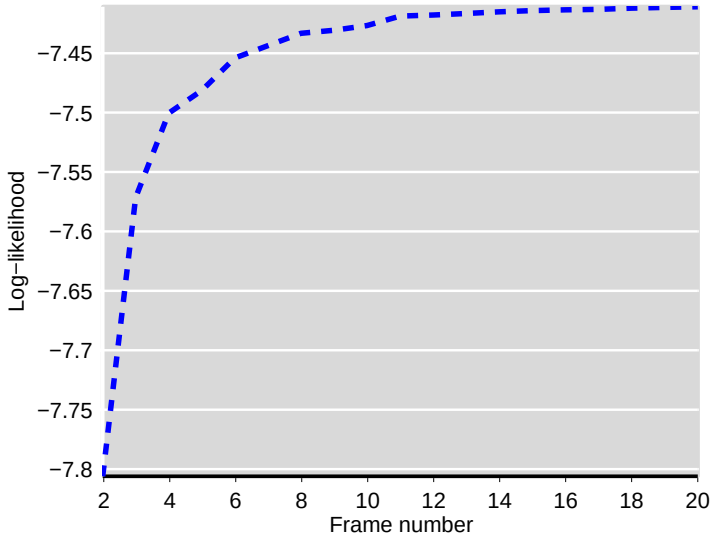


Figure 5.9: Model building with complete model (multiple objects, real data). The y-axis shows model accuracy, measured by the log-likelihood of the model given the observed displacements and corresponding object types in the testing frames. The x-axis shows the frame number. We can see the model accuracy increasing, and then leveling off, as more frames are acquired.

with each training frame acquired.

5.4 Class Models

5.4.1 Building the Model

A class model is simply the collection of cell models from that class. Hence, we can view the class model as a mixture model, where its constituent cell models form the components of that mixture. To build the class model, we need to build a cell model for each available time series in the class, modeling distance instead of displacement as in Section 5.3.2. We discuss alternative ways to build class models in Section 5.6.

5.4.2 Validating the Model

The primary way to validate a class model is with classification, as we discuss in Section 5.5. However, we can also use a log-likelihood method similar to that described in Section 5.3. The procedure is to build a class model using K training cells of that class. We then measure the likelihood of this model given an observed testing cell of the same class. As K increases, we expect the likelihood of the resulting model to also increase.

To find the log-likelihood of a class model given an observed cell model, we first measure the likelihood of each of the class's constituent cell models. This is similar to the process used in Section 5.3, except now we can use all frames from the testing cell as testing frames. The likelihood of a constituent cell model m' given the observed cell model m is found by looking at each component of the model individually:

$$l = l_\lambda, l_{\lambda,\lambda'}, l_{\lambda,d}, l_{\lambda,\emptyset}, \tag{5.7}$$

where:

$$l_\lambda = \prod_{\lambda=1}^{k} (m_\lambda)^{N_\lambda}(1-m_\lambda)^{(N-N_\lambda)}, \tag{5.8a}$$

$$l_{\lambda,\lambda'} = \prod_{\lambda=1}^{k}\prod_{\lambda'=1}^{k} (m_{\lambda,\lambda'})^{N_{\lambda,\lambda'}}(1-m_{\lambda,\lambda'})^{(N_\lambda-N_{\lambda,\lambda'})}, \tag{5.8b}$$

$$l_{\lambda,d} = \prod_{\lambda=1}^{k}\prod_{d=1}^{d_{max}} (m_{\lambda,d})^{N_{\lambda,d}}(1-m_{\lambda,d})^{(N_\lambda-N_{\lambda,d})}, \tag{5.8c}$$

$$l_{\lambda,\emptyset} = \prod_{\lambda=1}^{k} (m_{\lambda,\emptyset})^{N_{\lambda,\emptyset}}(1-m_{\lambda,\emptyset})^{(N_{\lambda_t}-N_{\lambda,\emptyset})}, \tag{5.8d}$$

where m_λ, $m_{\lambda,d}$, $m_{\lambda,\emptyset}$, $m_{\lambda,\lambda'}$, are from (5.5)-(5.6) respectively, but replacing (x, y, z) with d because we are modeling distance instead of displacement. We then take the likelihood of the class model, l_C, to be the mean of the likelihoods of its constituent cell models, l_i. Hence:

$$l_C = \frac{1}{n_C}\sum_{i\in C} l_i, \tag{5.9}$$

where there are n_C cell models making up class model C. In practice, one term generally dominates this expression, and we observed no

change in accuracy when replacing (5.9) with (5.10):

$$l_C = \max_{i \in C} l_i, \tag{5.10}$$

This latter formulation is more convenient when working with log-likelihoods.

We now perform the validation experiment, varying x from 1 up to the total number of cells for that class (leaving one cell for testing). The results of this experiment are highly dependent on the specific training and testing cells chosen, and so we average them over 100,000 trials, randomly choosing the training and testing cells each time. In Fig. 5.10, we show the results for the Hspa9a class. The solid red line shows the log-likelihood of the model increasing as more training cells from the Hspa9a class are used to build it. As a comparison, the dotted black line shows the results when we use an Hspa9a cell for testing, but use training cells from the Tctex1 class. Not surprisingly, using training cells from a different class gives a much lower log-likelihood, and this forms the basis of the classifier that we now describe.

5.5 Classification

We now discuss how to classify a time series of an unknown protein, using only the cell model of that time series and the class model of each protein. Successful classification provides an additional source of validation that our cell models and class models capture relevant discriminative information. We have already shown, in Section 5.4, how to find the likelihood of a class model given a particular cell model. For classification, we simply do this with all 12 class models and choose the class of maximum likelihood.

We evaluate our classification method using leave-one-out cross-validation (see Section 3.3.3), giving a classification accuracy of 84.5% on our data set. We described the previous best classifier on this data set in Section 3.5.1, which used a range of morphological, texture, and temporal features, yielding an accuracy of 83.9%. Therefore, in bettering the classification accuracy, we verify that our models are capturing relevant discriminative information present in the time series.

The number of object types, k, was chosen to maximize classification accuracy on the training set. This was achieved with $k = 18$

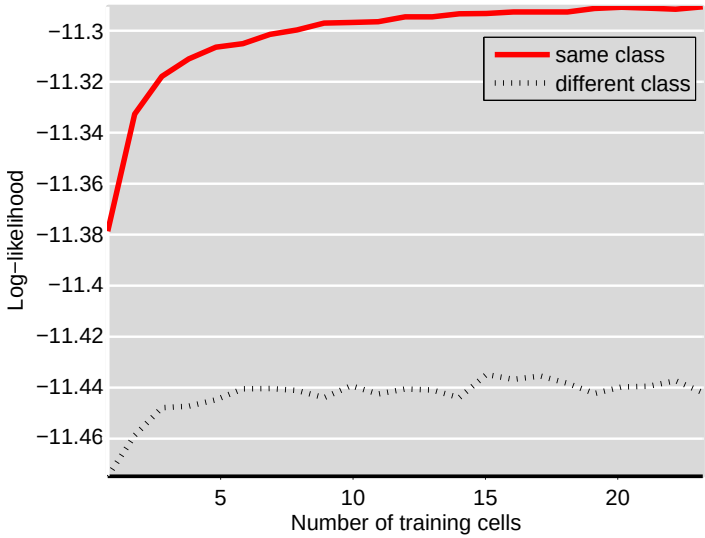


Figure 5.10: Model building (class models, real data). The y-axis shows the model accuracy, measured by the log-likelihood of the model given an observed Hspa9a testing cell. The x-axis shows the number of training cells used to build this model. The solid red line shows the case when the training cells are also from the Hspa9a class. The dotted black line shows the case when the training cells are from a different class (Tctex1). As expected, the accuracy is much higher when the training cells are from the same class as the testing cell.

for every fold of the leave-one-out cross-validation. However, the accuracy was not too sensitive to the choice of k. Anywhere in the range of $12-24$ resulted in a classification accuracy of at least 82.2% (see Section 5.6).

If we consider only one component of our model—the proportion of objects in each type (m_λ)—we can get 67.4% accuracy. Adding the components ($m_{\lambda,d}, m_{\lambda,\emptyset}$), which find the joint probability of the initial object type and the distance moved, we get 76.3% accuracy. The addition of the type transition model ($m_{\lambda,\lambda'}$) completes the model, giving our final result of 84.5%. These results are summarized in the bottom part of Table 5.3.

We recall from Section 5.2 that object types are assigned on the basis of 7 static features, which were listed in Table 5.2. The top part

of Table 5.3 shows the classification accuracy when only one of these static features is used to determine the object type. The accuracy ranges from 68.8% to 78.6% depending on the feature used. The most informative features were 3D-SOF1.1, the number of voxels in the object, and 3D-SOF1.9, the fraction of object voxels contained within the skeleton. However, using any one of these features to assign object types is significantly better than not assigning object types at all. When the model considered only the distance between an object and its nearby objects (and not the object types), classification accuracy dropped to 47.4%.

The confusion matrix for our best classifier is shown in Table 5.4. In both our method and in the method of the aforementioned previous work [26], Dia1 & Sdpr were commonly confused. However, the other classes of confusion differed between the two methods. For example, the previous work reported high confusion between Timm23 & Cav, with the resulting accuracies of these proteins being 70% and 68% respectively. In contrast, our method had no confusion between these classes, and their accuracies were 90% and 100% respectively. This suggests that, in some cases, the two methods are finding different discriminative information, and there may be potential to combine the classifiers to get a more accurate overall result.

Configuration	**Classification Accuracy**
Previous work [26]	83.9%
Distance	47.4%
Distance & 3D-SOF1.8	68.8%
Distance & 3D-SOF1.11	69.1%
Distance & 3D-SOF1.6	72.0%
Distance & 3D-SOF1.7	73.4%
Distance & 3D-SOF1.4	74.3%
Distance & 3D-SOF1.1	78.0%
Distance & 3D-SOF1.9	78.6%
Proportion of object types	67.4%
Object types & distance	76.3%
Object type transitions & distance	**84.5**%

Table 5.3: Classification accuracy for different configurations.

	Tctex1	Actn4	Cald1	Anxa5	Glut1	Hspa9a	Cav	Dia1	Sdpr	Atp5a1	Adfp	Timm23
Tctex1	**100**	0	0	0	0	0	0	0	0	0	0	0
Actin4	7	**79**	0	3	3	0	0	7	0	0	0	0
Cald1	13	0	**69**	6	0	0	0	0	6	0	0	6
Anxa5	0	22	0	**78**	0	0	0	0	0	0	0	0
Glut1	6	12	0	0	**65**	12	0	0	0	6	0	0
Hspa9a	0	4	0	0	4	**75**	0	4	4	4	0	4
Cav	0	0	0	0	0	0	**100**	0	0	0	0	0
Dia1	0	5	0	5	5	10	0	**65**	10	0	0	0
Sdpr	0	0	4	4	0	4	0	22	**65**	0	0	0
Atp5a1	5	5	5	0	0	0	0	0	5	**80**	0	0
Adfp	0	0	0	0	0	0	0	0	0	2	**98**	0
Timm23	0	0	0	0	0	5	0	0	5	0	0	**90**

Table 5.4: Confusion matrix of the best classifier. Each row corresponds to the true class, and each column corresponds to the predicted class. All numbers are percentages.

5.6 Discussion

In this section, we discuss alternative methods, extensions, and other considerations related to the multiple-object model-building.

Computation time. The modeling method outlined in this chapter has a fast computation time relative to the acquisition time. The object detection and feature extraction stages can be done in well under a second (Intel Core Duo 2.2GHz processor with 1.96GB of memory), and in real time while the frame is being acquired. The actual model building takes only 50ms per frame. This compares to an acquisition time of 45s per frame.

Higher-order models. Although we have shown that the modeling is sufficient to classify with comparable accuracy to previous work on this data set, there is still a lot of information that the model does not capture. In particular, it only looks at the locations of objects relative to their locations in the previous frame, meaning that it cannot distinguish between objects following a random walk model and objects following a constant velocity model. One

way to capture such higher-order information would be to extend the method to look at sets of three consecutive frames, rather than only two. However, the most detailed analysis on an object level would come from extending the particle filter method of Chapter 4 to properly handle multiple-object models. This was discussed in Section 4.4.

Object types. Grouping the objects into 18 types gave us our highest classification accuracy, 84.5%. Fig. 5.11 shows the accuracy when we group objects into any number of types in the range $1 - 30$. We can see that increasing the number of types gives a big increase in accuracy up to 8 types, but increasing the number of types beyond this point gives only a small improvement. This suggests that there are 8 major types, and beyond this we are splitting those major types into smaller sub-types, or perhaps identifying new but relatively uncommon types. The accuracy drops steadily when we increase the number of types beyond 22 or so. This might be avoided if we used a hierarchy of object types, allowing the major types to be identified, but retaining information from the smaller sub-types.

Although these object types were computed from all 7 of the static features given in Table 5.2, we still achieve a classification accuracy of 82.2% when we compute the types using only the features 3D-SOF1.1 and 3D-SOF1.4 (size and eccentricity). The addition of the features 3D-SOF1.7 and 3D-SOF1.9 takes us to 84.2% accuracy. If we use all features except for 3D-SOF1.6 (roundness), we get 84.9%, which is slightly higher than the 84.5% achieved when using all of the features. However, to conclusively determine the optimal combination of static features, we would need to repeat these experiments with a larger data set.

Restriction to same-type displacements. We also note that although we build a separate displacement model for each object type, these models are still based on displacements between objects of different types. An alternative method would be to base the model only on displacements between objects of the same type. This could increase the probability that a recorded displacement actually corresponds to the motion of a single object between consecutive frames. Unfortunately, we found that the models built with this method only gave a classification accuracy of 80.3% (compared to 84.5% with the earlier model). This lower accuracy may be because of the increased data sparsity that results from reducing the number of recorded displacements. It could also indicate that objects do actually change

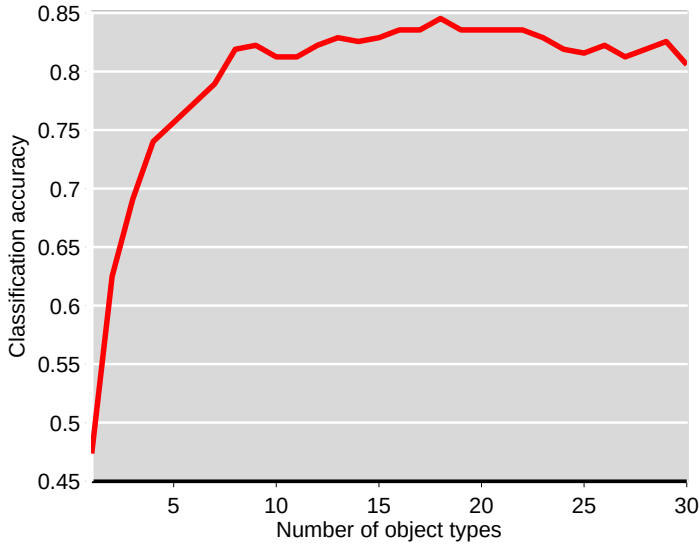


Figure 5.11: Number of object types. This graph shows the resulting classification accuracy when objects are grouped into a different number of types. The highest accuracy of 84.5% occurs with 18 types.

types between frames, whether due to errors in the object detection and type assignment process, or due to a physical change in that object. We can address this latter possibility by weighting the displacements according to their probability under the model, including the type transition model, as we now describe.

Weighting displacements. The model-building method described in this chapter records every observed displacement between frames, even if the same object is involved in two or more displacements. For example, if an object in frame t has n nearby objects in frame $t+1$, each at a different displacement, all n displacements are recorded with a weighting of 1. We also tested an alternative method that weighted the displacements, giving a cumulative weighting of 1 to all n displacements associated with a particular object in frame t. The simplest way to do this is to give each of them a weighting of $1/n$, but a better method is to weight them according to their probability under the current model. We then iterate between recalculating the model using the recorded displacements and their weights, and up-

dating the weights based on this recalculated model. This process is shown in Algorithm 1.

The first advantage of this method is that it typically gives more weight to displacements between objects of the same type, because these have a stronger probability of occurring under the type transition model (see Fig. 5.7). The second advantage is that it naturally gives more weight to smaller displacements as these typically have higher weighting under the displacement model (see Fig. 5.4). In fact, with this method, we do not even need to define a maximum displacement distance d_{max} because large displacements are naturally given small weighting.

Unfortunately, this alternative model-building method only gives a classification accuracy of 80.6%, whereas our chosen method gave 84.5%. The reason for this may be that our chosen model-building method implicitly captures which object types occur close together in space, which could be useful discriminative information. If we could add this information into the alternative mdoel building method, it might give higher classification accuracy. However, a larger dataset would be necessary to confidently test this hypothesis.

Because this method associates an overall weighting of 1 with the displacements from each object, it can be used to predict the probability with which pixels in a subsequent frame will contain objects. We test this in Fig. 5.12 by setting aside the first two frames as testing frames, and measuring the log-likelihood of the model given the locations of objects in these testing frames. We can see that this log-likelihood increases as more training frames are used to build the model. We also show that we get a lower log-likelihood if we use a uniform model (objects are equally likely to move anywhere up to distance d_{max} from their original position) or a Gaussian model of zero mean and equal variance in each dimension.

Class model extensions. Finally, we note that we simply represented class models as a mixture model, where each of the constituent cell models made up the components of that mixture. This formulation effectively assumes no similarity between the different cells in a class. At the other extreme, we could make a single model—either by taking the mean of all the cell models, or by concatenating all frames across all cells and then building a model from this concatenation. We found that such methods gave significantly lower results, which is why we prefer the mixture model method. However, a third and promising method lies between these two extremes, which is to

Algorithm 1
Input: locations of objects in frames $1, ..., t$.
Output: m, the model, a function giving the probability of each displacement.

```
for all objects o in frames 1, ..., (t − 1) do
  set D_o to the set of possible displacements for o
  initialize W_o to give constant weight to each displacement in D_o
end for
repeat
  learn m from (D,W)
  for all objects o in frames 1, ..., (t − 1) do
    for all displacements d in D_o do
      set W_o(d) to m(d)
      normalize W_o
    end for
  end for
until m no longer changing
return m
```

cluster the cells and then make a single model of each cluster. This allows us to exploit that many cells in a class will follow the same model, but without assuming that all cells in a class follow the same model.

A clustering of the cells requires that we measure the similarity between two cell models. This can be done by finding the log-likelihood of one model given the other, averaging to remove the assymetry. Cell models could be clustered within a class, or could be clustered across classes as we did when assigning types to each object. If we used the latter method, we would end up defining a set of cell types, and each class would contain a certain proportion of each cell type. Another method again is to define motion types on the object level, but this would require modeling each object individually, which we do not yet do in the multiple-object scenario.

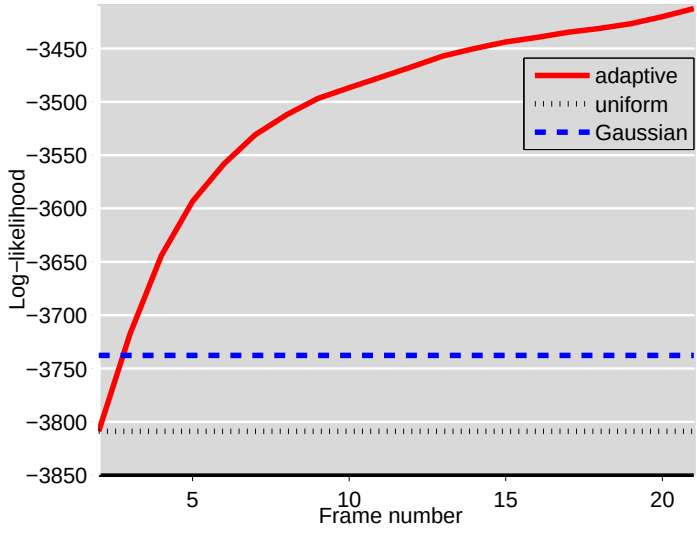


Figure 5.12: Alternative model-building method with weighted displacements (multiple objects, real data). The y-axis shows the log-likelihood of the model given the two observed testing frames. The x-axis shows the number of frames used to build the model. The solid red line shows the model learned using Algorithm 1. The dotted black line shows a uniform model that assumes objects are equally likely to move anywhere within a distance of d_{max}. The dashed blue line shows a model that assumes object motion is governed by a zero-mean Gaussian distribution with equal variance in each dimension. The learned model quickly surpasses the non-adaptive models.

Part III

Intelligent Acquisition

The goal of this part is to acquire cells and classes of cells intelligently so that we can build models as efficiently as possible. We try to maximize model accuracy while minimizing both the acquisition time and the photobleaching incurred during acquisition. The four chapters in this part each discuss a different acquisition strategy: where to acquire in a frame, when to acquire frames, when to stop acquiring frames, and how many cells to acquire.

The first two strategies—choosing where to acquire in a frame and choosing when to acquire frames—are only developed for the single-object scenario. The reason for this is that our multiple-object model-building method requires that every pixel in the frame is known and that the frame-rate is constant. These requirements are not compatible with an acquisition module that acquires only a subset of pixels in a frame, or varies the time at which acquisitions are made. Our method for determining when to stop acquiring frames is tested on both the single-object scenario and the multiple-object scenario. Furthermore, we show that this strategy differs depending on whether we are building a cell model or a class model. The method for determining how many cells to acquire is only applicable for class models, and so we test this method only on the multiple-object scenario (we did not develop class models for the single-object scenario).

With all of these strategies, we tried to develop methods that were not closely tied to the specific model-building procedure used. We believe that these strategies capture general principles that could be applied in many scenarios that require intelligent acquisition.

6

Where to Acquire in a Frame

This chapter looks at which pixels to acquire in each frame. Every time we acquire a pixel, photobleaching can occur, which eventually destroys the fluorescent signal. Our goal is to choose pixels such that we gain as much information as possible before this happens. We restrict discussion in this chapter to single-object models, because our multiple-object model-building algorithm requires the full set of pixels.

When choosing where to acquire, we want to find a combination of pixels that gives a lot of information about the model, but does not incur much photobleaching. To evaluate this, we present a method to estimate the *reward* and the *cost* of an acquisition. The reward is the amount of information that we expect to gain from the acquisition; because we evaluate the accuracy of single-object models using the log-likelihood of the true underlying model, we define the reward as the expected increase in this log-likelihood. The cost is made up of two components: the photobleaching cost, which estimates the expected photobleaching that will result from the acquisition, and the time cost, which is a constant cost associated with every frame. We search for the set of pixels that maximizes the reward relative to the cost.

6.1 Cost Evaluation

First, we look at the cost associated with an acquisition. Our cost function reflects two goals: to gain information as quickly as possible, and to gain as much information as possible before photobleaching destroys the fluorescent signal. Photobleaching occurs every time a pixel containing an object is acquired. Additionally, in 3D imaging with a laser scanning confocal microscope, acquisition of a pixel also causes photobleaching for objects in the out-of-focus planes. As an approximation, we assume that photobleaching of an object is proportional to the number of times any pixel with the same xy-coordinates of that object is acquired, regardless of the z-coordinate (this assumes constant illumination intensity and exposure time). In future, a better approximation would use a double-cone model to reflect the focusing of light onto a single pixel. Note that if we were to lower the illumination intensity or shorten the exposure time then we would reduce photobleaching. Adjusting these quantities could form part of the intelligent acquisition module. However, in our experiments we assume constant illumination intensity and exposure time because we do not have a good model of how varying these quantities would affect the quality of the image.

We do not define a generic phototoxicity model because it depends on the type of cells being imaged, and furthermore, phototoxicity is harder to measure directly. For the purposes of our simulations, we simply assume that the phototoxicity cost is proportional to the photobleaching cost.

The probability of a pixel containing an object is given by the proportion of particles in that pixel. Defining $Z(p)$ as the set of pixels with the same xy-coordinates as pixel p, and N_p as the number of particles in p, the cost of a frame acquisition is given by:

$$C = \tau t + \rho \sum_{p \in P} \frac{\sum_{p' \in Z(p)} N_{p'}}{N}. \tag{6.1}$$

In this equation, the first term reflects the time cost: t is the elapsed time since the last acquisition, and τ is the cost per unit time. If τ is high, then the system will try to minimize cost by finishing acquisition as quickly as possible. The second term is proportional to the overall probability of detecting an object using the set of pixels P, combined with the probability of acquiring any pixel with the same

xy-coordinates as the object. ρ represents the photobleaching cost associated with each exposure of the object, and so this second term reflects the expected photobleaching cost of the acquisition. The ratio τ/ρ determines the relative importance of minimizing acquisition time versus minimizing photobleaching.

6.2 Reward Evaluation

We now look at how to predict the reward—the expected increase in model accuracy—that will result from an acquisition. To do this, we first look at how the intelligent acquisition module can evaluate the model accuracy (which is the log-likelihood of the true underlying model) at any frame.

To make an estimate of the model at frame t, we use the models of all the surviving particles, as shown by (4.3). Ideally, the intelligent acquisition module could then evaluate the accuracy of this model using (4.7), which calculated the log-likelihood of the true underlying model. However, it is not possible to do this because the true underlying model is unknown. Instead, we can estimate the accuracy by finding the log-likelihoods of the models of each surviving particle. Assuming that each surviving particle's model is equally likely to be the true model, we can then estimate the model accuracy at t as the mean of these log-likelihoods:

$$E[log(l)_{t,e}] = \frac{1}{N} \sum_{q} log(l)_{q,e}, \tag{6.2}$$

where $log(l)_{q,e}$ is the log-likelihood of the model of particle q given the estimated model.

More importantly, we wish to estimate the expected increase in model accuracy following a particular acquisition. We consider this for the case of acquiring a single pixel p in frame $t+1$. There are two possible outcomes: that the object will be found in p, or that it will not be found in p. We can estimate the probability of each outcome by looking at the proportion of particles that move to p in frame $t+1$.

If the object is found in p then only those particles that moved to p in frame $t+1$ will survive. Our model estimate under this outcome will be found by applying (4.3) over those particles. Similarly,

our estimated model accuracy under this outcome can be found by applying (6.2) over those particles.

We can also predict the model accuracy under the outcome that the object is not found in p. We follow the same procedure as before, except now we use those particles that did *not* move to p (essentially, we are considering all pixels other than p as one super-pixel). Hence, we have now worked out the expected increase in model accuracy under the two possible outcomes, and because we know the probability of each outcome, we can estimate the reward associated with acquiring pixel p. We can extend this method to find the reward associated with acquiring a whole set of pixels simply by considering in turn the outcome that the object is found in each given pixel, as well as the outcome that the object is not found.

We may expect that the reward of acquiring a single pixel is proportional to the probability that that pixel contains the object. However, this is not the case. In Fig. 6.1, we simulate an object moving in 1D under an RW model. We do this in 1D solely for plotting purposes—a similar result can be shown in 3D. Plot (a) shows the first frame of the simulation, where the solid red line plots the reward associated with acquiring a pixel, and the dashed blue line plots the probability of finding the object in the given pixel. Although these curves are close to each other, their shapes are clearly different. We can see that there are two distinct points where the reward curve drops to nearly zero while the probability curve remains relatively high. The reason for this is that, even if the object is found in these pixels, we gain very little information about the rate parameter, Σ (which is just a single number in the 1D case). To understand this, Fig. 6.2 shows 10 Gaussian distributions with standard deviations between 1 and 3. We can see that all of these curves come very close at two distinct points, and thus observing such a displacement gives little information about the true underlying Gaussian distribution.

In Fig. 6.1(b), we consider the ratio of reward to cost, which we define as the benefit. When learning the model quickly is of paramount importance (that is, the time cost is high), the benefit curve follows that of the reward curve and is shown by the solid red line. However, when minimizing photobleaching is of paramount importance, the benefit curve instead follows that of the dotted black line. We can see that, theoretically, the highest benefit results from acquiring pixels that are far from the object—although these pixels do not give much information, they are even less likely to cause pho-

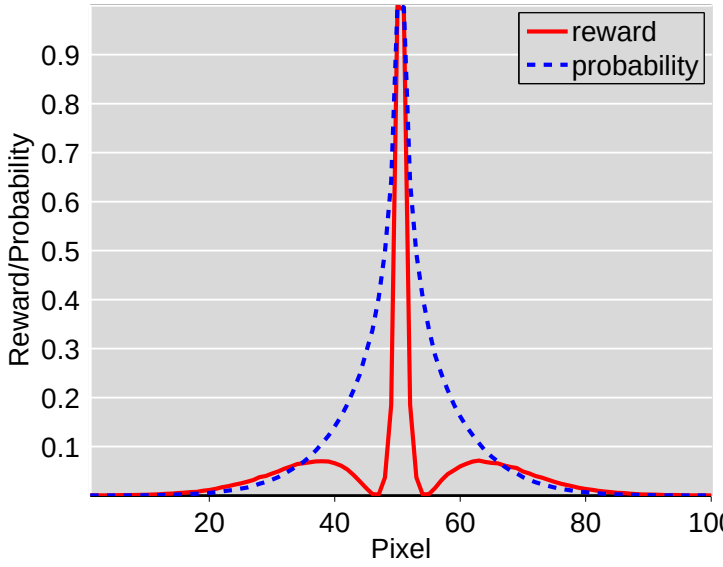
tobleaching, and thus the ratio of reward to cost is high. However, in practice, we always have some time cost associated with acquisition, and even a small time cost pulls the benefit curve back towards the solid red line. For example, when the time cost of one frame is defined to be 10% of the photobleaching cost of one exposure, the benefit curve follows the dashed blue line.

6.3 Choosing the Pixels

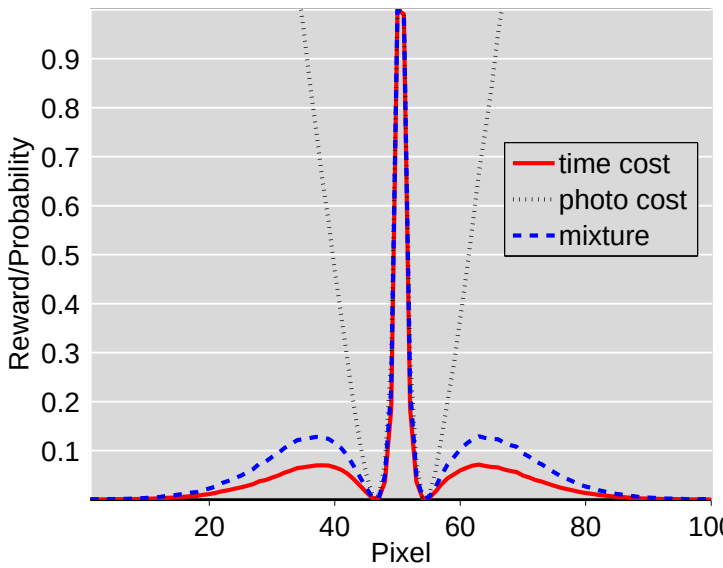
The actual acquisition algorithm aims to choose the set of pixels that maximizes the benefit (which we defined as the reward relative to the cost). We take a greedy approach and only try to maximize this benefit in the immediately subsequent frame. Note that this is not guaranteed to maximize the benefit in the long-term.

Even maximizing the benefit in just the subsequent frame is a computationally daunting task. In any given frame of N pixels, there are 2^N combinations of pixels we could acquire. Clearly it is too resource-intensive to estimate the reward and cost for all of these combinations, and so instead we use a greedy procedure in which we continue adding the pixels with the highest marginal benefit to our set until the overall benefit stops increasing. Algorithm 2 describes this in more detail.

To test this algorithm, we simulate an object track as described in Section 4.1. We then simulate acquisition of this data set using five different acquisition algorithms, and try to learn the object's model. Our results, which are shown in Fig. 6.3, are averaged over 1,000 trials. Plot (a) shows the model accuracy (log-likelihood) against the frame number under these five different acquisition scenarios. The dotted black line shows the case where all pixels are acquired in a frame. Not surprisingly, this method learns the model the fastest. The solid red line shows the case where our intelligent algorithm (Algorithm 2) is used for acquisition, and where the only cost considered by this algorithm is the time cost. We compare this line to the dashed blue line, which acquires the pixels that are most likely to contain the object (rather than those with the highest reward). For both of these scenarios, the same number of pixels are acquired. We can see that the intelligent algorithm performs best for the first 10 frames, but is then outperformed by the algorithm that acquires the most likely pixels. We suspect that the reason for this is that the intelli-



(a)



(b)

Figure 6.1: Reward associated with a pixel acquisition. In this 1D example, the object is assumed to move under an RW model. Plot (a) shows the first frame in the simulation. The dashed blue line shows the probability of finding the object in a given pixel. The solid red line shows the reward associated with acquiring this pixel. Plot (b) shows the ratio of reward to cost. The solid red line shows this ratio when we only consider the time cost. The dotted black line

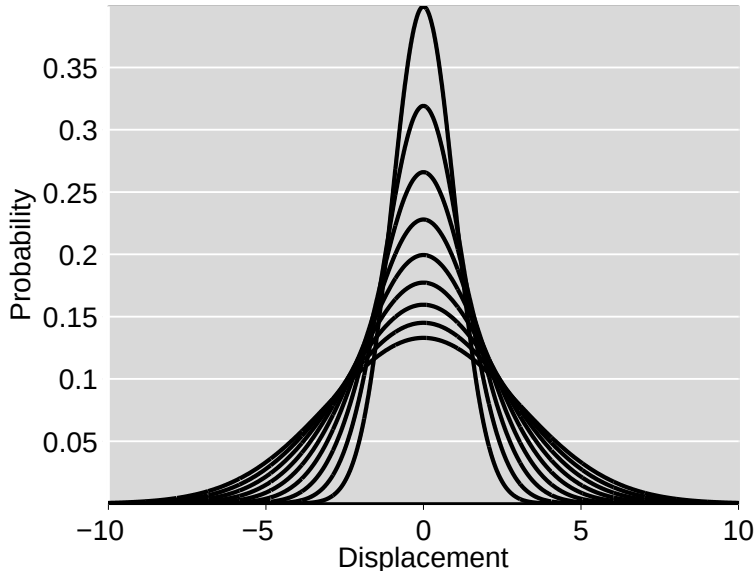


Figure 6.2: 10 Gaussian probability distributions, each with a different variance. These curves all come near two distinct points. If we observed a sample at one of these points, we would have little information about from which probability distribution the sample was drawn.

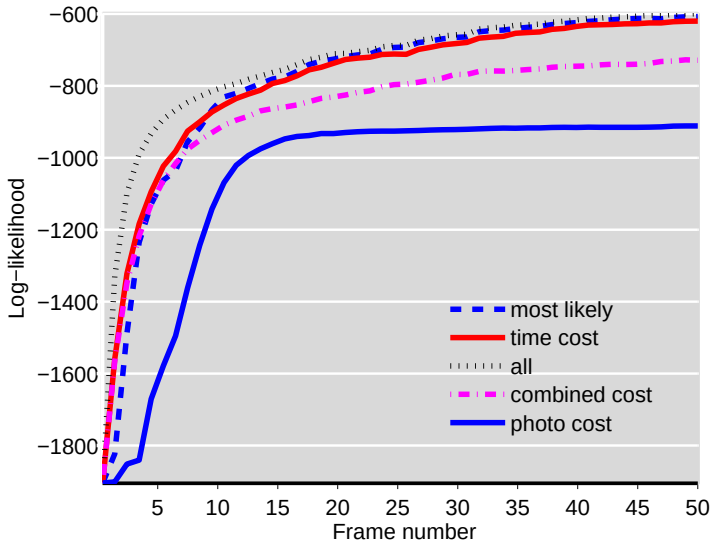
gent algorithm only maximizes the benefit in the subsequent frame. However, in the long-term, accurately knowing the object's location is important, and so acquiring the most likely pixels can perform better. Finally, the solid blue line shows results for our intelligent algorithm but where the only cost considered is the photobleaching cost. Model learning is slowest under this algorithm. The dash-dotted magenta line shows results for our intelligent algorithm when the time cost and the photobleaching cost are considered equal. As expected, this yields faster model learning than when time cost was ignored altogether (solid blue line), but is slower than when time was the only consideration (solid red line).

Fig. 6.3(b) shows the same results, but now the model accuracy is plotted against the photobleaching cost (instead of against the frame number). This shows us how well the model is learned for a given amount of photobleaching. We can see that all three configurations of our intelligent algorithm (considering just time cost,

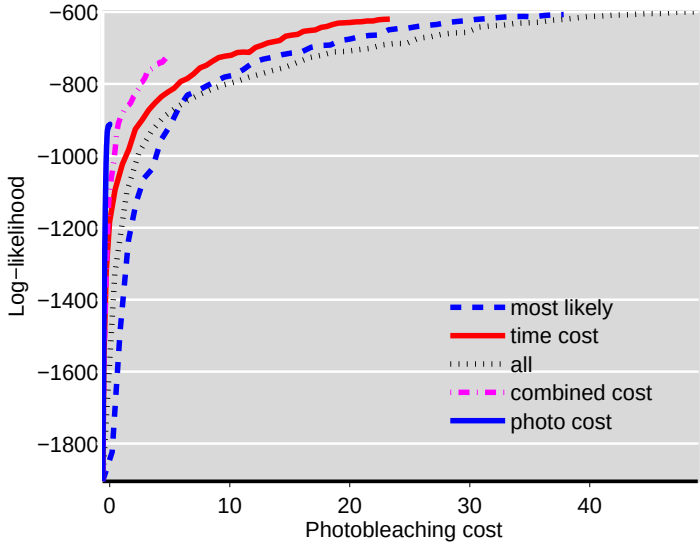
Algorithm 2
Input: the set of particles in frame t.
Output: P_{acq}, the set of pixels to acquire in frame t.

$P_{acq} = \emptyset$
$B = 1$
repeat
  for all pixels p in $\overline{P_{acq}}$ **do**
    set R_p to reward of acquiring $\{P_{acq}, p\}$
    set C_p to cost of acquiring $\{P_{acq}, p\}$
    $B_p = R_p / C_p$
  end for
  $p' = \text{argmax}_p \, B_p$
  if $B_{p'} >= B$ **then**
    $P_{acq} = \{P_{acq}, p'\}$
    $B = B_{p'}$
  end if
until $B_{p'} < B$
return P_{acq}

just photobleaching cost, and a mixture of the two) perform better than acquiring all pixels, or acquiring the pixels most likely to contain an object. This is an important experiment because the object can only be observed a certain number of times before its fluorescent signal is destroyed by photobleaching. Hence, these curves show how well the model can be learned before this happens. The curves in this figure stop at different points on the x-axis because, over the 50 frames, a different amount of photobleaching occurred for each of them. As expected, the intelligent algorithm that considers only the photobleaching cost (solid blue line) gives the highest model accuracy relative to photobleaching, but the slow learning of this algorithm meant that the model accuracy was still low after 50 frames. At the other extreme, model learning is fastest when we consider only the time cost (solid red line), but accuracy relative to photobleaching is lower. We can obtain curves in between these extremes by considering a mixture of time cost and photobleaching cost. An example, as mentioned, is the dash-dotted magenta line—this algorithm considers the photobleaching cost and the time cost to be equal.



(a)



(b)

Figure 6.3: Where to acquire (single object, synthetic data). These curves show the rate at which a model is learned under different acquisition strategies. Plot (a) shows log-likelihood against frame number. Plot (b) shows log-likelihood against photobleaching cost. The dotted black line shows the method where every pixel in every frame is acquired. The solid red line shows the intelligent algorithm (Algorithm 2) where only the time cost is considered. The dashed

6.4 Discussion

The main weakness of the intelligent algorithm is that it only tries to maximize the benefit in the immediately subsequent frame. This may cause it to underestimate the importance of keeping track of the object's location. We showed an example of this in Fig. 6.3(a), where the algorithm that focused on tracking the object's location (by acquiring the pixels most likely to contain the object) actually worked better than our intelligent algorithm after the first 10 frames. Ideally we would extend our algorithm to maximize the benefit in the subsequent k frames, but this is computationally infeasible with our current method. Probably the simplest way to improve our algorithm would be to adjust the benefit function to give more weighting to those pixels most likely to contain an object. The specific weights could be learned from a training set.

We did not apply our algorithms to the multiple-object scenario because the model-building module for multiple objects requires the full set of pixels. However, there is a very simple method that we can use in the multiple-object scenario: this method is to skip over a pixel if we are sure that it will not contain an object. This does not affect the model-building procedure because we can set the value of this pixel to zero, and hence still provide the model-building module with a full set of pixels as required. The benefit is that it reduces photobleaching of any objects with the same x,y-coordinates as the skipped pixel (regardless of the z-coordinate).

There are several other things to consider when acquiring a subset of pixels of real data. First, if objects have size and shape, we must consider the implications of acquiring a partial object, or splitting an object in two. Second, the object detection process may be more difficult if less background pixels are available. Third, we must consider whether the microscope is capable of acquiring an arbitrary set of pixels as we have assumed in this chapter, or whether, for example, it can only acquire rectangular regions. Despite these considerations, in scenarios where photobleaching is a major limitation, intelligently choosing where to acquire can undoubtedly yield more information from a cell.

7

When to Acquire Frames

In addition to deciding where to acquire, we must also decide when to acquire. If we wait longer to acquire a new frame, more will have changed, which may mean that the acquisition will provide more information. This is beneficial, because to reduce photobleaching we want to get as much information as possible from each acquisition. However, in some cases, waiting too long to acquire a frame results in losing information that could best be gained with high temporal resolution. Furthermore, we then take longer to learn the model, and in the case of multiple objects, tracking performance is degraded. However, we test our algorithm here solely on the single-object case.

7.1 Method and Results

In Chapter 6, we showed how to find the expected reward of acquiring any set of pixels in the subsequent frame, $t + 1$. If we expand this set to include every pixel in the image, then we have the expected reward of acquiring the entire frame $t + 1$. We can easily find the expected reward of skipping frame $t + 1$ and acquiring frame $t + 2$: To do this, we simply consider the positions of the particles in $t + 2$ instead of $t + 1$, and proceed in the same way.

We assume that acquiring any frame has the same photobleaching cost. Therefore, to maximize the reward relative to photobleaching cost, we just need to assess whether the expected reward of $t + 1$ is higher than that of $t + 2$; if so, we acquire frame $t + 1$, and if not, we skip $t + 1$, and then compare the rewards of acquiring $t + 2$ and $t + 3$, and so forth. The problem with this method is that waiting

another frame might always increase the expected reward, and so this method might never actually choose to acquire a frame. This problem is solved when we introduce the time cost of an acquisition, which is proportional to the number of frames elapsed since the last acquisition. Recalling that the benefit is the ratio of reward to cost, and that the cost consists of both the photobleaching cost and the time cost, our method is to acquire a frame if the expected benefit is higher than if we waited for the following frame. Algorithm 3 details this procedure.

Algorithm 3
Input: t, the frame number of the last acquired frame.
Output: $t + k$, the frame number of the next frame to acquire.

```
k = 0
repeat
  k = k + 1
  set B_now to benefit of acquiring frame t + k
  set B_later to benefit of acquiring frame t + k + 1
until B_later < B_now
return t + k
```

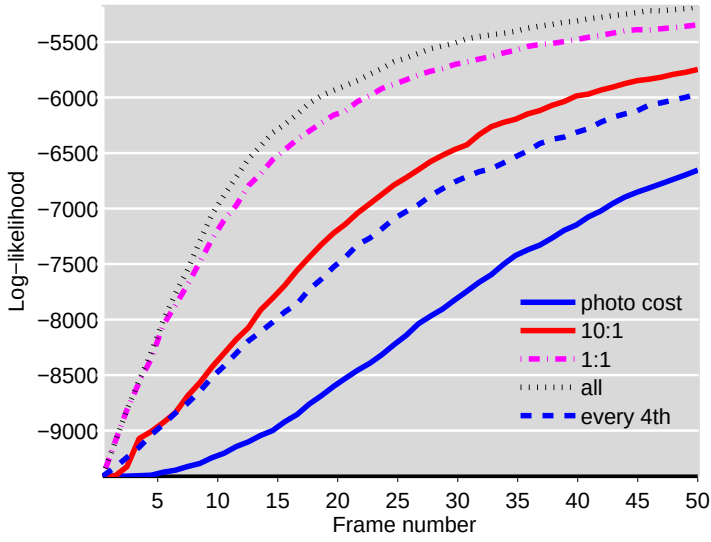
To test this algorithm, we simulate an object track as described in Section 4.1. We then try to learn the object's model, using different acquisition algorithms to acquire the data. Our results, which are shown in Fig. 7.1, are averaged over 10,000 trials. Plot (a) shows the log-likelihood against the frame number. The frame number is incremented regardless of whether that frame was acquired or not. The dotted black line shows the log-likelihood for the case when every frame was acquired. Not surprisingly, this method learns the model the fastest. The dash-dotted magenta line shows the results of our intelligent algorithm when the photobleaching cost and time cost were considered equal. The solid red line shows the results when photobleaching cost was weighted at 10x the time cost. Because less importance was given to time, this results in slower model learning. This method acquired roughly 25% of the frames. The dashed blue line shows a naive algorithm that acquires every 4th frame. The significance of this is that it acquires the same total number of frames as the solid red line, and yet we can see that the final model accuracy is lower. Finally, the solid blue line shows the intelligent algorithm when only photobleaching cost is considered. This results in the

slowest model learning, although it is still steady.

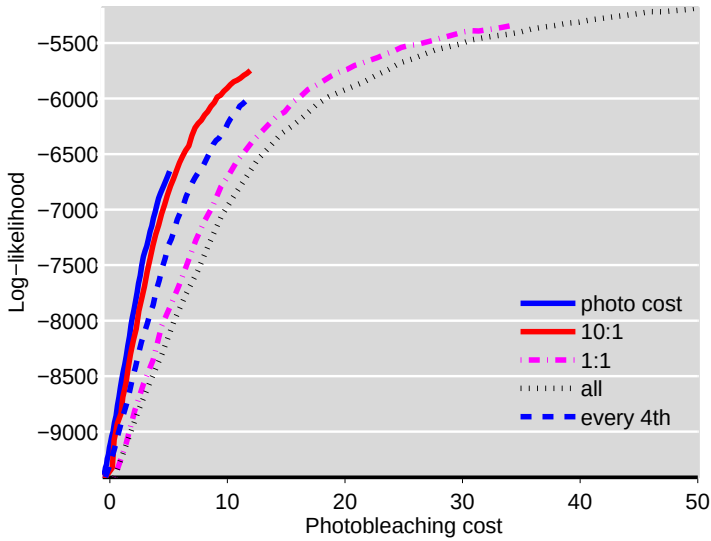
Plot (b) now plots the same log-likelihood against photobleaching incurred, which in this case, is equivalent to the number of frames acquired. We see that the intelligent algorithm that considers only the photobleaching cost does best, but the algorithm that considers photobleaching cost at 10x the time cost performs nearly as well and has much faster model learning. All methods performed better than the method that acquired every frame.

7.2 Discussion

The results in this chapter show that intelligently choosing when to acquire results in a higher model accuracy for a given amount of photobleaching. We also showed that acquiring at a variable frame rate can give a higher accuracy than a constant frame rate, even if the average number of frames acquired is the same (as seen by the solid red line and the dashed blue line in Fig. 7.1(a)). The algorithm is general and could be applied to the multiple-object scenario if we had a method to build multiple-object models with a variable frame rate. It could also be combined with the algorithm that determines where to acquire in a frame (see Chapter 6). To do this, we find the maximum benefit achievable when acquiring a subset of pixels from frame $t + 1$ or $t + 2$, and then proceed as before.



(a)



(b)

Figure 7.1: When to acquire (single object, synthetic data). These curves show the rate at which a model is learned under different acquisition strategies. Plot (a) shows log-likelihood against frame number. Plot (b) shows log-likelihood against photobleaching cost. The dotted black line shows the method where every frame is acquired. The dash-dotted magenta line shows the intelligent algorithm when the photobleaching cost and the time cost are considered equal. The

8

When to Stop Acquiring Frames

This chapter looks at when to stop acquiring frames. The motivation is to save time and avoid unnecessary photobleaching. Ideally, we would acquire until the point when additional frames are unlikely to provide much new information. This could mean that the model has been learned with high accuracy, or it could simply mean that—perhaps due to high photobleaching—additional frames will not help. When building class models, we want to stop acquiring a cell at the point when switching to another cell of that class would provide significantly more information.

We consider this topic both for the single-object scenario and for the multiple-object scenario; the chapter is split accordingly. We further subdivide the multiple-object scenario into cell models and class models, because the optimal stopping point differs depending on which of these we are trying to build.

8.1 Single Object

We test two methods for choosing when to stop acquiring in the single-object scenario. The first method is to predict the reward associated with acquiring the next frame, and to stop acquiring when this reward falls below a threshold. This is our preferred method, and the reward of the next frame can be predicted in the same way as in Chapter 7. The second method is to estimate the reward that was gained from the most recent frame acquisition, and to stop acquiring

when this falls below a threshold. The assumption is that the reward associated with the most recent frame acquisition will be similar to that of the subsequent frame acquisition. The motivation for the second method is that it is faster to compute, but, as we will show, it does not work well in the single-object scenario.

To test these methods, we generated 1,000 synthetic time series as described in Section 4.1, with 50 frames in each time series. We then simulated acquisition of these time series using the two intelligent algorithms described above, as well as a standard algorithm that acquired a fixed number of frames in each time series. For each algorithm, we recorded the average number of frames acquired over the 1,000 time series, and the average accuracy of the resulting models. The results are shown in Fig. 8.1. The solid red line shows our preferred intelligent algorithm that stops acquiring when the reward associated with acquiring the next frame falls below a threshold. To obtain the different points on this curve, we varied the value of the threshold. The dashed blue line shows the intelligent algorithm that stops acquiring when the reward associated with acquiring the previous frame falls below a threshold. Once again, we varied the value of the threshold to obtain the different points on the curve. Finally, the dotted black line shows the standard algorithm, that acquires a fixed number of frames from each time series.

We see that our preferred intelligent algorithm requires fewer frames on average for any given average accuracy. For example, it requires an average of 5 frames from each time series to achieve an average log-likelihood of -1850, whereas the standard algorithm requires 10 frames to achieve the same average log-likelihood. Our other intelligent algorithm performs well when the stopping threshold is low (which means that few frames are acquired on average), but not as well when the stopping threshold is high. The reason for this poor performance may be because the reward of consecutive frames varies greatly when there is only a single object in the cell—in the multiple-object scenario, we show that a similar algorithm works fairly well.

8.2 Multiple Objects

In Chapter 5, we described two ways to validate the accuracy of the multiple-object models. The first way was to measure the log-

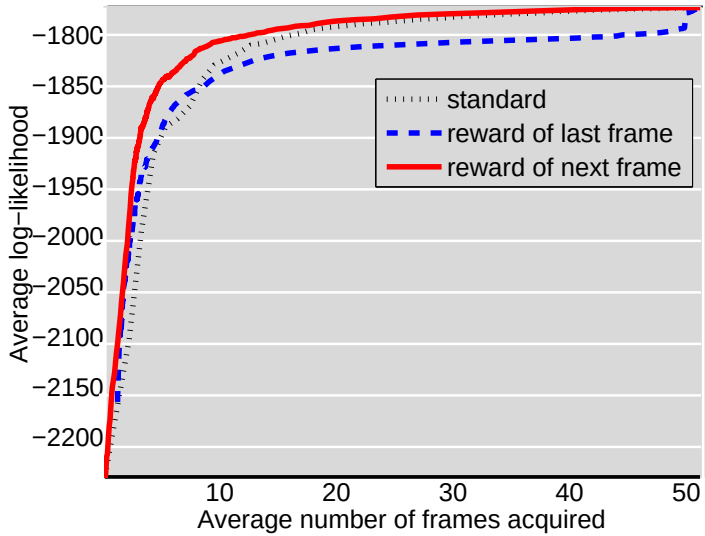


Figure 8.1: When to stop acquiring (single object, synthetic data). The y-axis shows the log-likelihood of the final model given the true model, averaged across all time series. The x-axis shows the average number of frames acquired to get this final model. The solid red line shows the intelligent algorithm that stops acquiring when the reward associated with acquiring the next frame falls below a threshold. The dashed blue line shows the intelligent algorithm that stops acquiring when the reward associated with acquiring the previous frame falls below a threshold. The different points on these curves are obtained by varying the threshold. The dotted black line shows the standard algorithm that acquires a fixed number of frames from each time series. We see that the first intelligent algorithm gives the highest average log-likelihood for any given average number of frames acquired.

likelihood of the model given a separate set of testing frames, and the second way was to measure the classification accuracy. When designing an algorithm to specify when to stop acquiring frames in the multiple-object scenario, we must specify whether our goal is to maximize the log-likelihood or to maximize the classification accuracy. We discuss these cases separately, and then discuss a further modification that is useful when building a class model.

8.2.1 Maximizing Likelihood

In the single-object scenario, the best method worked by predicting the reward of the next frame acquisition. However, in the multiple-object scenario, there is no easy way to predict this future reward, and instead we approximate it with the estimated reward of the last frame acquisition. Although this approximation does not work well for single-object scenarios, it works better with multiple objects; we discuss the reason for this in Section 8.3.

To estimate the reward of the last frame acquisition, we proceed much as we did when we validated the model in Section 5.3. We set aside the first two frames as validation frames, and learn the model from the remaining frames. After each new frame acquisition, we test our model on the validation frames and record the accuracy. The reward of a frame is defined as the improvement in accuracy resulting from that frame's acquisition. Once this reward drops below a threshold, we stop acquiring, and then build the final model using all the frames (including the validation frames). Algorithm 4 shows this process in algorithmic form.

We test this method using the 3T3 data set described in Section 5.1. For each time series, we proceed as follows: First, we set aside the first two frames as the testing frames. We then use the training frames to build a model and intelligently determine when to stop acquiring. The intelligent algorithm uses the first two training frames as validation frames. When acquisition stops, the final model is built from all acquired training frames, and the resulting accuracy is measured on the testing frames.

This method acquires a different number of frames for each time series in the data set, and the resulting model of each time series has a different accuracy. In Fig. 8.2, the solid red line plots the average accuracy against the average number of acquired frames, where the different points on the curve are created by varying the stopping

Algorithm 4
Input: β, the threshold below which to stop acquisition.
Output: m, the final cell model $f_{acquired}$, the number of frames acquired.

```
acquire frames 1 and 2
set f_acquired to 2
set l_new to 0
repeat
  acquire next frame
  increment f_acquired
  build model m with frames 3, ..., f_acquired
  set l_old to l_new
  set l_new to log-likelihood of m given frames 1 and 2
until l_new − l_old < β
build model m with frames 1, ..., f_acquired
return m, f_acquired
```

threshold. In contrast, the dashed blue line plots the average accuracy against the average number of acquired frames when we use the standard method of acquiring k frames every time. In this case, the different points on the curve are created by varying k. We can see that our intelligent method gives a higher accuracy for the same average number of frames, but the difference between the curves is not large. Fortunately, we show a much bigger improvement when the goal is to maximize classification accuracy.

8.2.2 Maximizing Classification Accuracy

Our intelligent stopping algorithm gives much stronger results when the goal is to maximize classification accuracy. Here, we try to stop acquiring at the point where the classification decision is unlikely to change. We distinguish between two scenarios:

1. *Global scenario.* We have access to the class models during acquisition. This means that we can classify the cell after acquiring each frame, and stop acquiring when the classification result reaches a sufficient confidence.

2. *Local scenario.* We do not have access to the class models during acquisition. Hence, we cannot classify during acquisition,

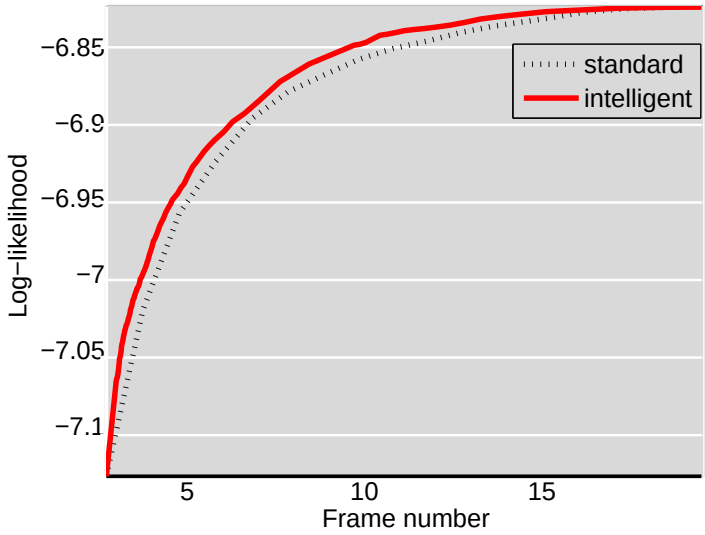


Figure 8.2: When to stop acquiring (multiple objects, real data, maximizing likelihood). The y-axis shows the log-likelihood of the final model given the testing frames, averaged across all time series. The x-axis shows the average number of frames acquired to get this final model. The solid red line shows the algorithm that intelligently chooses when to stop acquiring, with the different points on the curve obtained by varying the stopping threshold. The dotted black line shows the standard algorithm that acquires a fixed number of frames from each time series. We see that the intelligent algorithm gives a higher average accuracy for any given average number of frames acquired, but the improvement is small.

and our choice of when to stop acquiring the cell must be based solely on the data from that cell.

Global Scenario

As stated in Section 5.5, we classify a cell by finding the log-likelihood of each class model, and choosing the class of maximum log-likelihood. We now show that we can find the standard errors of each of these log-likelihoods, and use them to compute our confidence in the classification decision.

In Section 5.4.2, we showed how to find the likelihood of a cell

model given an observed cell using (5.7)-(5.8). We reproduce these equations below, but taking logarithms of each side:

$$\log(l) = \log(l_\lambda) + \log(l_{\lambda,\lambda'}) + \log(l_{\lambda,d}) + \log(l_{\lambda,\emptyset}), \tag{8.1}$$

where:

$$\log(l_\lambda) = \sum_{\lambda=1}^{k} N_\lambda m_\lambda + (N - N_\lambda)(1 - m_\lambda), \tag{8.2a}$$

$$\log(l_{\lambda,\lambda'}) = \sum_{\lambda=1}^{k} \sum_{\lambda'=1}^{k} N_{\lambda,\lambda'} m_{\lambda,\lambda'} + (N_\lambda - N_{\lambda,\lambda'})(1 - m_{\lambda,\lambda'}), \tag{8.2b}$$

$$\log(l_{\lambda,d}) = \sum_{\lambda=1}^{k} \sum_{d=1}^{d_{max}} N_{\lambda,d} m_{\lambda,d} + (N_\lambda - N_{\lambda,d})(1 - m_{\lambda,d}), \tag{8.2c}$$

$$\log(l_{\lambda,\emptyset}) = \sum_{\lambda=1}^{k} N_{\lambda,\emptyset} m_{\lambda,\emptyset} + (N_{\lambda_t} - N_{\lambda,\emptyset})(1 - m_{\lambda,\emptyset}). \tag{8.2d}$$

These expressions are based around the counts N_λ, $N_{\lambda,\lambda'}$, $N_{\lambda,d}$ and $N_{\lambda,\emptyset}$, defined in Section 5.3. To calculate the standard error of the log-likelihood, note that these counts come from the N individual object observations, and that the log-likelihood given the overall cell is the sum of the log-likelihoods given each individual object observation. To make this clearer, we introduce the binary variable $n_{j,\lambda}$, defined to be 1 when object j is of type λ, and 0 otherwise. Hence, $N_\lambda = \sum_{j=1}^{N} n_{j,\lambda}$. We define similar variables to correspond with the counts $N_{\lambda_t,\lambda_{t+1}}$, $N_{\lambda_t,d}$, and $N_{\lambda_t,\emptyset}$. Hence, $n_{j,\lambda,\lambda'}$ is 1 if object j is of type λ and there is a nearby object of type λ' in the subsequent frame; $n_{j,\lambda,d}$ is 1 if object j is of type λ and there is a nearby object distance d away in the subsequent frame; $n_{j,\lambda,\emptyset}$ is 1 if object j is of type λ and there are no nearby objects in the subsequent frame.

Using these binary variables, we can rewrite (8.2) as:

$$log(l_\lambda) = \sum_{j=1}^{N} x_{j,\lambda}, \tag{8.3a}$$

$$log(l_{\lambda,\lambda'}) = \sum_{j=1}^{N} x_{j,\lambda,\lambda'}, \tag{8.3b}$$

$$log(l_{\lambda,d}) = \sum_{j=1}^{N} x_{j,\lambda,d}, \tag{8.3c}$$

$$log(l_{\lambda,\emptyset}) = \sum_{j=1}^{N} x_{j,\lambda,\emptyset}, \tag{8.3d}$$

where:

$$x_{j,\lambda} = \sum_{\lambda=1}^{k} n_{j,\lambda} log(m_\lambda) + (1 - n_{j,\lambda}) log(1 - m_\lambda), \tag{8.4a}$$

$$x_{j,\lambda,\lambda'} = \sum_{\lambda=1}^{k} \sum_{\lambda'=1}^{k} n_{j,\lambda,\lambda'} log(m_{\lambda,\lambda'}) + (1 - n_{j,\lambda,\lambda'}) log(1 - m_{\lambda,\lambda'}), \tag{8.4b}$$

$$x_{j,\lambda,d} = \sum_{\lambda=1}^{k} \sum_{d=1}^{d_{max}} n_{j,\lambda,d} log(m_{\lambda,d}) + (1 - n_{j,\lambda,d}) log(1 - m_{\lambda,d}), \tag{8.4c}$$

$$x_{j,\lambda,\emptyset} = \sum_{\lambda=1}^{k} n_{j,\lambda,\emptyset} log(m_{\lambda,\emptyset}) + (1 - n_{j,\lambda,\emptyset}) log(1 - m_{\lambda,\emptyset}). \tag{8.4d}$$

Finally, we can rewrite (8.1) as:

$$log(l_i) \quad = \quad \sum_{j=1}^{N} y_j, \tag{8.5}$$

where:

$$y_j = x_{j,\lambda} + x_{j,\lambda,\lambda'} + x_{j,\lambda,d} + x_{j,\lambda,\emptyset}. \tag{8.6}$$

The reason for this rearranging is that we can now view $y_1, ..., y_N$ as instantiations of a random variable Y. Learning the mean of Y is sufficient for learning the log-likelihood of the cell model, and our estimate of this mean improves as the number of object observations

increases. Specifically, the standard error of the mean is given by $\sigma/\sqrt{N}$, where σ is the standard deviation of $y_1, ..., y_N$.

We have described how to find the log-likelihood of a *cell model* and its standard error given an observed cell. However, what we actually want to find is the log-likelihood of a *class model* and its standard error. To do this, we simply take the class's constituent cell models and find the one with the highest log-likelihood. This log-likelihood, along with its standard error, becomes the log-likelihood and standard error of the class model.

Now, if c_1 is the class of highest log-likelihood, and c_2 is the class of second highest log-likelihood, we define the classification confidence, C, as:

$$C = \frac{\log(l_{c_1}) - \log(l_{c_2})}{\sqrt{e_{c_1}^2 + e_{c_2}^2}}, \tag{8.7}$$

where $\log(l_{c_1})$ and $\log(l_{c_2})$ are the log-likelihoods of classes c_1 and c_2 respectively, and e_{c_1} and e_{c_1} are their standard errors. Our intelligent stopping algorithm continues to acquire frames until C exceeds some threshold that we call the *classification confidence threshold.*

Local Scenario

In the local scenario, we cannot compute the log-likelihood of a class model, because the class models are unavailable. However, we can still compute the log-likelihood of some sample model, including its standard error. This standard error provides an indication of how accurately we expect to predict the log-likelihood of the class models when they become available. The natural choice of a sample model is the model of the cell being acquired. We define the standard error of its log-likelihood as S, and acquire frames until S falls below some threshold that we call the *likelihood uncertainty threshold.*

Results

We now test our intelligent stopping algorithms both for the extrinsic and the local scenario. As before, we take each time series in the data set and determine how many frames would have been acquired for a given stopping algorithm and threshold. We then measure the average number of frames acquired, and the average classification accuracy. For example, in the extrinsic scenario with a classification confidence threshold of 0.7, we acquire 3-35 frames per cell, with

an average of 8.0 frames, yielding 80.3% classification accuracy. We compare this to the standard method that acquires exactly 8 frames for each cell, yielding only 75.0% classification accuracy. To reach 80% classification accuracy using the standard method, we would need to acquire over 14 frames per cell—almost twice as many as with our intelligent method.

In Fig. 8.3, we compare the two intelligent methods with the standard method, using a range of stopping thresholds to get the different points on the curves. The solid red line plots the intelligent algorithm for the global scenario, the dashed blue line plots the intelligent algorithm for the local scenario, and the dotted black line plots the standard algorithm that acquires a fixed number of frames in each time series. The intelligent algorithms achieve significantly higher accuracy for the same average number of frames acquired, with the best results for the global scenario.

8.2.3 Class Models

Finally, we discuss when to stop acquiring frames from a cell when building a class model. The primary change is that, in the global scenario, we now stop acquiring a cell if we recognize that it is similar to a previous cell in that class. This allows us to focus our time and resources on acquiring cells that are different from previously acquired cells of that class, as these are the cells that provide the most new information.

To assess whether a cell is similar to previous cells in the class, we simply classify it. If it is correctly classified with confidence exceeding the classification confidence threshold, we know that it must be similar to previously acquired cells of that class, and we stop acquiring. In addition, as we did previously, we also stop acquiring when S falls below the likelihood uncertainty threshold. This ensures that we do eventually stop acquisition even when the cell is very different from previously acquired cells of that class (or when we are acquiring the first cell of that class).

This method implicitly assumes that acquisition alternates between cells of all the classes, such that we can build and refine a class model of every class simultaneously. In the local scenario, when classes are not available before starting the acquisition, this is not possible. However, we can still use the likelihood uncertainty threshold as before, which gives almost as good results for small numbers

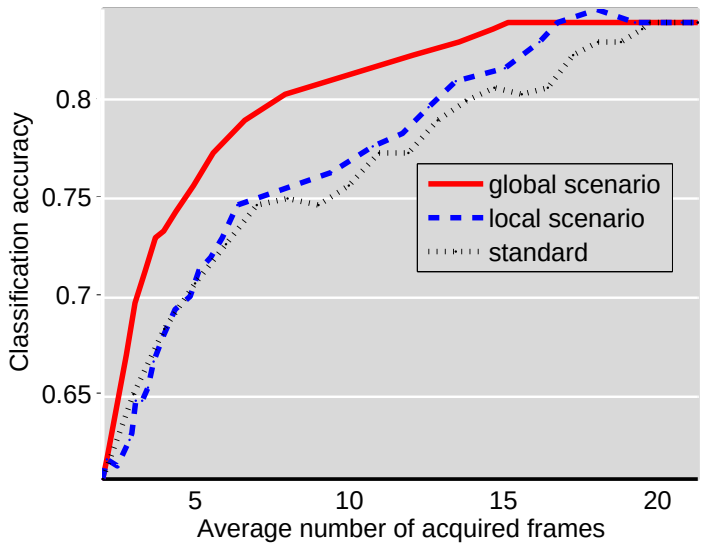


Figure 8.3: When to stop acquiring (multiple objects, real data, maximizing classification accuracy). The y-axis shows the classification accuracy of the final model, averaged across all time series. The x-axis shows the average number of frames acquired to get the final model. We use one of three methods to choose when to stop acquiring. The first method (solid red line) uses the intelligent algorithm under the global scenario. The second method (dashed blue line) uses the intelligent algorithm under the local scenario. The third standard method (dotted black line) acquires the same number of frames for each time series. We see that the two intelligent methods outperform the standard method, with the best results in the global scenario.

of cells.

To test these intelligent acquisition algorithms, we set aside 10 time series as our testing set. We take the remaining 294 training time series and determine which frames would have been acquired for each intelligent stopping method and threshold. We then use these to build a class model for each of the 12 classes, and attempt to classify the testing set. We use every frame for time series in the testing set. We ran 10,000 trials, randomizing the testing and training sets, including their order, in each trial. In Fig. 8.4, we show the resulting classification accuracy against the average number of frames acquired per cell when (1) using our intelligent acquisition algorithm for the global scenario, varying the classification confidence threshold to get the different points on the curve, (2) using our intelligent acquisition algorithm for the local scenario, varying the likelihood uncertainty threshold to get the different points on the curve, (3) using a standard acquisition algorithm that acquires a fixed number of frames for every cell. Once again, we see that the intelligent algorithms achieve significantly higher accuracy than the standard method for the same average number of frames acquired. Moreover, the results for the global scenario are best. In the global scenario, the intelligent algorithm requires only 10 frames per cell to reach a classification accuracy of 80%, whereas the standard algorithm requires 18 frames. The results for the local scenario are almost as good as those for the global scenario, but we expect that the difference between the two scenarios will increase as the number of cells per class increases.

8.3 Discussion

In the single-object scenario, we showed that predicting the reward of the next frame, and ceasing acquisition when this reward falls below a threshold, is a more effective stopping strategy than just acquiring a fixed number of frames. The limitation is that this algorithm only looks at the very next frame, $t+1$. However, as sometimes $t+1$ may have a low reward but subsequent frames may have high rewards, a useful extension would be a stopping criterion that considers the reward of the next several frames. We could do this by combining the current stopping algorithm with the *when to acquire* algorithm of Chapter 7. The latter determines the best frame to acquire, and the former determines whether the benefit associated with acquiring

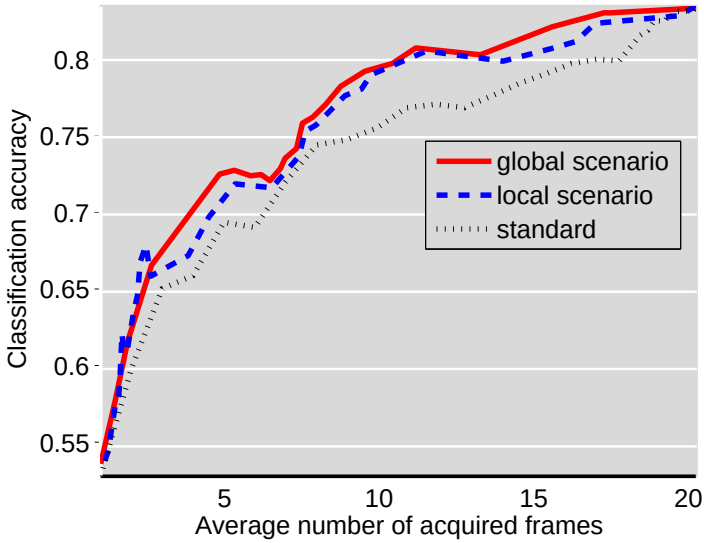


Figure 8.4: When to stop acquiring (class models, real data, maximizing classification accuracy). The y-axis shows the classification accuracy of the final model, averaged across all time series. The x-axis shows the average number of frames acquired to get the final model. We use one of three methods to choose when to stop acquiring. The first method (solid red line) uses the intelligent algorithm for the global scenario. The second method (dashed blue line) uses the intelligent algorithm for the local scenario. The third method (dotted black line) acquires the same number of frames for each time series. We see that the two intelligent methods outperform the standard method, with the best results in the global scenario.

this frame is high enough to justify continued acquisition.

We also tried using the reward associated with the last frame acquisition to predict the reward of the next frame acquisition. We showed that this did not work well in the single-object scenario (Fig. 8.1), but gave better results in the multiple-object scenario (Fig. 8.2). The reason for its failure in the single-object scenario is that the reward associated with successive frames varies greatly due to pixelation and noise in the object's motion; therefore, the reward of the most recent frame acquisition may be very different from the reward of the next frame acquisition. However, these variations tend

to cancel each other out in the multiple-object scenario, and so the rewards of successive frames are more similar.

For maximizing classification accuracy, we got good results by ceasing acquisition at the point where the classifier had high confidence in its decision. It is important to realize that this confidence does not refer to the probability that the decision is correct, but rather to the probability that the decision will not change. Our results might be even better if we considered the extent of photobleaching in the cell: If photobleaching is extensive then new frames contribute less information, and so the classification decision is even more unlikely to change.

To find the confidence in the classification decision, we converted the likelihoods of each class into a Gaussian random variable. An alternative method is to use the likelihood values directly as a measure of confidence. However, because these likelihoods are the product of all the different variables in the model, and because there are dependencies between these variables, this method tends to give confidences that are highly inflated and not useful as a stopping criterion.

Finally, we note that the best time to stop acquiring depends on our goal—whether to maximize log-likelihood, maximize classification accuracy, or maximize some other quantity. Although we did not show results, we actually found that the criterion for maximizing log-likelihood gave worse results in terms of classification accuracy. Hence, tailoring the algorithm to the specific application is highly desirable. For situations where the specific application is unknown during acquisition, it would be useful to search for a quantity whose maximization gives good results for most real-world scenarios.

9

How Many Cells to Acquire

Our final strategy is to decide how many cells to acquire when building a class model. Because cells must be cultured prior to image acquisition, we focus on batch additions. Hence, rather than re-evaluating after every cell, we first acquire, say, 10 cells, and then ask whether we should acquire another 10 cells.

Our approach here is similar to that described in Section 8.2.1 for deciding how many frames to acquire. The basic idea is to measure the reward gained from acquiring the last cell, and to use this as a basis for deciding whether to acquire more cells. The difference is that, unlike frames, the cells have no inherent ordering. Hence, instead of only focusing on the reward of the most recently acquired cell, we can measure the reward of each of the acquired cells and then average across all of them. To measure the reward of a cell, we measure the resulting drop in model accuracy when that cell is removed from the set used for model building.

As in Chapter 8, our method differs depending on whether we want to maximize the likelihood of the class model given a newly observed cell, or to maximize the classification accuracy of that newly observed cell. We discuss each case in turn.

9.1 Maximizing Likelihood

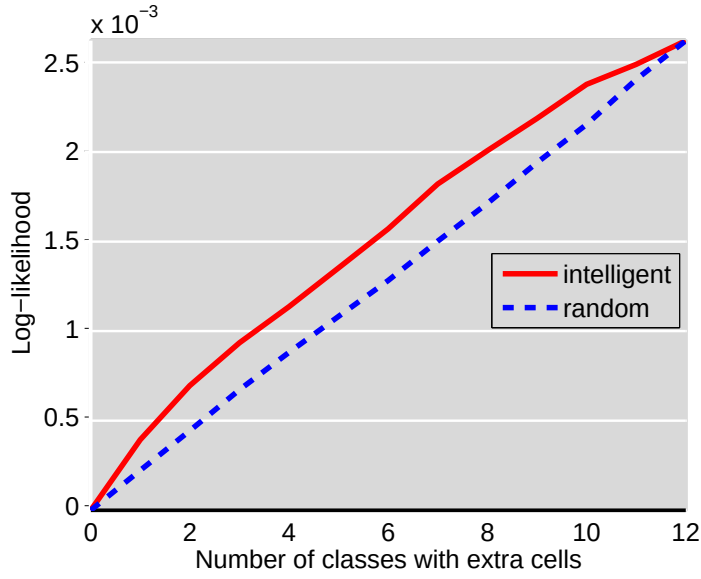
Suppose we have acquired k_1 cells from a class and want to decide whether to acquire more. Our method is to first measure the accuracy

of the model when $k_1 - 2$ cells are used to build it, and then measure the accuracy of the model when $k_1 - 1$ cells are used to build it. If that extra cell significantly improved the accuracy, then we would decide to acquire more cells. (Ideally we would measure the improvement in accuracy from $k_1 - 1$ cells to k_1 cells, but we have no way to measure the accuracy when all k_1 cells are used.)

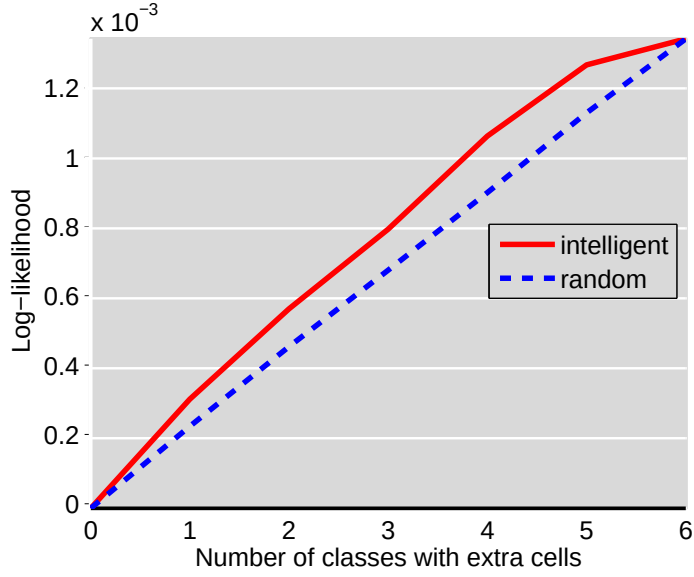
To measure the accuracy of the model when $k_1 - 2$ cells are used to build it, we first select any $k_1 - 2$ cells from the set and build the model. Then, we measure the log-likelihood of this model given the remaining two cells. We repeat this using every possible combination of $k_1 - 2$ cells (there are $k_1(k_1 - 1)$ such combinations), and compute the average log-likelihood over all of them, which we denote a_2.

To measure the accuracy of the model when $k_1 - 1$ cells are used to build it, we repeat this process, but now selecting $k_1 - 1$ cells from the set to build the model, and measuring the log-likelihood given the remaining one cell. There are k_1 combinations of $k_1 - 1$ cells, and again we compute the average log-likelihood across all of them, which we denote a_1. If $(a_1 - a_2)$ exceeds a threshold, we choose to acquire more cells from this class. Algorithm 5 shows this whole process in algorithmic form.

We tested this method using the 3T3 data set described in Section 5.1. For each class, we first set aside one testing cell. Then, we acquired k_1 cells from that class, and used the method above to decide whether to acquire an additional k_2 cells. Finally, we measured the likelihood of the model (which was built with either k_1 cells or $k_1 + k_2$ cells) given the testing cell. We ran this experiment on every class and repeated it 10,000 times with different training and testing cells. In Fig. 9.1, the y-axis measures the average log-likelihood of the model and the x-axis indicates the number of classes for which an additional k_2 cells were acquired. The solid red line shows the average log-likelihood when we intelligently chose whether to acquire an additional k_2 cells, whereas the dashed blue line shows the average log-likelihood when we randomly chose whether to acquire an additional k_2 cells. Plot (a) shows the results when $k_1 = 10$ and $k_2 = 5$. Plot (b) shows the results when $k_1 = 10$ and $k_2 = 10$. In the latter plot, we only used 6 of the 12 classes (Sdpr, Adfp, Timm23, Hspa9a, Tctex1 and Actn4), because we did not have enough cells available for the other 6 classes. We can see from these experiments that acquiring with the intelligent method results in a higher average log-likelihood for the same average number of cells acquired.



(a)



(b)

Figure 9.1: How many cells to acquire (class models, real data, maximizing likelihood). The y-axis measures the average likelihood on the testing cells. The x-axis indicates the number of classes for which the additional k_2 cells were acquired. The solid red line shows the average likelihood when we used our intelligent acquisition method to choose whether to acquire an additional k_2 cells for a class. The dashed blue line is for when we randomly chose whether to acquire

Algorithm 5
Input: c, the class from which to acquire, k_1, the initial number of cells to acquire from the class, k_2, the number of extra cells to acquire from the class, γ, the threshold over which these extra cells are acquired.
Output: S_c, set of acquired cells.

```
initialize set of acquired cells S_c to ∅
acquire any k_1 cells from c and add to S_c
for all cells s in S_c do
  build class model M_c from all S_c except s
  set x_s to log-likelihood of M_c given s
end for
set accuracy, a_1, to mean of x
for all cells s in S_c do
  for all cells t in S_c except s do
    build class model M_c from all S_c except s and t
    set x_t to log-likelihood of M_c given t
  end for
  set b_s to mean of x
end for
set accuracy, a_2, to mean of b
set reward, r, to a_2 − a_1
if r > γ then
  acquire k_2 extra cells from c and add to S_c
end if
return S_c
```

9.2 Maximizing Classification Accuracy

We use the same method when our goal is to maximize classification accuracy, but now we estimate the reward of acquiring more cells from a class in terms of classification accuracy instead of log-likelihood. Here, as in Chapter 8, we distinguish between two scenarios:

1. *Global scenario.* We first acquire k_1 cells from every class, and can then look at all cells when deciding from which classes to acquire an additional k_2 cells.

2. *Local scenario.* We acquire k_1 cells from a class, and must

decide whether to acquire an additional k_2 cells from that class without accessing the cells from other classes.

Global Scenario

Suppose we have acquired k_1 cells from every class. Now, for a class c, we want to decide whether to acquire more cells. Similarly to Section 9.1, our process is to first measure the accuracy of the class model when $k_1 - 2$ cells are used to build it, and then measure the accuracy of the class model when $k_1 - 1$ cells are used to build it. If that extra cell significantly improved the accuracy, then we would acquire more cells from that class.

To implement this idea, we first we build class models for all classes except c (using all k_1 cells available). Second, we build a class model for c with $k_1 - 2$ cells, and measure the classification accuracy on the remaining two cells in c. We repeat this with every possible combination of $k_1 - 2$ cells (there will be $k_1(k_1 - 1)$ such combinations) and compute the average accuracy, which we denote a_2. Third, we build a class model for c with $k_1 - 1$ cells, and measure the classification accuracy on the remaining cell. We repeat this with every possible combination of $k_1 - 1$ cells (there are k_1 such combinations), and compute the average accuracy, which we denote a_1. If $(a_1 - a_2)$ exceeds a threshold, we choose to acquire more cells from c. Algorithm 6 shows this process in algorithmic form.

Local Scenario

In the local scenario, we do not have access to cells from other classes, and so we cannot find the classification accuracy as we did for the global scenario. Nevertheless, we still try to estimate the drop in classification accuracy that would result from removing a cell from the class. To do this, we take each cell model m in the class, and determine its closest match, m', which is the cell model that has the highest likelihood given m. We then find p—the proportion of cell models in that class that are chosen at least once as a closest match. If a cell model is chosen as a closest match, then its removal is likely to reduce the classification accuracy of that class. Hence, if p is higher, removing a cell from the class will result in a bigger drop in classification accuracy, which means that adding more cells to the class may cause a bigger increase in classification accuracy. Thus, if

Algorithm 6

Input: k_1, the initial number of cells to acquire from each class, k_2, the number of extra cells to acquire from a class, γ, the threshold over which these extra cells are acquired.

Output: S, the set of acquired cells from all classes.

```
for all classes c do
  initialize S_c to ∅
  acquire k_1 cells from c and add to S_c
  build class model M_c from S_c
end for
for all classes c do
  for all cells s in S_c do
    build class model M_c from all S_c except s
    classify s using current class models
    set x_s to 1 if s is classified correctly, 0 otherwise
  end for
  set accuracy, a_1, to mean of x
  for all cells s in S_c do
    for all cells t in S_c except s do
      build class model M_c from all S_c except s and t
      set x_t to 1 if t is classified correctly, 0 otherwise
    end for
    set b_s to mean of x
  end for
  set accuracy, a_2, to mean of b
  set reward, r, to a_2 − a_1
  if r > γ then
    acquire k_2 extra cells from c and add to S_c
  end if
  build class model M_c from all S_c
end for
return S
```

p exceeds a threshold, we choose to acquire more cells from the class. Algorithm 7 shows this process in algorithmic form.

Algorithm 7

Input: c, the class from which to acquire, k_1, the initial number of cells to acquire from the class, k_2, the number of extra cells to acquire from the class, γ, the threshold over which these extra cells are acquired.

Output: S_c, set of acquired cells.

initialize set of acquired cells S_c to $\emptyset$
acquire any k_1 cells from c and add to S_c
for all cells s in S_c **do**
  set x_s to 0
end for
for all cells s in S_c **do**
  find cell in S_c, s', whose cell model has the highest likelihood given s
  set $x_{s'}$ to 1
end for
set reward, r, to mean of x
if $r > \gamma$ **then**
  acquire k_2 extra cells from c and add to S_c
end if
return S_c

Results

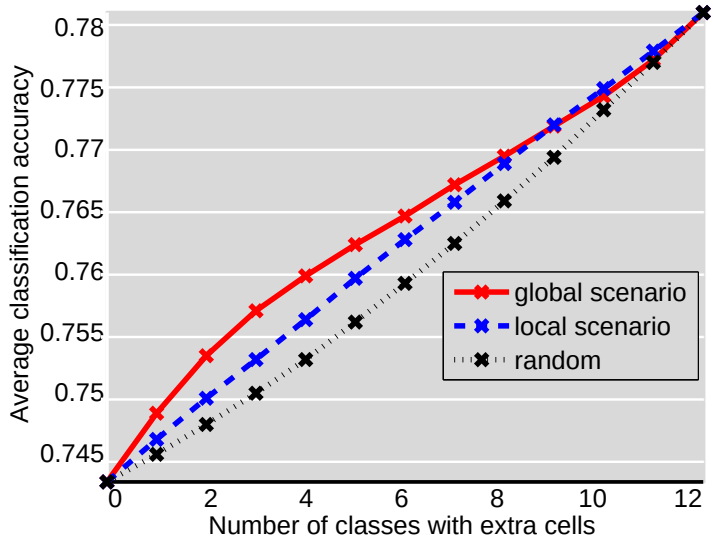
We tested this method using the 3T3 data set described in Section 5.1. For each class, we first set aside one testing cell. Then, we acquired k_1 cells from each class, and used the methods above to decide from which classes to acquire an additional k_2 cells. Finally, we built a model using all the acquired cells from all classes, and measured the classification accuracy on the testing cells. We repeated this experiment 10,000 times with different training and testing cells. In Fig. 9.2, the y-axis measures the average classification accuracy of the model, and the x-axis indicates the number of classes for which an additional k_2 cells were acquired. The solid red line shows the average classification accuracy under the global scenario when we intelligently chose whether to acquire an additional k_2 cells. The dashed blue line shows the average classification accuracy under the local scenario when we intelligently chose whether to acquire an additional k_2 cells. The dotted black line shows the classification ac-

curacy when we randomly chose whether to acquire an additional k_2 cells. Plot (a) shows the results when $k_1 = 10$ and $k_2 = 5$. Plot (b) shows the results when $k_1 = 10$ and $k_2 = 10$. In the latter plot, we only used 6 of the 12 classes (Sdpr, Adfp, Timm23, Hspa9a, Tctex1 and Actn4), because we did not have enough cells available for the other 6 classes. We can see from these experiments that acquiring with the intelligent method results in a significantly higher average classification accuracy for the same average number of cells acquired, with the best results for the global scenario.

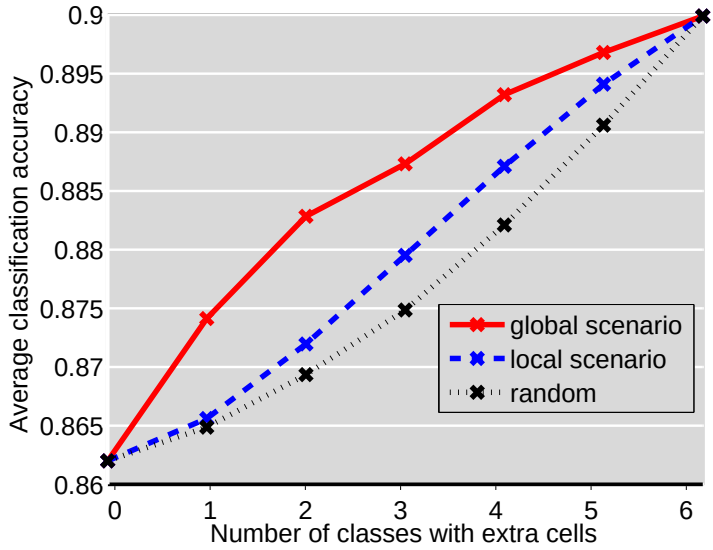
9.3 Discussion

This chapter showed that we get higher accuracy when we intelligently choose from which classes to acquire extra cells than when we randomly choose. We tested this both for maximizing log-likelihood and for maximizing classification accuracy, achieving the best results in the latter scenario. Because classification is a real-world application, we expect that the classification results will be more indicative of the results on other real-world applications.

One way to view this problem is to say that a class has multiple sub-classes, and that to learn a class model, we need to continue acquiring cells until we believe we have at least one cell from each sub-class. In that case, we can consider the algorithms in this chapter as giving the probability that there are still major undiscovered sub-classes. These algorithms work well in conjunction with the *when to stop acquiring frames* algorithm for class models from Section 8.2.3. That algorithm stopped acquiring frames from a cell if it observed that that cell was similar to a previously acquired cell (hence, in the same sub-class), allowing us to find the cells from undiscovered sub-classes more quickly.



(a)



(b)

Figure 9.2: How many cells to acquire (class models, real data, maximizing classification accuracy). The y-axis measures the average classification accuracy on the testing cells. The x-axis indicates the number of classes for which the additional k_2 cells were acquired. The solid red line shows the classification accuracy when we use the intelligent algorithm in the global scenario to choose from which classes to acquire extra cells. The dashed blue line shows the classification

Part IV

Conclusions

Although automated analysis of fluorescence microscope images is becoming more common, up until now, this analysis has always taken place as a post-processing step. This means that the acquisition process has been unable to adapt to the task and data at hand. In this work, we have explored the benefits of building models while the data is being acquired, thereby opening up the potential for intelligent acquisition. We have developed a new acquisition framework which combines model building and intelligent acquisition, and we have shown that this framework results in either time savings or reduced photobleaching without loss of accuracy, or alternatively, a higher accuracy for the same total acquisition time or photobleaching cost.

Model Building We described methods to build models both in the single-object scenario and in the multiple-object scenario. We tested the single-object method on synthetically generated data where we could use accurate ground truth to demonstrate its correctness, and tested the multiple-object method on a real data set of 3T3 time series to show useful results on real data. Because of the difficulty of matching objects between frames, the multiple-object method does not model each individual object, but instead models the average behavior of each type of object in the cell. Future research will look at ways of bringing the modeling down to an individual object level, which would, in turn, allow for more intelligent acquisition methods.

The cells in our 3T3 data set came from 12 cell lines, each with a different protein labeled. We defined each of these cell lines as a unique class, and built class models by combining the models from all cells in a class. We showed that these class models could classify a cell of unknown class with 84.5% accuracy—higher than any previously reported result on this 3T3 data set. Future research will look at clustering cells within a class so that classes could be represented as a set of sub-classes. This would provide more insight into their behavior as well as allowing for more sophisticated intelligent acquisition algorithms.

Although it is convenient to build the final models while we acquire, note that the only requirement is that the model building is accurate enough to guide the intelligent acquisition module. If necessary, we can then build more accurate models as a post-processing step, perhaps with a more sophisticated and time-consuming algo-

rithm.

Intelligent Acquisition We discussed four different strategies for intelligent acquisition. These were evaluated with respect to maximizing log-likelihood and maximizing classification accuracy. For both of these goals, we showed that intelligent acquisition was more effective than standard acquisition, but generally the difference was greater when the goal was maximizing classification accuracy. This is encouraging because classification is a real-world application, suggesting that our framework and algorithms will have real practical benefits.

- *Where to Acquire in a Frame.* For the single-object scenario, we showed that we could learn an object's motion model with less photobleaching when we intelligently chose which pixels to acquire. Interestingly, the best pixels to acquire were not necessarily those most likely to contain the object, because those pixels were also the most likely to cause photobleaching. We would like to apply our method to the multiple-object scenario as well, but this would require modeling multiple-object cells on an individual object level, which we cannot yet do efficiently. In the meantime, a simple strategy for the multiple-object scenario is to skip any pixels highly unlikely to contain an object, because this does not affect the model accuracy, but does still lead to faster frame acquisition as well as reducing photobleaching in out-of-focus z-slices.

- *When to Acquire Frames.* We also showed that we can reduce photobleaching by sampling in time—that is, by intelligently choosing when to acquire frames. Intuitively, if objects are move slowly relative to the spatial resolution, then we want to acquire infrequently, whereas if objects move quickly relative to spatial resolution, we want a much higher frame rate.

- *When to Stop Acquiring Frames.* This is perhaps the most immediately applicable of our strategies, because it was tested on real data, and requires no special microscope features. Here, by intelligently choosing from which cells to acquire more frames and from which cells to acquire less, we achieved up to a twofold reduction in acquisition time without any loss in classification accuracy.

- *How Many Cells to Acquire.* Finally, we presented a method for determining from which classes we should acquire more cells when building a class model. This lets us know which cells we should culture in the first place. With the goal of modeling all proteins in all cell-types and under all conditions, it is clearly important to know from which classes we need more cells. Intelligent acquisition can provide significant savings: In one example, we showed that we got a higher classification accuracy by intelligently choosing just two classes from which to acquire extra cells than by randomly choosing four such classes.

As large amounts of 2D and 3D time series are being acquired every day, reducing the acquisition time and total light exposure is a desirable goal. The framework that we have introduced, which combines online model building with intelligent acquisition, is an effective way to achieve this goal. As computing power continues to increase in accordance with Moore's Law, even more sophisticated online algorithms will become possible, making intelligent acquisition a valuable tool in the future of fluorescence microscopy.

Bibliography

[1] M. S. Arulampalam, S. Maskell, N. Gordon, T. Clapp, D. Sci, T. Organ, and S. A. Adelaide. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Trans. Signal Proc.*, 50(2):174–188, 2002.

[2] R. G. Baraniuk. Compressive Sensing. *IEEE Signal Proc. Mag.*, 24(4):118–121, 2007.

[3] C. Berzuini, N. G. Best, W. R. Gilks, and C. Larizza. Dynamical conditional independence models and markov chain monte carlo methods. *Journ. Amer. Stat. Assoc.*, 92(440), 1997.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[5] M. V. Boland, M. K. Markey, and R. F. Murphy. Automated recognition of patterns characteristic of subcellular structures in fluorescence microscopy images. *Cytometry*, 33:366–375, Nov. 1998.

[6] M. V. Boland and R. F. Murphy. A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of HeLa cells. *Bioinformatics*, 17(12):1213–1223, 2001.

[7] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[8] E. J. Candès. Compressive sampling. In *Int. Congr. of Mathematicians*, Madrid, Spain, 2006.

[9] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inform. Th.*, 52(2):489–509, Feb. 2006.

[10] A. Chebira, Y. Barbotin, C. Jackson, T. E. Merryman, G. Srinivasa, R. F. Murphy, and J. Kovačević. A multiresolution approach to automated classification of protein subcellular location images. *BMC Bioinformatics*, 8(210), 2007. http://www.andrew.cmu.edu/user/jelenak/Repository/-07_ChebiraBJMSMK/07_ChebiraBJMSMK.html.

[11] X. Chen, M. Velliste, S. Weinstein, J. W. Jarvik, and R. F. Murphy. Location proteomics - building subcellular location trees from high resolution 3d fluorescence microscope images of randomly-tagged proteins. In *Proc. SPIE*, volume 4962, pages 298–306, San Jose, CA, 2003.

[12] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Mach. Learn.*, 15(2):201–221, 1994.

[13] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[14] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[15] J.V. Davis and I. Dhillon. Differential Entropic Clustering of Multivariate Gaussians. *Advances in Neural Information Processing Systems*, 19:337–344, 2007.

[16] D. L. Donoho. Compressed sensing. *IEEE Trans. Inform. Th.*, 52(4):1289–1306, Apr. 2006.

[17] A. Doucet. On sequential simulation-based methods for bayesian filtering. Technical report, Cambridge Univ., Cambridge, UK, 1998.

[18] A. Frank. On Kuhn's Hungarian method-a tribute from Hungary. *Naval Research Logistics*, 52(1):2–5, 2005.

[19] E. Glory and R. F. Murphy. Automated subcellular location determination and high throughput microscopy. *Developmental Cell*, 12:7–16, 2007.

[20] N. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Radar and Signal Processing, IEEE Proceedings*, 140(2):107–113, 1993.

[21] R. M. Haralick. Statistical and structural approaches to texture. *Proc. IEEE*, 67:786–804, 1979.

[22] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall, 2008.

[23] R. A. Hoebe, C. H. Van Oven, T. W. Gadella Jr, P. B. Dhonukshe, C. J. Van Noorden, and E. M. Manders. Controlled light-exposure microscopy reduces photobleaching and phototoxicity in fluorescence live-cell imaging. *Nature Biotech.*, 25(2):249–53, 2007.

[24] D.W. Hosmer and S. Lemeshow. *Applied logistic regression*. Wiley-Interscience, 2004.

[25] C.W. Hsu and C.J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[26] Y. Hu. *Automated analysis of protein subcellular locations in time series images*. PhD thesis, Carnegie Mellon Univ., Carnegie Mellon Univ., 2007.

[27] Y. Hu, J. Carmona, and R.F. Murphy. Application of temporal texture features to automated analysis of protein subcellular locations in time series fluorescence microscope images. In *Proc. of the 2006 IEEE International Symposium on Biomedical Imaging (ISBI 2006)*, pages 1028–1031, 2006.

[28] K. Huang and R. F. Murphy. Boosting accuracy of automated classification of fluorescence microscope images for location proteomics. *BMC Bioinformatics*, 5(78), 2004.

[29] J. W. Jarvik, S. A. Adler, C. A. Telmer, V. Subramaniam, and A. J. Lopez. CD-tagging: A new approach to gene and protein discovery and analysis. *Biotechniques*, 20:896–904, 1996.

[30] I.D. Johnson. *Handbook of Biological Confocal Microscopy*, chapter Practical considerations in the selection and application of fluorescent probes. Springer, third edition, 2006.

[31] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. *Proc. ACM SIGIR Conf. on research and development in information retrieval*, pages 3–12, 1994.

[32] Z.-P. Liang and P. C. Lauterbur. An efficient method for dynamic magnetic resonance imaging. *IEEE Trans. Med. Imag.*, 13(4):677–686, 1994.

[33] T. E. Merryman and J. Kovačević. Adaptive multiresolution acquisition of fluorescence microscopy data sets. *IEEE Trans. Image Proc., sp. iss. Molecular and Cellular Bioimaging*, 14(9):1246–1253, Sep. 2005.

[34] P. Muller. Monte Carlo integration in general dynamic models. *Contemp. Math.*, 115:145–163, 1991.

[35] R. F. Murphy. Location proteomics: A systems approach to subcellular location. *Biochem. Soc. Trans.*, 33:535–538, Mar. 2005.

[36] R. F. Murphy, M. Velliste, and G. Porreca. Robust classification of subcellular location patterns in fluorescence microscope images. In *Proc. IEEE Int. Workshop Neur. Netw. for Signal Proc.*, pages 67–76, Sep. 2002.

[37] A.Y. Ng and M.I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 2:841–848, 2002.

[38] L. P. Panych, C. Oesterle, G. P. Zientara, and J. Hennig. Implementation of a fast gradient-echo SVD encoding technique for dynamic imaging. *Magn. Res. Imaging*, 35(4):554–62, 1996.

[39] M. K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journ. Amer. Stat. Assoc.*, 94(446):590–591, 1999.

[40] JR Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[41] D.B. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, 1979.

[42] Burr Settles. Active learning literature survey. Computer Science Technical Report 1648, University of Wisconsin–Madison, 2009.

[43] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. *Proc. Workshop Comp. Learn. Theory*, pages 287–294, 1992.

[44] D.L. Taylor and E.D. Salmon. *Fluorescence microscopy of living cells in culture: Fluorescent analogs, labeling cells, and basic microscopy*, chapter Basic Fluorescence Microscopy. Academic Press, 1989.

[45] J. Tsao, P. Boesiger, and K. P. Pruessmann. kt BLAST and kt SENSE: dynamic MRI with high frame rate exploiting spatiotemporal correlations. *Magn. Res. Imaging*, 50(5):1031–1042, 2003.

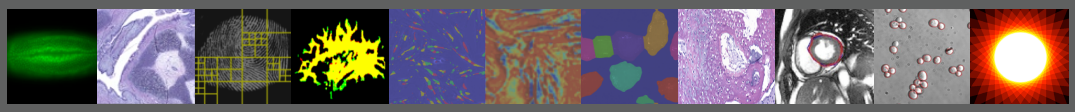
[46] M. Velliste. *Image interpretation methods for a systematics of protein subcellular location.* PhD thesis, Carnegie Mellon Univ., Carnegie Mellon Univ., 2002.

[47] M. Velliste and R. F. Murphy. Automated determination of protein subcellular locations from 3D fluorescence microscope images. In *Proc. IEEE Int. Symp. Biomed. Imaging*, pages 867–870, Washnigton, DC, 2002.

[48] M. West and J. Harrison. *Bayesian Forecasting and Dynamic Models.* Springer, 1997.

[49] H. Zhang. The optimality of naive Bayes. *Proceedings of the Seventeenth Florida Artificial Intelligence Research Society Conference*, pages 562–567, 2004.

[50] T. Zhao, M. Velliste, M. V. Boland, and R. F. Murphy. Object type recognition for automated analysis of protein subcellular location. *IEEE Trans. Image Proc., sp. iss. Molecular and Cellular Bioimaging*, 14(9), Sep. 2005.



Abstract

This thesis presents a new acquisition framework that models fluorescence microscope data during acquisition, and uses these learned models to intelligently guide future acquisitions. This framework results in significant time savings, as well as in reducing the photobleaching and phototoxicity incurred during acquisition.

Fluorescence microscopy is a popular tool for live-cell imaging, and in recent years, there has been an explosion in the amount of data acquired with this technique. Visual inspection of this data is time-consuming and not reproducible, motivating the goal of automated image analysis. Furthermore, we would ideally like to acquire all types of cells under all conditions, but standard acquisition methods are too time-consuming to achieve this feat. This work proposes to address these problems with a new acquisition framework that builds models of the data while it is being acquired, and uses these models to carry out intelligent acquisition. The goal is to reduce total acquisition time by identifying and acquiring only the data that is necessary for building the model, as well as to acquire in a way that reduces photobleaching and phototoxicity – two fundamental limitations associated with fluorescence microscopy.

We evaluate the framework experimentally on synthetic and real data. First, we present a possible method to build models of a single object within a cell, of multiple objects in a cell, and of a population of cells. Then, we present intelligent acquisition algorithms to determine where to acquire in a cell, when to acquire in a cell, when to stop acquiring from a cell, and how many cells to acquire from a population. We show that the combination of model building and intelligent acquisition results in time savings, reduced photobleaching, and reduced phototoxicity, without loss of accuracy.