UC Santa Barbara

UC Santa Barbara Electronic Theses and Dissertations

Title

Statistical and Generative Modeling Approaches for Cell Microscopy Image Segmentation and Synthesis

Permalink

https://escholarship.org/uc/item/8g3405nf

ISBN

9798186386254

Author

King, Blythe

Publication Date

2026-06-11

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA

SANTA BARBARA

Statistical and Generative Modeling Approaches for Cell Microscopy Image Segmentation and Synthesis

A dissertation submitted in partial satisfaction of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

STATISTICS AND APPLIED PROBABILITY

by

Blythe King

Committee in charge:

Professor Mengyang Gu, Chair

Professor Alex Franks

Professor Sang-Yun Oh

June 2026

The Dissertation of Blythe King is approved:

Professor Alex Franks

Professor Sang-Yun Oh

Professor Mengyang Gu, Chair

May 2026

Copyright © by

Blythe King

2026

To my parents and sister, who have always supported my academic and personal pursuits.

To my grandparents, who helped me discover my passion for analytics and coding, and provided advice for making it through graduate school.

To my dog, Arlo, who has sat through enough Zoom classes and meetings to earn his PhD, as well.

Acknowledgments

Although I somewhat last-minute decided to make the switch away from computer science and to apply to graduate programs in statistics during my senior year of college, my academic growth during the last five-and-a-half years has been invaluable, and I will be forever grateful to have made that decision. Being able to develop advanced mathematical and statistical knowledge next to the beach, with a view of the Santa Ynez mountains from your office, is an experience someone would only be able to have at UC Santa Barbara. However, although the school's location is hard to beat, the help and support of those around me is what truly made my time here special.

First, I want to express my gratitude for my advisor, Dr. Mengyang Gu. His enthusiasm for research deeply impacted me, and I am grateful for the support and consideration he extends to his students. His detailed feedback and insightful approaches proved crucial for my academic progress. I hope to take what I've learned from him into my next position and use it to continue to grow as a statistician.

I would also like to extend thanks to my internship mentors and co-workers at Johnson & Johnson and Bayer, as they helped me gain industry knowledge and experience in pharmaceutical data science and clinical statistics. I thoroughly enjoyed my summers working with them. Additionally, I would like to thank my committee members, Dr. Alex Franks and Dr. Sang-Yun Oh. I appreciate that they have taken the time to review my research and provide feedback to strengthen my analysis. I also would not have been able to pursue my PhD without my undergraduate research mentors, who introduced me to biological and analytical research. My experience in their labs and research groups helped shape my research interests today.

I also thank the UCSB graduate division for their financial support during my first year through the Regents fellowship. Furthermore, development of the unsupervised cell seg-

mentation algorithm was partially supported by National Science Foundation (NSF) Awards No. OAC-2411043 and No. DMR-2308708.

I'm so happy to have been able to complete my degree alongside my classmates. In particular, I am thankful for the support Erika McPhillips, Doris Padilla, and Ethan Marzban have provided, from getting late-night study snacks together to celebrating each other's achievements, and for the warm welcome and advice our upperclassman, Megan Elcheikhali, provided. I would also like to recognize the support and collaboration my co-authors, Haoran Yan and Laura Baracaldo Lancheros, provided. Furthermore, I'd like to thank my outside-of-the-statistics-department friends, who have made sure I had a reprieve from my studies when needed and always checked in on me. They have been there with me for so many of the great experiences I've had over the past five-and-a-half years.

Finally, I'd like to thank my family. First and foremost, thank you to my parents, Leigh and Rob King, who are always willing to listen and remind me that they are there for me. They have helped me through many cross-country drives between the east and west coasts, internship and job applications, and exam seasons. I would not have been able to finish my PhD without their encouragement. I'd also like to thank my sister, Elizabeth King, for sending lots of pictures of her cats, as they always put me in a good mood. I'm grateful for my grandmother, Mary Williams, who helped build my mathematical foundation by giving extracurricular math lessons during my childhood, and my grandparents, Drs. Nina and David King, who have provided guidance on teaching and the graduate school experience. Of course, I'd also like to thank my dog, Arlo, who always provides comfort, joy, and a lot of dog hair.

Curriculum Vitae

Blythe King

Education

2026	Ph.D. in Statistics & Applied Probability, University of California, Santa Barbara Advisor: Dr. Mengyang Gu Dissertation: *Statistical and Generative Modeling Approaches for Cell Microscopy Image Segmentation and Synthesis*
2022	M.A. in Statistics, Data Science Specialization, University of California, Santa Barbara
2020	B.A. in Computer Science, University of North Carolina at Chapel Hill

Experience

	Prenosis
2026 - present	*Machine Learning Data Scientist*
	Bayer
Summer 2025	*Statistician Intern*
	Johnson & Johnson
Summer 2024	*Data Science Intern, Advanced Analytics*
	University of California, Santa Barbara
2020 - 2024, 2025 - 2026	*Teaching Assistant*
2024 - 2025	*DataLab Graduate Student Consultant*
Summer 2023	*Teaching Associate*
Winter & Spring 2023	*Head Teaching Assistant*
	University of North Carolina at Chapel Hill
2018 - 2020	*Gillings School of Public Health, Student Research Assistant*
2017 - 2018	*Department of Biochemistry & Biophysics, Student Research Assistant*
	Environmental Protection Agency
Summer 2019	*Summer Research Assistant*

Honors & Awards

2024	Johnson & Johnson Inspire Award
2022	NSF BioPACIFIC Materials Innovation Platform (MIP) Associate
2020	Regents Fellowship
2018	Phillips Ambassador Scholarship
2017	Alpha Epsilon Delta Pre-Health Honors Society

Publications

1. Baracaldo, L.*, **King, B.***, Yan, H.*, Lin, Y., Miolane, N., & Gu, M. (2026). Unsupervised cell segmentation by fast Gaussian processes. *The New England Journal of Statistics in Data Science*, pp. 1-16.

2. Farjat, A., Ji, Y., Kaiser, A., Lu, C., Pap, A., Potts, J., **King, B.**, & Wang, M. (2025). A comprehensive Bayesian double-adjustment approach to dynamic borrowing of external data. Submitted manuscript.

3. Dhingra, R., Nwanaji-Enwerem, J., Aiello, A., Sentilles, F., **King, B.**, & Ward-Caviness, C. (2020). 'Epigenetics, Aging, and Early Life' in Fry, R. (ed.) *Environmental Epigenetics in Toxicology and Public Health.*

**Co-first authors*

Software

1. **King, B.**, Yan, H., Baracaldo, L., and Gu, M. (2026). "FastSegmentation," an `R` package for automatic and unsupervised cellular image segmentation by fast Gaussian processes. Available on the Comprehensive R Archive Network (`CRAN`).

2. Gelbart, B., with contributions from **King, B.** (2025). "Plotzing," an `R` package for customizable, publication-quality plots in a single line of code. Available on `GitHub`.

Presentations

1. Unsupervised Cell Segmentation by Fast Gaussian Processes. 2026 Biostatistics Symposium on Navigating New Frontiers: Biostatistics, AI, and Regulatory Evolution in Biomedical Research, Biostatistics Symposium of Southern California, Newport Beach, CA, January 2026. (Poster).

2. Methods for Supplementing Clinical Trial Data with External Electronic Healthcare Records. The 12th American Statistical Association (ASA) New Jersey Chapter / Bayer Workshop, ASA/Bayer, Whippany, NJ, October 2025. (Poster).

3. Unsupervised Cell Segmentation by Fast Gaussian Processes. International Women in Mathematics Day Research Seminar, UCSB Department of Statistics & Applied Probability, Santa Barbara, CA, May 2025. (Talk).

4. Unsupervised Cellular Boundary Detection by Gaussian Processes. Accepted Students Day, UCSB Department of Statistics & Applied Probability, Santa Barbara, CA, February 2025. (Talk).

5. Time Series Feature Engineering for Clinical Trial Site Selection. Johnson & Johnson Summer Intern Research Symposium, Johnson & Johnson Innovative Medicine, Spring House, PA (remote), August 2024. (Poster).

6. Detecting Cell Boundaries via Unsupervised Machine Learning. Accepted Students Day, UCSB Department of Statistics & Applied Probability, Santa Barbara, CA, February 2024. (Talk).

Abstract

Statistical and Generative Modeling Approaches for Cell Microscopy Image Segmentation and Synthesis

by

Blythe King

Cell boundary detection and segmentation enable the quantitative analysis of cellular organization, morphology, and dynamics, which are fundamental to understanding biological processes and disease mechanisms. Accurate detection and segmentation from time-lapse microscopy videos are essential for providing longitudinal information of cell boundaries. Popular unsupervised cell segmentation tools, such as ImageJ, require manual tuning of parameters and rely on restrictive assumptions about the shape of the objects. While recently developed supervised segmentation tools based on convolutional neural networks enhance accuracy, they depend on high-quality labeled images for training, making them unsuitable for segmenting new types of objects not in existing databases.

To address these limitations, we developed an unsupervised cell segmentation algorithm from noisy microscopy images using fast Gaussian processes without the need for parameter tuning or restrictive assumptions about the shape of the object. We derived robust thresholding criteria that are adaptive for heterogeneous images containing distinct brightness at different parts to separate objects from the background, and employed watershed segmentation to distinguish touching cell objects. Both simulated studies and real-data analysis of large microscopy images demonstrate the scalability and accuracy of our approach compared with existing alternatives. Our method achieves consistently higher average precision scores than ImageJ for both whole cell and cell nuclei segmentation. This unsupervised approach is

particularly valuable for segmenting novel cell types where annotated training data is limited or unavailable, making it broadly applicable to diverse microscopy imaging tasks. We have also implemented this method into an `R` package, FastSegmentation, for scalable and accessible application.

Beyond segmentation, we apply diffusion-based generative models to cell microscopy image analysis. We review denoising diffusion probabilistic models (DDPMs) and score-based generative models using stochastic differential equations (SDEs), including their theoretical foundations and training procedures. Through controlled experiments on simulated Gaussian mixture models and real cell microscopy images, we demonstrate that these models can effectively generate realistic cell images for data augmentation and denoise corrupted microscopy images while preserving cellular structures. These capabilities address critical challenges in biomedical imaging, where acquiring large labeled datasets is expensive, and images are often degraded by various noise sources during acquisition.

Chapter 1 introduces cell microscopy segmentation challenges and available segmentation and smoothing methods, including the application of Gaussian processes (GP) for smoothing 2-dimensional (2D) data. Chapter 2 details the proposed unsupervised segmentation algorithm leveraging smoothing via GP, a novel data-driven threshold, and watershed for final segmentation. Chapter 3 presents the `R` package FastSegmentation, which contains readily available functions for implementing the segmentation process described in Chapter 2. Chapter 4 reviews DDPMs and score-based generative modeling using SDEs, presents simulation analysis of Gaussian mixture models, and demonstrates applications to cell microscopy image generation and denoising. Chapter 5 provides concluding remarks and future directions.

Table of Contents

Dedication **iv**

Curriculum Vitae **vii**

Abstract **x**

List of Figures **xv**

List of Tables **xx**

1 Introduction **1**
1.1 Scalar-valued Gaussian processes 4
1.1.1 Parameter estimation for GP models 6
1.1.2 Gaussian processes with vectorized outputs 9
1.2 Image segmentation tools 11
1.2.1 ImageJ 11
1.2.2 Cellpose and U-net architecture 15
1.3 Outline 19

2 Unsupervised cell segmentation by fast Gaussian processes **21**
2.1 Introduction 22
2.2 Gaussian process models of images 25
2.2.1 The likelihood function and predictive distribution 25
2.2.2 Fast computation for images 28
2.3 Image segmentation from Gaussian processes 29
2.3.1 Image denoising from Gaussian processes on lattice 30
2.3.2 Generating binary cell masks 31
2.3.3 Labeling cell masks by the watershed algorithm 35
2.4 Numerical studies of image denoising 37
2.5 Numerical studies of cell segmentation 42
2.5.1 Criteria and methods 42
2.5.2 Image segmentation results for cell nuclei 44
2.5.3 Image segmentation results for whole cells 46

3 FastSegmentation: an R package for unsupervised cell segmentation by fast Gaussian processes **49**
3.1 Package overview . . . 49
3.2 Main functions of FastSegmentation . . . 50
3.2.1 The generate_GP_Masks function . . . 50
3.2.2 The separable_GP_param_est and predict_separable_GP functions . . . 52
3.2.3 The criterion_1 function . . . 53
3.2.4 Supporting functions . . . 55
3.3 Methods and examples . . . 57
3.3.1 Nuclear channel image segmentation . . . 57
3.3.2 Whole cell channel image segmentation . . . 58
3.3.3 Visualization with compressed output . . . 61
3.3.4 Predictive mean by separable GP with sub-image parameter estimation . . . 62
3.3.5 Predictive mean by separable GP with whole image parameter estimation 63
3.3.6 Segmentation of nonuniform images . . . 65
3.3.7 Segmentation of jammed cells . . . 65

4 Diffusion-based generative models: theory, validation, and applications to cell microscopy **68**
4.1 Introduction . . . 69
4.1.1 Variational autoencoder . . . 70
4.2 Denoising diffusion probabilistic models . . . 71
4.2.1 Forward process . . . 72
4.2.2 Reverse process . . . 73
4.2.3 Training and data generation for discrete-time denoising formulation . . . 74
4.3 Score-based generative modeling using stochastic differential equations . . . 77
4.3.1 Forward SDE . . . 77
4.3.2 Reverse SDE . . . 78
4.3.3 Training and data generation for continuous-time denoising formulation . 79
4.4 Simulations from known distributions . . . 81
4.5 Generating and denoising cell images from experimental data . . . 84
4.5.1 Generating new cell images . . . 85
4.5.2 Denoising cell images . . . 87

5 Conclusion and future work **95**
5.1 Future work and extensions . . . 96
5.1.1 Database of cell characteristics across experimental conditions . . . 96
5.1.2 Cell object tracking of GP-based segmentation results with the linear assignment problem . . . 98
5.1.3 GP-based segmentation for nucleation images . . . 99
5.1.4 Time-conditional generative models for sequential cell microscopy images 100
5.1.5 Flow matching models for cell microscopy image generation . . . 101

A Appendix of chapter 2 **102**
A.1 Proof for Lemma 1 . . . 102
A.2 Segmentation without GP smoothing . . . 105
A.3 Sensitivity analysis of threshold α_k^* . . . 107
A.4 Re-thresholding noisy sub-images with abnormally high object counts . . . 109

A.5 Flood fill algorithm . . . 111
A.6 Additional figures for watershed simulation . . . 112
A.7 Additional results of simulated studies . . . 112
A.8 Cell culturing and imaging for real data collection . . . 113
A.9 Benchmark ImageJ segmentation steps for real data analysis . . . 114
A.10 Ground truth cell mask retrieval for real data analysis . . . 116
A.11 Specifications of microscopy images . . . 117

B Appendix of chapter 4 **119**
B.1 Diffusion Model Functions . . . 119
B.2 Sensitivity of denoising prediction for score-based generative model timestep selection . . . 119

Bibliography **124**

List of Figures

1.1 Nuclear cell channel example (*left*) and whole cell channel example (*right*) [1] . . 2
1.2 U-net architecture, adapted from [2]. 16

2.1 Workflow for segmenting and labeling cell images: (A) Divide the image into different sub-images to enable locally estimated mean and variance parameters for capturing local properties such as change of brightness. (B) Compute the predictive mean of fast GPs in Section 2.2.2 to each sub-image, which greatly reduces image noise. (C) Threshold each smoothed sub-image based on the criterion discussed in Section 2.3.2 to produce binary images, separating cells from the background. The optimal threshold is estimated for each sub-image. (D) Recombine the binary sub-images into a single binary image and apply the watershed algorithm discussed in Section 2.3.3 to the image for segmentation and labeling, with each cell marked by a unique color. 30
2.2 Comparison of binary image results across thresholds based on the absolute second difference in foreground pixels. The threshold, which ranges from 0-1, refers to the proportion of the maximum value of the predictive mean that is set as the binary cutoff. Images A, B, C, D, and E are the binary images generated by setting the corresponding threshold on cropped predictive mean image F. Note that the Image B corresponds with $\text{argmax}_m \Delta c_k(\alpha_m)$ in Equation (2.11). 32
2.3 (A) Frequency of the predictive mean of the intensity values over all pixels from the same microscopy image shown in Figure 2.2 F. The optimal threshold is annotated and lies right after the bulk of the background pixel values. The thresholds that generate images A, B, C, D, and E from Figure 2.2 are represented as vertical lines with the same color. All pixel intensity values less than the optimal threshold are background pixels and have a symmetric distribution. (B) The predictive mean is shown for each pixel with the optimal threshold plotted as the horizontal plane. 34

2.4 An image of two cell nuclei (upper row) and heights of negative distances to the nearest background pixels (lower row). (A) Heights of negative distances along the red straight line in the upper panel are plotted in the lower panel before the watershed algorithm starts. (B) At water level = -13, both catch basins are partially filled, as both local minima are less than the current water level. The two separate water sources have unique labels and are not yet touching. (C) At water level at around -10.2, the water sources from the two catch basins flow into each other and the watershed line is formed at water level at around -10.2. (D) At water level = 0, the cell objects are filled with water, and the watershed operation is complete. Each cell is labeled and separated. 35
2.5 Violin plots of RMSE for five methods applied to the Branin function and the linear diffusion equation across various noise levels. Each experiment is repeated 10 times. The Fast-Mat and Fast-Exp represent the fast GPs with Matérn kernels in Equation (1.11) and exponential kernels, respectively. 38
2.6 Violin plots of RMSE for the five methods applied to images of noisy cell nuclei and whole cells across noise levels $\sigma_0^2 \in [0.1^2, 0.3^2, 0.5^2]$. Each experiment is repeated 10 times. The Fast-Mat and Fast-Exp represent the fast GPs with Matérn kernels in Equation (1.11) and exponential kernels, respectively. 39
2.7 (A) True mean of cell nuclei image. (B) Noisy observation of cell nuclei image ($\sigma_0 = 0.1$). (C) Predictive mean of cell nuclei by fast GPs on lattice data with Matérn kernels in Equation (1.11). (D) True mean of the whole cell image. (E) Noisy observation of the whole cell image ($\sigma_0 = 0.1$). (F) Predictive mean of the whole cell by Fast GPs on lattice data with Matérn kernels. 42
2.8 Comparison of the computational time in seconds for evaluating the profile likelihood of $\mathbf{Y}_v$ using direct and fast computation under varying sample sizes N. The direct computation with the exponential kernel (Direct-Exp) is plotted as orange dots; the direct computation with the Matérn kernels in Equation (1.11) (Direct-Mat) is plotted with green triangles; the fast computation with the exponential kernel (Fast-Exp) is plotted with pink squares; the fast computation with the Matérn kernels in Equation (1.11) (Fast-Mat) is plotted with blue crosses. The left panel shows the computational time in seconds, which is in the original untransformed scale, and the right panel displays the logarithmic scale. 43
2.9 Results for segmenting cell nuclei. (A) AP scores for GP-based segmentation and ImageJ across different thresholds. (B) Boxplots of AP scores for each method. These plots are created using the APs for each of the 5 images using each method across different thresholds. (C) True boundaries of the fifth test image with 1024×1024 pixels. (D) Boundaries generated by the GP-based method for the fifth test image. (E) Boundaries generated using ImageJ for the fifth test image. 45
2.10 Results for segmenting whole cells. (A) AP scores for GP-based segmentation and ImageJ across different thresholds. (B) Boxplots of AP scores for each method. (C) True boundaries of the first test image with 602×600 pixels. (D) Boundaries generated by the GP-based method of the first test image. (E) Same for panel (D) but generated using ImageJ. 47

3.1 Schematic overview of the FastSegmentation package. 51

3.2 (A) Input nuclear cell image, with color legend presenting how pixel intensity corresponds with visualized color. (B) Predictive mean of input image. (C) Binary matrix after thresholding by criterion 1, with 0 (blue) being background and 1 (red) being foreground. (D) Generated cell masks after watershed and noise filtering, with the background corresponding with 0 (blue), and each cell mask is represented by a different color corresponding with unique cell ID. 59
3.3 (A) Input whole cell image, with color legend giving how pixel intensity corresponds with visualized color. (B) Predictive mean of input image. (C) Binary matrix after thresholding by criterion 1, with 0 (blue) being background and 1 (red) being foreground. (D) Generated cell masks after watershed and noise filtering, with the background corresponding with 0 (blue), and each cell mask is represented by a different color corresponding with unique cell ID. 60
3.4 (A) Predictive mean. (B) Horizontal gradient of predictive mean, with pixel color corresponding to gradient intensity. (C) Vertical gradient of predictive mean, with pixel color corresponding to gradient intensity. (D) Gradient magnitude, with pixel color corresponding to gradient intensity. 64
3.5 Segmentation results for cell images with nonuniform cell densities across space. (A) Original 799×448 image with nonuniform cell densities. This image contains a cluster of cells, increasing local cell density, and other regions of lower cell density. (B) Segmentation results for (A). Although some regions of the image are sparsely concentrated, there is still enough information for threshold calibration by criterion 1 to be successful. (C) Cropped 799×181 image from (B). (D) Segmentation results for (C). Although each partitioned section of this image has different numbers of foreground pixels, the resulting cell masks still reflect accurate cell boundaries. 66
3.6 (A) Pixel intensity matrix for cell microscopy image with jamming. (B) Segmentation results for jammed image (A). 67

4.1 (A) Samples from the true GMM distribution (*left*), generated samples (*middle*), and overlay comparison (*right*) for the trained DDPM model. (B) Samples from the true GMM distribution, generated samples, and overlay comparison for the trained score-based SDE model. 83
4.2 (A) Sample images from training data. (B) Cell image generated by trained DDPM. (C) Cell image generated by trained score-based SDE model. 86
4.3 Original (left), noisy (middle), and denoised via DDPM (right) cell images across Gaussian noise levels $\sigma_0^2 \in [0.1^2, 0.3^2, 0.5^2]$. 88
4.4 Original (left), noisy (middle), and denoised via score-based SDE model (right) cell images across Gaussian noise levels $\sigma_0^2 \in [0.1^2, 0.3^2, 0.5^2]$. 89
4.5 (A) Original image. (B) Gaussian noise with $\sigma_0 = 2$ is added to the original image and clipped to valid pixel value $[0, 1]$. (C) DDPM prediction for the noisy image. (D) score-based SDE model prediction for the noisy image. (E) GP prediction for the noisy image. 91
4.6 Boxplots for the (A) RMSE and (B) SSIM for using DDPM (orange), score-based SDE model (green), and fast GP (blue) to denoise 500 noisy test images with Gaussian noise variance $\sigma_0^2 = 0.3^2$ compared to the original image. 92

A.1 Results for segmenting cell nuclei using the proposed GP method, and the method without GPs for image noise reduction in Section 2.3.1 and for smoothing the difference in pixel counts in Section 2.3.2. (A) AP scores for GP-based segmentation and segmentation method with GP models removed across different thresholds. (B) Boxplots of AP scores for each method. (C) True boundaries of the fifth test image with 1024×1024 pixels. (D) Boundaries generated by the GP-based method for the fifth test image. (E) Boundaries generated using the segmentation method with GP models removed for the fifth test image. 106
A.2 Results for segmenting cell nuclei using the proposed GP method, and the method without GPs for image noise reduction. (A) AP scores for GP-based segmentation and the segmentation method with GP image smoothing removed across different thresholds. (B) Boxplots of AP scores for each method. (C) True boundaries of the fifth test image with 1024×1024 pixels. (D) Boundaries generated by the GP-based method for the fifth test image. (E) Boundaries generated using the segmentation method with GP image smoothing removed for the fifth test image. 108
A.3 Empirical sensitivity analysis for α_k^* cutoff. (A) When the α_k^* cutoff is set to $0.01\tau_k$, the cell mask information does not describe the entire shape, as the threshold lies well above the base of the cell (see bottom row). (B) The α_k^* cutoff is set to $0.05\tau_k$ as described in the study methods. This cutoff retrieves binary masks that are a strong representation of the true cell shape. (C) When the α_k^* cutoff is set to $0.1\tau_k$, the cell masks are still somewhat representative of the true cell shape. However, the outlines become slightly more jagged and irregular, particularly in the bottom cells in the binary mask output. (D) When the α_k^* cutoff is set to $2.5\tau_k$, noise distorts the binary mask output, indicating that a larger cutoff is not effective. 109
A.4 Workflow of determining the true optimal threshold for selected sub-images. (A) The original image is cropped into six equal parts, and the predictive means are generated via GPs. (B) The initial thresholds α_k^* for the predictive means of two cropped images are determined using the criterion described in Section 2.3.2. (C) The number of objects in the binary sub-images is detected by flood fill. In the bottom row, despite there only being 12 cells, image noise coerces random pixel groups to be detected as foreground, leading to over 500 objects being detected in the binary sub-image. This rare scenario only happened once. (C.2) By inequality in (A.5), this sub-image is determined to have an unusually high object count. The sub-image threshold is reset to the average of all non-outlier sub-image thresholds. (C.3) Twelve objects are detected after watershed and filtering based on the cell sizes discussed in Section 2.3.3. (D) The binary sub-images are stitched, maintaining the spatial continuity of the original image. 111
A.5 (A) An example binary image of two connecting cells. (B) Corresponding negative distance map in the watershed algorithm. 113
A.6 (A) True mean generated by Branin function. (B) Noisy observations of Branin function ($\sigma_0 = 10$). (C) Predictive mean of Branin function by GPs on lattice data with Matérn kernel. (D) True mean generated by linear diffusion equation. (E) Noisy observations of linear diffusion equation ($\sigma_0 = 0.3$). (F) Predictive mean of linear diffusion equation by GPs on lattice data with Matérn kernel. . . 114

B.1 (*Left*) Noisy image. (*Right*) Predicted output when using timesteps $t = 10$ (*top*), $t = 50$ (*middle*), and $t = 150$ (*bottom*) for denoising with DDPM. 122
B.2 (*Left*) Noisy image. (*Right*) Predicted output when using timesteps $t = 0.1$ (*top*), $t = 0.6$ (*middle*), and $t = 0.75$ (*bottom*) for denoising with score-based SDE model. 123

List of Tables

1.1 Description of U-net functions. 17

4.1 Model specifications, like number of epochs, training data image count, and MLP shape, for DDPM and score-based SDE model in generating 2D GMM data. . . . 82
4.2 Model specifications, like number of epochs, training data image count, and layers in U-net, for DDPM and score-based generative model using SDE in generating and denoising cell image data. 85

A.1 Name, dimensions, channel utilized, and relevant figures for the microscopy images used in this study. 118

B.1 Function or object class name and utility for Keras DDPM implementation. . . . 120
B.2 Function or object class name and utility for the implementation of the score-based SDE model. 121

List of Notations and Symbols

$y(\mathbf{x})$	Image intensity function.
$\circ$	Element-wise product.
$\otimes$	Kronecker product.
N	Number of pixels in an image.
$\ldots^T$	Transpose.
$\ldots^{-1}$	Inverse.
$\|\|.\|\|$	Euclidean distance or Euclidean norm.
$\mathbb{E}(\ldots)$	Expected value.
Δ	Difference of functions or variables.
∇	Gradient operator.
$L(.,.)$	Laplacian.
$\mathrm{LoG}(.,.)$	Laplacian of Gaussian.
$z(.)$	Sigmoid function.
$\mathcal{D}$	Max-pooling.
$\mathcal{U}$	Transpose convolution.
$\mathrm{Vec}(\ldots)$	Vectorization operator.
$K(.,.)$	Correlation function.
$\min(.)$	Minimum.
$\max(.)$	Maximum.
$\mathrm{Var}(.)$	Variance.
$\mathrm{Cov}(.,.)$	Covariance.

Chapter 1

Introduction

Microscopy videos consist of hundreds of individual cell images and contain a vast amount of space-time data and measurements [3, 4]. Therefore, accurate and automated data processing and collection techniques, such as image segmentation [5, 6, 7, 8], object tracking [9], smoothing [10, 11], and scattering analysis [12, 13] are crucial for robust interpretations, as manual data manipulation is inconsistent and time-consuming [14]. Cell microscopy data are typically presented as large grayscale images, often acquired across two fluorescence channels [1, 15, 16]. The whole cell channel contains the cell cytoplasm, and the nuclear channel contains only the cell nuclei (see Figure 1.1). Two separate channels are achieved by staining the cell cytoplasm and nuclei with different colored dyes [17, 18].

Cell alignment refers to how cells orient with respect to some point of reference, such as the division axis in cell proliferation [19]. This attribute plays an important role in many biological processes. For example, high cell alignment in cardiac muscle tissue is required for many electro-physiological processes in the heart [20, 21]. Multiple techniques have been utilized to control cell alignment in vitro, such as changing the surrounding physical environment of cells [22]. Cell-cell and cell-substrate interactions also play a role in directing cell alignment

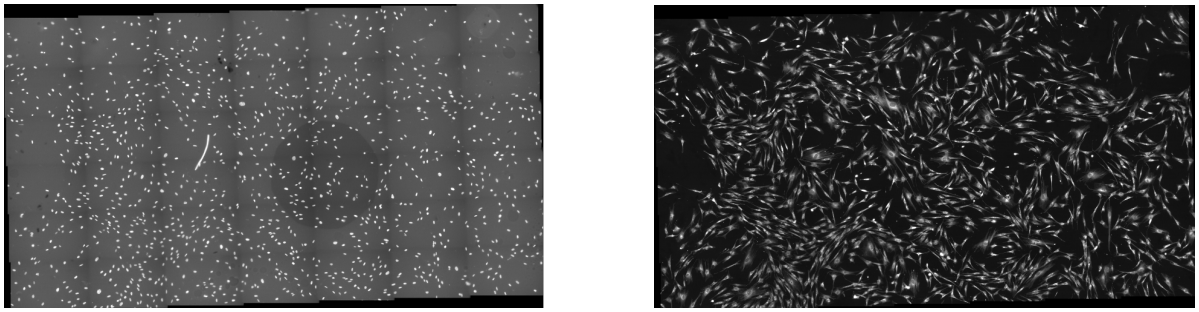


Figure 1.1: Nuclear cell channel example (*left*) and whole cell channel example (*right*) [1]

in both two-dimensional and three-dimensional space [23, 1]. Cell-substrate interactions lead to alignment via focal adhesion [24], which occurs when cells interact with the extracellular matrix due to integrins present in the plasma membrane [25]. To study how alignment changes across experimental conditions, including substrate order, seeding density, and crosslinker to monomer ratio, numerous large-scale cell cultures were observed and imaged, and cell angle and ultimately alignment were calculated with the assistance of cell image segmentation tools, specifically ImageJ. This analysis resulted in a comprehensive experimental cell alignment database, and a preliminary analysis was run to investigate how experimental conditions are associated with cell alignment across time.

The challenge of extracting cell boundary information from these cell microscopy images lies not only in their scale but also in the inherent variability of cellular appearances [26, 27]. Cells exhibit substantial heterogeneity in morphology, size, and intensity, even in homogeneous populations [28], and these characteristics can drastically vary across experimental conditions [29], time points [30], and location within a single image [31]. Traditional segmentation approaches often rely on human-driven image pre-processing [6] or threshold-based methods [32, 33, 34, 35] that perform well under specific imaging conditions but may not generalize well across different data. Furthermore, modern supervised techniques, like Cellpose [5], may require

large amounts of labeled data for model training.

The first focus of this dissertation is on the workflow and implementation of an unsupervised cell segmentation algorithm leveraging fast Gaussian processes. Unsupervised segmentation methods offer a compelling solution by learning representations directly from the data without the need to manually annotate training sets [36, 37]. Generating ground truth labels can be prohibitively time-consuming, as experimental videos can consist of hundreds of individual frames, so eliminating annotation requirements is particularly valuable in this context. Furthermore, unsupervised approaches can adapt to the specific features of each dataset, potentially detecting signal from characteristics that might be overlooked in supervised methods trained on different cell types or imaging conditions [38, 39]. This adaptability is essential for handling the diversity of cellular phenotypes and generalizing well to novel cell types.

The second focus is to discuss diffusion models [40] as a tool for image generation and denoising. Cell microscopy experiments are expensive in resources and time, so generating realistic and reasonable cell images to augment a cell image database could help reduce this cost [41]. Furthermore, diffusion models additively incorporate noise during training, making them a powerful tool for denoising in the event of image corruption [42]. Diffusion models are able to learn cell image features, such as morphology, size, and density, to recreate images that match well with the training set, making them a powerful tool for increasing the amount of data available for analysis.

In this chapter, we will introduce the background of Gaussian process (GP) models, which is necessary for understanding how they can be leveraged for image smoothing and noise filtering in subsequent chapters. We will also discuss the vector-valued partial parallel GP for efficiently estimating outputs over wide spatial domains. Finally, two of the most popular cell image analysis tools, ImageJ [6] and Cellpose [5] are reviewed, including segmentation algorithms

and tracking methodology. In particular, the background for the U-net [2] architecture of Cellpose is essential to understand the training methods of diffusion models.

1.1 Scalar-valued Gaussian processes

Let us consider a Gaussian stochastic process (GaSP), or Gaussian process (GP), for some p-dimensional input $\mathbf{x} \in \mathbb{R}^p$ (e.g., $p = 2$ for image data) and outcome $y(\mathbf{x}) \in \mathbb{R}$. As microscopy images often contain noise [43], we start with a GP model for noisy outcome $y(.)$:

$$y(.) \sim \mathcal{GP}(\mu(.), \sigma^2 K(.,.) + \sigma_0^2) \tag{1.1}$$

where $\mu(.)$ is the mean function, σ^2 is the variance parameter, $K(.,.)$ is the correlation function, and σ_0^2 is the variance of the noise in the observed output. For a set of inputs $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$ for $i = 1, ..., n$ with n being the number of inputs, the marginal distribution of the outcome vector follows the multivariate normal distribution [44]:

$$(y(\mathbf{x}_1), ..., y(\mathbf{x}_n))^T \mid \boldsymbol{\beta}, \sigma^2, \sigma_0^2, \mathbf{R} \sim \mathcal{MN}((\mu(\mathbf{x}_1), ..., \mu(\mathbf{x}_n))^T, \sigma^2\mathbf{R} + \sigma_0^2\mathbf{I}_n), \tag{1.2}$$

where $\mathbf{R}$ is the correlation matrix, with each (i, i')-th element given by $K(\mathbf{x}_i, \mathbf{x}_{i'})$, and $\mathbf{I}_n$ is a size n identity matrix. The mean function $\mu(\mathbf{x})$ is typically modeled as the following linear combination of basis functions:

$$\mu(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \sum_{t=1}^{q} h_t(\mathbf{x})\beta_t, \tag{1.3}$$

where $\mathbf{h}(\mathbf{x}) = (\mathbf{h}_1(\mathbf{x}), ..., \mathbf{h}_q(\mathbf{x}))$ is a vector of basis functions, and $\boldsymbol{\beta} = (\beta_1, ..., \beta_q)^T$ is a vector of regression parameters for each basis function in $\mathbf{h}(\mathbf{x})$.

Most commonly, the correlation function $K(.,.)$ will either be an isotropic correlation or a product correlation [45]. If $K(.,.)$ is isotropic, then it is a function of the Euclidean distance:

$$K(\mathbf{x}_i, \mathbf{x}_{i'}) = ||\mathbf{x}_i - \mathbf{x}_{i'}||, \tag{1.4}$$

for any $\mathbf{x}_i = (x_{i,1}, ..., x_{i,p})^T$ and $\mathbf{x}_{i'} = (x_{i',1}, ..., x_{i',p})^T$, and $d = ||.||$ is the Euclidean distance.

Using isotropic correlation may not be optimal when variables have different units, as Euclidean distance is not scale-invariant. Instead, the product correlation may be utilized and is especially common for emulating computer model experiments [46]:

$$K(\mathbf{x}_i, \mathbf{x}_{i'}) = \prod_{l=1}^{p} K_l(x_{i,l}, x_{i',l}), \tag{1.5}$$

where $K_l(.,.)$ is the one-dimensional (1D) correlation function for the l-th coordinate of the input vector. Using the product correlation function to calculate $K(.,.)$, the correlation matrix is decomposed as:

$$\mathbf{R} = \mathbf{R}_1 \circ \mathbf{R}_2 \circ ... \circ \mathbf{R}_p, \tag{1.6}$$

where $\mathbf{R}_l$ is the correlation matrix for the l-th input, and $\circ$ is the element-wise product. In order to calculate the correlation matrices, however, the function for $K(x_{i,l}, x_{i',l})$ must be specified.

The power exponential correlation function is as follows [47]:

$$K(x_{i,l}, x_{i',l}) = \exp\left[-(\frac{d_l}{\gamma_l})^{\alpha_l}\right], \tag{1.7}$$

where γ_l is the range parameter for the l-th input, and α_l is the roughness parameter. The parameter d_l is the Euclidean distance $||x_{i,l} - x_{i',l}||$. The power exponential correlation function is equivalent to the exponential correlation function when $\alpha_l = 1$ [47]:

$$K(x_{i,l}, x_{i',l}) = \exp\left[-\frac{d_l}{\gamma_l}\right]. \tag{1.8}$$

The power exponential correlation function is equivalent to the square exponential (SE) correlation function (also referred to as the Gaussian kernel) when $\alpha_l = 2$ [47]:

$$K(x_{i,l}, x_{i',l}) = \exp\left[-\frac{d_l^2}{\gamma_l^2}\right]. \tag{1.9}$$

GPs using the SE correlation function are infinitely mean square differentiable [48], but the resulting correlation matrix can be low rank. Thus, it can be difficult to inverse the covariance

matrix to determine the likelihood and predictive distribution in this scenario. Therefore, a power exponential correlation function with a roughness parameter close to 2 (e.g, $\alpha = 1.9$) is often selected to return a similar correlation matrix that is more easily invertible [49, 50]. However, when $\alpha < 2$, the correlation function is not differentiable and often leads to a rougher process than desired for certain applications.

The Matérn correlation [51] function enables the control of roughness and differentiability with roughness parameter α_l:

$$K(x_{i,l}, x_{i',l}) = \frac{1}{2^{\alpha_l - 1}\Gamma(\alpha_l)}(\frac{d_l}{\gamma_l})^{\alpha_l}\mathcal{K}_{\alpha_l}(\frac{d_l}{\gamma_l}), \tag{1.10}$$

where $\Gamma(.)$ is the Gamma function $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$, $\mathcal{K}(.)$ is the modified Bessel function of the second kind, and roughness parameter α_l is set such that $\alpha_l \in (0, \infty)$. This correlation function is $(\lfloor \alpha \rfloor - 1)$ times mean square differentiable [47], and it is often selected as the default choice in software packages of GP surrogate models [52, 53]. The function also converges to the SE correlation function when α_l approaches ∞. Furthermore, the Matérn correlation function has a closed-form expression when the roughness parameter is a half-integer, or $\alpha_l = m_l + \frac{1}{2}$, $m_l \in \mathbb{Z}^+$. For instance, when roughness parameter is $\alpha_l = \frac{5}{2}$, the Matérn correlation follows [54]

$$K(x_{i,l}, x_{i',l}) = \left(1 + \frac{\sqrt{5}d_l}{\gamma_l} + \frac{5d_l^2}{3\gamma_l^2}\right)\exp\left[-\frac{\sqrt{5}d_l}{\gamma_l}\right]. \tag{1.11}$$

1.1.1 Parameter estimation for GP models

Next, methods for parameter estimation in the GP model will be discussed. We will assume the GP, as given in Equation (1.2), uses a Matérn correlation function with roughness parameter $\frac{5}{2}$, as given in Equation (1.11). For this model configuration, the parameters to estimate are the mean parameter $\boldsymbol{\beta}$, variance parameter σ^2, noise variance σ_0^2, and range parameters $\boldsymbol{\gamma} = \{\gamma_1, ..., \gamma_p\}$. For a GP model with n dimensional outcome sequence $\mathbf{y} = (y(\mathbf{x}_1), ..., y(\mathbf{x}_n))^T$,

the likelihood function is expressed as:

$$p(\mathbf{y} \mid \boldsymbol{\beta}, \sigma^2, \boldsymbol{\gamma}, \eta) = (2\pi\sigma^2)^{-\frac{n}{2}} \mid \tilde{\mathbf{R}}|^{-\frac{1}{2}} \exp\left\{-\frac{(y-\mathbf{H}\boldsymbol{\beta})^T\tilde{\mathbf{R}}^{-1}(y-\mathbf{H}\boldsymbol{\beta})}{2\sigma^2}\right\}, \tag{1.12}$$

where $\tilde{\mathbf{R}} = \mathbf{R} + \eta\mathbf{I}_n$ is the covariance matrix, and the nugget parameter η is the ratio of the noise to signal variance, $\eta = \frac{\sigma_0^2}{\sigma^2}$. $\mathbf{H} = (\mathbf{h}(\mathbf{x}_1), ..., \mathbf{h}(\mathbf{x}_n))^T$ is an $n \times q$ matrix of basis functions.

Bayesian inference is often used to estimate the parameters of interest, and the standard reference prior for the mean and variance is [55]

$$\pi(\boldsymbol{\beta}, \sigma^2) \propto \frac{1}{\sigma^2}. \tag{1.13}$$

After integrating out the mean $\boldsymbol{\beta}$ and variance σ^2 parameters, the marginal distribution of the nugget η and range $\boldsymbol{\gamma}$ parameters is

$$p(\mathbf{y} \mid \boldsymbol{\gamma}, \eta) \propto \int p(\mathbf{y} \mid \boldsymbol{\beta}, \sigma^2, \boldsymbol{\gamma}, \eta)\pi(\boldsymbol{\beta}, \sigma^2)d\boldsymbol{\beta}d\sigma^2, \tag{1.14}$$

which also has the explicit expression [55]:

$$p(\mathbf{y} \mid \boldsymbol{\gamma}, \eta) \propto \mid \tilde{\mathbf{R}} \mid^{-\frac{1}{2}} \mid \mathbf{H}^T\tilde{\mathbf{R}}^{-1}\mathbf{H} \mid^{-\frac{1}{2}} (S^2)^{-\frac{n-q}{2}}, \tag{1.15}$$

where $S^2 = (\mathbf{y} - \mathbf{H}\hat{\boldsymbol{\beta}})^T\tilde{\mathbf{R}}^{-1}(\mathbf{y} - \mathbf{H}\hat{\boldsymbol{\beta}})$, and $\hat{\boldsymbol{\beta}} = (\mathbf{H}^T\tilde{\mathbf{R}}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\tilde{\mathbf{R}}^{-1}\mathbf{y}$ is the generalized least squares estimator of $\boldsymbol{\beta}$.

The range parameters $\boldsymbol{\gamma}$ and the nugget parameter η can be estimated through maximizing the marginal likelihood [56]:

$$(\hat{\gamma}, \hat{\eta}) = \arg\max_{(\gamma,\eta)} \{p(\mathbf{y} \mid \boldsymbol{\gamma}, \eta)\pi(\boldsymbol{\gamma}, \eta)\}, \tag{1.16}$$

where $\pi(\boldsymbol{\gamma}, \eta)$ is a reference prior or a jointly robust prior [57] with robust parameterization [56].

After estimating the $\boldsymbol{\gamma}$ and η parameters, we can obtain the predictive distribution $y(\mathbf{x}^*)$, where $\mathbf{x}^*$ is a new input, by integrating out the mean $\boldsymbol{\beta}$ and variance σ^2 parameters:

$$p(y(\mathbf{x}^*) \mid \mathbf{y}, \hat{\boldsymbol{\gamma}}, \hat{\eta}) = \int p(y(\mathbf{x}^*) \mid \mathbf{y}, \boldsymbol{\beta}, \sigma^2, \hat{\boldsymbol{\gamma}}, \hat{\eta})\pi(\boldsymbol{\beta}, \sigma^2)d\boldsymbol{\beta}d\sigma^2 \tag{1.17}$$

Following the standard reference prior in Equation (1.13), the predictive distribution $p(y(\mathbf{x}^*) \mid \mathbf{y}, \hat{\boldsymbol{\gamma}}, \hat{\eta})$ has a Student t-distribution with $n - q$ degrees of freedom:

$$y(\mathbf{x}^*) \mid \mathbf{y}, \boldsymbol{\beta}, \sigma^2, \hat{\boldsymbol{\gamma}}, \hat{\eta} \sim \mathcal{T}(\hat{y}(\mathbf{x}^*), \sigma^2 K^{**}, n - q), \tag{1.18}$$

where

$$\hat{y}(\mathbf{x}^*) = \mathbf{h}(\mathbf{x}^*)\hat{\boldsymbol{\beta}} + \mathbf{r}^T(\mathbf{x}^*)\tilde{\mathbf{R}}^{-1}(\mathbf{y} - \mathbf{H}\hat{\boldsymbol{\beta}}), \tag{1.19}$$

$$\hat{\sigma}^2 = \frac{1}{n-q}(\mathbf{y} - \mathbf{H}\hat{\boldsymbol{\beta}})^T\tilde{\mathbf{R}}^{-1}(\mathbf{y} - \mathbf{H}\hat{\boldsymbol{\beta}}), \tag{1.20}$$

and,

$$\begin{aligned} K^{**} = K(\mathbf{x}^*, \mathbf{x}^*) + \hat{\eta} - \mathbf{r}^T(\mathbf{x}^*)\tilde{\mathbf{R}}^{-1}\mathbf{H} + \\ (\mathbf{h}(\mathbf{x}^*) - \mathbf{r}(\mathbf{x}^*)^T\tilde{\mathbf{R}}^{-1}\mathbf{H})(\mathbf{H}^T\tilde{\mathbf{R}}^{-1}\mathbf{H})^{-1}(\mathbf{h}(\mathbf{x}^*) - \mathbf{r}(\mathbf{x}^*)^T\tilde{\mathbf{R}}^{-1}\mathbf{H})^T. \end{aligned} \tag{1.21}$$

The term $\hat{\boldsymbol{\beta}}$ is the generalized least squares estimator for the mean parameter $\boldsymbol{\beta}$:

$$\hat{\boldsymbol{\beta}} = (\mathbf{H}^T\tilde{\mathbf{R}}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\tilde{\mathbf{R}}^{-1}\mathbf{y},$$

and $\mathbf{r}(\mathbf{x}^*) = (K(\mathbf{x}^*, \mathbf{x_1}), ..., K(\mathbf{x}^*, \mathbf{x}_n))^T$ can be calculated by inputting the estimated range parameter $\hat{\boldsymbol{\gamma}}$. The `R` package `RobustGaSP` can be utilized to estimate the parameters and predictive distribution of a GP model [53].

Another popular way to estimate parameters is the maximum likelihood estimation (MLE). As the likelihood of the GP model follows Equation (1.12), given the range $\boldsymbol{\gamma}$ and nugget η parameters, the MLE of the mean parameter $\boldsymbol{\beta}$ is

$$\hat{\boldsymbol{\beta}}_{MLE} = (\mathbf{H}^T\tilde{\mathbf{R}}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\tilde{\mathbf{R}}^{-1}\mathbf{y}, \tag{1.22}$$

and the MLE of the variance parameter σ^2 is

$$\hat{\sigma}^2_{\text{MLE}} = \frac{S^2}{n}, \tag{1.23}$$

where $S = (\mathbf{y}-\mathbf{H}\hat{\boldsymbol{\beta}}_{MLE})\tilde{\mathbf{R}}^{-1}(\mathbf{y}-\mathbf{H}\hat{\boldsymbol{\beta}}_{MLE})$. After inputting estimated parameters $(\hat{\boldsymbol{\beta}}_{MLE}, \hat{\sigma}^2_{MLE})$ and ignoring the constant term, the likelihood in Equation (1.12) reduces to

$$p(\mathbf{y} \mid \hat{\boldsymbol{\beta}}_{MLE}, \hat{\sigma}^2_{MLE}, \gamma, \eta) \propto |\tilde{\mathbf{R}}|^{-\frac{1}{2}} \ (S^2)^{-\frac{n}{2}}. \tag{1.24}$$

The MLEs of range parameters $\boldsymbol{\gamma}$ and nugget parameter η do not have closed form expressions and are instead estimated through numerical optimization of the logarithm of the profile likelihood in Equation (1.24), such as quasi-Newton optimization methods [58]. Both MLE and marginal posterior mode estimation have been implemented in the RobustGaSP package [56].

1.1.2 Gaussian processes with vectorized outputs

Gaussian process emulators are powerful tools to reproduce data from complex numerical systems, such as environmental processes, biological behaviors, and sensing mechanics [59, 60, 61], providing both predictions and uncertainty quantification estimates for simulated output. However, many emulations interpret and produce massive amounts of data to cover a spatial-temporal domain and can be extremely computationally expensive. The partial-parallel Gaussian process (PP-GP) [50] addresses this computational bottleneck using the assumption that emulation outputs across different spatial-temporal coordinates are likely highly correlated. In this approach, the parallel emulators for different locations are still independent, as in the Many Single (MS) emulator approach [62, 63, 64], but the correlation functions are shared.

Consider the entire simulation output $\mathbf{y}(\mathbf{x}) = (y_1(\mathbf{x}), ..., y_k(\mathbf{x}))^T$ for input $\mathbf{x}$. For PP-GP emulation, it is assumed that each coordinate is modeled via an independent Gaussian process:

$$y(.) \sim \mathcal{GP}(\mu(.), \sigma^2 K(.,.)). \tag{1.25}$$

The mean is commonly modeled using regression, as in Equation (1.3). Applying this to the

simulator output at the j-th coordinate, $\mathbf{y}_j(\mathbf{x})$, $\mathbf{h}(\mathbf{x})\boldsymbol{\beta}_j$ consists of the prior mean functions of regression, where $\mathbf{h}(\mathbf{x})$ is a common vector of basis functions with length q, and $\boldsymbol{\beta}_j$ are different unknown regression coefficients for the basis function. Furthermore, the unknown variance priors σ_0^2 differ, and the estimated correlation parameters, or range parameters, $\hat{\boldsymbol{\gamma}}$, are common. Assuming $\mathbf{h}(\mathbf{x})$ and $\hat{\boldsymbol{\gamma}}$ are shared across independent Gaussian processes leads to better computational efficiency.

Consider a set of n selected inputs from the simulator, $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, and the corresponding $k \times n$ matrix of simulator outputs, $\mathbf{y} = (\mathbf{y}(\mathbf{x}_1), ..., \mathbf{y}(\mathbf{x}_n))$ to develop the emulator. The objective prior for the mean and variance parameters follows [50]

$$\pi(\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_k, \sigma_1^2, ..., \sigma_k^2) \propto \frac{1}{\prod_{j=1}^{k} \sigma_j^2}. \tag{1.26}$$

As, given $\boldsymbol{\gamma}$, the coordinate-wise GPs are independent, and $\hat{\boldsymbol{\gamma}}$ is common, the overall GP for a new input $\mathbf{x}^*$ at the j-th coordinate a Student's t-distribution with $n - q$ degrees of freedom:

$$y_j(\mathbf{x}^*) \mid \mathbf{y}, \hat{\boldsymbol{\gamma}} \sim \mathcal{T}(\hat{y}_j(\mathbf{x}^*), \hat{\sigma}_j^2 c^{**}, n - q), \tag{1.27}$$

where the predictive mean and variance parameters follow:

$$\hat{y}_j(\mathbf{x}^*) = \mathbf{h}(\mathbf{x}^*)\hat{\boldsymbol{\beta}}_j + \mathbf{r}^T(\mathbf{x}^*)\mathbf{R}^{-1}(\mathbf{y}_j - \mathbf{H}\hat{\boldsymbol{\beta}}_j), \tag{1.28}$$

and

$$\hat{\sigma}_j^2 = (n - q)^{-1}(\mathbf{y}_j - \mathbf{H}\hat{\boldsymbol{\beta}}_j)^T\mathbf{R}^{-1}(\mathbf{y}_j - \mathbf{H}\hat{\boldsymbol{\beta}}_j), \tag{1.29}$$

with $\hat{\boldsymbol{\beta}}_j = (\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{y}_j$. As the nugget parameter is not being considered, c^{**} follows:

$$\begin{aligned} c^{**} = K(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{r}^T(\mathbf{x}^*)\mathbf{R}^{-1}\mathbf{r}(\mathbf{x}^*) + (\mathbf{h}(\mathbf{x}^*) - \mathbf{H}^T\mathbf{R}^{-1}\mathbf{r}(\mathbf{x}^*))^T \\ \times (\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}))^{-1}(\mathbf{h}(\mathbf{x}^*) - \mathbf{H}^T\mathbf{R}^{-1}\mathbf{r}(\mathbf{x}^*)), \end{aligned} \tag{1.30}$$

where K is a correlation function.

The advantages of PP-GP lie on its computational scalability which takes only $\mathcal{O}(n^3)+\mathcal{O}(n^2k)$ operations for training the model and making predictions, and its flexiblity of having different mean and variance parameters on each output coordinates. Because of these advantages, PP-GP surrogate models have been used in distinct areas of applications [65, 66, 67, 68, 69, 70, 71, 72, 73, 74]. The robust estimation for both scalar-valued GP and PP-GP for vectorized output is implemented in the RobustGaSP package [53].

When the sample size n is large, the primary computational bottleneck of the PP-GP emulator is the inversions of $\mathbf{R}^{-1}$, which increases cubically fast along with n. When the design is lattice or incomplete lattice, such as for image input data, then the correlation matrix $\mathbf{R}$ has a separable or Kronecker product structure [62] to reduce matrix inversion cost, as will be shown in Chapter 2. The separable structure can be extended to semi-separable structure [75, 76, 77], where Kalman filter [78] and inverse Kalman filter [79] were used to accelerate the computation.

1.2 Image segmentation tools

1.2.1 ImageJ

ImageJ [6, 80] is an unsupervised cell segmentation user interface developed by the National Institute of Health (NIH) that contains multiple cell image processing, tracking, and visualization tools. It models cells as ellipses by its default settings, returning the shape's angle, major and minor axes, and area. However, obtaining these estimates requires user-defined parameters, like estimated cell size, and user-driven pre-processing steps [6], like applying a Gaussian blur for denoising [81], which convolves the original image $y(x_1, x_2)$ with a Gaussian kernel, $K_{\text{Gaussian}}(x_1, x_2) = \frac{1}{2\pi\sigma}\exp\left[-\frac{x_1^2+x_2^2}{2\sigma^2}\right]$, for smoothing. The output is extremely sensitive

to the user-defined input, and ImageJ's default parameters in the program are often not reasonable. For example, the default estimated cell size is 0 to ∞ pixels, so the best-fit ellipse would be found for any detected object, including small areas of noise and large foreign objects [6].

The three main steps to detect cell objects using default methods in ImageJ are as follows. First, the binary matrix must be generated from the original grayscale cell image. By the default method, the threshold is set to approximately half the sum of the average cell pixel value and the average background pixel value, Threshold $\approx \frac{\mu_{\text{cell}}+\mu_{\text{background}}}{2}$, which is determined iteratively through one of several available global thresholding methods [82, 32]. Pixel values lying above the threshold are cell pixels, and pixel values lying below are background pixels,

$$B(x_1, x_2) = \begin{cases} 1, & y(x_1, x_2) \geq \text{Threshold} \\ 0, & y(x_1, x_2) < \text{Threshold}. \end{cases} \tag{1.31}$$

The algorithm then scans the binary matrix until it reaches the edge of a cell object, given by a group of foreground pixels, and it traces around the object until the start point is reached to determine the boundary [83]. ImageJ then fits an ellipse by matching the second central moments of the region of interest (ROI). For a single connected region of interest, let $(\bar{x}_1, \bar{x}_2)$ be the centroid, and let $x_1 \in \mathbf{x}_{1,\,\text{ROI}}$ be the x_1-coordinates in the ROI, and let $x_2 \in \mathbf{x}_{2,\,\text{ROI}}$ be the x_2-coordinates in the ROI. Major and minor axes are then derived from the eigenvectors of the covariance matrix [84]:

$$\mathbf{\Sigma} = \frac{1}{N_{\text{ROI}} - 1} \sum_{x_1 \in \mathbf{x}_{1,\,\text{ROI}}} \sum_{x_2 \in \mathbf{x}_{2,\,\text{ROI}}} \begin{pmatrix} x_1 - \bar{x}_1 \\ x_2 - \bar{x}_2 \end{pmatrix} \begin{pmatrix} x_1 - \bar{x}_1 \\ x_2 - \bar{x}_2 \end{pmatrix}^T, \tag{1.32}$$

where N_{ROI} is the total number of pixels in each ROI.

Tracking cell trajectories with TrackMate

TrackMate [9] is a plug-in for the ImageJ user-interface that tracks cell trajectories across a series of frames. It is useful for calculating the cell velocity or the change in angle across time, as it will link cell identities between subsequent images in a cell video. It first leverages the Laplacian of Gaussian (LoG) for detecting cell objects on a frame, and then the linear assignment problem (LAP) is used to link cells. TrackMate still requires many user-defined parameters, such as the estimated cell radius.

All spot detectors available in the TrackMate algorithm are based on LoG filtering [85]. The Laplacian of an image matrix with pixel intensity values y is given as

$$L(x_1, x_2) = \frac{\partial^2 y}{\partial x_1^2} + \frac{\partial^2 y}{\partial x_2^2}. \tag{1.33}$$

The Laplacian of Gaussian filter combines $L(x_1, x_2)$ with a Gaussian kernel to reduce image noise before differentiation:

$$L(K_{\text{Gaussian}}) = \frac{\partial^2 K}{\partial^2 x_1^2} + \frac{\partial^2 K}{\partial x_2^2}, \tag{1.34}$$

and calculating the second-order derivative for each of x_1 and x_2 yields

$$\begin{aligned} \frac{\partial^2 K}{\partial x_1^2} &= \frac{1}{2\xi^4\pi}\left(\frac{x_1^2}{\xi^2} - 1\right)\exp\left[-\frac{x_1^2 + x_2^2}{2\xi^2}\right], \\ \frac{\partial^2 K}{\partial x_2^2} &= \frac{1}{2\xi^4\pi}\left(\frac{x_2^2}{\xi^2} - 1\right)\exp\left[-\frac{x_1^2 + x_2^2}{2\xi^2}\right], \end{aligned} \tag{1.35}$$

where ξ is the standard deviation of the Gaussian kernel and is selected to control the degree of smoothing. Since the image matrix is discrete, the LoG can be approximated using the following convolutional kernel:

$$\text{LoG}(x_1, x_2) = -\frac{1}{\pi\xi^4}\left(1 - \frac{x_1^2 + x_2^2}{2\xi^2}\right)\exp\left\{-\left(\frac{x_1^2 + x_2^2}{2\xi^2}\right)\right\}. \tag{1.36}$$

The LoG is the equivalent of applying Gaussian smoothing before edge detection, and the default parameter in the ImageJ implementation is $\xi = 5$ [9, 6]. At the edges of objects in

an image, the resulting matrix after convolution will be 0, so cell objects are detected by the location of their boundaries.

Cell trajectories are then tracked across frames via LAP [86], which links cells with global combinatorial optimization. The objective is to minimize

$$\hat{A}_s = \sum_{s_1} \sum_{s_2} A_{s_1 s_2} C_{s_1 s_2} \tag{1.37}$$

such that

$$\sum_{s_1=1}^{\text{row \#}} A_{s_1 s_2} = 1, \qquad \sum_{s_2=1}^{\text{col \#}} A_{s_1 s_2} = 1.$$

First, this algorithm creates a series of track segments based on location measures that informed the values in cost matrix $C_{s_1 s_2}$, where s_1 indexes cells on frame $t-1$ and s_2 indexes cells on frame t. The assignments are tracked in an assignment matrix $A_{s_1 s_2}$, where $A_{s_1 s_2} = 1$ if cell s_1 is linked to cell s_2 and 0 otherwise. However, these track segments were largely incomplete due to cells disappearing, being erroneously dropped, or moving out of microscope view, so track segments are also linked using cost matrix $C_{S_1 S_2}$, where S_1 indexes the end of a track segment, and S_2 indexes the start of a track segment. These assignments are tracked in matrix $A_{S_1 S_2}$. This creates an additional objective for minimization:

$$\hat{A}_S = \sum_{S_1} \sum_{S_2} A_{S_1 S_2} C_{S_1 S_2}. \tag{1.38}$$

The values of the cost matrices $C_{s_1 s_2}$ and $C_{S_1 S_2}$ are based on the location and spatial features of the cells. For example, both track initiation, where a cell on frame t cannot be linked to another cell on frame $t-1$, and track termination, where a cell on frame $t-1$ cannot be linked to another cell on frame t, both have a cost of 1.05 $\times$ the maximum cost of all previous links. The cost value d_l^2 is the squared Euclidean distance between cell s_1 on frame t and cell s_2 on frame $t-1$. The cost value d_g^2 is equal to the Euclidean distance between the end of track segment S_1 and the start of track segment S_2 squared. Additional cost values include track

merging, as cells may overlap or become stuck together, and splitting, as cells divide during the cell cycle.

After the cost matrices are populated, the assignments that minimize Equations (1.37) and are determined using the Hungarian algorithm [87]. A simplified explanation of this method is as follows. For each iteration, the cost matrix row and column minimums are subtracted from each row and column. The zeroes in the altered matrix are crossed using the smallest number of lines. If the number of lines matches the number of columns or rows, an optimal assignment can be made among the zeroes. If not, the smallest non-zero element is subtracted from all other elements, and the process is repeated. The frame-to-frame cell linking is optimized and assigned in $\hat{A}_s$ before track-to-track linking, as track segments are dependent on the frame-to-frame cell linking attributes.

1.2.2 Cellpose and U-net architecture

Cellpose [5, 88] is a deep learning–based framework for cell segmentation that is designed to work robustly across a wide range of cell types and experimental conditions. It contains multiple pre-trained models for different cell types and channels, including segmentation models for whole cell and nuclear channels, referred to as a model zoo. Unlike traditional unsupervised segmentation approaches, Cellpose predicts spatial flow fields and cell probability maps using a U-net–style architecture, enabling accurate delineation of individual cells without extensive manual tuning or dataset-specific retraining.

U-net

U-net is a popular convolutional neural network widely used for biological image segmentation problems [2]. The U-net architecture, illustrated in Figure 1.2, consists of two main

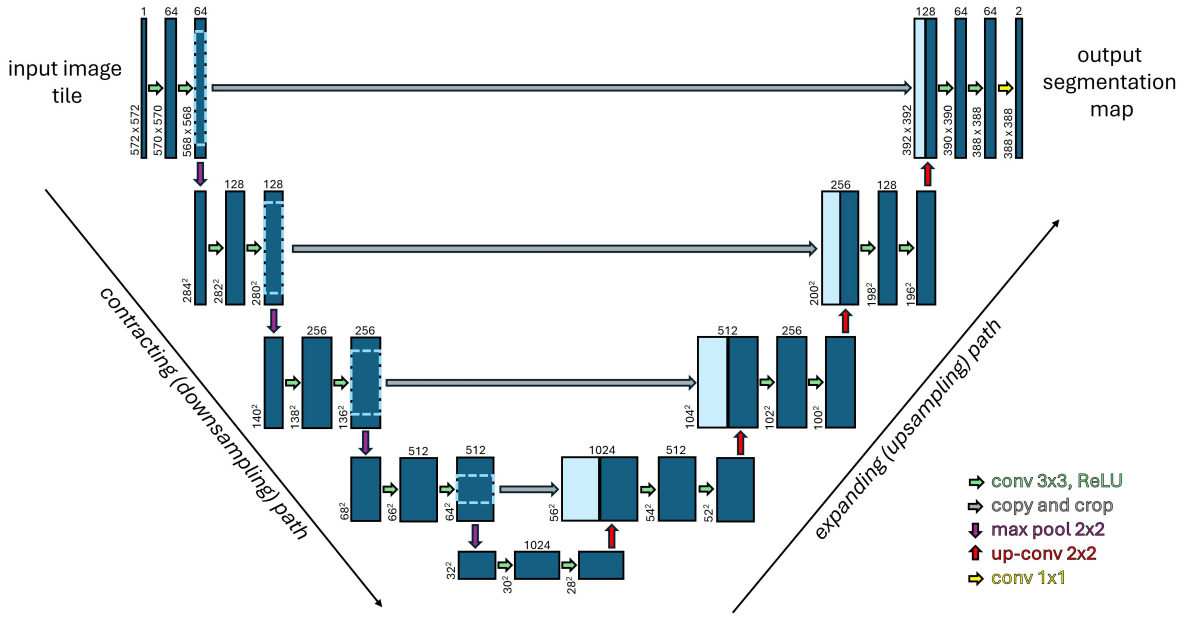


Figure 1.2: U-net architecture, adapted from [2].

steps: the contracting (downsampling) path, corresponding to the "downwards" segment of the U-shape, and expanding (upsampling) path, corresponding to the "upwards" segment of the U-shape. The downsampling path consists of repeated applications of convolutional layers followed by ReLU and max-pooling, defined in Table 1.1, progressively reducing the spatial resolution while learning increasingly abstract feature representations. The upsampling path consists of repeated up-convolutions, or transposed convolutions as defined in Table 1.1, followed by feature concatenations with corresponding layers from the downsampling path via skip connections. This design enables the network to recover spatial resolution and refine precise features after dimensionality reduction. A list of common functions used in U-net training and modeling is given in Table 1.1.

Table 1.1: Description of U-net functions.

Function	**Description**
max-pooling ($\mathcal{D}_{a\times b}$)	Selects the maximum value in an $a \times b$ window as it slides across the matrix to reduce dimensionality [89]
batch normalization	Normalizes input to mean 0 and unit variance, and then applies scaling and shifting using learned parameters [90]
ReLU	$\max(0, x)$, applied to the input element-wise [89]
$a \times b$ convolution	Passes a kernel with window size $a \times b$ and learned weights over input, summarizing each local patch into one value [91]
$a \times b$ transpose convolution	Passes a kernel with window size $a \times b$ and learned weights over input, and outputs the value into an $a \times b$ panel to increase dimensionality [92]

Cellpose architecture

The input of the Cellpose U-net is a two-channel cell image with the whole cell channel and an optional nuclear channel. The architecture of the Cellpose U-net closely follows that in Figure 1.2. For all main analyses [5], a base of 32 feature maps was used in the first layer, which grew by a factor of two at each downsampling step. Up to 256 feature maps were in the layer with the lowest resolution, at the "bottom" of the U-shape.

For each layer in the downsampling pass, the following operations are performed:

$$\mathbf{x}'_t = \mathcal{D}_{2\times 2}(\mathbf{x}_{t-1})$$

$$\mathbf{x}^*_t = \mathcal{F}(\mathcal{F}(\mathbf{x}'_t)) + \mathcal{P}_{1\times 1}(\mathbf{x}'_t)$$

$$\mathbf{x}_t = \mathcal{F}(\mathcal{F}(\mathbf{x}^*_t)) + \mathbf{x}^*_t,$$

where $\mathbf{x}_t$ are the feature maps at layer t, where $\mathbf{x}_t[k]$ represents a single feature map k of dimension (h, w). $\mathcal{D}_{2\times 2}$ is the downsampling operation max-pooling, as defined in Table 1.1, and $\mathcal{F}$ combines batch normalization, a rectified linear unit (ReLU), and 3×3 convolution operations, all defined in 1.1. The number of feature maps at each layer for each input image is k, and (h, w) are the height and width of the image and depend on the dimensions of the input.

Parameters for each $\mathcal{F}$ (ReLU weights, convolutional filter values) are trained via gradient descent. $\mathcal{P}_{1\times1}$ is 1x1 convolution.

Between the downsampling and upsampling passes, at the layer with the lowest resolution, the image style is computed, which gives the global average for each of the feature maps for each image. The normalized averages given are in style vector $\mathbf{s}^*$:

$$\mathbf{s}[k] = \sum_{i,j} \mathbf{x}_4[k,i,j]$$

$$\mathbf{s}^* = \frac{\mathbf{s}}{||\mathbf{s}||}.$$

For the given architecture, $\mathbf{s}^*$ contains 256 values, as there are 256 feature maps at the layer with the lowest resolution for this analysis.

For each layer in the upsampling pass, the following operations are performed:

$$\mathbf{z}'_t = \mathcal{U}_{2\times2}(\mathbf{z}_{t+1})$$

$$\mathbf{z}^*_t = \mathcal{G}(\mathbf{s}^*, \mathcal{F}(\mathbf{z}'_t) + \mathbf{x}_t) + \mathcal{P}_{1\times1}(\mathbf{z}'_t)$$

$$\mathbf{z}_t = \mathcal{G}(\mathbf{s}^*, \mathcal{G}(\mathbf{s}^*, \mathbf{z}^*_t)) + \mathbf{z}^*_t,$$

where $\mathcal{U}_{2\times2}$ is 2×2 transpose convolution, as defined in Table 1.1. Function $\mathcal{G}(\mathbf{x})[k,i,j]$ adds a projection of the style vector and is equal to $\mathcal{G}(\mathbf{x})[k,i,j] = \mathcal{F}(\mathbf{x})[k,i,j] + \mathcal{P}(\mathbf{s}^*)[k]$. $\mathcal{F}$ and $\mathcal{P}_{1\times1}$ remain the same as in the downsampling pass.

The output of final layer is run through three 1×1 convolutions. The first two outputs $(\mathbf{y}_0, \mathbf{y}_1)$ are used to directly calculate horizontal and vertical gradients using an L2 loss. The last output $(\mathbf{y}_2)$ is passed through a sigmoid to calculate the probability each pixel is inside a true cell mask. The cost function for the Cellpose training algorithm is:

$$\text{Cost} = ||\mathbf{y}_0 - 5\mathbf{C}||^2 + ||\mathbf{y}_1 - 5\mathbf{V}||^2 + \text{L}(z(\mathbf{y}_2), \mathbf{P}), \tag{1.39}$$

where L is binary cross-entropy loss, $\mathbf{P}$ is the hand-drawn binary cell mask, $\mathbf{V}$ and $\mathbf{C}$ are the vertical and horizontal heat source gradients, respectively, and $z(x)$ is the sigmoid $z(x) = \frac{1}{1+\exp(-x)}$. Cell masks can be retrieved from the final outputs using gradient flow tracking. Considering only pixels within the original image with a greater than or equal to 50% chance of belonging to a cell mask, the gradient for each pixel is traced over 200 iterations to determine the approximate cell center.

To train cellpose, the training data is generated via heat diffusion of hand-drawn binary cell masks. The centers of each cell mask, defined as the median x- and y-coordinates of each mask, act as the heat sources, and the edges of the masks were set to 0. For N_{iter} iterations, where N_{iter} is twice the sum of the horizontal and vertical ranges, a constant value of 1 is added to the mask centers, and each pixel within the cell mask is set to the average of the pixels in its surrounding 3×3 block of pixels. The final output of this process is used to calculate the vertical and horizontal heat source gradients, based on the centers of the cell masks, respectively.

1.3 Outline

Chapter 2 introduces an automatic and unsupervised cell segmentation algorithm leveraging fast Gaussian processes [11]. We begin by introducing a method for calculating the predictive mean of an image via fast, separable GPs. We then detail the workflow for the unsupervised cellular image segmentation algorithm, including image smoothing by GP, binary thresholding using a robust data-driven criterion, and final segmentation using a watershed operation. We also generate cell masks for both the cell nuclear and whole cell channels, by our methods and a benchmark ImageJ segmentation method to compare their accuracy with the manual, ground-truth labels.

Chapter 3 presents the `R` package FastSegmentation, which provides an automatic and user-friendly function for running the entire segmentation workflow detailed in Chapter 2. Multiple examples of image segmentation with this package are provided. This package works similarly for multiple file types, in particular `JPEG`, `PNG`, and `TIFF`, and the cell mask information can be output and visualized in multiple different fashions. Additionally, example code for calculating the predictive mean of an image using fast GPs, as well as outputting gradient information to analyze the rate and direction of changes in pixel intensity in the predictive mean, is detailed.

Chapter 4 introduces two diffusion models: denoising diffusion probabilistic models (DDPMs) and score-based generative models using stochastic differential equations (SDEs). This chapter details the mathematical theory underlying each approach and validates model performance using simulated bimodal data, comparing generated results to the known distribution. The practical usage of these models is demonstrated through analysis of real-world cell microscopy data, where both approaches realistically generated image data. Furthermore, both approaches are able to denoise images effectively, and images can be recovered when corrupted with both minor noise and noise that almost completely obscures the original features.

Chapter 5 summarizes the main discoveries and contributions and suggests future directions for research. Extensions for cell tracking using segmentation results from Chapter 2, unsupervised segmentation for nucleation and crystallization image data, and generative models for predicting future frames in microscopy video sequences are outlined. As the ultimate goal of image segmentation is to retrieve cell boundary information to inform biological processes, a preliminary database containing cell density and cell alignment measurements across different experimental conditions is introduced.

Chapter 2

Unsupervised cell segmentation by fast Gaussian processes

In this chapter, we present a novel unsupervised cell segmentation algorithm leveraging fast GPs for image smoothing and denoising, a data-driven threshold for automatic foreground object detection, and watershed for final segmentation and object labeling [11]. Our approach is entirely free of manual parameter selection or user-driven pre-processing. It takes advantage of the separable covariance structure of image data to reduce computational cost of computing the predictive mean for noise reduction. Additionally, the lattice structure of image data yields closed-form expressions for the predictive distribution. We demonstrate that our method outperforms a benchmark segmentation ImageJ method, which requires user-driven tuning and processing at each step, when compared to ground-truth cell masks.

2.1 Introduction

The spatial organization of cells, such as orientation, density, and shape, are fundamental to characterizing physical mechanisms and biological functions [93, 94], as the arrangement of cells influences their functions, including proliferation and differentiation [95]. For example, high cell density can signal cells to die to maintain tissue homeostasis [96], and cell shape has been linked to DNA synthesis and cell growth [97]. Advances in microscopy techniques enable detailed video recordings of cellular motions. Image segmentation approaches have been developed for microscopy images [7], where the segmentation results, including cell locomotion, alignment, and density, can be used for modeling cell behaviors [98, 99, 79, 100]. Therefore, precise cell image segmentation is essential for quantifying cell location, alignment, and morphology, as these statistics enhance our understanding of how cellular spatial organization influences human health outcomes [101, 102].

Cellular segmentation tools can be broadly classified as unsupervised and supervised methods, based on whether labeled data is needed for training the models. One of the most popular tools of unsupervised cellular segmentation is the ImageJ [103, 80, 104], which enables segmenting both cell nuclei and whole cells, and linking cells across frames of a microscopy video via plug-ins such as TrackMate [9]. Prior to using ImageJ for cell detection and segmentation, a series of user-driven image pre-processing steps must be completed in the program. For instance, the image is typically converted into a binary image first, based on the threshold chosen either manually or automatically via several global threshold setting methods [82, 32]. Additional image processing steps, such as despeckling for noise reduction [83], typically need to be run before segmentation. To detect cell objects within an image, the algorithm scans the corresponding binary image until the edge of an object is reached. Then, the algorithm traces around the edge of the object until the starting point is reached. This process is repeated

for pixels of each object, or the foreground pixels, in the binary image. The centroid of each cell object can be identified by fitting an ellipse for each object using the second-order central moments [80], which assumes the cell boundary is elliptical. The parameters for the best-fit ellipse, including the major and minor axes and centroids, and matrices for the object masks for each object boundary, are recorded in ImageJ [83]. Additional functionality, such as adjusting brightness automatically or calculating elliptical outlines across an entire video sequence, can be added to the user interface by writing macro code to streamline image analysis or adding third-party plug-ins [80]. However, achieving reasonable segmentation results is often labor-intensive and requires human interaction at each step. Furthermore, the assumption of the shape of the cells in ImageJ is restrictive. Though the ellipse can approximate nuclei of certain cells, it cannot approximate the shape of cytoplasm or the whole cell of frequently used cells in experiments, such as fibroblasts or epithelial cells.

Supervised cellular segmentation methods are typically built upon a convolutional neural network structure, such as U-net [2]. A popular tool is Cellpose, which builds U-nets for segmenting images that contain one type of cell that are trained on a dataset of over 70,000 segmented objects, including annotated images that contain numerous types of cells [5]. The model first compresses the image and then reconstructs the image to its original dimensions through a U-net to capture both local and global features. It further generates a topological map from diffusion simulation to product gradient maps for the model to learn the horizontal and vertical gradients of the image. The final layer of Cellpose produces three feature maps, including the horizontal and vertical directional gradients, and the sigmoid transformation of the probability map of the output image from the U-net, and the parameters of the Cellpose are then estimated by fitting a loss function of the gradient vector and binary label of each pixel of the image. As the Cellpose segmentation requires massive training data, it may not work well

for detecting novel cell types, where high-labeled image data that may not be readily available. Furthermore, training Cellpose and using it for segmenting large images with several million pixels can be computationally expensive.

In response to the challenges the current methods face, we present a novel unsupervised cell image segmentation method for segmenting images containing one type of cells, based on separable Gaussian processes, automated thresholding, and watershed operations for image segmentation. The proposed method has several advantages. First, the separable covariance structure on image data enables us to decompose the large covariance into a Kronecker product of two small covariances, which substantially reduces the computational cost for computing the matrix inversion and the logarithm of the determinant of the covariance matrix. Second, unlike conventional unsupervised image segmentation methods such as ImageJ, which require manual tuning of multiple hyperparameters, including Gaussian blur radius, thresholding method, and minimum object size, our proposed segmentation pipeline is free of manual parameter selection. We developed fast algorithms for maximum likelihood estimation of the parameters in GPs and the predictive distribution. The predictive mean surface naturally facilitates foreground-background separation by applying a data-driven threshold, with the optimal threshold automatically estimated based on the absolute second difference in foreground pixel counts between the thresholds. Third, the cell masks are segmented using the watershed algorithm, which simulates water flowing through a topographical map of the image to separate cell objects. Fourth, our approach is entirely unsupervised, negating any need for annotated training data, in contrast to supervised methods, such as Cellpose. Thus, the proposed method is particularly suitable for segmenting new cell types or other biological structures. We studied our methodology by simulated data for noise filtering. We also compared our approaches with the alternatives using real microscopy images for segmenting both the cell nucleus and whole cell

in optically dense images, a challenging scenario even when labeled data are available. These studies demonstrate that our approach substantially improves the alternatives.

The remainder of the chapter is organized as follows. In Section 2.2, we discuss GP models of images and a fast algorithm for parameter estimation and predictive distribution. In Section 2.3, we introduce our workflow for cellular image segmentation from Gaussian processes, including image smoothing, binary thresholding, and segmentation. In Section 2.4, we used simulated data and microscopy images of cells to compare our methods and other alternatives for denoising, including principal component analysis and dynamic mode decomposition. We also demonstrate the computational advantage of the fast algorithm against direct computation. Finally, we generate cell masks for both the cell nucleus and cytoplasm, or the whole cell, by our methods and ImageJ segmentation method to compare their accuracy with the manual labels in Section 2.5. Our findings not only present a novel cell segmentation technique but also highlight the applicability of fast Gaussian processes in image processing tasks. The data and code of this article are made publicly available: `https://github.com/UncertaintyQuantification/cell_segmentation`.

2.2 Gaussian process models of images

2.2.1 The likelihood function and predictive distribution

Consider a GP model for a two-dimensional (2D) microscopy image, where the intensity at the pixel location $\mathbf{x}$ is modeled as $y(\mathbf{x}) = f(\mathbf{x}) + \epsilon$, where $f(\cdot)$ is a latent GP and $\epsilon \sim \mathcal{N}(0, \sigma_0^2)$ is Gaussian noise [47]. Consider an $n_1 \times n_2$ image $\mathbf{Y}$, where the (i, j)th term is $y(\mathbf{x}_{i,j})$ the pixel intensity for the pixel location $\mathbf{x}_{i,j} = (x_{i,1}, x_{j,2})$. The data matrix can be

written as:

$$\mathbf{Y} = \mathbf{F} + \mathbf{E}, \tag{2.1}$$

where $\mathbf{F}$ follows a matrix-normal distribution $\mathbf{F} \sim \text{Matrix-Normal}\left(\mu\mathbf{1}_{n_1\times n_2},\, \sigma^2\mathbf{R}_1, \mathbf{R}_2\right)$. The mean parameter is μ, and $\mathbf{1}_{n_1\times n_2}$ is a $n_1 \times n_2$ matrix of ones. The variance parameter is σ^2, and $\mathbf{R}_1$ and $\mathbf{R}_2$ are the correlation matrices for two inputs. The vectorized noise matrix follows $\text{Vec}(\mathbf{E}) \sim \mathcal{MN}(0, \sigma_0^2\mathbf{I}_{N\times N})$, with $N = n_1 n_2$. Additional trend structure can be modeled in the mean function, and the covariance can be generalized to be semi-separable in the model [105].

The observation matrix can be vectorized to $\mathbf{y}_v = \text{Vec}(\mathbf{Y})$. After marginalizing out the latent signal matrix $\mathbf{F}$, the observation vector of the image follows a multivariate normal distribution:

$$(\mathbf{y}_v \mid \sigma^2, \eta, \mathbf{R}_1, \mathbf{R}_2) \sim \mathcal{MN}(\mu\mathbf{1}_N, \sigma^2\tilde{\mathbf{R}}), \tag{2.2}$$

where $\tilde{\mathbf{R}} = \mathbf{R} + \eta\mathbf{I}_N$, $\mathbf{R} = \mathbf{R}_2 \otimes \mathbf{R}_1$, $\otimes$ is a Kronecker product, and $\eta = \sigma_0^2/\sigma^2$ is a nugget parameter. For this equation, the correlation matrices are parameterized kernel functions $R_1(i, i') = K_1(x_{i,1}, x_{i',1})$ and $R_2(j, j') = K_2(x_{j,2}, x_{j',2})$. Frequently used kernel functions include the power exponential kernel and Matérn kernel [47]. This output model is similar to that described in Equation (1.2).

Denote $K_l(x_{i,l}, x_{i',l}) = K_l(d_l)$ with $d_l = |x_{i,l} - x_{i',l}|$. For instance, the Matérn kernel with roughness parameter $\frac{5}{2}$ follows Equation (1.11), where γ_l is a range parameter to be estimated for $l = 1, 2$. We use the kernel function in Equation (1.11) for demonstration purposes.

After specifying the model, we compute the likelihood function and predictive distribution for parameter estimation and prediction, sequentially. The parameters of the model in Equation (2.2) include the mean, variance, range and nugget parameters $\{\mu, \sigma^2, \boldsymbol{\gamma}, \eta\}$, with $\boldsymbol{\gamma} = (\gamma_1, \gamma_2)$ being the range parameters in the kernel function in Equation (1.11). In order to reduce computational time due to the large image size, we employ the maximum likelihood

estimator (MLE) to estimate the parameters, as the result is similar to the robust estimation by marginal posterior mode [56]. Given the range and nugget parameters, the MLE of the mean and variance parameters has a closed-form expression: $\hat{\mu} = (\mathbf{1}_N^T \tilde{\mathbf{R}}^{-1} \mathbf{1}_N)^{-1} \mathbf{1}_N^T \tilde{\mathbf{R}}^{-1} \mathbf{y}_v$, where $\hat{\sigma}^2 = \frac{S^2}{N}$ with $S^2 = (\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)^T \tilde{\mathbf{R}}^{-1} (\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)$. The logarithm of the profile likelihood after plugging in the MLE of the mean and variance parameters follows:

$$\log(L(\boldsymbol{\gamma}, \eta)) = C - \frac{1}{2}\log(\mid \tilde{\mathbf{R}} \mid) - \frac{N}{2}\log(S^2), \tag{2.3}$$

where C is a constant and not related to $(\boldsymbol{\gamma}, \eta)$.

The range and nugget parameters are then estimated by maximizing the logarithm of the profile likelihood function:

$$(\hat{\boldsymbol{\gamma}}, \hat{\eta}) = \underset{(\boldsymbol{\gamma}, \eta)}{\operatorname{argmax}} \log(L(\boldsymbol{\gamma}, \eta)). \tag{2.4}$$

After vectorizing the signal $\mathbf{f}_v = \mathrm{vec}(\mathbf{F})$ from model (2.1) and plugging in the MLE of the parameters, the posterior distribution of the image vector follows:

$$(\mathbf{f}_v \mid \mathbf{y}_v, \hat{\mu}, \hat{\sigma}^2, \hat{\eta}, \hat{\boldsymbol{\gamma}}) \sim \mathcal{MN}(\mathbf{f}_v^*, \hat{\sigma}^2 \mathbf{R}^*), \tag{2.5}$$

where the predictive mean $\mathbf{f}_v^*$ and predictive covariance $\mathbf{R}^*$ are:

$$\mathbf{f}_v^* = \hat{\mu}\mathbf{1}_N + \mathbf{R}\tilde{\mathbf{R}}^{-1}(\mathbf{y}_v - \hat{\mu}\mathbf{1}_N), \tag{2.6}$$

$$\mathbf{R}^* = \mathbf{R} - \mathbf{R}\tilde{\mathbf{R}}^{-1}\mathbf{R}. \tag{2.7}$$

The predictive mean and predictive variances, which are given in the diagonal terms of $\hat{\sigma}^2\mathbf{R}^*$, are required for prediction and uncertainty quantification. However, direct computation of the likelihood function involves inverting the covariance matrix, which takes $\mathcal{O}(N^3)$ operations, and the number of pixels N can be at the order of $10^6 - 10^7$ for microscopy images. A fast computational way without approximation is introduced in Section 2.2.2 to solve this computational challenge.

2.2.2 Fast computation for images

A wide range of approximation approaches for GPs have been developed [106, 107, 108, 109, 110, 111, 112, 113, 114]. For GPs with product covariances on image data, no approximation is required. The eigendecomposition of the subcovariance is denoted as $\mathbf{R}_l = \mathbf{U}_l \mathbf{\Lambda}_l \mathbf{U}_l^T$, where $\mathbf{U}_l$ and $\mathbf{\Lambda}_l$ are matrices of eigenvectors and a diagonal matrix of eigenvalues for $l = 1, 2$, respectively. Furthermore, the n_l-vector of the diagonal terms in $\mathbf{\Lambda}_l$ is denoted as $\boldsymbol{\lambda}_l = (\lambda_{1,l}, ..., \lambda_{n_l,l})^T$, for $l = 1, 2$.

The log profile likelihood in Equation (2.3) and the predictive distributions in Equation (2.5) can be written as a function of eigenpairs, as shown in Lemma 1 below. Lemma 1 eliminates the need to invert the $N \times N$ covariance matrix and reduces the computational complexity to $\mathcal{O}(n_1^3 + n_2^3)$, which is significantly smaller than $\mathcal{O}(n_1^3 n_2^3)$. The proof of Lemma 1 is provided in the Appendix.

Lemma 1. *1. (Profile likelihood). Denote the transformed likelihood vector* $\tilde{\mathbf{y}}_v = Vec(\tilde{\mathbf{Y}}) = Vec(\mathbf{U}_1^T(\mathbf{Y} - \hat{\mu}\mathbf{1}_{n_1 \times n_2})\mathbf{U}_2)$. *The logarithm of the profile likelihood in Equation (2.3) can be written as*

$$\begin{aligned} \log(L(\boldsymbol{\gamma}, \eta)) =& C - \frac{1}{2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\log\left(\lambda_{i,1}\lambda_{j,2} + \eta\right) - \\ & \frac{N}{2}\log\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\frac{\tilde{Y}_{i,j}^2}{\lambda_{i,1}\lambda_{j,2} + \eta}\right), \end{aligned} \tag{2.8}$$

where

$$\begin{aligned} \hat{\mu} = & \left(\sum_{j=1}^{n_2}\sum_{i=1}^{n_1}\frac{\tilde{u}_{i,1}^2\tilde{u}_{j,2}^2}{\lambda_{i,1}\lambda_{j,2} + \eta}\right)^{-1} \times \\ & \left(\sum_{j=1}^{n_2}\sum_{i=1}^{n_1}\frac{\tilde{u}_{i,1}\tilde{Y}_{i,j,0}\tilde{u}_{j,2}}{\lambda_{i,1}\lambda_{j,2} + \eta}\right), \end{aligned} \tag{2.9}$$

with $\tilde{Y}_{i,j}$ *being the* (i,j)*th entry in* $(\mathbf{U}_1^T(\mathbf{Y} - \hat{\mu}\mathbf{1}_{n_1 \times n_2})\mathbf{U}_2)$, $\tilde{Y}_{i,j,0}$ *being the* (i,j)*th entry of*

the $n_1 \times n_2$ *matrix* $\mathbf{U}_1^T \mathbf{Y} \mathbf{U}_2$, $\tilde{u}_{i,1}$ *being the* i*th term of the* n_1 *vector* $\mathbf{1}_{n_1}^T \mathbf{U}_1$ $i = 1, ..., n_1$ *and* $\tilde{u}_{j,2}$ *being the* j*th term of the* n_2 *vector* $\mathbf{1}_{n_2}^T \mathbf{U}_2$, *for* $j = 1, ..., n_2$.

2. *(Predictive distribution). The predictive mean vector in Equation (2.6) follows*

$$\mathbf{f}_v^* = \hat{\mu} \mathbf{1}_N + Vec(\mathbf{U}_1 \mathbf{\Lambda}_1 \tilde{\mathbf{Y}}_0 \mathbf{\Lambda}_2 \mathbf{U}_2^T), \tag{2.10}$$

where $\tilde{\mathbf{Y}}_0$ *is a* $n_1 \times n_2$ *matrix with the* (i, j)*th entry being* $\frac{\tilde{Y}_{i,j}}{\lambda_{i,1} \lambda_{j,2} + \eta}$ *for* $i = 1, \ldots, n_1$ *and* $j = 1, \ldots, n_2$. *The predictive variance at the* (i, j)*th pixel follows*

$$\hat{\sigma}^2 c_{i,j}^* = \hat{\sigma}^2 \left(1 - \sum_{j'=1}^{n_2} \sum_{i'=1}^{n_1} \frac{\tilde{r}_{i,i',1}^2 \tilde{r}_{j,j',2}^2}{\lambda_{i,1} \lambda_{j,2} + \eta} \right)$$

where $\tilde{r}_{i,i',1}$ *is the* i'*th term of the vector* $\mathbf{r}_{i,1}^T \mathbf{U}_1$ *and* $\tilde{r}_{j,j',2}$ *is the* j'*th term of the vector* $\mathbf{r}_{j,2}^T \mathbf{U}_2$, *for* $i' = 1, ..., n_1$ *and* $j' = 1, ..., n_2$, *with* $\mathbf{r}_{i,1}^T = (K_1(x_{i,1}, x_{1,1}), ..., K_1(x_{i,1}, x_{n_1,1}))^T$ *and* $\mathbf{r}_{j,2}^T = (K_2(x_{j,2}, x_{1,2}), ..., K_2(x_{j,2}, x_{n_2,2}))^T$.

2.3 Image segmentation from Gaussian processes

Although Gaussian processes have been widely applied in surrogate models to emulate expensive computer simulations, nonparametric regression, and modeling spatially correlated data [47, 115], the flexibility and scalability of GPs in unsupervised image segmentation have not been studied yet. The overall flow of our cellular image segmentation process is summarized in Figure 2.1. First, some microscopy images of cells are particularly large with $10^6 - 10^7$ pixels. Similar to other image segmentation methods [80, 5], we segment each large image into multiple sub-images. We then apply the fast algorithm of GPs in Lemma 1 to get the predictive mean of each sub-image, illustrated in Parts (A) and (B) in Figure 2.1, respectively. We assume that the range parameters and nugget of the GPs are the same for each sub-image estimated by MLE in Lemma 1, but we do not maintain this assumption for the mean and variance

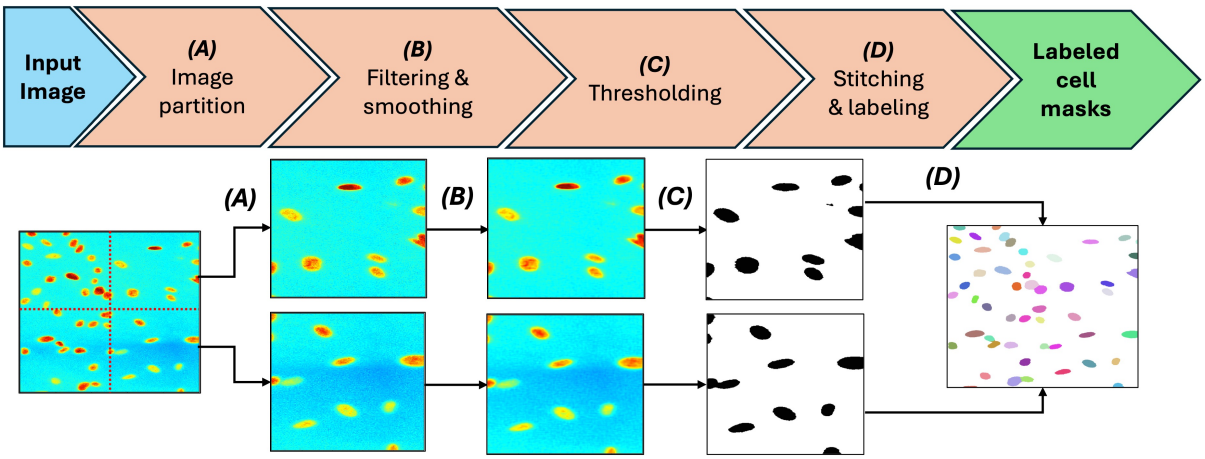


Figure 2.1: Workflow for segmenting and labeling cell images: (A) Divide the image into different sub-images to enable locally estimated mean and variance parameters for capturing local properties such as change of brightness. (B) Compute the predictive mean of fast GPs in Section 2.2.2 to each sub-image, which greatly reduces image noise. (C) Threshold each smoothed sub-image based on the criterion discussed in Section 2.3.2 to produce binary images, separating cells from the background. The optimal threshold is estimated for each sub-image. (D) Recombine the binary sub-images into a single binary image and apply the watershed algorithm discussed in Section 2.3.3 to the image for segmentation and labeling, with each cell marked by a unique color.

parameters. Instead, we assume that they are distinct for each separable sub-image, and the estimation of these parameters has a closed-form expression. This assumption enables the separably estimated means and variances to capture the change in optical properties, such as differences in brightness and out-of-focus blur from distinct parts of a large microscopy image, which is crucial in practice.

2.3.1 Image denoising from Gaussian processes on lattice

Let $\mathbf{Y}$ represent the original cell image, which has size $N = n_1 \times n_2$. We denote $y(\mathbf{x}_{i,j})$ as the value of the image at pixel $\mathbf{x}_{i,j} = (x_{i,1}, x_{j,2})$. All possible pixel values are between 0 and 1, as each image in this analysis is in grayscale. Cell microscopy images can contain more than 10^6 pixels, and some areas of an image can have substantially higher pixel values

than other areas. Thus, for a large image $\mathbf{Y}$, it is normally partitioned into a set of cropped images, denoted $\{\mathbf{Y}_k\}_{k=1}^{\tilde{K}}$, where $\tilde{K}$ is the total number of cropped images as illustrated in Figure 2.1. This step can substantially reduce the algorithm's computational cost and enables more adaptive segmentation outcomes to the features of the local areas in a large image.

As experimental images often contain substantial noise, for each cropped image, we first use GPs with fast computation introduced in Section 2.2.1 for denoising the images. The range and nugget parameters of GPs are estimated by maximizing the likelihood in Equation (2.8), and they are held the same for all sub-images as they have similar smoothness properties. The mean and variance parameters are estimated differently for each cropped image to capture the local change of optical properties. As the sub-images will be turned into a binary image, the discontinuity of the boundary between different sub-images does not impact the image segmentation task.

2.3.2 Generating binary cell masks

The predictive mean at the interior pixel of a cell object typically has a larger value than the one in the background of a microscopy image. Inspired by the IsoData algorithm [116] in ImageJ [103], we developed an automated and robust way to determine the threshold value for separating objects from background noise using the predictive mean from GP models.

Suppose each cropped image is $n_{k,1} \times n_{k,2}$ with predictive mean $\mathbf{F}_k^*$ with the (i,j)th entry being $F_k^*(\mathbf{x}_{i,j})$ and $\text{Vec}(\mathbf{F}_k^*) = \mathbf{f}_{k,v}^*$ given in Equation (2.10). As the intensity values of the pixels are between 0 to 1 and the predictive uncertainty is small, the range of $F_k^*(\mathbf{x}_{i,j})$ is also approximately between 0 to 1. As the image has only one type of cells, the change of pixel values $F_k^*(\mathbf{x}_{i,j})$ is large if $\mathbf{x}_{i,j}$ is the pixel at the boundary of a cell object, and small elsewhere.

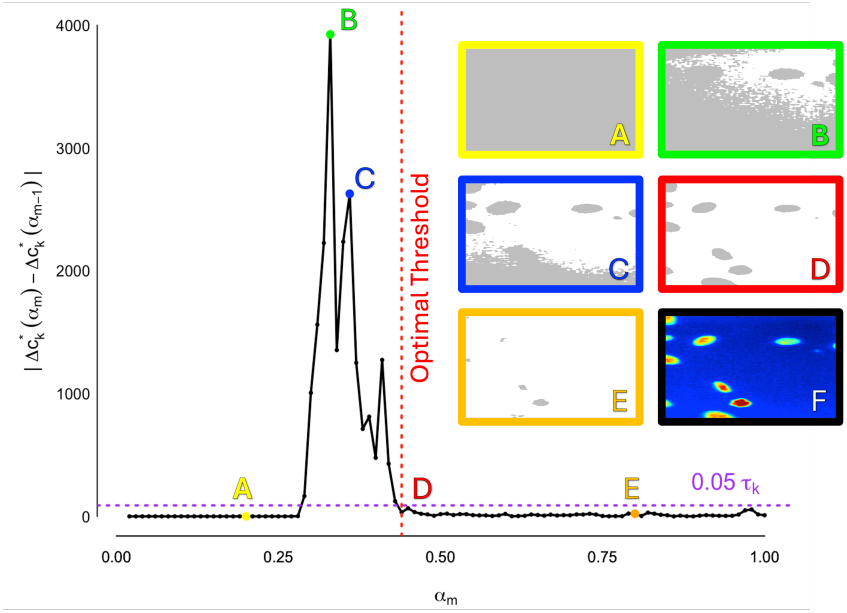


Figure 2.2: Comparison of binary image results across thresholds based on the absolute second difference in foreground pixels. The threshold, which ranges from 0-1, refers to the proportion of the maximum value of the predictive mean that is set as the binary cutoff. Images A, B, C, D, and E are the binary images generated by setting the corresponding threshold on cropped predictive mean image F. Note that the Image B corresponds with $\text{argmax}_m \Delta c_k(\alpha_m)$ in Equation (2.11).

Thus we first compute the normalized predictive mean exceeds each threshold value:

$$c_k(\alpha_m) = \sum_{i,j} 1_{\frac{F_k^*(\mathbf{x}_{i,j})}{\max(\mathbf{F}_k^*)} > \alpha_m},$$

where $\{\alpha_m\}_{m=1}^M$ a sequence of equally spaced thresholds from 0 to 1 with the default value $M = 100$, and $\max(\mathbf{F}_k^*)$ denotes the maximum value of the predictive mean $\mathbf{F}_k^*$ of the kth cropped image for $k = 1, ..., \tilde{K}$. In general, the range of $\{\alpha_m\}_{m=1}^M$ can be chosen to be the range of $F_k^*(\mathbf{x}_{i,j})/\max(\mathbf{F}_k^*)$. Subsequently, we calculate the difference $\Delta c_k(\alpha_m)$ in pixel counts:

$$\Delta c_k(\alpha_m) = c_k(\alpha_{m-1}) - c_k(\alpha_m).$$

As $\Delta c_k(\alpha_m)$ may not be smooth over α_m, we compute the predictive mean of $\Delta c_k(\alpha_m)$, denoted by $\Delta c_k^*(\alpha_m)$, via a GP model with the default Matérn kernel in Equation (1.11) by the RobustGaSP package [53].

We define α^* as the smallest threshold α_m after the point of maximum fluctuation, denoted as $\arg\max_m \Delta c_k^*(\alpha_m)$, where the smoothed differences $\Delta c_k^*(\alpha_m)$ stabilize within a specified tolerance relative to the variability of $\Delta c_k(\alpha_m)$, an example of which is plotted in Figure 2.2. Formally, let τ_k represent the standard deviation of the differences $\Delta c_k^*(\alpha_m)$. The optimal threshold α_k^* is given by

$$\alpha_k^* = \min_m \{\alpha_m :\mid \Delta c_k^*(\alpha_m) - \Delta c_k^*(\alpha_{m-1}) \mid < 0.05\tau_k,$$
$$m > \arg\max_m \Delta c_k^*(\alpha_m)\}. \tag{2.11}$$

Figure 2.2 illustrates this process for determining the optimal threshold for segmenting cell objects from the background. While the first difference $\Delta c_k(\alpha_m)$ measures the immediate change in the number of pixels above each threshold, the second difference more precisely captures the stabilization process of these changes. Initially, at extremely small thresholds, almost all pixels are classified as foreground or cell objects, leading to minimal changes in pixel counts between threshold values, shown as A in Figure 2.2. However, once the threshold starts to intersect the main background distribution, even small increments can trigger substantial variations, resulting in high second differences. As the threshold increases, the second difference reaches its maximum value at B in Figure 2.2 and then becomes erratic between B and C in Figure 2.2. This erratic behavior occurs because, in this transitional range, as the predictive mean values near the decision boundary are very similar, even minor changes in the threshold result in disproportionately large shifts in the number of pixels classified as foreground or cell objects. Beyond this range, the second difference begins to stabilize and falls below a predetermined tolerance level show as E in Figure 2.2, indicating that further increases yield only negligible changes in segmentation. Any further increase in the threshold results in over-segmentation.

Panel (A) in Figure 2.3 plots the location of different threshold values in the distri-

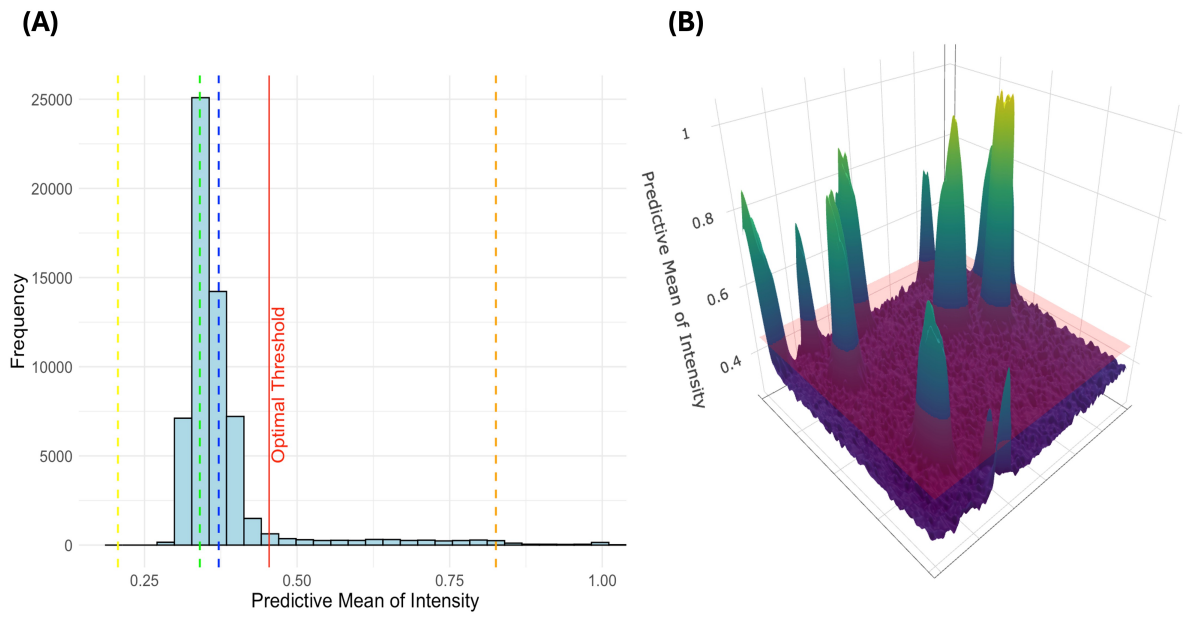


Figure 2.3: (A) Frequency of the predictive mean of the intensity values over all pixels from the same microscopy image shown in Figure 2.2 F. The optimal threshold is annotated and lies right after the bulk of the background pixel values. The thresholds that generate images A, B, C, D, and E from Figure 2.2 are represented as vertical lines with the same color. All pixel intensity values less than the optimal threshold are background pixels and have a symmetric distribution. (B) The predictive mean is shown for each pixel with the optimal threshold plotted as the horizontal plane.

bution of the image's pixel intensity. The optimal threshold selected by the algorithm lies on the right side of the mode. Panel (B) of Figure 2.3 plots a horizontal plane of the optimal threshold value. A threshold value much smaller than this value misclassifies the background as cell objects, whereas a threshold value much larger than the threshold value misses some small cell objects with low intensity peaks.

After estimating the threshold value α_k^*, we construct a binary image $\mathbf{B}_k^*$ for each cropped image at each pixel:

$$\mathbf{B}_k^*(\mathbf{x}_{i,j}) = \begin{cases} 1 & \text{if } F_k^*(\mathbf{x}_{i,j}) > \alpha_k^* \max(\mathbf{F}_k^*) \\ 0 & \text{otherwise} \end{cases}$$

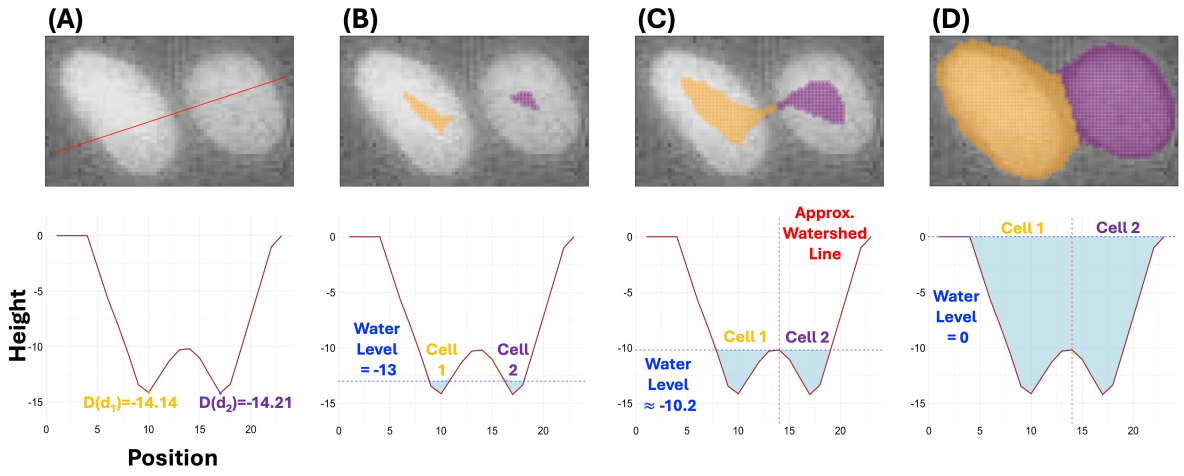


Figure 2.4: An image of two cell nuclei (upper row) and heights of negative distances to the nearest background pixels (lower row). (A) Heights of negative distances along the red straight line in the upper panel are plotted in the lower panel before the watershed algorithm starts. (B) At water level $= -13$, both catch basins are partially filled, as both local minima are less than the current water level. The two separate water sources have unique labels and are not yet touching. (C) At water level at around -10.2, the water sources from the two catch basins flow into each other and the watershed line is formed at water level at around -10.2. (D) At water level $= 0$, the cell objects are filled with water, and the watershed operation is complete. Each cell is labeled and separated.

for $k = 1, ..., \tilde{K}$. We complete panel (C) in Figure 2.1 by generating binary cell masks for all sub-images. To address a rare scenario with a higher-than-expected object count, which only happened once for all sub-images, we implemented a re-thresholding step discussed in Appendix A.4. Finally, each object group is separated and detected via flood fill [117], as described in Section A.5 in the Appendix.

2.3.3 Labeling cell masks by the watershed algorithm

After obtaining the binary image, we apply the `watershed` algorithm to separate cell objects from `EBImage` package in `R` [118], which is particularly useful for segmenting objects connecting to each other. The watershed algorithm creates a topological representation of an image based on a distance map, which is derived from the binary image representing foreground

objects (cells). The distance map assigns a value to each foreground pixel based on its Euclidean distance to the nearest background pixel; higher values represent larger distances from the background. These distance values act as heights in a topological landscape, providing a basis for the watershed function to simulate water flowing through. The negative distance map is then calculated by taking the negative of the distance map. Examples of the binary image and negative distance map are provided in Appendix A.6.

The local minima of the negative distance map generally correspond to the centers of cell objects and are used as the starting point for the "flooding" process in the watershed algorithm. The water fills the cell object from these local minima as the negative distance map is immersed into water, so the cell object acts as a catch basin as the map floods. As water from different starting points meets, watershed lines are formed, which delineate boundaries for distinct binary cell objects. Additionally, whenever the water from a catch basin meets a background pixel, a watershed line is formed [119]. These watershed lines separate and create distinct cell objects, and these cell objects can be assigned unique labels.

An example of the watershed algorithm at distinct water levels is provided in Figure 2.4 with two cells denoted by d_1 and d_2. The top row of Figure 2.4 plots the cell images where the negative heights of a slice over the red line are plotted in the bottom row. Figure 2.4 (A) shows depths for the two local minima for the 1D example are $D(d_1) = -14.14$ and $D(d_2) = -14.21$. Each height acts as a water level, and the local minima act as seed points for water to flow through into each valley or catch basin. When the water level rises to touch a local minimum, the water is given the same label as the local minimum. As the water level rises, all unidentified pixels at or below the water level with neighboring labeled pixels are given the same label.

Figure 2.4 (B) shows the water levels at -13. Each catch basin is partially filled, as

the water level is higher than the two local minima. As the water in each valley is still well separated, all emerged pixels in each valley are either labeled corresponding with d_1 or d_2, with no neighboring pixels containing a different label.

At water level around -10.2, shown in Figure 2.4 (C), the water from the two different basins begins to touch. The water from the two catch basins corresponding to d_1 and d_2 meet at the unlabeled pixel p_0 with a position close to 14, plotted in the x-coordinate. The Euclidean distances between p_0 and the local minima are utilized to determine its assignment, and a watershed line is formed. The Euclidean distance between p_0 and d_2 (≈ 5.01 units) is smaller than the distance between p_0 and d_1 (≈ 5.62 units), so p_0 is grouped with cell 2. The watershed line will serve as the boundary between the two cells.

At the water level 0, the objects are completely segmented, shown in Figure 2.4 (D). The watershed operation then halts, as this height is used as a stopping point since all cell objects have negative heights in the negative distance map. Additional details of the watershed algorithm, including definitions and edge cases, can be found in [119].

Once all objects are segmented by watershed, the average foreground object pixel count is calculated. To filter out any noise that may have been included as a foreground object, all objects with pixel counts less than or equal to 15% of the average are removed. If an object is touching the image boundary, the threshold count for removal is reduced to 5% of the average, as some objects could be partially cut off during imaging.

2.4 Numerical studies of image denoising

In this section, we numerically compare the performance of the proposed fast GPs with several alternative methods using simulated experiments. The comparison includes the fast GP of a Matérn kernel with roughness parameter $\alpha = 2.5$ in Equation (1.11) and the

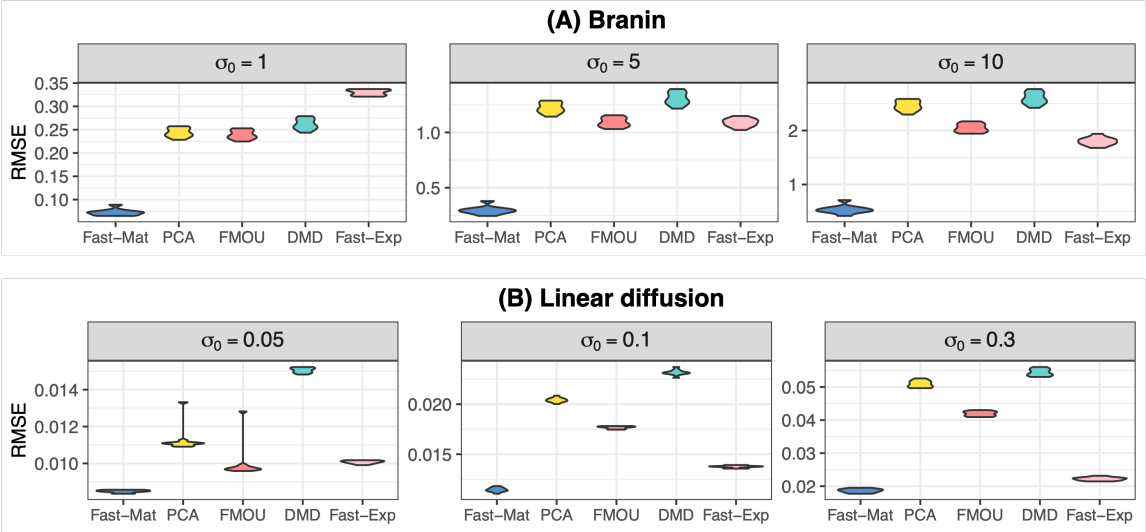


Figure 2.5: Violin plots of RMSE for five methods applied to the Branin function and the linear diffusion equation across various noise levels. Each experiment is repeated 10 times. The Fast-Mat and Fast-Exp represent the fast GPs with Matérn kernels in Equation (1.11) and exponential kernels, respectively.

exponential kernel in Equation (1.8), for $l = 1, 2$. We also include three other alternative methods for comparison. First, we consider principal component analysis (PCA) [120], which is typically used in image denoising by projecting image data into low-dimensional subspaces defined by the dominant eigenvectors of the data covariance matrix. Secondly, we consider the fast algorithm of multivariate Ornstein-Uhlenbeck processes (FMOU) [121], a fast-expectation maximization (E-M) algorithm for parameter estimation in latent factor models. For model $\mathbf{Y} = \mathbf{U}_0\mathbf{Z} + \boldsymbol{\epsilon}$, where $\mathbf{U}_0$ is an orthogonal factor loading matrix and and $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_d]^T$ is a vector of d independent latent processes, modeled as an Ornstein-Uhlenbeck process with unique correlation and variance parameters. Parameter ϵ represents independent Gaussian noise. For each iteration of the E-M algorithm, all parameters in the given model are updated using closed-form expressions. Finally, we consider dynamic mode decomposition (DMD) [122], a popular video processing tool for extracting spatiotemporal structure in complex, time-varying systems. It assumes a linear transition matrix $\mathbf{A}$, estimated as $\hat{\mathbf{A}} = \text{argmin}_{\mathbf{A}} ||\mathbf{Y}_{2:n_2} - \mathbf{A}\mathbf{Y}_{1:n_2-1}||$, where

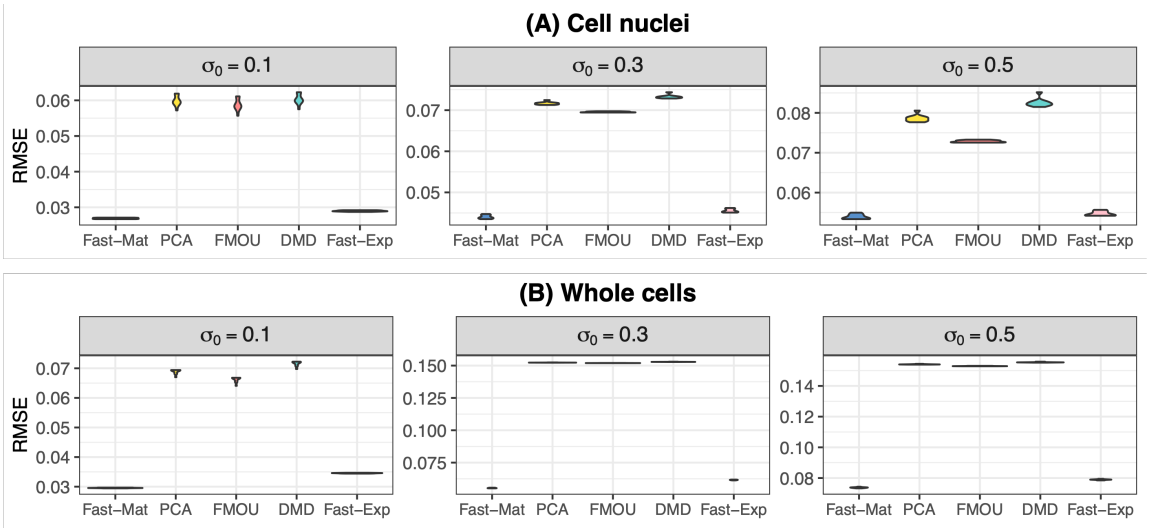


Figure 2.6: Violin plots of RMSE for the five methods applied to images of noisy cell nuclei and whole cells across noise levels $\sigma_0^2 \in [0.1^2, 0.3^2, 0.5^2]$. Each experiment is repeated 10 times. The Fast-Mat and Fast-Exp represent the fast GPs with Matérn kernels in Equation (1.11) and exponential kernels, respectively.

$\mathbf{Y}_{2:n_2}$ and $\mathbf{Y}_{1:n_2-1}$ represent the last and first $n_2 - 1$ columns of $\mathbf{Y}$, respectively. Exact DMD [123] is an efficient algorithm that identifies the leading eigenpairs of $\mathbf{A}$ with computational complexity of $\min(\mathcal{O}(n_1 n_2^2), \mathcal{O}(n_1^2 n_2))$.

We study two examples to numerically compare the proposed fast GP with similar algorithms. In the first example, we simulate and compare two sets of data, where the means of the observations in the first set follow the Branin function [124], and the means of the observations in the second set follow the linear diffusion equation [125]. In the second example, we directly use experimental whole cell and nuclear image data.

Example 1. *In this example, we generate image means using the Branin function and the linear diffusion equation.*

1. *(Branin function) The mean of the observation is defined by the Branin function*

$$f(x_1, x_2) = a(x_2 - bx_1^2 + cx_1 - r)^2 +$$

$$s(1-t)\cos(x_1) + s,$$

where $x_1 \in [-5, 10]$ *and* $x_2 \in [0, 15]$. *The default model parameter values are:* $a = 1$, $b = \frac{5.1}{4\pi^2}$, $c = \frac{5}{\pi}$, $r = 6$, $s = 10$ *and* $t = \frac{1}{8\pi}$. *The input domain is discretized into a uniform grid with* $n_1 \times n_2 = 100 \times 100$ *points. Observations are generated as* $y(x_1, x_2) = f(x_1, x_2) + \epsilon$, *where* ϵ *is an independent Gaussian noise with the noise variance* $\sigma_0^2 \in \{1^2, 5^2, 10^2\}$.

2. *(Linear diffusion) The observation mean is governed by the partial differential equation*

$$\frac{\partial f(x,t)}{\partial t} = D\frac{\partial^2 f(x,t)}{\partial x^2},$$

where $f(x, t)$ *represents the concentration of the diffusing material at location* x *and time* t, *and* D *is the diffusion coefficient. We set* $D = 1$ *and discretize the spatial domain* $[0, 1]$ *into* 200 *equally spaced points. The initial condition is* $f(x, 0) = 0$, *with a boundary condition at one end maintaining a constant external concentration of 1. The signal is simulated over* $t \in [0, 0.2]$ *using* 200 *timesteps via a numerical solver [126]. Observations are generated as* $y(x, t) = f(x, t) + \epsilon$, *where* ϵ *is an independent Gaussian noise. We evaluate the model performance under three noise variance configurations:* $\sigma_0^2 \in \{0.05^2, 0.1^2, 0.3^2\}$.

Example 2 (Cell images)**.** *We consider two noisy cell microscopy images [1]: one for the nuclear channel and one for the whole cell channel. Independent Gaussian noise with variances* $\sigma_0^2 \in \{0.1^2, 0.3^2, 0.5^2\}$ *is added to each image to generate the observations.*

To compare model performance in noise filtering, we quantify prediction error using the root mean squared error (RMSE):

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}(\hat{y}(\mathbf{x}_{i,j}) - \mathbb{E}[y(\mathbf{x}_{i,j})])^2}{n_1 n_2}},$$

where $\hat{y}(\mathbf{x}_{i,j})$ is the estimated mean of $y(\mathbf{x}_{i,j})$.

Figure 2.5 presents the RMSE for mean estimation based on the data-generating processes in Example 1. Our fast GPs with the Matérn kernel (Equation 1.11) consistently achieve

the highest accuracy across all noise levels in both scenarios. This superior performance stems from the separable kernel structure, which captures spatial dependencies along both input dimensions, a critical feature for image modeling. In contrast, PCA implicitly assumes independence in one direction, as its probabilistic formulation treats latent factors as standard Gaussian random variables [127]. While DMD and FMOU model latent factors as Gaussian processes that capture correlation along one input dimension, they estimate the factor loading matrix without explicitly modeling correlation along the other dimension via a kernel function. The fast GPs with Matérn kernel directly model correlations across both inputs through a product kernel, yielding more accurate signal predictions. Additionally, DMD's implicit assumption of a noise-free model [73] leads to performance degradation as noise variance increases. Figure A.6 in the Appendix displays predicted signals from fast GPs with Matérn kernel applied to noisy observations from both the Branin function and linear diffusion equation, demonstrating close alignment with the ground truth.

Figure 2.6 compares the denoising accuracy of the five approaches on cell nuclei and whole cell images from Example 2. The fast GPs with Matérn and exponential kernels substantially outperform PCA, FMOU, and DMD across all noise levels, demonstrating the advantages of estimating correlation parameters along both spatial dimensions. Figure 2.7 illustrates the true signal, noisy observations, and predicted mean from the fast GPs with a Matérn kernel (1.11) for the cell image denoising task. The predicted mean closely approximates the underlying latent signal.

We further evaluate the computational efficiency of the profile likelihood calculation, comparing direct and fast computation methods. Simulations span sample sizes ranging from 10^2 to 80^2 grid points. As demonstrated in Figure 2.8, the fast algorithm (Equation 2.8) dramatically reduces computation time relative to the direct approach, with gains becoming

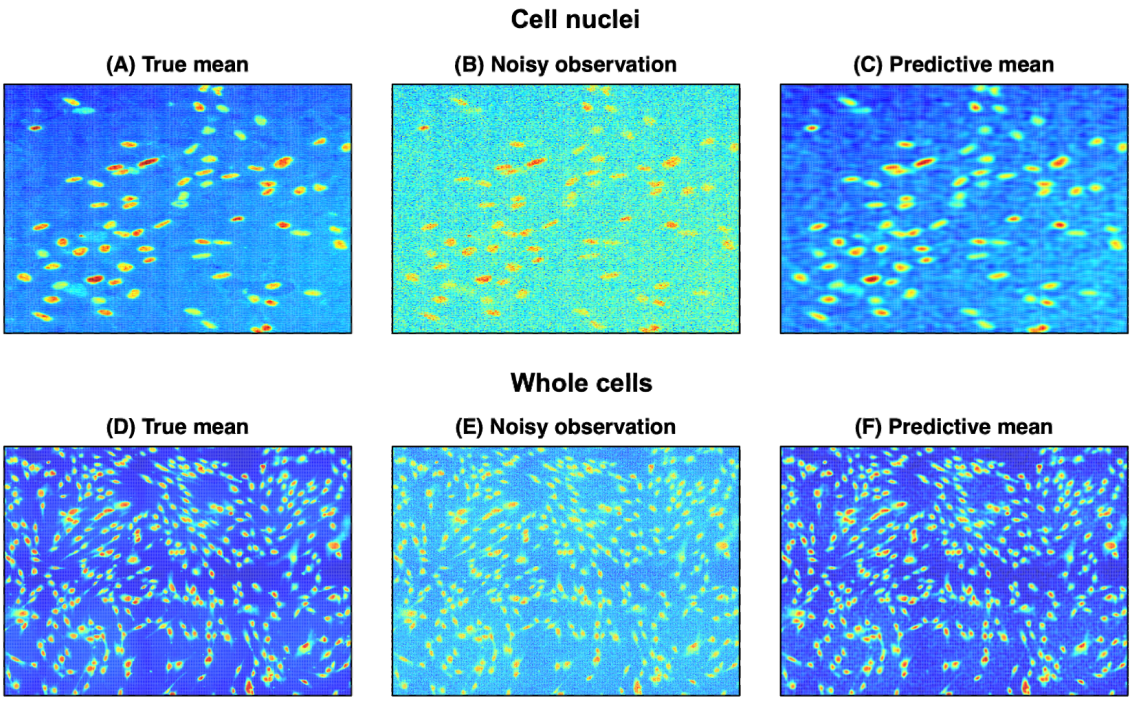


Figure 2.7: (A) True mean of cell nuclei image. (B) Noisy observation of cell nuclei image ($\sigma_0 = 0.1$). (C) Predictive mean of cell nuclei by fast GPs on lattice data with Matérn kernels in Equation (1.11). (D) True mean of the whole cell image. (E) Noisy observation of the whole cell image ($\sigma_0 = 0.1$). (F) Predictive mean of the whole cell by Fast GPs on lattice data with Matérn kernels.

increasingly pronounced at larger sample sizes.

2.5 Numerical studies of cell segmentation

2.5.1 Criteria and methods

We use microscopy images of human dermal fibroblasts (hDFs) in [1] for testing different approaches. More details on the experimental conditions and techniques can be found in Appendix A.8. These images contain both nuclear and cytoplasm channels, and the evaluation of the proposed GP-based segmentation method was performed on both channels.

To evaluate the accuracy of the truth and segmentation results, we compare the area of overlap between the detected object and the true object relative to the overall union area

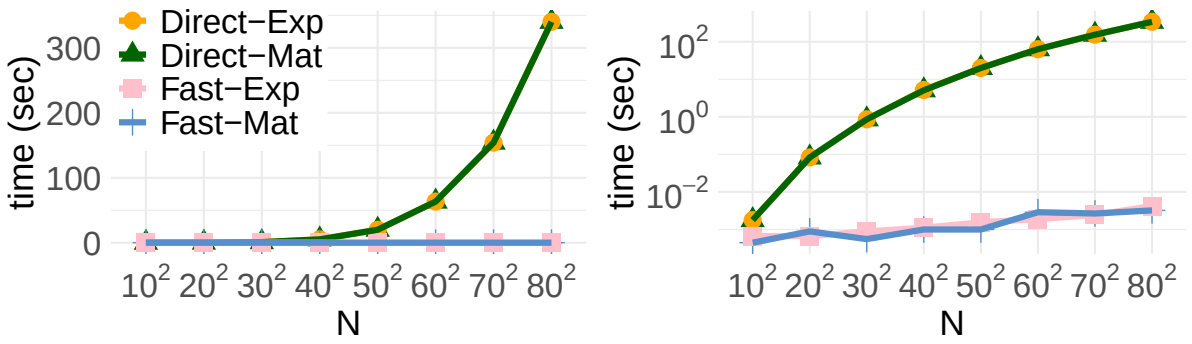


Figure 2.8: Comparison of the computational time in seconds for evaluating the profile likelihood of $\mathbf{Y}_v$ using direct and fast computation under varying sample sizes N. The direct computation with the exponential kernel (Direct-Exp) is plotted as orange dots; the direct computation with the Matérn kernels in Equation (1.11) (Direct-Mat) is plotted with green triangles; the fast computation with the exponential kernel (Fast-Exp) is plotted with pink squares; the fast computation with the Matérn kernels in Equation (1.11) (Fast-Mat) is plotted with blue crosses. The left panel shows the computational time in seconds, which is in the original untransformed scale, and the right panel displays the logarithmic scale.

of the two objects [128]. Let g represent a ground truth mask and p a predicted mask. The Intersection over Union (IoU) metric between g and p is defined as

$$\text{IoU}(g, p) = \frac{|\, g \cap p \,|}{|\, g \cup p \,|}. \tag{2.12}$$

A higher IoU indicates a better match between the predictive mask and true mask of an object, with the perfect match occurring at an IoU of 1. We test several levels of IoU for each method: $\alpha = [0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8]$. Level $\alpha = 0.6$, for instance, means that whenever the IoU defined in Equation (2.12) of a cell is larger than 0.6, this object will be classified as a true positive for the corresponding segmentation method. The range of thresholds allows us to assess image segmentation performance under distinct precision levels for defining true positives.

The average precision metric (AP(α)) at each IoU threshold $\alpha \in [0, 1]$ is often used for evaluating the correction of image segmentation [5]. For $\alpha \in [0, 1]$, a true positive (TP) means that the ground truth mask g is matched to a predicted mask p with $\text{IoU}(g, p) \geq \alpha$. Otherwise, unmatched predicted masks are counted as false positives (FP), and unmatched ground truth

masks are counted as false negatives (FN). The AP(α) is calculated as follows:

$$\mathrm{AP}(\alpha) = \frac{\mathrm{TP}(\alpha)}{\mathrm{TP}(\alpha) + \mathrm{FP}(\alpha) + \mathrm{FN}(\alpha)}. \tag{2.13}$$

We compare ImageJ segmentation and the GP-based unsupervised detection algorithm developed in this work. We followed the conventional processing steps in ImageJ to generate cell masks [80]. All input images are converted to grayscale. To detect the foreground and background pixels, a threshold was set for each image using the default methods. To reduce noises in ImageJ, a despeckle operation, which applies a 3×3 median filter, was then applied to each image. A watershed algorithm was used to ensure better separation of connecting cells. Finally, the cell masks were generated. The cell size is selected to be larger than 30 pixels. Then, the binary image and the labeled mask can be generated in ImageJ. A detailed description of each step in this process for ImageJ can be found in Appendix A.9.

2.5.2 Image segmentation results for cell nuclei

We first use five images of cell nuclei to test different methods. We choose a wide range of image sizes to test methods for images with different sizes. The smallest dimension of these images is 374×250, and the largest image is 962×1128. The experimental details of these images are shown in Table A.1 in the Appendix.

In panel (A) of Figure 2.9, we plot the AP(α) between the GP-based segmentation and ImageJ segmentation methods by solid lines and dashed lines, respectively, for 5 test images of nuclei at distinct threshold α. For any image and threshold level, the solid line is almost always above the dashed line with the same color, as the GP-based method consistently achieves higher AP than ImageJ at any threshold level α. The performance gap becomes more obvious at intermediate IoU thresholds between 0.6 to 0.7, which means that the GP-based method is better at recovering the overall structure of the object than ImageJ. The AP score of both

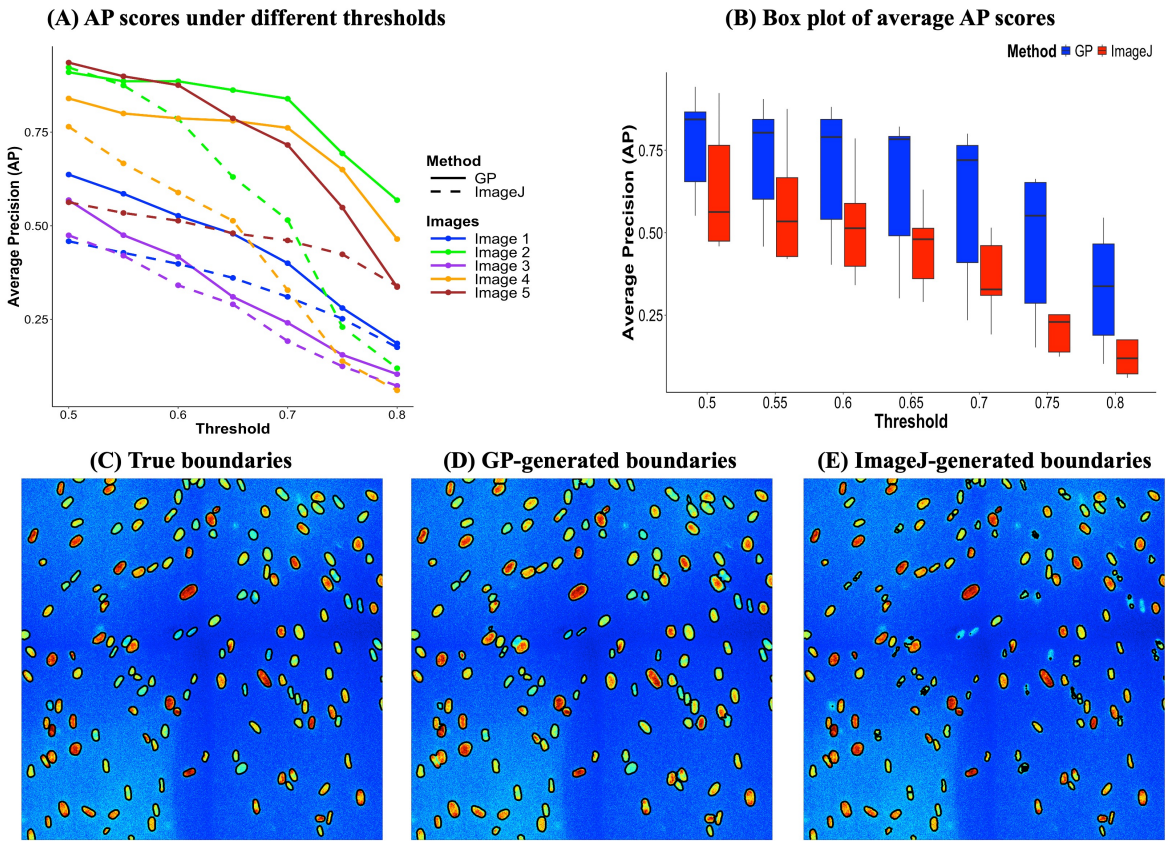


Figure 2.9: Results for segmenting cell nuclei. (A) AP scores for GP-based segmentation and ImageJ across different thresholds. (B) Boxplots of AP scores for each method. These plots are created using the APs for each of the 5 images using each method across different thresholds. (C) True boundaries of the fifth test image with 1024×1024 pixels. (D) Boundaries generated by the GP-based method for the fifth test image. (E) Boundaries generated using ImageJ for the fifth test image.

methods drops at a high IoU threshold, as the annotated object may not be fully accurate, which can affect defining the truth. Panel (B) in Figure 2.9 shows the distribution of these AP scores from GPs and ImageJ segmentation methods averaged at each threshold. Similar to the results in panel (A), the GP-based method achieves higher median AP scores across all thresholds. The difference between the GP-based method and ImageJ is more pronounced at a high threshold level, such as $\alpha = 0.75$ or $\alpha = 0.8$, where the boxes of two colors nearly have no overlap, indicating that GP-based method better capture the fine-scale boundary details than ImageJ.

The true annotated boundaries, those generated by the GP-based method, and those generated by ImageJ of the fifth test image are plotted in Figure 2.9 (C)-(E). The GP-based image segmentation method generally produces smoother, more continuous boundaries capturing the actual cell nuclei, even in cases where cells were closely clustered. In contrast, the ImageJ segmentation method sometimes produces fragmented boundaries and erroneously segmented individual nuclei into multiple smaller regions, especially in regions where the variations of pixel intensity are large. Additionally, ImageJ fails to detect multiple cells entirely. Further image refinement may be achieved using additional plug-ins or filters from ImageJ. However, this process would require extensive human intervention for trial and error, which would substantially increase the image processing time.

2.5.3 Image segmentation results for whole cells

We compare the performance of the GP-based segmentation method and the ImageJ method for five whole-cell images. The smallest image is 602×600, and the largest is 1000×1000.

Panel (A) of Figure 2.10 shows the AP with different thresholds by the two methods for five images of whole cells. The variation of AP by both methods for the five images of whole cells is larger than the five images of the nuclei images. This is expected as segmenting whole cells is much more challenging due to the distinct shapes of the whole-cell images and more objects connecting to each other. Nonetheless, the solid line representing the AP for the GP-based method is again almost always above the dashed line representing the AP of the ImageJ segmentation results with the same color for all thresholds, meaning that the GP method consistently achieves higher AP scores than ImageJ. Both methods show much lower precision at a smaller IoU threshold, as correctly identifying irregular boundary shapes in connecting whole cells is extremely hard to capture. Such boundaries also pose challenges for humans to

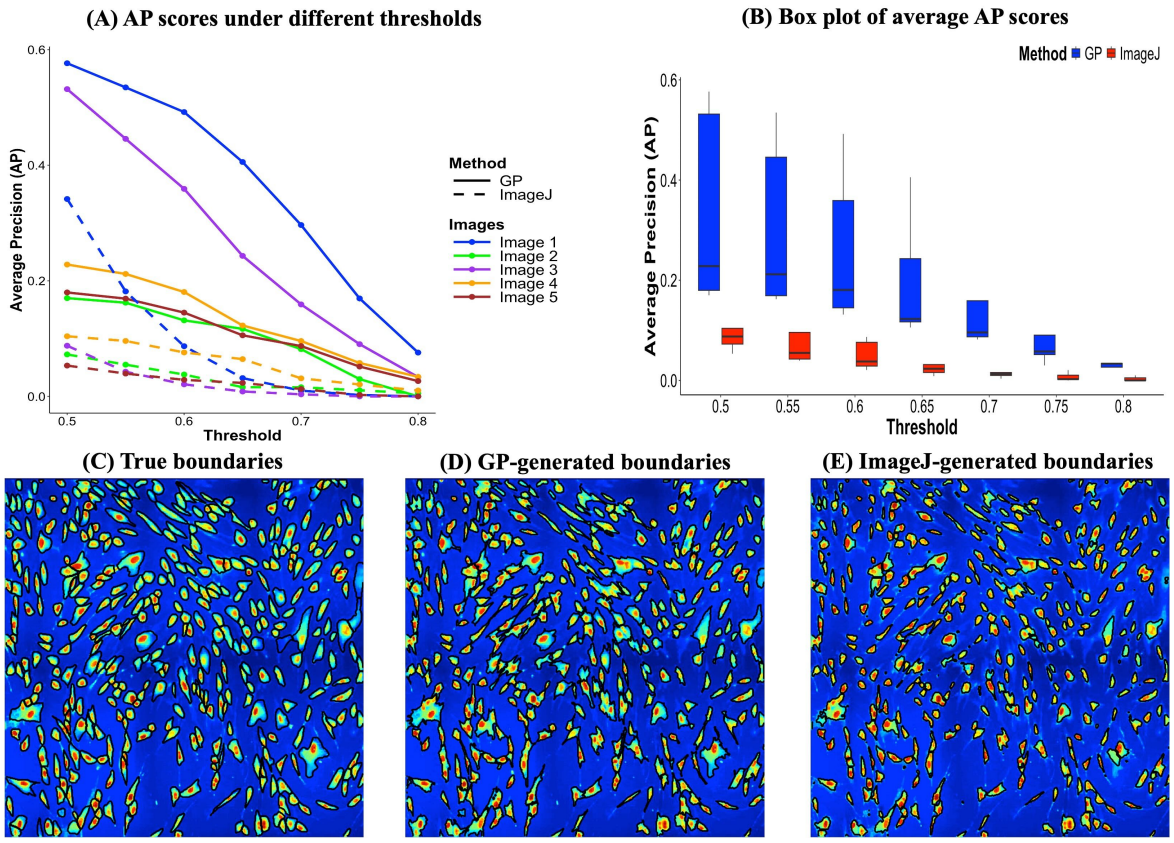


Figure 2.10: Results for segmenting whole cells. (A) AP scores for GP-based segmentation and ImageJ across different thresholds. (B) Boxplots of AP scores for each method. (C) True boundaries of the first test image with 602×600 pixels. (D) Boundaries generated by the GP-based method of the first test image. (E) Same for panel (D) but generated using ImageJ.

annotate, and for supervised methods to segment, even when annotated data are available.

Panel (B) of Figure 2.10 shows the distribution of the AP scores between the two methods. Compared to the segmentation results of the cell nuclei, the improvement by the GP-based segmentation methods is more pronounced for images of whole cells. The variation of AP by the GP-based segmentation is larger for whole-cell images than the ones for the images of nuclei, due to the difficulty in identifying the whole-cell shape either by statistical learning methods or by humans.

The ground truth boundaries, the GP-based method generated boundaries, and the ImageJ generated boundaries of the first text image are given in panels (C)-(E) in Figure 2.10.

The GP-based method generally produces boundaries that better fit the actual cell shapes, preserving the irregular shapes of the cell cytoplasm. Although the GP-based method generally better preserves the true cell shapes, cell objects were also often grouped together. In contrast, ImageJ segmentation often produces fragmented boundaries and splits individual cell shapes into multiple smaller regions.

By leveraging fast Gaussian processes, automated thresholding and watershed operations, we show the process is not only more automated, but the results are more precise than other popular unsupervised segmentation methods, such as ImageJ. A key strength of the proposed segmentation algorithm is that it does not rely on parameter tuning or labeled training datasets. The results of our unsupervised method demonstrate versatility across different cell channels, making it an appealing option for more general segmentation tasks beyond cell segmentation.

Chapter 3

FastSegmentation: an R package for unsupervised cell segmentation by fast Gaussian processes

In this chapter, we present FastSegmentation, an R package developed to implement the methodology described in Chapter 2. This package automatically calculates GP parameters for image smoothing and noise reduction, applies the data-driven threshold for foreground-background segmentation, and implements the watershed operation for separating connected cell objects, all within one function. This tool enables the efficient unsupervised segmentation of one or many cell images in R without manual parameter tuning or image processing.

3.1 Package overview

The R package FastSegmentation [129] was developed to facilitate implementation of the unsupervised cell segmentation algorithm described in Chapter 2. This methodology is

a multi-step, yet fully automated, method for calculating cell object boundary information on grayscale cell microscopy images using fast GPs for denoising, a data-driven threshold for foreground object detection, and watershed for segmentation.

Figure 3.1 provides a schematic overview of the main pipeline function of this package and mirrors the segmentation workflow visualized in Figure 2.1. The entire segmentation pipeline can be run using the `generate_GP_Masks()` function, with the only required input being a file path to a cell microscopy image. The runtimes for most segmentation tasks using this function are typically less than 15 seconds per image. The primary output is a 2D array containing labeled cell masks (see blue box in Figure 3.1 on the right). This function contains all steps and functions related to the segmentation pipeline; however, intermediate steps of each segmentation phase can also be run individually with certain functions, as detailed in the middle of Figure 3.1. Image partitioning information is collected using `get_proportions()`, and image smoothing is run using both `separable_GP_param_est()` for range and nugget parameter estimation and `predict_separable_GP` for predictive mean calculation. Image thresholding is conducted using `criterion_1()` for optimal threshold detection and `threshold_image()` for retrieving the binary output. After image restitching, final segmentation is run using the `watershed()` function from the EBImage [118] package.

3.2 Main functions of FastSegmentation

3.2.1 The `generate_GP_Masks` function

The `generate_GP_Masks` function serves as the primary function for users and runs the complete segmentation pipeline outlined in Chapter 2. This function accepts a file path to a cell microscopy image with size $N = n_1 \times n_2$ with format compatible with the `image_read()`

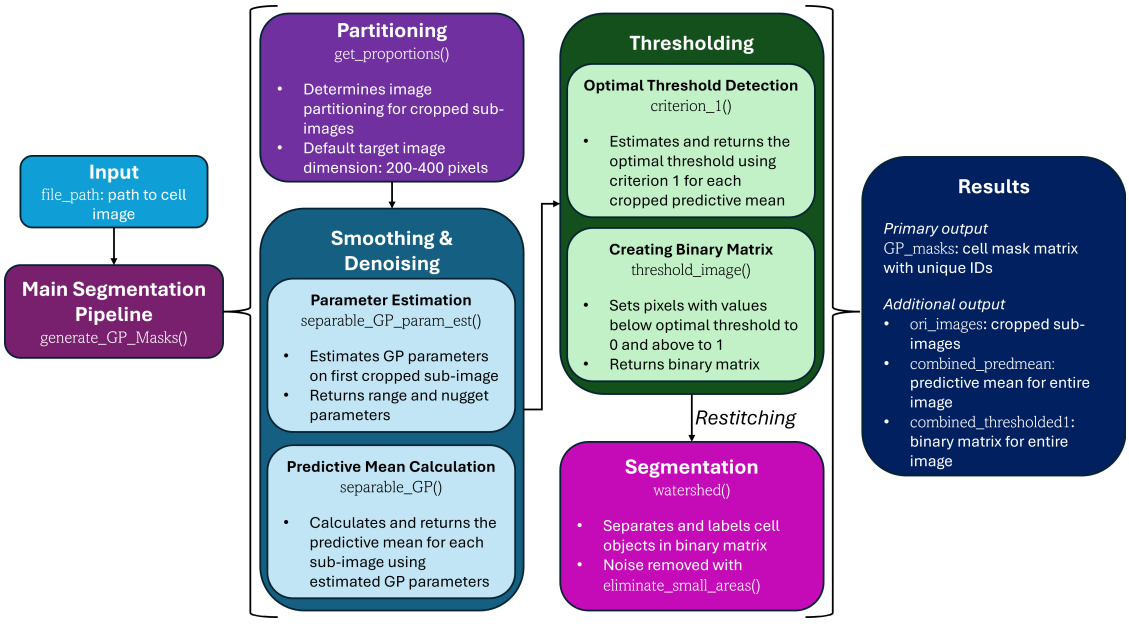


Figure 3.1: Schematic overview of the FastSegmentation package.

function from the magick package [130], such as `TIF`, `PNG`, or `JPEG`, and returns comprehensive segmentation results, including cell masks, predictive means, and diagnostic information. The file path is the only required parameter. However, a number of parameters have default values that can be altered: 1) `delta` $= 0.01$, which is the step size for percentage thresholds $\{\alpha_m\}_{m=1}^{M}$ to be tested when determining the optimal threshold by criterion 1, 2) `nugget` = `TRUE`, which indicates if the nugget parameter should be estimated when smoothing the absolute differences in pixel counts between thresholds when determining the optimal threshold by criterion 1, 3) `middle_threshold` $= 0.15$, which is the size threshold for filtering object masks not touching the image boundary, 4) `boundary_threshold` $= 0.05$, which is the size threshold for filtering object masks touching the image boundary, 5) `compress_output` = `FALSE`, which indicates that the true matrices for the cell mask and binary matrix should be output, instead of data frames listing each 2D coordinate and corresponding value, and 6) `return_gradient = FALSE`, which

indicates that the gradients of the predictive mean should not be returned.

This function calls the series of functions in the order described by the schematic diagram in Figure 3.1. The primary output of this function is the `GP_masks` matrix, a matrix of size N that directly gives the estimated cell masks, with a unique numerical ID for each separate cell object. Other major outputs include `ori_images`, a list of matrices corresponding to the sub-images $\{\mathbf{Y}_k\}_{k=1}^{\tilde{K}}$, `combined_predmean`, which provides the predictive mean matrix $\mathbf{F}^*$ for the entire original image, `combined_thresholded1`, the binary matrix $\mathbf{B}_k^*$ for the total image after determining the optimal threshold using criterion 1. Other outputs can be useful for model diagnostics and interpretation, including `processed_images`, a list of predictive mean matrices $\{\mathbf{F}_k^*\}_{k=1}^{\tilde{K}}$ for each sub-image, `crit_1_opt_thresholds`, a list of the optimal thresholds α_k^* detected by criterion 1 for each cropped predictive mean $\mathbf{F}_k^*$, `connected_parts_count`, a list of the number of unique objects for each image section after thresholding for each cropped binary matrix $\mathbf{B}_k^*$, and `outliers`, a list of indices the for sub-images with unusually high object counts after thresholding.

3.2.2 The `separable_GP_param_est` and `predict_separable_GP` functions

The `separable_GP_param_est` and `predict_separable_GP` functions work in tandem to compute the predictive mean of each partitioned sub-image for image smoothing and denoising. For the first sub-image $\mathbf{Y}_k$, the function `separable_GP_param_est` estimates the range and nugget parameters by maximizing the likelihood in Equation (2.8), outputting the list of parameters, $(\gamma_1, \gamma_2, \eta)$. The range and nugget parameters are then kept the same across all sub-images $\{\mathbf{Y_k}\}_{k=1}^{\tilde{K}}$ when calculating the predictive means $\{\mathbf{F}_k^*\}_{k=1}^{\tilde{K}}$ using Lemma 1. This function outputs the predictive mean $\mathbf{F}_k^*$ for each cropped image k, as well as gradient information for $\mathbf{F}_k^*$: $\mathbf{G}_x$, $\mathbf{G}_y$, and $\mid \nabla \mathbf{F}^* \mid$.

Matrices G_x and G_y provide the horizontal and vertical gradients of the predictive mean, respectively, and are calculated as [84]

$$\mathbf{G}_x(\mathbf{x}_{i,j}) = \frac{\mathbf{F}^*(\mathbf{x}_{i,j+1}) - \mathbf{F}^*(\mathbf{x}_{i,j-1})}{2} \tag{3.1}$$

and

$$\mathbf{G}_y(\mathbf{x}_{i,j}) = \frac{\mathbf{F}^*(\mathbf{x}_{i+1,j}) - \mathbf{F}^*(\mathbf{x}_{i-1,j})}{2}. \tag{3.2}$$

The values of the gradients represent the rate of change of pixel intensity in the horizontal and vertical directions, respectively. A positive gradient value indicates that pixel intensity is increasing (e.g., moving from a low intensity background pixel to a higher intensity cell pixel). The gradient magnitude, represented by $\mid \nabla \mathbf{F}^* \mid$, is a function of the two directional gradients,

$$\mid \nabla \mathbf{F}^* \mid = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}. \tag{3.3}$$

Gradient magnitude values represent the intensity of change in any direction. The magnitude values are high in areas where pixel intensity is rapidly changing, such as an cell object edge, and lower in flat or uniform areas, such as in the middle of the background or cell object.

Gradients are automatically returned when using the `predict_separable_GP` function individually. Gradients are by default not returned in the overall segmentation pipeline but can be retrieved by setting `return_gradient` to `TRUE`.

```
generate_GP_Masks(file_path, return_gradient = TRUE)
```

3.2.3 The `criterion_1` function

Image thresholding is driven by the `criterion_1` function, which uses Equation (2.11) to determine the optimal threshold α_k^* for each input predictive mean $\mathbf{F}_k^*$. By default the function settings, the tested thresholds are $\{\alpha_m\}_{m=1}^M = \{0, 0.1, 0.2, \ldots, 0.99, 1\}$, as the `delta`

parameter that determines the step size is set to 0.01. More stringent threshold testing can be performed by decreasing `delta`; for example, if `delta` is set to 0.001, then $\{\alpha_m\}_{m=1}^{M} = \{0, 0.001, \ldots, 0.999, 1\}$. Although `delta` $= 0.01$ was found to be robust for all cell images tested, this implementation is recommended in the event that the default `delta` does not lead to an accurate optimal threshold detection due to noise or smaller variations in pixel intensity between foreground and background pixels. The code below runs the entire segmentation pipeline for an image with this specified `delta`.

```
generate_GP_Masks(file_path, delta = 0.001)
```

Alternatively, laxer testing can be performed by increasing `delta`, leading to lower computational time. For example, if `delta` is set to 0.1, then $\{\alpha_m\}_{m=1}^{M} = \{0, 0.1, 0.2, \ldots, 0.9, 1\}$.

```
generate_GP_Masks(file_path, delta = 0.1)
```

This implementation is recommended when there are greater variations in pixel intensity between foreground and background objects, as subsequent thresholds for smaller `delta` settings will likely yield similar binary matrices. Note that 0 is excluded as a possible threshold, as it is assumed that each cropped predictive mean will contain at least one cell.

Furthermore, the default method smoothing $\Delta c_k(\alpha_m)$ via the `rgasp()` function from the RobustGasp package [53] includes nugget estimation, with the `nugget` parameter being set to `TRUE`. This parameter accounts for noise in the image. Although the predictive mean is a smoothed version of the original image, it may still include some noise; therefore, it is recommended to maintain `nugget` = `TRUE`. However, if a cell image is known to contain no noise, such as in a completely simulated or human-constructed image, `nugget` can be set to `FALSE` for better computational efficiency, as demonstrated in the code below.

```
generate_GP_Masks(file_path, nugget = FALSE)
```

3.2.4 Supporting functions

This package also contains a number of functions to support the segmentation workflow implementation. Each one is briefly described below.

The `get_proportion` function

This function helps determine the optimal subdivision of an image based on the desired dimension range of the cropped sub-images. The target width and length range for a sub-image is set to a minimum of 200 pixels and a maximum of 400 pixels by default. For a scalar input, which is in practice either the original image width w or height h, this function determines if splitting this range into fourths, thirds, or halves, or if not splitting at all, achieves each section being within the optimal pixel count:

$$200 \leq v \times r \leq 400, \tag{3.4}$$

where v is either h or w, and $r \in \{\frac{1}{4}, \frac{1}{3}, \frac{1}{2}, 1\}$. If this boolean is not true for any r, then this function will declare that the range should be split into fourths. The output of this function is the fraction of the scalar input range that defines where the original image will be cropped in that dimension. For example, for image width $w = 200$ and height $h = 400$, this function will return $r_w = 1$ and $r_h = \frac{1}{2}$. Therefore, the image height will be split in half, resulting in two sub-images in that dimension, whereas the width will not be split, resulting in 2 cropped sub-images overall.

The `threshold_image` function

This function applies a threshold to an image, both of which must be specified by the user, to create a binary matrix. Any pixel values greater than the threshold are set to 1, and any pixel values less than or equal to the threshold are set to 0. In practice, the threshold is

the output of the `criterion_1` function, and the input image is the cropped predictive mean calculated by the `separable_GP` function. The output binary matrix separates the foreground, or cell, pixels from the background. This function has two possible outputs. If the argument `count` is `TRUE`, as it is by default, the function returns the sum of foreground pixels in the binary matrix. This output is used in the `criterion_1` function to determine the optimal threshold. If `count` is instead set to `FALSE`, the binary matrix is instead returned. In the segmentation pipeline, this output is the input for the watershed segmentation process to retrieve the cell masks.

The `eliminate_small_areas` function

This function eliminates any foreground objects with smaller-than-expected object counts based on the average cell object size. The main input of this function is the matrix of cell masks determined using the watershed algorithm. Although using the predictive mean as a smoothed image helps reduce noise, it may not eliminate all noise for every image, so noise may still be present as small objects in the cell mask matrix. By default settings, any cell object with a smaller pixel count than 15% of the average object pixel count in the image is excluded. For objects touching the image boundary, this threshold is reduced to 5% of the average object pixel count, as these objects may have been cut off during imaging. This function returns a filtered cell mask matrix, which is the final output of the segmentation workflow.

The `matrix_which` function

This function compresses an $N = n_1 \times n_2$ matrix into a data frame listing the N 2D coordinates which corresponding value. This function allows for more efficient storage of binary matrix and cell mask coordinates, especially when segmenting multiple images within the same

program. To save the binary and cell mask coordinates using this method, the `compress_output` argument in `generate_GP_Masks()` to `TRUE`,

```
generate_GP_Masks(file_path, compress_output = TRUE).
```

3.3 Methods and examples

3.3.1 Nuclear channel image segmentation

This package achieves cell image segmentation with one function, with the only user-defined input required being the file path to a cell microscopy image, typically `TIF`, `JPEG`, or `PNG`. This section provides an example of this process with default parameters and includes visualization for the major function outputs using the `image2D()` function from the `plot3D` package [131]. Custom settings, like size threshold for cell object detection, can be specified within the main segmentation workflow function, `generate_GP_Masks`. The example input image is a grayscale `JPEG` image of the cell nuclear channel with dimension 799×448. The `R` code below provides the base-use case of this function, along with visualization recommendations for all major outputs of the function.

```
img_path <- system.file("extdata", "example_cells.jpg",
                package = "FastSegmentation")
gp_masks_result <- generate_GP_Masks(img_path)

# Libraries for visualization
library(plot3D)
library(magick)

# Original image
```

```
img <- image_read(img_path)
ori_img_matrix <- as.numeric(img[[1]])[,,1]
image2D(ori_img_matrix, main = "Original Image")

# Predictive mean
image2D(gp_masks_result$combined_predmean, main = "Predictive Mean")

# Binary image after thresholding
image2D(gp_masks_result$combined_thresholded1,
    main = "Thresholded Image")

# Final cell masks
image2D(gp_masks_result$GP_masks, main = "Cell Masks")
```

The default visualization output is given in Figure 3.2. Additional adjustments to the `plot3D()` functions, such as specifying a color palette or value range, can be implemented for preferred visualization settings, but the figures are all displayed with largely the default argument settings in order to display the base-use case.

3.3.2 Whole cell channel image segmentation

Segmentation for the whole cell channel of an image can be executed using the same method as the previous example. The input for this example is a whole cell `TIF` image with dimension 500×600. Note that this `TIF` file only contains one sub-file.

```
img_path <- system.file("extdata", "whole_cell_example.tif",
                package = "FastSegmentation")
gp_masks_result <- generate_GP_Masks(img_path)
```

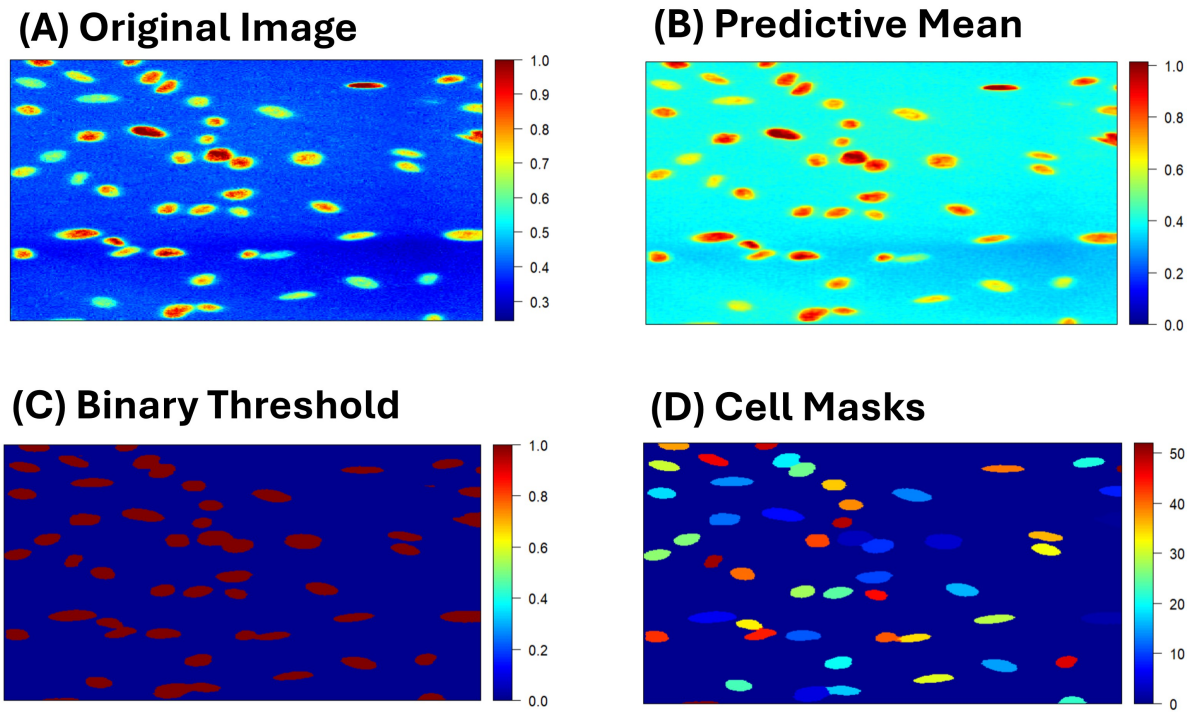


Figure 3.2: (A) Input nuclear cell image, with color legend presenting how pixel intensity corresponds with visualized color. (B) Predictive mean of input image. (C) Binary matrix after thresholding by criterion 1, with 0 (blue) being background and 1 (red) being foreground. (D) Generated cell masks after watershed and noise filtering, with the background corresponding with 0 (blue), and each cell mask is represented by a different color corresponding with unique cell ID.

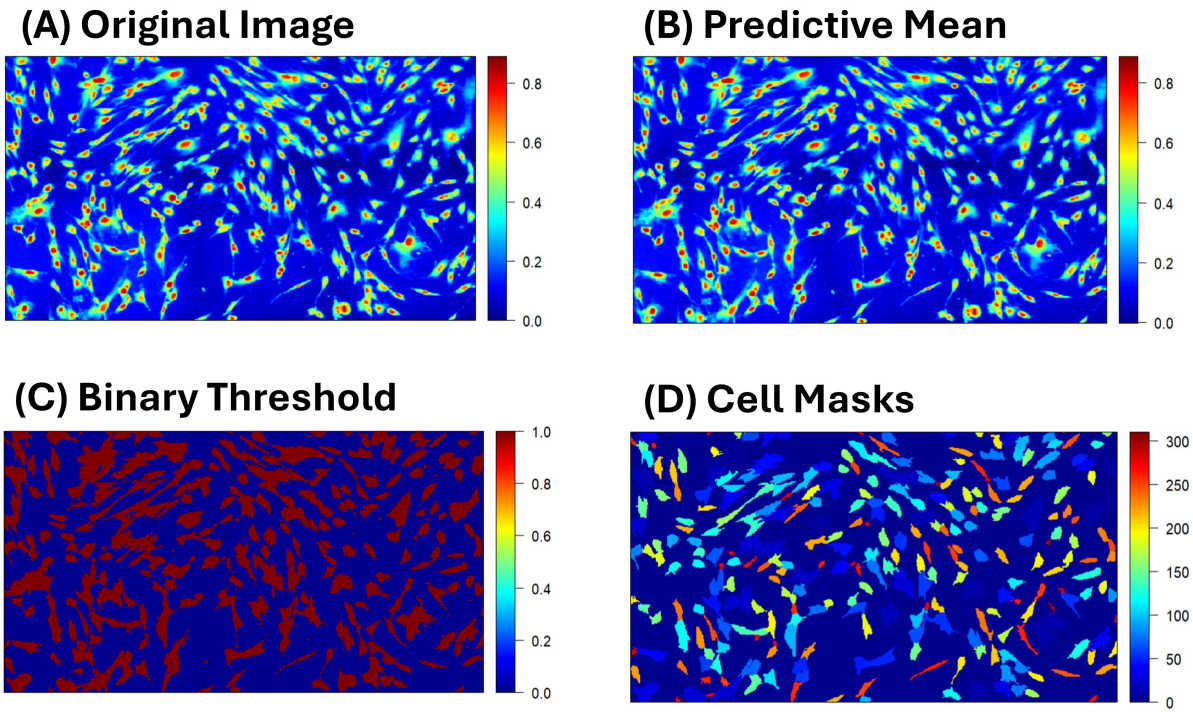


Figure 3.3: (A) Input whole cell image, with color legend giving how pixel intensity corresponds with visualized color. (B) Predictive mean of input image. (C) Binary matrix after thresholding by criterion 1, with 0 (blue) being background and 1 (red) being foreground. (D) Generated cell masks after watershed and noise filtering, with the background corresponding with 0 (blue), and each cell mask is represented by a different color corresponding with unique cell ID.

Visualizations for the main outputs after running for the segmentation pipeline are presented in Figure 3.3. Limitations for whole cell segmentation are discussed in Section 2.5, as even ImageJ struggles to fully capture cell boundary information for the whole cell channel. Although the cell masks, shown in Figure 3.3 (D), are less precise than they are for the nuclear channel, this package is still able to retrieve acceptable results for foreground-background detection, as shown in Figure 3.3 (C).

3.3.3 Visualization with compressed output

Although compressing the binary and cell object mask outputs takes up less storage, visualization requires additional data manipulation. Figures 3.2 (C) and (D) can be recreated from the compressed output using the following method.

```
img_path <- system.file("extdata", "example_cells.jpg",
    package = "FastSegmentation")

# Generate compressed outputs
gp_masks_result <- generate_GP_Masks(img_path, compress_output = TRUE)

# Reconstruct binary matrix
binary_mask <- matrix(NA,
    nrow = nrow(gp_masks_result$combined_predmean),
    ncol = ncol(gp_masks_result$combined_predmean))
binary_indx <- cbind(gp_masks_result$combined_thresholded1$row,
    gp_masks_result$combined_thresholded1$col)

# Reconstruct labeled GP masks
labeled_mask <- matrix(NA,
```

```
    nrow = nrow(gp_masks_result$combined_predmean),
    ncol = ncol(gp_masks_result$combined_predmean))
labeled_indx <- cbind(gp_masks_result$GP_masks$row,
    gp_masks_result$GP_masks$col)
labeled_mask[labeled_indx] <- gp_masks_result$GP_masks$value

# Visualize
image2D(binary_mask, main = "Thresholded Image")
image2D(labeled_mask, main = "Cell Masks")
```

3.3.4 Predictive mean by separable GP with sub-image parameter estimation

The FastSegmentation package does not have a direct function for calculating the predictive mean for each cropped sub-image and re-stitching to retrieve the whole predictive mean without running the full segmentation pipeline. However, by setting `return_gradient` to `TRUE`, the gradients of the predictive mean, as described in equations (3.2), (3.1), and (3.3) can be returned and visualized, with the output of the following example being presented in Figure 3.4.

```
# Load image
img_path <- system.file("extdata", "example_cells.jpg",
                package = "FastSegmentation")

# Generate compressed outputs
gp_masks_result <- generate_GP_Masks(img_path, return_gradient = TRUE)
```

```
# Visualize
library(plot3D)
image2D(gp_masks_result$combined_predmean, main = "Predictive Mean",
        axes=FALSE, xlab="", ylab="")
image2D(gp_masks_result$gradients$horizontal, main = "Horizontal
                Gradient", axes=FALSE, xlab="", ylab="")
image2D(gp_masks_result$gradients$vertical, main = "Vertical Gradient",
        axes=FALSE, xlab="", ylab="")
image2D(gp_masks_result$gradients$magnitude, main = "Gradient
                Magnitude", axes=FALSE, xlab="", ylab="")
```

3.3.5 Predictive mean by separable GP with whole image parameter estimation

Using the `separable_GP_param_est` and `predict_separable_GP` functions outside of the segmentation pipeline allows for the GP parameter estimation and predictive mean calculation of an image without partitioning. The code implementation is provided below, and the visualization results are similar to those presented in Figure 3.4.

```
img_path <- system.file("extdata", "example_cells.jpg",
            package = "FastSegmentation")
img_matrix <- as.numeric(img[[1]])[,,1]

# Estimate GP parameters
img_gp_parameters <- separable_GP_param_est(img_matrix)

# Calculate predictive mean
```

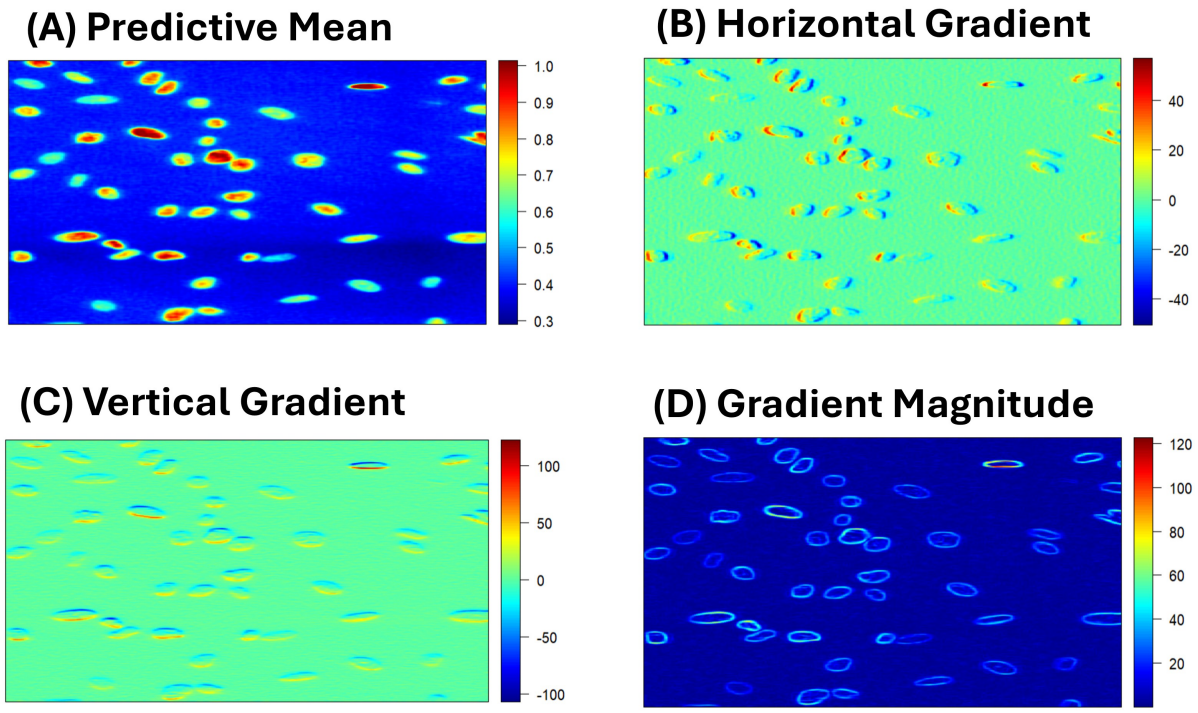


Figure 3.4: (A) Predictive mean. (B) Horizontal gradient of predictive mean, with pixel color corresponding to gradient intensity. (C) Vertical gradient of predictive mean, with pixel color corresponding to gradient intensity. (D) Gradient magnitude, with pixel color corresponding to gradient intensity.

```
predmean_results <- predict_separable_GP(img_matrix,
            img_gp_parameters$param)

# Visualize
library(plot3D)

image2D(predmean_results$predmean_mat, main = "Predictive Mean",
    axes=FALSE, xlab="", ylab="")
image2D(predmean_results$grad1, main = "Horizontal Gradient",
    axes=FALSE, xlab="", ylab="")
image2D(predmean_results$grad2, main = "Vertical Gradient",
    axes=FALSE, xlab="", ylab="")
image2D(predmean_results$grad_magnitude, main = "Gradient Magnitude",
    axes=FALSE, xlab="", ylab="")
```

3.3.6 Segmentation of nonuniform images

This algorithm is recommended for larger images with somewhat uniform cell density across space, as these images will have the most consistent segmentation results. Detecting edge cases when thresholding criterion 1 rely on information collected in other sub-regions of the image, as discussed in Appendix A.4. However, this algorithm has been empirically shown to still be robust to non-uniform distributions of cells, even in lower dimensions, as seen in Figure 3.5.

3.3.7 Segmentation of jammed cells

Cell jamming occurs in extremely high cell densities, with most cells touching or overlapping [132], such as in Figure 3.6 (A). Collecting cell boundary information from jammed

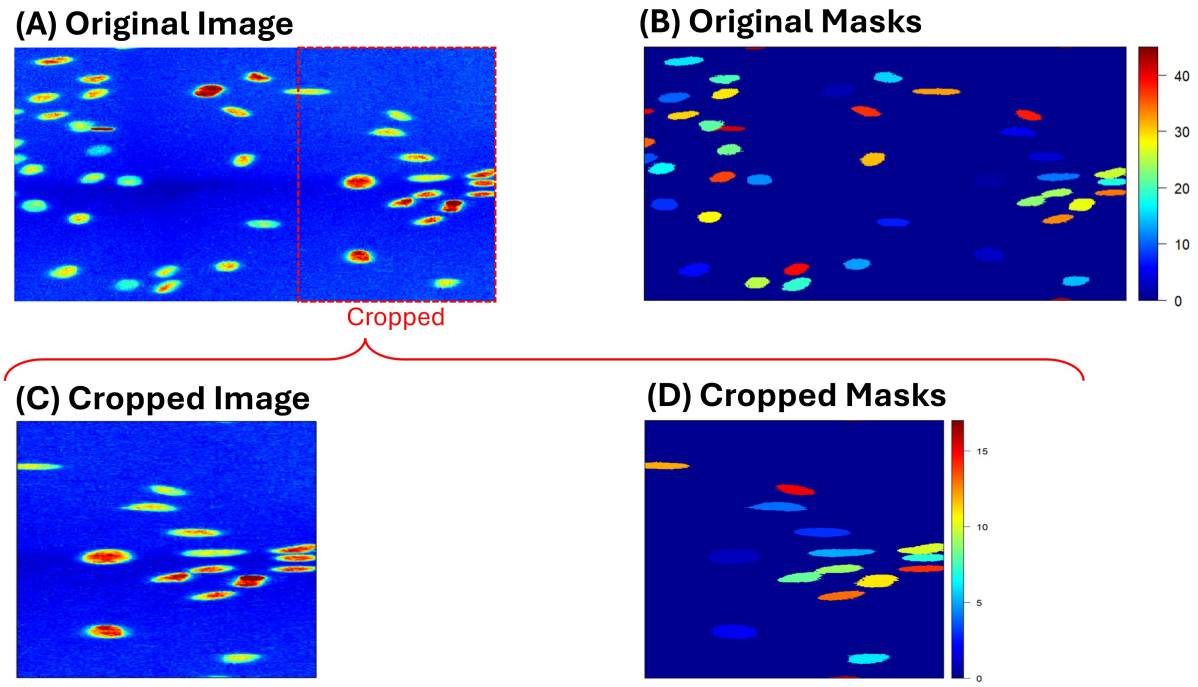


Figure 3.5: Segmentation results for cell images with nonuniform cell densities across space. (A) Original 799×448 image with nonuniform cell densities. This image contains a cluster of cells, increasing local cell density, and other regions of lower cell density. (B) Segmentation results for (A). Although some regions of the image are sparsely concentrated, there is still enough information for threshold calibration by criterion 1 to be successful. (C) Cropped 799×181 image from (B). (D) Segmentation results for (C). Although each partitioned section of this image has different numbers of foreground pixels, the resulting cell masks still reflect accurate cell boundaries.

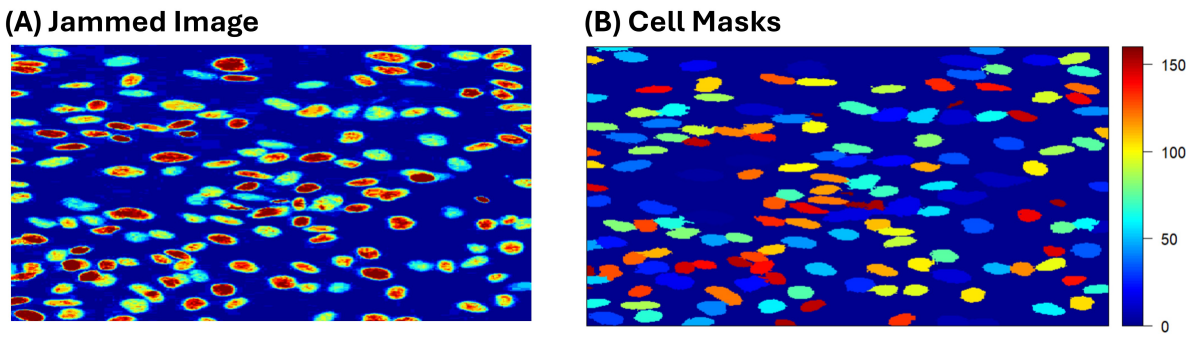


Figure 3.6: (A) Pixel intensity matrix for cell microscopy image with jamming. (B) Segmentation results for jammed image (A).

images is a more complicated task, as boundaries will regularly run together and overlap. Even in these high density conditions, the package algorithm is still able to successfully separate touching and overlapping cell objects, as demonstrated in Figure 3.6 (B).

Chapter 4

Diffusion-based generative models: theory, validation, and applications to cell microscopy

In this chapter, we present an introduction to two diffusion-based generative models: denoising diffusion probabilistic models (DDPMs) and score-based generative models. We discuss the mathematical overview and training methods of both models and provide data generation results when learning from samples from a known Gaussian mixture model. We then apply the models to real-world cell microscopy images to assess data generation and denoising performance. Overall, both models have strong results for generating data, creating synthetic images that closely resemble the input data, and are accurately able to denoise corrupted images from a test image set.

4.1 Introduction

Collecting high-quality cell images for machine learning applications has significant practical challenges in both cost and availability. Cell microscopy imaging requires expensive equipment, specialized facilities, and trained personnel [133]. Additionally, cell culture experiments require constant maintenance and supervision to ensure controlled experimental conditions [134], and cell images still may be corrupted during screening from inadequate equipment calibration and bleaching, potentially causing large amounts of image noise [135, 136, 137]. Generating comprehensive datasets that capture image data for cells across multiple experimental conditions can take weeks to months of continuous imaging, limiting scalability and replicability. Publicly available repositories [138, 139] often contain images from specific cell lines or experimental contexts, but this data could be limited or fail to include the target conditions. These constraints in data availability create a compelling argument for the use of generative models, which can augment limited datasets and denoise corrupted images, potentially reducing the experimental burden of image collection.

Generative modeling aims to learn the underlying probability distribution of observed data to generate new samples that resemble the training data [140]. While traditional approaches like generative adversarial networks (GANs) [141] and variational autoencoders (VAEs) [142] have shown considerable success, diffusion-based generative models are a powerful alternative for image generation and denoising. This chapter will focus on two closely related frameworks for diffusion-based generative modeling: denoising diffusion probabilistic models (DDPMs) [143] and score-based generative models using stochastic differential equations (SDEs), referred to as score-based SDE models in this chapter [144]. Although these frameworks are fundamentally connected, they offer unique perspectives (discrete-time for DDPM versus continuous-time formulations for the SDE approach) with different advantages and insights.

Knowledge of the theoretical foundation is essential for understanding how each could be used for generating and denoising cell image data.

4.1.1 Variational autoencoder

Autoencoders [145] are the mathematical and conceptual precursor for diffusion models. They contain an encoder network for to compress input to a lower-dimension and a decoder to reconstruct the original data, in the same way a U-net model uses upsampling and downsampling. VAEs [142] incorporate relationships between the input and latent space by encoding inputs as probability distributions and provide the variational framework on which DDPMs are built [140]. VAEs assume that each observed data point $\mathbf{y} \in \mathbb{R}^k$ is generated from a latent variable $\mathbf{z}$, where $\mathbf{z}$ is typically sampled from a standard Gaussian prior, $\mathbf{z} \sim \mathcal{MN}(0, \mathbf{I})$ [40]. The VAE latent-variable generative model follows the marginal distribution:

$$p_\phi(\mathbf{y}) = \int p_\phi(\mathbf{y} \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}, \tag{4.1}$$

where $p_\phi(\mathbf{y} \mid \mathbf{z})$ is the decoder network, and ϕ are the learnable decoder parameters. Since the integral over $\mathbf{z}$ does not typically have a closed-form expression, ϕ cannot be estimated through standard methods like MLE, so VAEs instead use the encoder as an approximate posterior. The posterior distribution by Bayes is

$$p_\phi(\mathbf{z}|\mathbf{y}) = \frac{p_\phi(\mathbf{y}|\mathbf{z}) p(\mathbf{z})}{p_\phi(\mathbf{y})}, \tag{4.2}$$

but since estimating the marginal likelihood $p_\phi(\mathbf{y})$ would require integration over all possible latent variables, the encoder $q_\theta(\mathbf{z} \mid \mathbf{y})$ is introduced to approximate the posterior, $q_\theta(\mathbf{z} \mid \mathbf{y}) \approx p_\phi(\mathbf{z} \mid \mathbf{y})$, where θ are parameters learned by a neural network.

Since the marginal likelihood is intractable, maximizing the evidence lower bound

(ELBO) is instead used as a surrogate training objective, as

$$\log(p_\phi(\mathbf{y})) \geq \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{y}). \tag{4.3}$$

Using Jensen's inequality, the ELBO is as follows:

$$\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{y}) = \mathbb{E}_{\mathbf{z}\sim q_\theta(\mathbf{z}|\mathbf{y})}\left[\log(p_\phi(\mathbf{y} \mid \mathbf{z})\right] - \mathcal{D}_{\text{KL}}\left(q_\theta(\mathbf{z} \mid \mathbf{y})||p(\mathbf{z})\right), \tag{4.4}$$

where $\mathbb{E}_{\mathbf{z}\sim q_\theta(\mathbf{z}|\mathbf{y})}\left[\log(p_\phi(\mathbf{y} \mid \mathbf{z})\right]$ is the reconstruction term, and $\mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z} \mid \mathbf{y})||p(\mathbf{z}))$ is the Kullbeck-Leibler divergence term for latent regularization. The reconstruction term encourages accurate reconstruction of input data $\mathbf{y}$ from the latent data $\mathbf{z}$, and the divergence term penalizes the approximate posterior $q_\theta(\mathbf{z} \mid \mathbf{y})$ for deviation away from the prior $p(\mathbf{z})$.

Diffusion models are a hierarchical expansion of VAEs [140]. In the DDPM framework [143], discussed in Section 4.2, the discrete latent variable sequence $\mathbf{y}_1, \ldots, \mathbf{y}_T$ is obtained by the forward noising process, and the encoder $q(\mathbf{y}_{1:T}|\mathbf{y}_0)$ is fixed. The decoder $p_\theta(\mathbf{y}_{0:T})$ is learned by the reverse Markov chain, and training optimizes the same ELBO objective as in equation (4.4) expanded over T latent levels. This hierarchy is extended to a continuous-time formulation in the score-based SDE model framework [144], as outlined in Section 4.3. The latent variable sequence is the continuous stochastic process $\mathbf{y}(t)_{t\in[0,T]}$, and the forward process is a forward SDE. In both frameworks, the learned reverse process, whether a discrete Markov chain or a continuous reverse-time SDE, corresponds to the VAE encoder, and the training goal in both cases is to maximize a variational lower bound on $\log\left(p_\theta(\mathbf{y}_0)\right)$.

4.2 Denoising diffusion probabilistic models

Denoising diffusion probabilistic models are generative models that transform data into noise through a fixed forward diffusion process and learn to invert this process via a reverse-time Markov chain, or reverse process [143]. The goal is to construct a sequence of increasingly noisy

random variables that converge to a tractable distribution, typically Gaussian, while ensuring that reverse transitions can be learned efficiently.

4.2.1 Forward process

The forward diffusion process is a discrete-time Gaussian diffusion process that progressively adds noise to data over a sequence of timesteps according to some variance schedule $\Upsilon_1, ..., \Upsilon_T$ that denotes the amount of noise added at each timestep [143]. Let $\mathbf{y}_t \in \mathbb{R}^k$ be a vector of k dimensions, with k equaling the number of pixels after vectorization when $\mathbf{y}_t$ represents an image. The forward process defines a conditional distribution $q(\mathbf{y}_t \mid \mathbf{y}_{t-1})$ given by

$$\mathbf{y_t} = \sqrt{1 - \Upsilon_t}\mathbf{y}_{t-1} + \sqrt{\Upsilon_t}\mathbf{z}_t, \qquad \mathbf{z}_t \sim \mathcal{MN}(\mathbf{0}, \mathbf{I}_k). \tag{4.5}$$

The term $\sqrt{1 - \Upsilon_t}\mathbf{y}_{t-1}$ preserves signal from the previous state according to the variance schedule, and $\sqrt{\Upsilon_t}\mathbf{z}_t$ is Gaussian noise to inject randomness. In the original DDPM formulation, described in [143], the variance schedule is set to linearly increase from $\Upsilon_1 = 10^{-4}$ to $\Upsilon_T = 0.02$, with the data scaled between $[-1, 1]$.

Defining $a_t = 1 - \Upsilon_t$, and $\bar{a_t} := \prod_{s=1}^{t} a_s$, the forward model has a closed-form expression for the marginal distribution of $\mathbf{y}_t$ conditioned on the original data $\mathbf{y}_0$:

$$\mathbf{y}_t \mid \mathbf{y}_0 \sim \mathcal{MN}(\sqrt{\bar{a}_t}\mathbf{y}_0, (1 - \bar{a}_t)\mathbf{I}_k). \tag{4.6}$$

This result implies that $\mathbf{y}_t$ can be directly sampled from the original data $\mathbf{y}_0$:

$$\mathbf{y}_t = \sqrt{\bar{a}_t}\mathbf{y}_0 + \sqrt{1 - \bar{a}_t}\boldsymbol{\zeta}, \tag{4.7}$$

where $\boldsymbol{\zeta} \sim \mathcal{MN}(\mathbf{0}, \mathbf{I})$. This closed-form marginal distribution is a crucial property of DDPMs, as it enables efficient training by allowing samples at any diffusion step to be generated without simulating the entire forward process [143].

4.2.2 Reverse process

While the forward process gradually transforms structured data into Gaussian noise, the reverse process seeks to recover the original data distribution by iteratively removing noise. The reverse process is what allows the model to generate new images or denoise corrupted images at some noise level [143, 146].

The reverse process is modeled as a Markov chain parameterized by θ, with transition probabilities $p_\theta(\mathbf{y}_{t-1} \mid \mathbf{y}_t)$ for $t = T, ..., 1$. When conditioned on both $\mathbf{y}_t$ and $\mathbf{y}_0$, the posterior distribution of the forward process $q(\mathbf{y}_{t-1} \mid \mathbf{y}_t, \mathbf{y}_0)$ is also Gaussian, due to the Gaussian structure of the forward transitions [140]. However, since $\mathbf{y}_0$ is unknown during generation, DDPMs approximate the reverse conditionals using a neural network, typically U-net [2, 143]. The reverse process is therefore parameterized as a Gaussian distribution with form

$$p_\theta(\mathbf{y}_{t-1} \mid \mathbf{y}_t) = \mathcal{MN}(\mathbf{y}_{t-1} : \boldsymbol{\mu}_\theta(\mathbf{y}_t, t), \sigma_t^2 \mathbf{I}), \tag{4.8}$$

where $\boldsymbol{\mu}_\theta(.)$ is a function to predict the forward process posterior mean, σ_t^2 is the variance, and t denotes the diffusion timestep.

The parameters of $\boldsymbol{\mu}_\theta$ are learned using the neural network, or U-net [2, 143], conditioned on both the noisy input $\mathbf{y}_t$ and the timestep t [147]. To see the motivation for this, we can rearrange equation (4.7) to express the noise in terms of $\mathbf{y}_t$ and $\mathbf{y}_0$:

$$\boldsymbol{\zeta} = \frac{\mathbf{y}_t - \sqrt{\bar{a}_t}\mathbf{y}_0}{\sqrt{1 - \bar{a}_t}}. \tag{4.9}$$

The neural network $\boldsymbol{\zeta}_\theta(\mathbf{y}_t, t)$ is trained to estimate this noise term, and the timestep t is incorporated into the network via embedding. From this estimate, the predicted mean $\boldsymbol{\mu}_\theta(\mathbf{y}_t, t)$ can be determined analytically [143, 140].

4.2.3 Training and data generation for discrete-time denoising formulation

During training (see Algorithm 1), the objective is to minimize the mean squared error between the true noise $\zeta \sim \mathcal{MN}(\mathbf{0}, \mathbf{I})$ and the predicted noise $\boldsymbol{\zeta}_\theta(\mathbf{y}_t, t)$ [143]:

$$L_\theta := \mathbb{E}_{t,\mathbf{y}_0,\boldsymbol{\zeta}}[||\boldsymbol{\zeta} - \boldsymbol{\zeta}_\theta(\sqrt{\bar{a}_t}\mathbf{y}_0 + \sqrt{1-\bar{a}_t}\boldsymbol{\zeta}, t)||^2]. \tag{4.10}$$

This objective has deep connections to denoising score matching [148], as minimizing the objective function is equivalent to denoising score matching across different noise levels. Furthermore, during training in diffusion models, the variational lower bound is optimized on the negative log-likelihood, or, equivalently, optimizing the ELBO [143, 140, 40]:

$$\mathbb{E}[-\log p_\theta(\mathbf{y}_0)] \leq \mathbb{E}_q \left[-\log \frac{p_\theta(\mathbf{y}_{0:T})}{q(\mathbf{y}_{1:T} \mid \mathbf{y}_0)} \right]. \tag{4.11}$$

The predicted noise network $\boldsymbol{\zeta}_\theta(\mathbf{y}_t, t)$ is implemented as a U-Net, as described in Chapter 1.2.2, with specific modifications for time-conditional denoising. The model inputs are the noisy image $\mathbf{y}_t$, corrupted according to the forward diffusion process, and the discrete timestep $t \in \{1, ..., T\}$, indicating the noise level [143]. To condition the network on the timestep t, a sinusoidal positional embedding is employed, similar to those used in transformer architectures [147]. For embedding dimension d, the timestep is encoded as:

$$PE(t,j) = \begin{cases} \sin\left(\frac{t}{10000^{2i/d}}\right) & \text{if } j = 2i \\ \cos\left(\frac{t}{10000^{2i/d}}\right) & \text{if } j = 2i+1 \end{cases}, \tag{4.12}$$

where $j \in \{0, 1, ..., d-1\}$ and $i = \lfloor \frac{j}{2} \rfloor$. This encoding uses different frequency components $\omega_i = 10000^{-2i/d}$ spanning wavelengths from 2π to $10000 \times 2\pi$, enabling the model to distinguish between timesteps at multiple scales. The sinusoidal embedding is them passed through a small multi-layer perceptron (MLP) to obtain a time embedding vector $\mathbf{t}_{\text{emb}} \in \mathbb{R}^{B \times D}$, where D is typically four times the base channel width and B is the number of convolutional feature maps

in the first residual block [149]. For each residual block, the time embedding vector is projected to match the dimensions of the feature maps in the layer:

$$\mathbf{t}_{\text{proj}} = g(\mathbf{t}_{\text{emb}} \cdot \mathbf{W}^T + \mathbf{c}),$$

where g is the activation function, $\mathbf{W}$ is a learned weight matrix and $\mathbf{c}$ is the bias created for the corresponding layer [149]. This projection can then be added element-wise to each feature map in the layer, effectively conditioning the layers of the U-net on time t [143].

Each residual block contains operations for group normalization (which normalizes across channels for each sample), non-linear activations, and the injection of the time embedding. The model architecture follows the U-net [2] structure given in Figure 1.2, with the image dimensions and number of layers differing depending on the input image data. Self-attention layers are incorporated at lower spatial resolutions (closer to the bottleneck of the U-net) to highlight important global features to accurately reconstruct long-range dependencies [150]. The output of this network is a predicted noise with the same dimensions as the input, with the final convolutional layer abstracting this outcome from the feature maps.

Algorithm 1 Model training for DDPMs, as proposed in [143].

1: **repeat**

2: Sample $\mathbf{y}_0 \sim q(\mathbf{y}_0)$

3: Sample $t \sim \text{Unif}(\{0, \ldots, T\})$

4: Sample $\boldsymbol{\zeta} \sim \mathcal{MN}(\mathbf{0}, \mathbf{I})$

5: Compute gradient, $\nabla_\theta ||\boldsymbol{\zeta} - \boldsymbol{\zeta}_\theta(\sqrt{\bar{a}_t}\mathbf{y}_0 + \sqrt{1-\bar{a}_t}\boldsymbol{\zeta}, t)||^2$

6: **until** convergence

After learning the predicted noise network $\boldsymbol{\zeta}_\theta(\mathbf{y}_t, t)$, images matching the learned distribution are generated by initializing $\mathbf{y}_T$ from a standard multivariate normal distribution and iteratively applying the learned reverse transitions, referred to as sampling (see Algorithm

2) [143]. For each timestep t, a sample from $p_\theta(\mathbf{y}_{t-1} \mid \mathbf{y}_t)$ is obtained using the predicted mean and variance, gradually transforming noise into a structured sample $\mathbf{y}_0$ that approximates pulling from the learned data distribution.

Algorithm 2 Data generation in DDPM by sampling, as proposed in [143].

Initialize some discrete $t = T$
Sample $\mathbf{y}_T \sim \mathcal{MN}(\mathbf{0}, \mathbf{I})$
for $t = T$ to 1 **do**
    if $t > 1$ **then**
        Sample $\mathbf{z} \sim \mathcal{MN}(\mathbf{0}, \mathbf{I})$
    else
        $\mathbf{z} = \mathbf{0}$
    end if
    $\mathbf{y}_{t-1} = \frac{1}{\sqrt{a_t}} \left(\mathbf{y}_t - \frac{1-a_t}{\sqrt{1-\bar{a}_t}} \boldsymbol{\zeta}_\theta(\mathbf{y}, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{y}_0$

The most popular training and generation implementation of DDPM is exactly described in the Keras package example [149], which will serve as the basis for the implemented DDPM models described in this chapter. A list of implemented functions and a brief description of their purpose is provided in Appendix B.1.

4.3 Score-based generative modeling using stochastic differential equations

Score-based generative models using SDEs provide a continuous-time formulation of diffusion-based generative modeling, in which data are transformed into noise through a forward SDE and recovered by solving a learned reverse-time SDE [144]. This framework generalizes DDPMs by replacing the discrete-time diffusion process with a continuous-time stochastic process, offering a more flexible theoretical perspective.

4.3.1 Forward SDE

Let $\mathbf{y}(t) \in \mathbb{R}^k$ denote a continuous-time stochastic process indexed by time $t \in [0, T]$, where k is the dimensionality of the data, and the upper bound of the time indexing is often $T = 1$. The forward process is defined by an Itô stochastic differential equation [151, 144]:

$$d\mathbf{y} = \mathbf{f}(\mathbf{y}, t)dt + g(t)d\mathbf{w}, \tag{4.13}$$

where $\mathbf{f}(., t)$ is the drift coefficient of $\mathbf{y}(t)$, $g(.)$ is the scalar diffusion coefficient of $\mathbf{y}(t)$, and $\mathbf{w}$ is the standard Wiener process. The original formulation of the score-based SDE models proposes complimentary SDEs, specifically the variance-preserving (VP) SDE [144]

$$d\mathbf{y} = -\frac{1}{2}\Upsilon(t)\mathbf{y}dt + \sqrt{\Upsilon(t)}d\mathbf{w}, \tag{4.14}$$

and the sub-VP SDE:

$$d\mathbf{y} = -\frac{1}{2}\Upsilon(t)\mathbf{y}dt + \sqrt{\Upsilon(t)(1 - e^{-2\int_0^t \Upsilon(s)ds})}d\mathbf{w}, \tag{4.15}$$

where $\Upsilon(t)$ is a continuous noise schedule that controls the rate at which noise is injected into the system. The forward SDE (4.15) performs particularly well on likelihoods, but the original

formulation for training utilizes the variance exploding (VE) SDE [144]:

$$d\mathbf{y} = \sqrt{\left(\frac{d\left[B^2(t)\right]}{dt}\right)}d\mathbf{w}, \tag{4.16}$$

where B is a function of t, like $B(t) = b^t$ where b is a constant as implemented in the original code base [144]. Under these forward SDEs, the marginal distributions of $\mathbf{y}(t)$ evolve smoothly from the data distribution at $t = 0$ towards tractable prior distributions as $t \to T$,

$$p_{0t}(\mathbf{y}(t) \mid \mathbf{y}(0)) = \begin{cases} \mathcal{MN}\left(\mathbf{y}(t); \mathbf{y}(0)e^{-\frac{1}{2}\int_0^t \Upsilon(s)ds}, [1 - e^{-\int_0^t \Upsilon(s)ds}]^2\mathbf{I}\right) & \text{for sub-VP SDE,} \\ \mathcal{MN}\left(\mathbf{y}(t); \mathbf{y}(0)e^{-\frac{1}{2}\int_0^t \Upsilon(s)ds}, [\mathbf{I} - \mathbf{I}e^{-\int_0^t \Upsilon(s)ds}]^2\right) & \text{for VP SDE,} \\ \mathcal{MN}\left(\mathbf{y}(t); \mathbf{y}(0), \left[B^2(t) - B^2(0)\right]\mathbf{I}\right) & \text{for VE SDE.} \end{cases} \tag{4.17}$$

The transition densities $p_{0t}(\mathbf{y}(t) \mid \mathbf{y}(0))$ have closed-form expressions, which is essential for efficient training.

4.3.2 Reverse SDE

A key theoretical result underlying the score-based SDE model is that the forward SDE admits a reverse-time SDE describing the dynamics of the process backward in time, as the reverse of a diffusion process is still a diffusion process [146]. The reverse SDE is given by:

$$d\mathbf{y} = [\mathbf{f}(\mathbf{y}, t) - g(t)^2\nabla_{\mathbf{y}} \log p_t(\mathbf{y})]dt + g(t)d\bar{\mathbf{w}}, \tag{4.18}$$

where $\bar{\mathbf{w}}$ is a standard Wiener process flowing backwards from T to 0, and $\nabla_{\mathbf{y}} \log p_t(\mathbf{y})$ is the score function of the marginal distribution at time t. For the VE SDE, the reverse process is expressed as [144, 146]

$$d\mathbf{y} = -\frac{d\left[B^2(t)\right]}{dt}\nabla_{\mathbf{y}} \log\left(p_t(\mathbf{y})\right) dt + \sqrt{\left(\frac{d\left[B^2(t)\right]}{dt}\right)}d\bar{\mathbf{w}}. \tag{4.19}$$

Because the true score function is unknown, it is approximated using a neural network, typically a U-net, $\mathbf{s}_\theta(\mathbf{y}(t), t)$, which is trained to estimate the gradient of the log-density at each time t.

The score function points in the direction of steepest ascent in the log-probability space and characterizes the direction of the data distribution at each noise level [152].

4.3.3 Training and data generation for continuous-time denoising formulation

The continuous score network $\mathbf{s}_\theta(\mathbf{y}(t), t)$ is trained using denoising score matching [148, 152, 153]. The training procedure is summarized by Algorithm 3. For a fixed time t, the objective minimizes the expected squared error between the network's score prediction and the true score over random times t, initial data $\mathbf{y}_0$, and noisy image $\mathbf{y}_t$ [144]:

$$\boldsymbol{\theta}^* = \arg\min_\theta \mathbb{E}_t \left\{ \lambda(t) \mathbb{E}_{\mathbf{y}(0)} \mathbb{E}_{\mathbf{y}(t)|\mathbf{y}(0)} \left[||\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{y}(t), t) - \nabla_{\mathbf{y}(t)} \log p_{0t}(\mathbf{y}(t) \mid \mathbf{y}(0))||_2^2 \right] \right\} \tag{4.20}$$

where $\lambda(.)$ is a positive weighting function, and $\mathbf{y}(0) \sim p_0(\mathbf{y})$ and $\mathbf{y}(t) \sim p_{0t}(\mathbf{y}(t) \mid \mathbf{y}(0))$. For training, timestep t is sampled uniformly over the interval $[0, T]$, and the noisy image $\mathbf{y}_t$ is generated by adding noise corresponding to timestep t to an input image $\mathbf{y}_0$.

The setup for the continuous score network $\mathbf{s}_\theta(\mathbf{y}(t), t)$ is almost identical to the discrete score network $\epsilon_\theta(\mathbf{y}_t, t)$. The inputs for $\mathbf{s}_\theta(\mathbf{y}(t), t)$ are some noisy image $\mathbf{y}(t)$ and continuous time t, which corresponds to the noise level of the image [151]. Instead of using sinusoidal time embedding [147], the score-based generative model uses Gaussian random features [154] to encode each timestep t:

$$\phi(t) = [\cos(2\pi\Omega t); \sin(2\pi\Omega t)], \tag{4.21}$$

where random sample $\Omega \sim \mathcal{MN}(\mathbf{0}, \iota^2\mathbf{I})$ is explicitly defined for each model, and ι is a set scale parameter (e.g., $\iota = 30$ in corresponding code repository). This encoding is then injected into the U-net feature maps using the same process as DDPM [151, 143, 147]. The output of the U-net is the score

$$\mathbf{s}(\mathbf{y}(t), t) = \nabla_{\mathbf{y}} \log p_t(\mathbf{y}), \tag{4.22}$$

with the score being abstracted in the final convolutional layer of the U-net.

Algorithm 3 Model training in score-based SDE models, as proposed in [144].

1: **repeat**

2: Sample $\mathbf{y}(0) \sim q(\mathbf{y}(0))$

3: Sample $t \sim \text{Unif}(0, T)$

4: Sample $\mathbf{z} \sim \mathcal{MN}(\mathbf{0}, \mathbf{I})$

5: Compute $\mathbf{y}(t) \sim p_{0t}(\mathbf{y}(t) \mid \mathbf{y}(0))$

6: Compute gradient, $\nabla_\theta ||\mathbf{s}_\theta(\mathbf{y}(t), t) - \nabla_{\mathbf{y}(t)} \log p_{0t}(\mathbf{y}(t), \mathbf{y}(0))||$

7: **until** convergence

To generate new samples, as outlined in Algorithm 4, the reverse-time SDE is initialized at $\mathbf{y}(T) \sim \mathcal{MN}(\mathbf{0}, \mathbf{I})$ and is numerically integrated backward in time to $t = 0$. This integration is often performed using numerical methods such as the Euler–Maruyama method [155]. The resulting value $\mathbf{y}(0)$ corresponds to a sample from the learned data distribution.

An alternative deterministic numerical method proposed by the original score-based SDE model [144] is obtained from the reverse SDE once scores are known, yielding an ordinary differential equation (ODE) that shares the same marginal distributions as the reverse-time SDE:

$$d\mathbf{y} = \left[\mathbf{f}(\mathbf{y}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{y}} \log p_t(\mathbf{y}) \right] dt. \tag{4.23}$$

This deterministic formulation enables exact likelihood computation via the instantaneous change of variables formula, facilitates latent space manipulation, and provides a deterministic mapping between noise and data [144].

The authors of the original manuscript provides a recommended model walkthrough and implementation using VE SDE [144], which will serve as the basis for training and implementing the score-based SDE model in this chapter.

Algorithm 4 Data generation in score-based SDE models, as proposed in [144].

1: Initialize continuous $t = T$

2: Sample $\mathbf{y}_T \sim \mathcal{MN}(\mathbf{0}, \mathbf{I})$

3: Choose timestep Δt

4: **while** $t > 0$ **do**

5: Compute $\mathbf{y}(t - \Delta t)$ using ODE solver (4.23), $\mathbf{y}(t - \Delta t) = \mathbf{y}(t) + \left[\mathbf{f}(\mathbf{y}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{y}} \log p_t(\mathbf{y})\right] \Delta t$

6: Update $t = t - \Delta t$

7: **end while**

8: **return** $\mathbf{y}(0)$

4.4 Simulations from known distributions

To evaluate the performance of diffusion models on low-dimensional data, we consider simulated data sampled from a 2D Gaussian mixture model (GMM) with two well-separated modes. The target distribution is defined as

$$p(\mathbf{y}_0) = 0.5\mathcal{MN}(\mathbf{y}_0 \mid (-0.7, 0.7)^T, 0.1\mathbf{I}_k) + 0.5\mathcal{MN}(\mathbf{y}_0 \mid (0.7, -0.7)^T, 0.1\mathbf{I}_k). \tag{4.24}$$

The modes are positioned at diagonally opposite locations with equal covariance, creating a bimodal distribution with a clear separation that tests the model's ability to capture complex data structure.

For modeling the score functions $\boldsymbol{\zeta}_\theta(\mathbf{y}_t, t)$ and $\mathbf{s}_\theta(\mathbf{y}(t), t)$, we employ a multi-layer perceptron (MLP) [145] rather than the U-Net architecture commonly used for image generation. The 2D GMM produces coordinate pairs with no inherent spatial structure or local correlations. In contrast, U-Nets are specifically designed to exploit spatial structures through convolutional operations that aggregate information from nearby pixels [2]. The MLP architecture provides a

Table 4.1: Model specifications, like number of epochs, training data image count, and MLP shape, for DDPM and score-based SDE model in generating 2D GMM data.

	DDPM	**SDE**
Training Data Input Shape	(1000, 2)	(1000, 2)
MLP Shape	[128, 128, 128]	[128, 128, 128]
Number of Epochs	1000	1000
Batch Size	64	64

more appropriate architecture for this task, as it has no spatial assumptions [145]. For a single layer of an MLP, the output is defined as:

$$\mathbf{l} = g(\mathbf{W}^T\mathbf{y} + \mathbf{c}), \tag{4.25}$$

where g is an activation function (typically ReLU), $\mathbf{W}$ is the weight matrix, $\mathbf{y}$ is the input vector, and $\mathbf{c}$ are the biases [156]. MLPs with k layers apply this function recursively:

$$\begin{aligned}
\mathbf{l}_1 &= g(\mathbf{W}_1^T\mathbf{y} + \mathbf{c}_1) \\
\mathbf{l}_2 &= g(\mathbf{W}_2^T\mathbf{l}_1 + \mathbf{c}_2) \\
&\vdots \\
\mathbf{l}_k &= g(\mathbf{W}_k^T\mathbf{l}_{k-1} + \mathbf{c}_k),
\end{aligned}$$

where the final layer output is in $\mathbb{R}^2$.

Both the DDPM and score-based SDE model are trained on 1,000 samples from the GMM distribution in Equation 4.24. The MLPs for both models share an identical architecture consisting of 3 hidden layers with 128 neurons each. The input is the concatenation of the 2D sample coordinates $\mathbf{y}_t$ and the time embedding t, and the output is 2D score estimates. All major model architectures features and hyperparameters are kept constant across both models, including MLP shape, batch size, and number of epochs. All specifications are documented in Table 4.1.

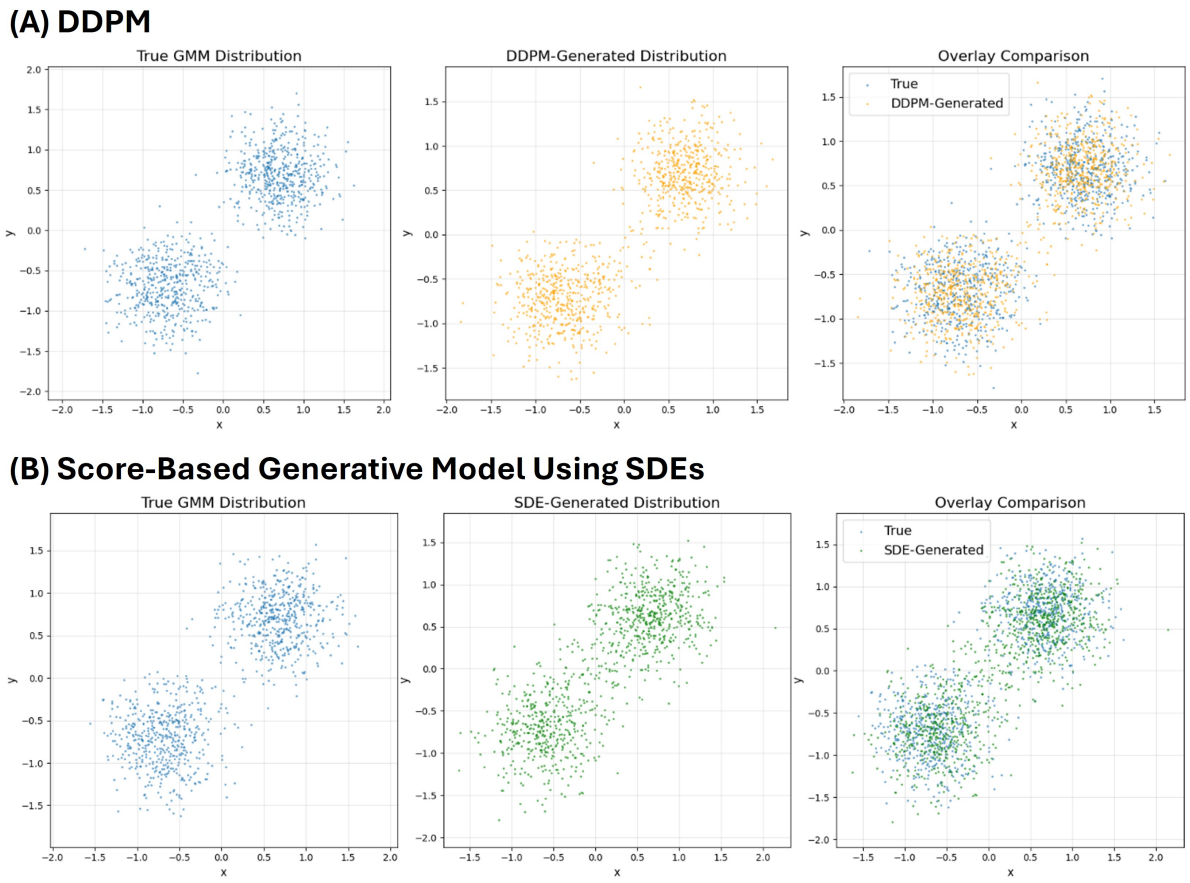


Figure 4.1: (A) Samples from the true GMM distribution (*left*), generated samples (*middle*), and overlay comparison (*right*) for the trained DDPM model. (B) Samples from the true GMM distribution, generated samples, and overlay comparison for the trained score-based SDE model.

After training, we generate 1,000 samples from each model and compare them to samples drawn from the true GMM distribution. Figure 4.1 presents a comparison of the generated distributions. Both models successfully reproduce the bimodal structure of the true distribution, demonstrating their ability to learn and sample from a multi-modal density. The DDPM exhibits sharper mode separation, more accurately capturing the region of low probability between the two modes, while the score-based SDE model produces some samples within that region. However, both achieve high-quality generation on this benchmark task.

4.5 Generating and denoising cell images from experimental data

To test the models' performance in real-world settings, we evaluate the performance of diffusion models on generating and denoising cell microscopy data. The training dataset consists of 1,500 images of size 256×256 pixels, randomly cropped from a time-lapse hDF cell microscopy video sequence [1]. This dataset contains images with features that can be challenging for biological imaging analysis tasks, such as subtle intensity variations and noise.

We pursue two complementary objectives with this dataset. We first assess the models' ability to learn and generate realistic cell images, with direct applications in cell image data augmentation. Second, we evaluate the models' denoising capabilities by testing their ability to reconstruct clean images from corrupted test samples. We add controlled amounts of noise to test images and assess whether the trained diffusion models can accurately recover the original images. This task is particularly relevant for microscopy applications, where various noise sources, like environmental interference and image bleaching, often degrade images. The ability to denoise microscopy images while preserving cellular structures could enhance downstream analysis pipelines for cell tracking, morphology quantification, and automated phenotyping.

For the DDPM implementation, we adapt the U-Net architecture from the reference Keras implementation [149], which follows the original DDPM structure from [143]. Similarly, the score-based SDE model employs U-Net adapted from the official code base for [144]. Due to computational and runtime constraints, the batch size from Table 4.1 is reduced from 64 to 16 for both models when training on cell image data. For DDPM, the number of epochs is reduced to 70, and the number of epochs for the score-based SDE model is reduced to 200. Furthermore, due to extreme computation time requirements, the U-net shape of DDPM is set

Table 4.2: Model specifications, like number of epochs, training data image count, and layers in U-net, for DDPM and score-based generative model using SDE in generating and denoising cell image data.

	DDPM	**SDE**
Training Data Image Count	1500	1500
Test Data Image Count	500	500
Image Shape	(256, 256, 1)	(256, 256, 1)
U-Net Shape	[64, 132, 256]	[32, 64, 128, 256]
Number of Epochs	70	200
Batch Size	16	16
Noise Scheduling	$[\Upsilon_{\text{start}}, \Upsilon_{\text{end}}] = [0.0001, 0.02]$	$B(t) = b^t,\ t \in [0, 1],$ with $b = 25$

to [64, 132, 256]. The score function for the image data is learned with U-net. All specifications are documented in Table 4.2.

All cropped images undergo pre-processing before training. First, a 3×3 median filter is applied for noise reduction and smoothing. Second, pixel intensities are normalized using min-max scaling:

$$y' = \frac{\mathbf{y} - \min(\mathbf{y})}{\max(\mathbf{y}) - \min(\mathbf{y})}, \tag{4.26}$$

which increases contrast between the background and cell regions. For model training, pixel values are linearly transformed to the range [-1, 1] to match the typical output range of the network's final activation.

4.5.1 Generating new cell images

Figure 4.2 presents a comparison of simulated cell images generated by the DDPM and score-based SDE approaches, alongside representative training samples. Both models successfully learned to generate images similar to the training images, and the generated images

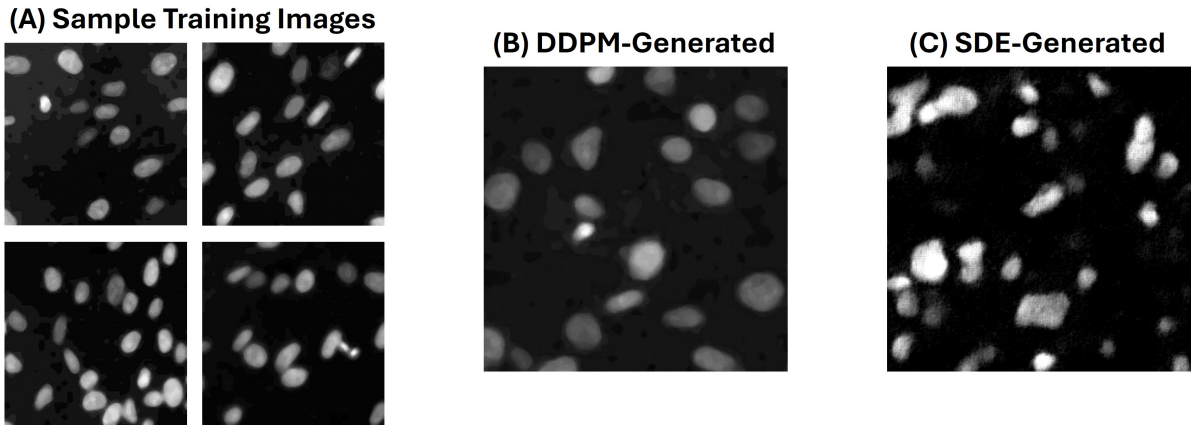


Figure 4.2: (A) Sample images from training data. (B) Cell image generated by trained DDPM. (C) Cell image generated by trained score-based SDE model.

have elliptical shapes for cells, realistic size variation, and reasonable cell density. However, the qualitative features of each generated image reveal distinct characteristics of each approach. The DDPM-generated images, shown in Figure 4.2 (B), exhibited smooth cell boundaries with consistent intensity values across cells. However, this smoothness and consistency also result in a minor loss of fine textural details when compared to the representative training images. The cells appear to have natural variation in size, shape, and luminescence, and the orientations and density appear to match well with the training images. In contrast, the SDE-generated images, shown in Figure 4.2 (C), produced sharper cellular structures with higher intensity variation between cells. However, this increased sharpness occasionally resulted in slightly more irregular cell boundaries compared to the DDPM-generated images.

This difference in generative output is likely due to the discrete nature of the DDPM noise variance schedule. Typically, this noise variance schedule will produce more conservative, averaged outputs [143]. The model averages over uncertainty at each timestep, which helps eliminate rapid pixel-to-pixel intensity fluctuations. Alternatively, the continuous formulation in score-based SDE model pursues sharper gradient estimates [144], which may overshoot towards

finer details that can look unnatural in cell structures. The continuous score function that the loss function is based on is much more sensitive to local intensity gradients [151]. The model may not have encountered sufficient structural diversity within the comparatively small training set to constrain high pixel intensities within generated cell boundaries [157, 158], thus leading to the observed irregularities in generated cell shapes.

4.5.2 Denoising cell images

To further validate the learned score functions, both models are also evaluated on image denoising performance. Test images are corrupted with independent Gaussian noise, $\tilde{\mathbf{y}} = \mathbf{y}' + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{MN}(0, \sigma_0^2\mathbf{I})$, with $\sigma_0^2 \in \{0.1^2, 0.3^2, 0.5^2\}$. These noise levels represent a progression from mild to intermediate image degradation, as would be observed in real microscopy conditions from fluctuations in environmental conditions and equipment calibration.

Figure 4.3 illustrates the denoising performance of the DDPM algorithm across the three noise levels. The left column shows the original clean cell image from the test set, the middle column displays noisy observations generated by adding Gaussian noise with variances $\sigma_0^2 \in [0.1^2, 0.3^2, 0.5^2]$, and the right column presents the corresponding denoised DDPM predictions.

For denoising, we initialize the reverse diffusion process from the noisy observation at an appropriate intermediate timestep t that roughly corresponds to the noise level σ_0^2. This allows the model to leverage the partial information present in the corrupted image. The choice of denoising timestep was determined empirically through visual inspection for each noise level for DDPM: $t = 25$ for $\sigma_0^2 = 0.1^2$, $t = 75$ for $\sigma_0^2 = 0.3^2$, and $t = 100$ for $\sigma_0^2 = 0.5^2$. This timestep selection represents a careful balance, as using too few steps results in incomplete noise removal, while using excessive steps can over-smooth fine details (see Figure B.1 in the Appendix for

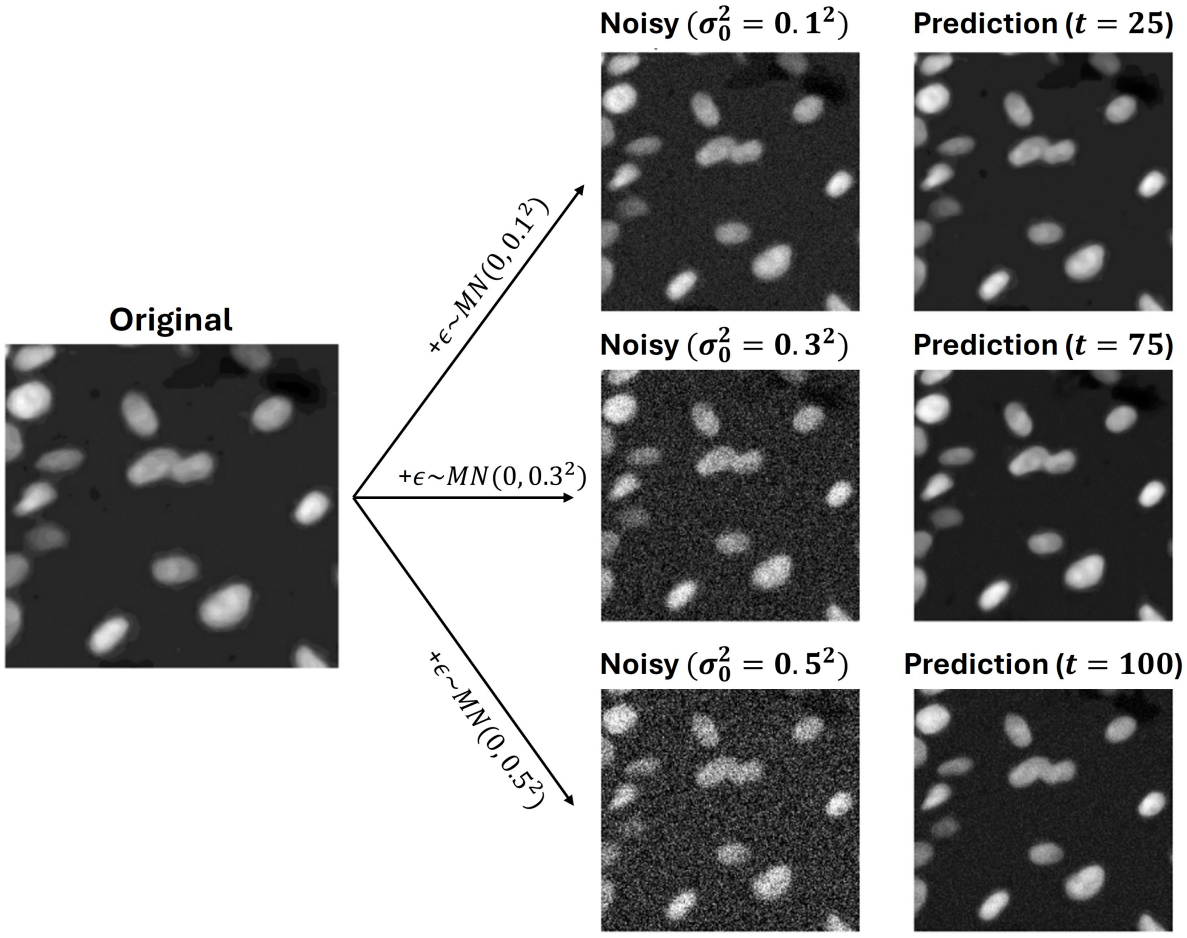


Figure 4.3: Original (left), noisy (middle), and denoised via DDPM (right) cell images across Gaussian noise levels $\sigma_0^2 \in [0.1^2, 0.3^2, 0.5^2]$.

sensitivity analysis).

Figure 4.4 illustrates the denoising performance of the score-based SDE model across the same noise levels. Similar to the DDPM approach, we initialize the reverse SDE from the noisy observation at an appropriate time t in the continuous diffusion process. The denoising timestep $t = 0.3$ was determined empirically through visual inspection to provide optimal performance across all noise levels tested. Using higher time values, corresponding to starting from noisier states, resulted in incomplete reconstructions with missing or faded cells (see Figure B.2 in the Appendix for sensitivity analysis).

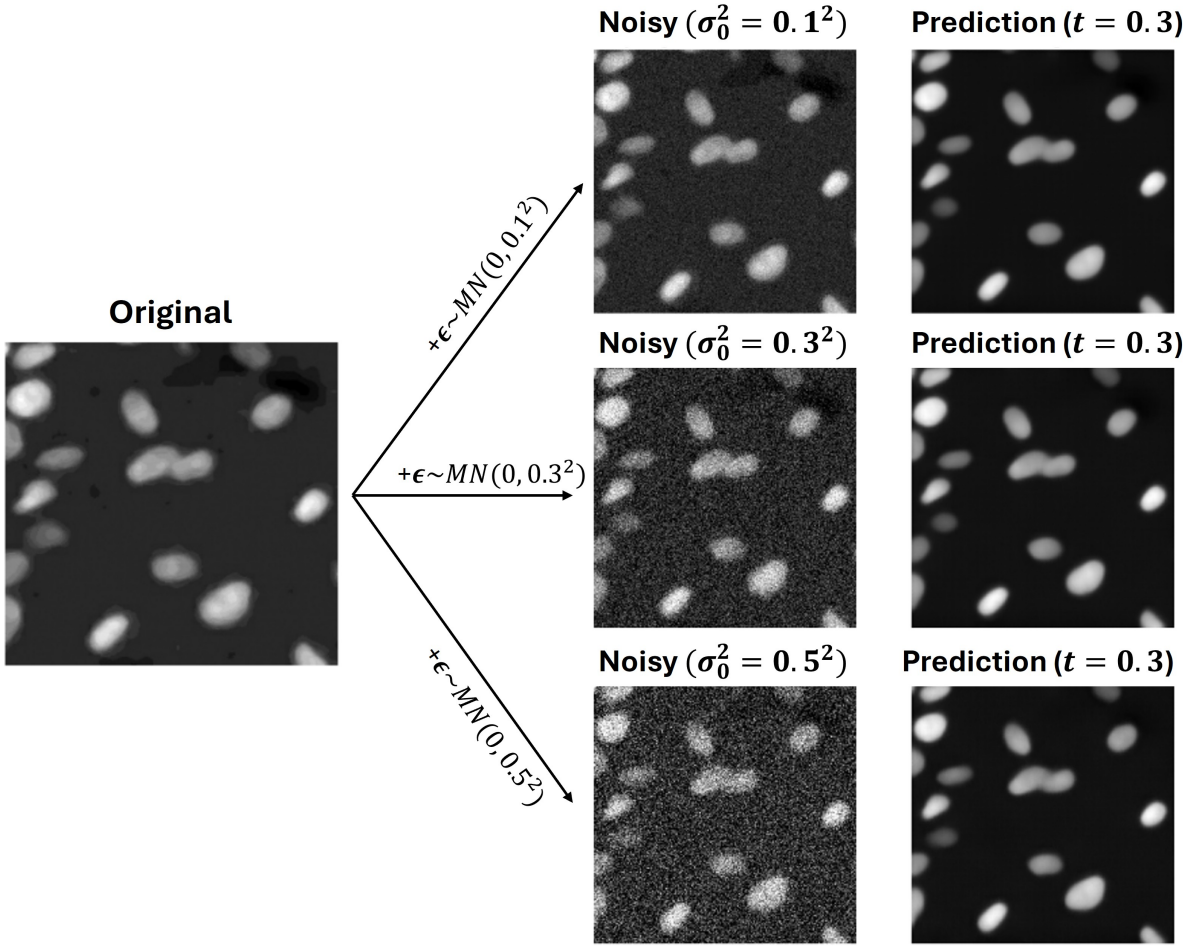


Figure 4.4: Original (left), noisy (middle), and denoised via score-based SDE model (right) cell images across Gaussian noise levels $\sigma_0^2 \in [0.1^2, 0.3^2, 0.5^2]$.

Both methods also yield strong results for denoising completely corrupted images, although exact one-to-one image reconstruction is difficult to achieve. Figure 4.5 presents the denoising results for both methods when the original image is corrupted with high-variance noise ($\sigma_0 = 2$), which is then clipped to be in the valid value range between $[0, 1]$. Cellular objects in these noisy images, given in Figures 4.5 (A) and (B), are almost completely obscured. Although noise of this level is unlikely to be observed in real experiments, it serves as a useful stress test to assess how well the generative models have learned the underlying image structures. Figures 4.5 (C) and (D) display the predicted images for DDPM and the score-based SDE model,

respectively, with empirically selected timesteps t. The DDPM result is similar to the original image in object shape and pixel intensity. However, the edges of the cell objects are not as well-defined, appearing somewhat "blurry" visually, and the background is a lighter intensity. The result for the score-based SDE model yields cell objects with somewhat similar object structures to the original image and defined edges. However, some cells are predicted to be much smaller than in the original image. Both methods missed the lighter cell in the top right corner of the original image. Although the results are not exact numerical representations of the original images, they are qualitatively good representations for understanding the original visualizations. Furthermore, both techniques outperform the GP-based smoothing method, as the GP method is unable to fully reconstruct the cell objects, given in Figure 4.5 (E). Although GP is able to detect the approximate locations of the cells, as the smoothed objects correspond to the original cell locations, the cell shape is almost completely lost. Furthermore, the amount of background noise is still extremely high in the smoothed image. The difference in performance between the generative models and fast GP demonstrates the inherent nature of each algorithm type. The generative algorithms are able to effectively "hallucinate" [159] plausible cell-like structures based on the learned distribution, as the noise corruption is so extreme that denoising relies on the generative abilities of these models. Alternatively, fast GP is largely an image smoothing mechanism [11], so any obscured cell structure signal cannot be recovered.

To quantitatively evaluate the models' denoising performance, we compute the RMSEs and structural similarity index measures (SSIM) for 500 test images corrupted with noise variance $\sigma_0^2 = 0.3^2$. This intermediate noise level is representative of realistic microscopy conditions and provides sufficient degradation to meaningfully test reconstruction quality. We measure pixel-wise reconstruction error using RMSE and perceived quality using the SSIM

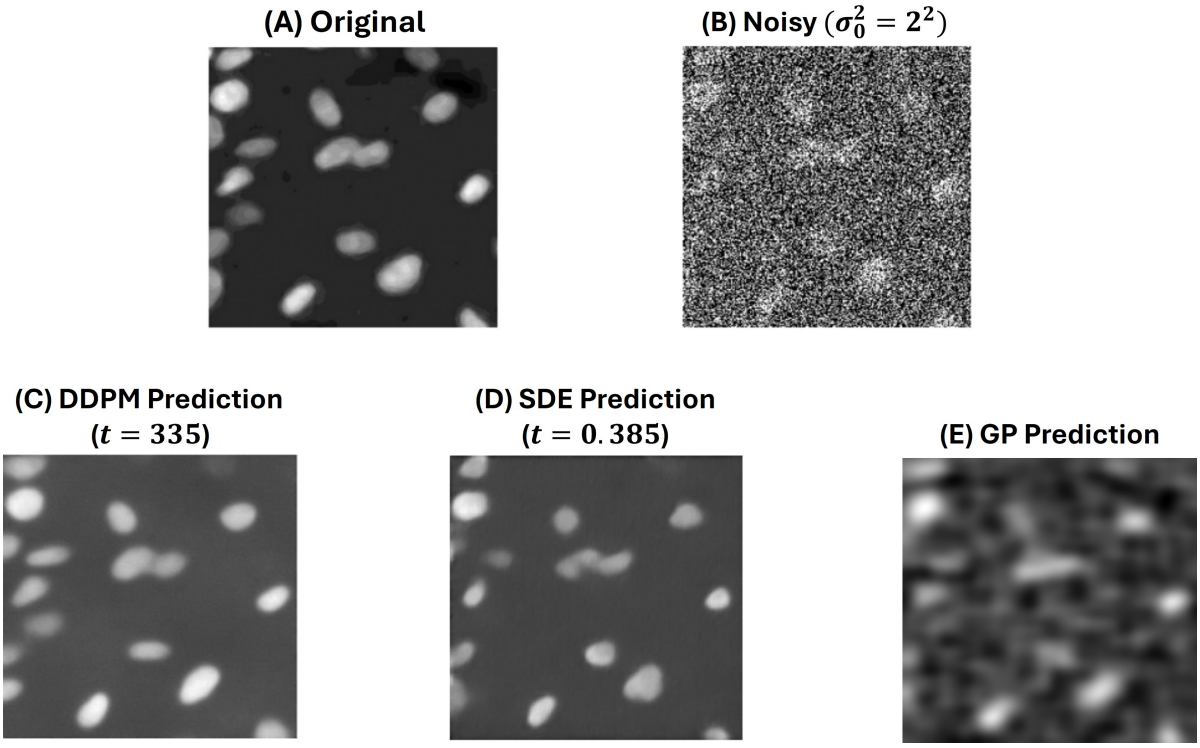


Figure 4.5: (A) Original image. (B) Gaussian noise with $\sigma_0 = 2$ is added to the original image and clipped to valid pixel value $[0, 1]$. (C) DDPM prediction for the noisy image. (D) score-based SDE model prediction for the noisy image. (E) GP prediction for the noisy image.

[160], and the results are given in Figure 4.6.

The SSIM between two images $\mathbf{y}'$ and $\tilde{\mathbf{y}}$ is defined as [160]:

$$\text{SSIM}(\mathbf{y}', \tilde{\mathbf{y}}) = \frac{(2\left(\bar{\mathbf{y}'} \times \bar{\tilde{\mathbf{y}}}\right) + C_1)(2 \times \text{Cov}(\mathbf{y}', \tilde{\mathbf{y}}) + C_2)}{(\bar{\mathbf{y}'}^2 + (\bar{\tilde{\mathbf{y}}})^2 + C_1)(\text{Var}(\mathbf{y}') + \text{Var}(\tilde{\mathbf{y}}) + C_2)}, \tag{4.27}$$

where $\bar{\mathbf{y}'}$, $\bar{\tilde{\mathbf{y}}}$ are the average pixel luminescence of images $\mathbf{y}'$ and $\tilde{\mathbf{y}}$, respectively, $\text{Var}(\mathbf{y}')$, $\text{Var}(\tilde{\mathbf{y}})$ are the pixel variances, and $\text{Cov}(\mathbf{y}', \tilde{\mathbf{y}})$ is the covariance of the pixel intensities between the two images. Constants C_1 and C_2 are added to prevent numerical instability when factors in the denominator approach zero. SSIM measures the quality of a modified image by comparing it with a clean reference image. SSIM values range from 0 to 1, with SSIM = 1 denoting perfect structural similarity to the original image, and SSIM = 0 indicating no similarity. This measure

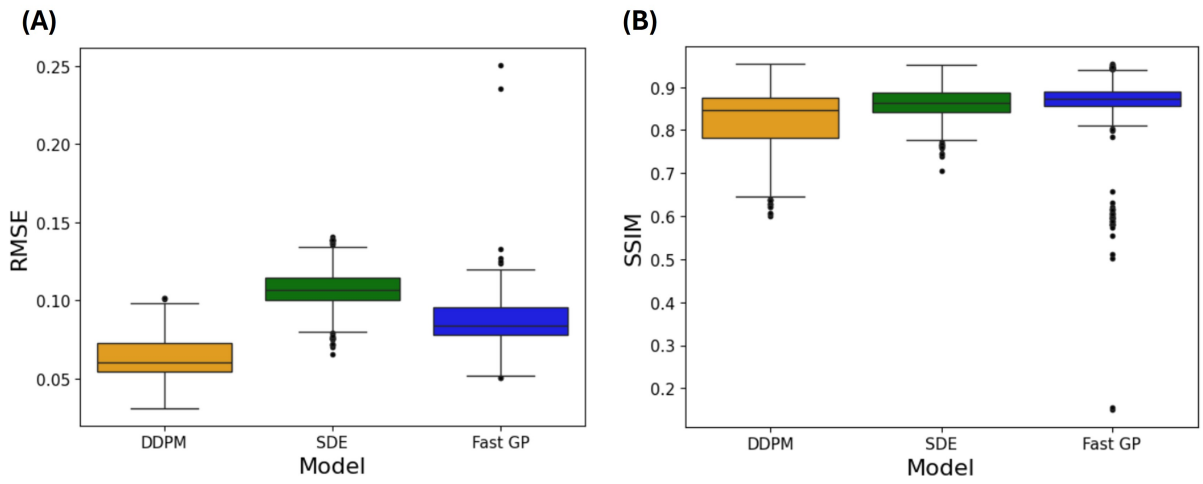


Figure 4.6: Boxplots for the (A) RMSE and (B) SSIM for using DDPM (orange), score-based SDE model (green), and fast GP (blue) to denoise 500 noisy test images with Gaussian noise variance $\sigma_0^2 = 0.3^2$ compared to the original image.

aligns more closely with human perception of image quality than RMSE because it incorporates features like luminescence, contrast, and structure rather than only pixel-wise reconstruction. All predicted pixel estimates are clipped to be within range $[0, 1]$ before computing evaluation metrics, as any values outside the valid intensity range may inflate error measurements.

Figure 4.6 presents box plots showing the distribution of RMSE and SSIM values for all 500 test images. The mean RMSEs for the denoised test images are approximately 0.064 ($\sigma_{\text{DDPM, RMSE}} = 0.014$) for DDPM predictions and approximately 0.11 ($\sigma_{\text{SDE, RMSE}} = 0.013$) for the score-based SDE model predictions. Denoising using fast GPs, as described in Chapter 2, is also included in the comparison and achieves an average RMSE of 0.086 ($\sigma_{\text{Fast GP, RMSE}} = 0.017$). The lower RMSE achieved by the DDPM indicates more accurate pixel-wise reconstruction. However, all three models perform similarly overall. The mean SSIMs for the denoised test images are approximately 0.822 ($\sigma_{\text{DDPM, SSIM}} = 0.075$) for DDPM predictions, approximately 0.864 ($\sigma_{\text{SDE, SSIM}} = 0.038$) for the score-based SDE model predictions, and approximately 0.857 ($\sigma_{\text{Fast GP, SSIM}} = 0.070$) for the Fast GP prediction.

The SSIM values indicate that all models generally preserve the structural content of the images during denoising. In particular, the GP model being well-suited for reducing noise, as it essentially provides a low-pass smoothing filter [161, 11], so denoising via GP results in an extremely high SSIM for many images. However, the variance of SSIM results is the largest, indicating that it may not result in a smoothed result in edge cases. In particular, some smoothed images have a middling to poor SSIM after denoising with GP, and the majority of these images had extremely weak contrast between the background and cell pixels. The object signal is lost even at this weaker noise level, and GP is unable to properly predict the object under these conditions. Alternatively, despite generative models sometimes hallucinating during the denoising process [159], which can cause small shifts in cell boundaries or differences in structure, the two generative models consistently achieved smoothed images extremely similar to the original input. The ability to "hallucinate" or complete predictions allows score-based models to effectively handle edge cases, as they are able to re-generate cell object boundaries when signal is poor like in the case of extreme noise.

Despite achieving lower RMSE, the DDPM exhibits a higher range and variance in SSIM compared to the score-based SDE model, as seen by the longer box plot depicted in Figure 4.6 (B). This wider SSIM distribution suggests increased variability in structure reconstruction. This behavior is consistent with the sensitivity of the DDPM reverse diffusion process to training duration [162], as this model was trained with fewer epochs due to computational time constraints (see Table 4.2). While pixel-wise intensity statistics tend to appear reasonable after limited training, accurate recovery of specific spatial structures may require longer training [162]. Furthermore, DDPM samples a different reverse noise trajectory for each image [143]. The stochastic reverse process, combined with the shorter training, may cause the spatial configuration of cells to differ even when average pixel intensities are consistent.

Overall, these metrics indicate that all three models are both able to accurately reconstruct noisy images and retrieve structurally similar outputs on average. All models demonstrate strong denoising capabilities suitable for practical microscopy applications. When data for a specific cell line or experimental condition is limited, DDPM demonstrates particularly strong generative abilities with a relatively short training schedule. Furthermore, the generative models outperform GP for smoothing and reconstructing images when the original image is corrupted with extreme noise or when the contrast between the cell and background pixels is weak. The generative models both have strong denoising capabilities with reasonable amounts of noise, yielding low RMSEs and high SSIMs on average when denoising images with moderate noise degradation.

Chapter 5

Conclusion and future work

In Chapter 2, we have presented a scalable cell image segmentation method for both the cell nucleus and cytoplasm. By leveraging fast Gaussian processes, data-driven thresholding, and the watershed operations, we show that the process is not only more automated, but the results are more precise than other unsupervised segmentation methods, particularly ImageJ. A key strength of the proposed segmentation algorithm is that it does not rely on parameter tuning or labeled training datasets. The results of our unsupervised method demonstrate versatility across different cell channels, making it an appealing option for more general segmentation tasks beyond cell segmentation.

In Chapter 3, the `R` package FastSegmentation for implementing the unsupervised segmentation by fast GP workflow, as covered in Chapter 2, is introduced. This chapter provides a summary of the package structure, functions, and relevant examples. By compiling the segmentation pipeline into a single `R` function with the only user-defined parameter being the file path, the unsupervised segmentation method is easily accessible to researchers across multiple domains and regardless of programming background. Furthermore, this package can be used to denoise cell images via fast GP, and the gradients of the predictive mean yield informa-

tion on fluctuations in pixel intensity across an image, which can be informative for detecting approximate object edge locations.

In Chapter 4, DDPM and score-based generative models are reviewed, and the mathematical and algorithmic framework is detailed. The models have first been validated by analyzing the generated data after training each with sampled data from a known GMM. Then, each model was trained with real-world cell microscopy images, and each was able to successfully generate realistic images and denoise corrupted test images. Both generative models were compared against GP for denoising tasks. Although all three models were on average able to accurately denoise images with weak to moderate noise conditions, the generative models were able to recover weak signal in edge cases where contrast between the cell object and background was low, whereas GP struggled. Furthermore, the generative models exhibited a strong performance in denoising completely corrupted images, as these models are able to generate boundaries to complete the image. Alternatively, GP was completely unable to recover the original input when corrupted with high noise.

5.1 Future work and extensions

5.1.1 Database of cell characteristics across experimental conditions

The ultimate goal of any cell image segmentation algorithm is to collect information on density, angle, and morphology for downstream analysis to better understand biological mechanisms [163, 164, 165]. We have prepared an initial database across 30 microscopy videos and 54 individual image frames and calculated cell densities, cell counts, and cell-cell order parameters describing how well neighboring cells align with each other. The cell angles are calculated from

the segmented cell boundaries, and the cell-cell order parameter S_{co} is determined by:

$$S_{co} = \frac{1}{n_c} \sum_{w=1}^{n_c} 2 * \cos\left(\frac{\theta_w}{180 - \pi}\right)^2 - 1, \tag{5.1}$$

where n_c is the number of cells in an image, and θ_w is the angle of cell w. The experiments represent a wide range of cell types, substrate types, seeding densities, and experimental durations, and all images and videos were preliminarily segmented using ImageJ.

Exploratory analysis provides initial evidence of an association between cell alignment and experimental conditions. A health index parameter was calculated to describe cell proliferation rate:

$$\text{Health Index} = \frac{\rho_{t_s}}{\rho_0}/t_s, \tag{5.2}$$

where t_s is the number of days since seeding, ρ_0 is the seeding density, and ρ_{t_s} is the cell density calculated for the frame corresponding to time t_s. To assess whether the health index can predict cell alignment, leave-one-out cross-validation was performed using `RobustGaSP` [53]. For the video data, final frame cell alignment was regressed against health index and other explanatory variables, and the exploratory `RobustGaSP` models typically achieved lower mean absolute errors (MAE) than the benchmark models. The same procedure was performed for the snapshot data separately with similar results. To further verify the association between health index and cell alignment, out-of-sample tests were performed for each model using a uniform sequence spanning the range of health index values in the original dataset as input. In the best performing models, the GP predictions exhibited a largely positive, albeit non-linear, trend between the input health indices and the predicted outputs, consistent with the hypothesis that a higher health index is associated with higher alignment.

5.1.2 Cell object tracking of GP-based segmentation results with the linear assignment problem

Modern microscopes provide time-lapse videos for tracking changes in cellular behavior over time, typically presented as a sequence of images corresponding to specified time points during the image acquisition process. A future goal is to integrate our segmentation methods with particle tracking and linking algorithms that solve a linear assignment problem [86], similar to that of TrackMate [9]. Due to the unsupervised nature of this algorithm, segmentation can be run in batch for a time-lapse cell video sequence, with unique parameter estimation for smoothing and thresholding on each frame.

The cell mask outputs allow us to determine the exact coordinates of each cell, so information like cell location, proximity of surrounding cells, or division can be detected to construct the LAP objective function $\hat{A}$. For example, the cell centroid can be determined by taking the median $x-$ and $y-$coordinates of the cell mask, and the distance between two cells on consecutive frames can be the Euclidean distance between the centroids. Beyond simple centroid calculation, having complete access to all coordinates within a cell mask allows for more in-depth cell morphology calculations. Whereas the LoG detector, or many other traditional unsupervised methods, typically use the spot radius to calculate area, the area of each cell mask in the proposed unsupervised segmentation method can instead be the number of pixels. Furthermore, the intensity profile within each cell can be easily quantified by calculating the mean or median pixel intensity value within each region of interest. The standard deviation of pixel intensity provides information on whether a cell has a consistent phenotype or if there are variations in the cell surface.

5.1.3 GP-based segmentation for nucleation images

As the cell segmentation approach in Chapter 2 does not restrict the shape of the object, the proposed approach can be extended for image segmentation with objects undergoing dynamical changes, including microscopy images for the nucleation and growth process in crystallization [166] and protein dynamics [167]. Image objects in these settings exhibit irregular morphologies that typically vary significantly with time [168], so object detection and segmentation can be significantly more challenging. Traditional segmentation techniques that rely on assumptions about object geometry, such as ImageJ [6] assuming cell objects are generally elliptical, therefore may be unable to adequately collect boundary information, but the flexibility of the proposed GP-based segmentation technique allows it to naturally adapt to new object types and environments.

In crystallization, nucleation is the process in which small, compact atom clusters grow and evolve into larger, irregularly shaped crystal structures [169]. Detecting and segmenting objects at each phase of growth can yield insights on nucleation kinetics and growth rate. The data-driven thresholding step is particularly well-suited for this task, as the intensity distributions of nucleation clusters could differ substantially across time [166], and the GP smoothing function can be applied independently to each frame to perform noise reduction. Finally, the nuclei of the crystal objects are relatively similar in shape to cell nuclei, so the algorithm should be easily adapted for this type of object.

For protein dynamics, objects of interest may undergo continuous structural reorganization across time [170], so shape complexity can be similarly difficult for traditional segmentation techniques. The pixel-level object mask outputs of the proposed GP-based method allow for detailed morphological quantification, including measurements about object area and pixel intensity profiles. These details can supplement a diverse set of downstream analyses for

characterizing the relationship between protein morphology and dynamic behavior.

5.1.4 Time-conditional generative models for sequential cell microscopy images

The diffusion models trained in Chapter 4 treat each cell image independently and temporal information from the time-lapse microscopy video sequence is not incorporated. A natural extension of this analysis is to condition the generative model on the frame number or preceding frames, enabling the generation of future frames in an image sequence. This model could be used to predict how a cell population behaves and evolves between time points, which could be useful for when frame acquisition is sparse or when frames are corrupted or missing. Furthermore, this type of model could be used to reduce the time required for cell experiments, allowing for shorter imaging times in favor of generating future frames after training.

One approach is to directly incorporate frame conditioning into the noise network $\boldsymbol{\zeta}_\theta$ either by concatenating the frame index or projecting a temporal embedding at each diffusion step, similar to how class labels are incorporated in conditional DDPMs [171, 172]. Alternatively, generation of the next frame could be directly conditioned on the current frame, as the model will learn to condition on the present state of the cell dynamics. This process would closely follow that of the video diffusion model framework [173], as this model framework operates over sequences of frames rather than individual images. Conditioning future generation on the current frame would help generate more consistent cell coordinates and counts, as both of these metrics vary greatly for each object across time.

5.1.5 Flow matching models for cell microscopy image generation

Both diffusion models introduced in Chapter 4 simulate a multi-step Markov chain, which can be computationally expensive when training on or generating large batches of images or high-resolution images. Flow matching [174] has been introduced as an alternative data generation method that instead learns a time-dependent target vector field that transports samples from a source distribution to a learned data distribution via a single continuous ODE. Flow matching has been shown to outperform other diffusion methods.

A natural expansion of the generative modeling analysis is to train a flow matching model on the same cell microscopy image dataset for a direct comparison of sample generational quality and denoising efficiency. Flow matching models can also be trained on conditional data, where inputs are conditioned, or coupled, with another source of information [175, 176, 177]. Therefore, flow matching methods directly connect to the previous extension for generating time-conditional images, and an image could be attached to the image at the previous time point during training. Furthermore, image data could also be connected to the detected cell mask matrix, such as the output by the GP-based segmentation method in Chapter 2, which could encourage cell objects to be generated within known boundaries.

Appendix A

Appendix of chapter 2

A.1 Proof for Lemma 1

Proof. We first prove part 1 of the Lemma 1. The eigendecompositions is given as $\mathbf{R}_l = \mathbf{U}_l\mathbf{\Lambda}_l\mathbf{U}_l^T$, where $\mathbf{U}_l \in \mathbb{R}^{n_l \times n_l}$ for $l = 1, 2$. Then, the Kronecker product $\tilde{\mathbf{R}} = \mathbf{R}_2 \otimes \mathbf{R}_1 + \eta\mathbf{I}_N$ can be written as,

$$\begin{aligned}
\tilde{\mathbf{R}} =& \mathbf{U}_2\mathbf{\Lambda}_2\mathbf{U}_2^T \otimes \mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1^T + \eta\mathbf{I}_N \\
=& [(\mathbf{U}_2\mathbf{\Lambda}_2) \otimes (\mathbf{U}_1\mathbf{\Lambda}_1)](\mathbf{U}_2^T \otimes \mathbf{U}_1^T) + \eta\mathbf{I}_N \\
=& (\mathbf{U}_2 \otimes \mathbf{U}_1)(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)(\mathbf{U}_2^T \otimes \mathbf{U}_1^T) + \eta\mathbf{I}_N \\
=& \mathbf{Q}\tilde{\mathbf{\Lambda}}\mathbf{Q}^T,
\end{aligned}$$

where $\mathbf{Q} = \mathbf{U}_2 \otimes \mathbf{U}_1$ is an orthogonal matrix, and $\tilde{\mathbf{\Lambda}} = (\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1 + \eta\mathbf{I}_N)$ is a diagonal matrix.

Thus,

$$| \tilde{\mathbf{R}} | = \prod_{i=1}^{n_1}\prod_{j=1}^{n_2}(\lambda_{i,1}\lambda_{j,2} + \eta). \tag{A.1}$$

Denote $\mathbf{1}_N = \mathbf{1}_{n_2} \otimes \mathbf{1}_{n_1}$. Then,

$$\hat{\mu} = \frac{\mathbf{1}_N^T\tilde{\mathbf{R}}^{-1}\mathbf{y}_v}{\mathbf{1}_N^T\tilde{\mathbf{R}}^{-1}\mathbf{1}_N}. \tag{A.2}$$

The numerator of Equation (A.2) can be written as

$$\begin{aligned}
&\mathbf{1}_N^T\tilde{\mathbf{R}}^{-1}\mathbf{y}_v \\
=&\mathbf{1}_N^T(\mathbf{Q}\tilde{\mathbf{\Lambda}}\mathbf{Q}^T)^{-1}\mathbf{y}_v \\
=&\mathbf{1}_N^T(\mathbf{U}_2\otimes\mathbf{U}_1)\tilde{\mathbf{\Lambda}}^{-1}(\mathbf{U}_2\otimes\mathbf{U}_1)^T\mathrm{Vec}(\mathbf{Y}) \\
=&[(\mathbf{1}_{n_2}^T\mathbf{U}_2)\otimes(\mathbf{1}_{n_1}^T\mathbf{U}_1)]\tilde{\mathbf{\Lambda}}^{-1}(\mathbf{U}_2^T\otimes\mathbf{U}_1^T)\mathrm{Vec}(\mathbf{Y}) \\
=&(\tilde{\mathbf{u}}_2^T\otimes\tilde{\mathbf{u}}_1^T)\tilde{\mathbf{\Lambda}}^{-1}\mathrm{Vec}(\mathbf{U}_1^T\mathbf{Y}\mathbf{U}_2) \\
=&\sum_{j=1}^{n_2}\sum_{i=1}^{n_1}\frac{\tilde{u}_{i,1}\tilde{Y}_{i,j,0}\tilde{u}_{j,2}}{\lambda_{i,1}\lambda_{j,2}+\eta}.
\end{aligned}$$

where $\tilde{Y}_{i,j,0}$ is the (i,j)-th entry of the $n_1\times n_2$ matrix $\mathbf{U}_1^T\mathbf{Y}\mathbf{U}_2$, $\tilde{u}_{i,1}$ is the i-th term of $\tilde{\mathbf{u}}_1 = \mathbf{U}_1^T\mathbf{1}_{n_1}$, and $\tilde{u}_{j,2}$ is the j-th term of $\tilde{\mathbf{u}}_2 = \mathbf{U}_2^T\mathbf{1}_{n_2}$, for $l = 1, 2$, $i = 1, ..., n_1$ and $j = 1, ..., n_2$.

Similarly, the denominator of Equation (A.2) can be written as

$$\begin{aligned}
&\mathbf{1}_N^T\tilde{\mathbf{R}}^{-1}\mathbf{1}_N \\
=&\mathbf{1}_N^T(\mathbf{U}_2\otimes\mathbf{U}_1)\tilde{\mathbf{\Lambda}}^{-1}(\mathbf{U}_2^T\otimes\mathbf{U}_1^T)\mathbf{1}_N \\
=&[(\mathbf{1}_{n_2}^T\mathbf{U}_2)\otimes(\mathbf{1}_{n_1}^T\mathbf{U}_1)]\tilde{\mathbf{\Lambda}}^{-1}[(\mathbf{U}_2^T\mathbf{1}_{n_2})\otimes(\mathbf{U}_1^T\mathbf{1}_{n_1})] \\
=&(\tilde{\mathbf{u}}_2^T\otimes\tilde{\mathbf{u}}_1^T)\tilde{\mathbf{\Lambda}}^{-1}(\tilde{\mathbf{u}}_2\otimes\tilde{\mathbf{u}}_1) \\
=&\sum_{j=1}^{n_2}\sum_{i=1}^{n_1}\frac{\tilde{u}_{i,1}^2\tilde{u}_{j,2}^2}{\lambda_{i,1}\lambda_{j,2}+\eta}.
\end{aligned}$$

The term S^2 in the log likelihood function in Equation (2.3) equals:

$$\begin{aligned}
S^2 =&(\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)^T\tilde{\mathbf{R}}^{-1}(\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)\\
=&(\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)^T\mathbf{Q}\tilde{\mathbf{\Lambda}}^{-1}\mathbf{Q}^T(\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)\\
=&[\mathbf{Q}^T(\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)]^T\tilde{\mathbf{\Lambda}}^{-1}\mathbf{Q}^T(\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)\\
=&[(\mathbf{U}_2^T \otimes \mathbf{U}_1^T)\mathrm{Vec}(\mathbf{Y} - \hat{\mu}\mathbf{1}_{n_1\times n_2})]^T\\
&\tilde{\mathbf{\Lambda}}^{-1} \times (\mathbf{U}_2^T \otimes \mathbf{U}_1^T)\mathrm{Vec}(\mathbf{Y} - \hat{\mu}\mathbf{1}_{n_1\times n_2})\\
=&[\mathrm{Vec}(\mathbf{U}_1^T(\mathbf{Y} - \hat{\mu}\mathbf{1}_{n_1\times n_2})\mathbf{U}_2)]^T \qquad \text{(A.3)}\\
&\tilde{\mathbf{\Lambda}}^{-1}\mathrm{Vec}(\mathbf{U}_1^T(\mathbf{Y} - \hat{\mu}\mathbf{1}_{n_1\times n_2})\mathbf{U}_2)\\
=&\tilde{\mathbf{y}}_v^T\tilde{\mathbf{\Lambda}}^{-1}\tilde{\mathbf{y}}_v\\
=&\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\frac{\tilde{Y}_{i,j}^2}{\lambda_{i,1}\lambda_{j,2}+\eta}, \qquad \text{(A.4)}
\end{aligned}$$

where $\tilde{\mathbf{y}}_v = \mathrm{Vec}(\mathbf{U}_1^T(\mathbf{Y} - \hat{\mu}\mathbf{1}_{n_1\times n_2})\mathbf{U}_2)$. By Equations (2.3), (A.1) and (A.4), we have Equation (2.8).

For part 2 of the lemma, we first calculate the predictive mean vector below:

$$\begin{aligned}
\mathbf{f}_v^* =&\hat{\mu}\mathbf{1}_N + \mathbf{R}\tilde{\mathbf{R}}^{-1}(\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)\\
=&\hat{\mu}\mathbf{1}_N + (\mathbf{U}_2 \otimes \mathbf{U}_1)(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)(\mathbf{U}_2^T \otimes \mathbf{U}_1^T)\times\\
&(\mathbf{U}_2 \otimes \mathbf{U}_1)\tilde{\mathbf{\Lambda}}^{-1}(\mathbf{U}_2^T \otimes \mathbf{U}_1^T)(\mathbf{y}_v - \hat{\mu}\mathbf{1}_N)\\
=&\hat{\mu}\mathbf{1}_N + (\mathbf{U}_2 \otimes \mathbf{U}_1)[(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)\tilde{\mathbf{\Lambda}}^{-1}]\times\\
&\mathrm{Vec}(\mathbf{U}_1^T(\mathbf{Y} - \hat{\mu}\mathbf{1}_{n_1\times n_2})\mathbf{U}_2)\\
=&\hat{\mu}\mathbf{1}_N + (\mathbf{U}_2 \otimes \mathbf{U}_1)[(\mathbf{\Lambda}_2 \otimes \mathbf{\Lambda}_1)\tilde{\mathbf{\Lambda}}^{-1}]\tilde{\mathbf{y}}_v\\
=&\hat{\mu}\mathbf{1}_N + (\mathbf{U}_2 \otimes \mathbf{U}_1)\mathrm{Vec}\left(\mathbf{\Lambda}_1\tilde{\mathbf{Y}}_0\mathbf{\Lambda}_2\right)\\
=&\hat{\mu}\mathbf{1}_N + \mathrm{Vec}\left(\mathbf{U}_1\mathbf{\Lambda}_1\tilde{\mathbf{Y}}_0\mathbf{\Lambda}_2\mathbf{U}_2^T\right),
\end{aligned}$$

where $\mathrm{Vec}(\tilde{\mathbf{Y}}_0) = \tilde{\mathbf{\Lambda}}^{-1}\tilde{\mathbf{y}}_v$.

To compute the predictive variance, the term $\mathbf{R}\tilde{\mathbf{R}}^{-1}\mathbf{R}$ equals:

$$\begin{aligned}\mathbf{R}\tilde{\mathbf{R}}^{-1}\mathbf{R} =&(\mathbf{R}_2 \otimes \mathbf{R}_1)(\mathbf{U}_2 \otimes \mathbf{U}_1)\tilde{\mathbf{\Lambda}}^{-1}(\mathbf{U}_2^T \otimes \mathbf{U}_1^T)\times \\ &(\mathbf{R}_2 \otimes \mathbf{R}_1) \\ =&(\mathbf{R}_2\mathbf{U}_2 \otimes \mathbf{R}_1\mathbf{U}_1)\tilde{\mathbf{\Lambda}}^{-1}(\mathbf{U}_2^T\mathbf{R}_2 \otimes \mathbf{U}_1^T\mathbf{R}_1)\end{aligned}$$

The $t = i + (j-1)n_1$ diagonal term of $\hat{\sigma}^2(\mathbf{R} - \mathbf{R}_2 \otimes \mathbf{R}_1)$ is the predictive variance at pixel location (i, j), which follows

$$\begin{aligned}\hat{\sigma}^2 c^*_{i,j} =&\hat{\sigma}^2 - \hat{\sigma}^2(\mathbf{r}_{j,2}^T\mathbf{U}_2 \otimes \mathbf{r}_{i,1}^T\mathbf{U}_1)\times \\ &\tilde{\mathbf{\Lambda}}^{-1}(\mathbf{U}_2^T\mathbf{r}_{j,2} \otimes \mathbf{U}_1^T\mathbf{r}_{i,1}) \\ =&\hat{\sigma}^2\left(1 - \sum_{j'=1}^{n_2}\sum_{i'=1}^{n_1}\frac{\tilde{r}^2_{i,i',1}\tilde{r}^2_{j,j',2}}{\lambda_{i,1}\lambda_{j,2} + \eta}\right),\end{aligned}$$

where $\tilde{r}_{i,i',1}$ is the i'th term of the vector $\mathbf{r}_{i,1}^T\mathbf{U}_1$ and $\tilde{r}_{j,j',2}$ is the j'th term of the vector $\mathbf{r}_{j,2}^T\mathbf{U}_2$, for $i' = 1, ..., n_1$ and $j' = 1, ..., n_2$, with $\mathbf{r}_{i,1}^T = (K_1(x_{i,1}, x_{1,1}), ..., K_1(x_{i,1}, x_{n_1,1}))^T$ and $\mathbf{r}_{j,2}^T = (K_2(x_{j,2}, x_{1,2}), ..., K_1(x_{j,2}, x_{n_2,2}))^T$.

□

A.2 Segmentation without GP smoothing

GP regression is used in two steps for image segmentation. The first step is to use a separable GP to denoise the image discussed in Section 2.3.1, and the second step is to smooth the difference of pixel counts satisfying a sequence of threshold values for generating binary cell masks discussed in Section 2.3.2. In order to demonstrate how incorporating GP into the thresholding workflow improves results, the numerical analysis outlined in Section 2.5 was repeated to compare the GP results with two approaches. The first approach removes GP regression in both steps, and the results are compared in Figure A.1. The second method only

removes GP regression for denosing the image as discussed in Section 2.3.1, and the results are compared in Figure A.2.

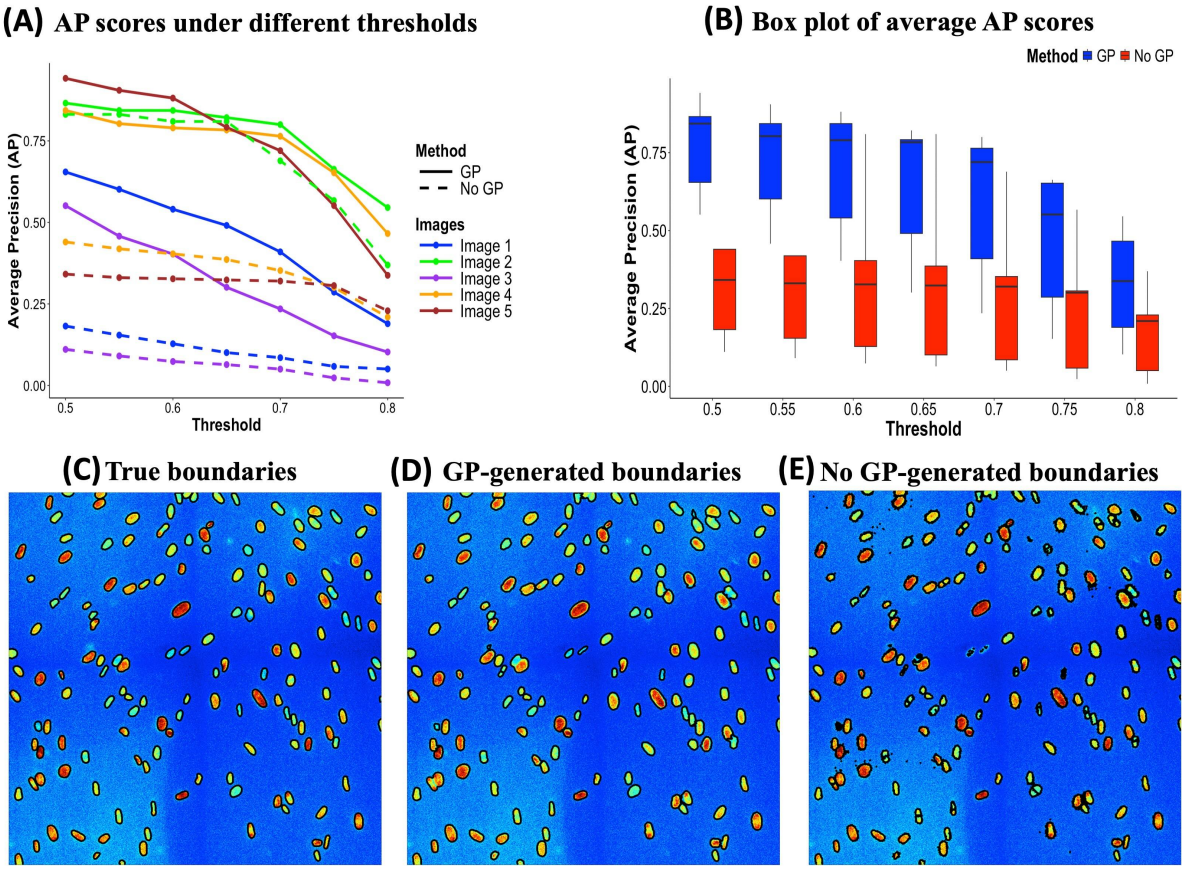


Figure A.1: Results for segmenting cell nuclei using the proposed GP method, and the method without GPs for image noise reduction in Section 2.3.1 and for smoothing the difference in pixel counts in Section 2.3.2. (A) AP scores for GP-based segmentation and segmentation method with GP models removed across different thresholds. (B) Boxplots of AP scores for each method. (C) True boundaries of the fifth test image with 1024×1024 pixels. (D) Boundaries generated by the GP-based method for the fifth test image. (E) Boundaries generated using the segmentation method with GP models removed for the fifth test image.

Panel (A) of Figure A.1 shows the AP with different thresholds by the two methods for five images of cell nuclei. The solid line representing the AP for the GP-based method is again almost always above the dashed line representing the AP of the segmentation results with GPs in two steps, with the same color for all thresholds, meaning that the segmentation

method with GPs consistently achieves higher AP scores than the segmentation without GPs. The two segmentation methods only perform similarly for one image (Test Image 2). Panel (B) in Figure A.1 shows the distribution of these AP scores from segmentation methods with and without GP at each threshold. Similar to the results in panel (A), the GP-based method achieves higher median AP scores across all thresholds. The true annotated boundaries, those generated by the GP-based method, and those generated by the segmentation method without GP of the fifth test image are plotted in Figure A.1 (C)-(E). The segmentation method without GP tends to over-segment cells, resulting in many cells being split in half.

This comparison is repeated for the segmentation method where only the GP used for image smoothing described in Section 2.2 is omitted, and the results are displayed in Figure A.2. Panel (A) of Figure A.2 demonstrates similar results to the previous comparison, and once again, the two segmentation methods only perform similarly for one image. For all other images, the GP-based segmentation results yield higher APs for each threshold. Panel (B) in A.2 shows the distribution of the APs of each segmentation method, and the GP-based method consistently achieves higher median AP scores across all thresholds. Finally, the true annotated boundaries, those generated by the GP-based method, and those generated by the segmentation method without the GP for image smoothing of the fifth test image are plotted in Figure A.2 (C)-(E). In Panel (E), many areas of the image have an extremely high concentration of erroneously detected boundaries. These results indicate the GPs used in both of the two steps are crucial for the segmentation method.

A.3 Sensitivity analysis of threshold α_k^*

As per Equation 2.11 in the main text, the optimal threshold α_k^* is set when the smoothed differences in foreground pixel counts $\Delta c_k^*(\alpha_m)$ stabilize, and the absolute difference

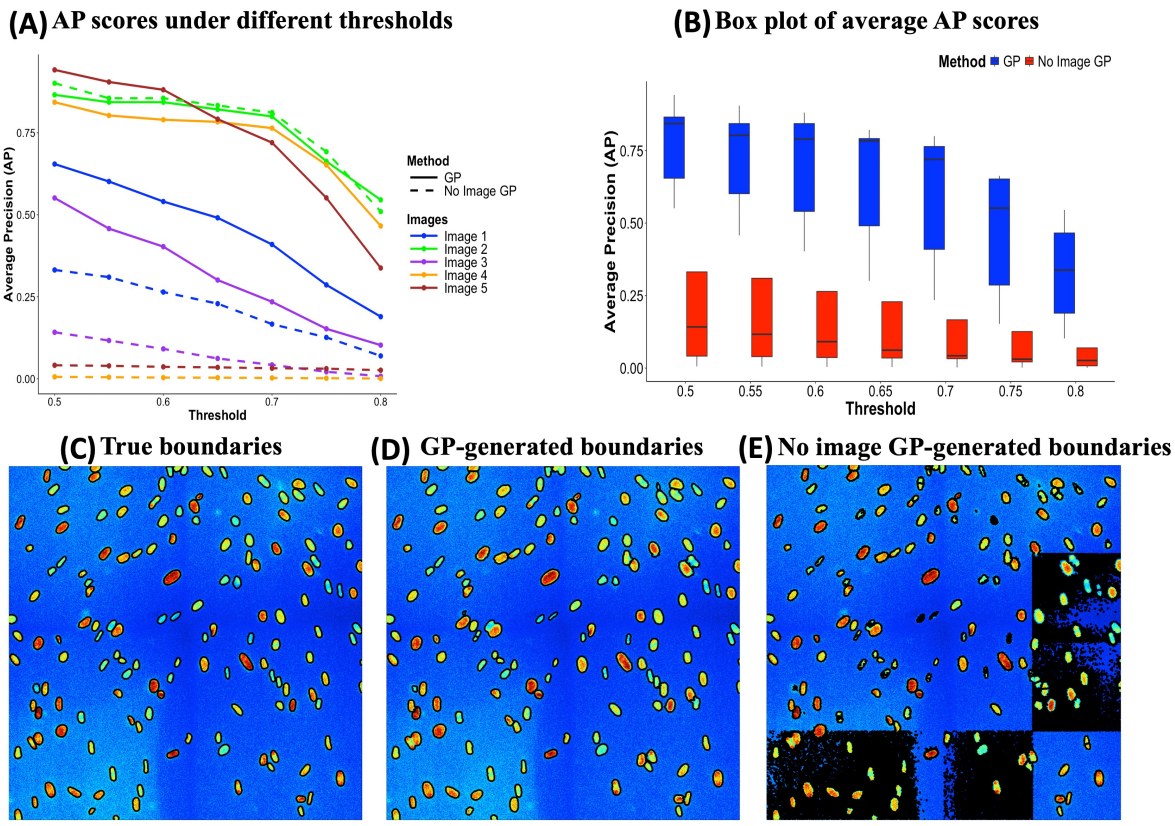


Figure A.2: Results for segmenting cell nuclei using the proposed GP method, and the method without GPs for image noise reduction. (A) AP scores for GP-based segmentation and the segmentation method with GP image smoothing removed across different thresholds. (B) Boxplots of AP scores for each method. (C) True boundaries of the fifth test image with 1024×1024 pixels. (D) Boundaries generated by the GP-based method for the fifth test image. (E) Boundaries generated using the segmentation method with GP image smoothing removed for the fifth test image.

of differences $\Delta c_k^*(\alpha_{m-1}) - \Delta c_k^*(\alpha_m)$ fall below $0.05\tau_k$, as described in Section 2.3.2. The cutoff $0.05\tau_k$ was determined empirically, as multiplying τ_k by a factor of 0.05 tends to retrieve binary images that preserve the true cell shape while still maintaining enough image information (see Figure A.3).

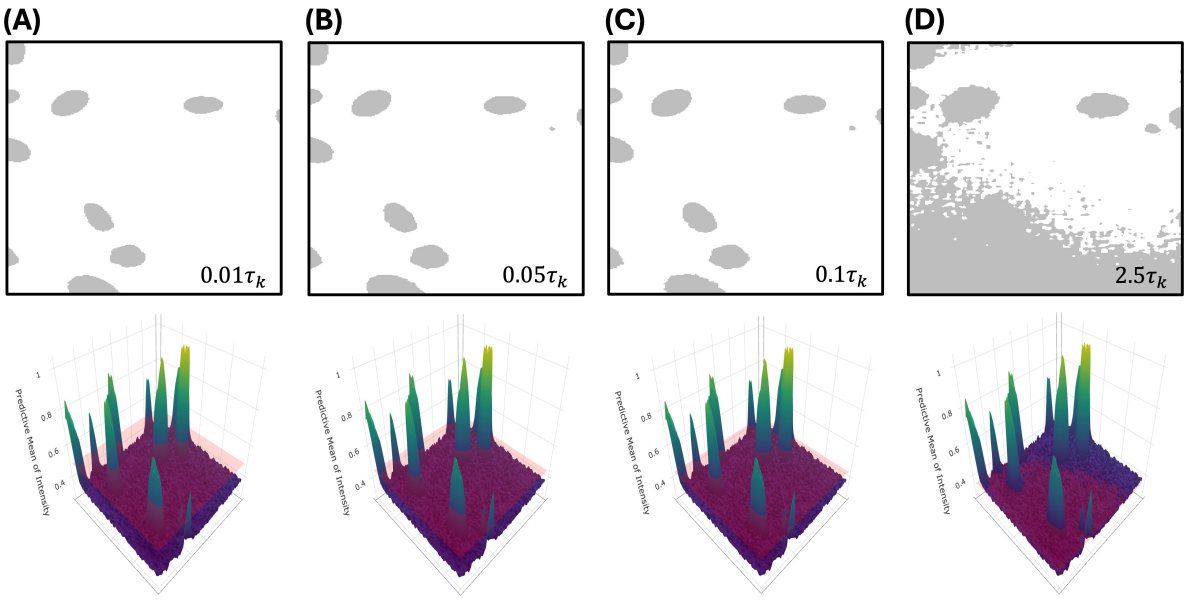


Figure A.3: Empirical sensitivity analysis for α_k^* cutoff. (A) When the α_k^* cutoff is set to $0.01\tau_k$, the cell mask information does not describe the entire shape, as the threshold lies well above the base of the cell (see bottom row). (B) The α_k^* cutoff is set to $0.05\tau_k$ as described in the study methods. This cutoff retrieves binary masks that are a strong representation of the true cell shape. (C) When the α_k^* cutoff is set to $0.1\tau_k$, the cell masks are still somewhat representative of the true cell shape. However, the outlines become slightly more jagged and irregular, particularly in the bottom cells in the binary mask output. (D) When the α_k^* cutoff is set to $2.5\tau_k$, noise distorts the binary mask output, indicating that a larger cutoff is not effective.

A.4 Re-thresholding noisy sub-images with abnormally high object counts

This section discusses a rare scenario where a sub-image has large noise that results in abnormally high object counts compared to other sub-images in the thresholding step discussed in Section 2.3.2. This scenario only happened once for all the sub-images in this study, and it can be easily avoided by applying the average threshold calculated from the other sub-images introduced below.

After a threshold is set for each sub-image to get the binary sub-images as described

in the main text in Section 2.3.2, each object group is separated and detected via flood fill [117] discussed in Section A.5. The flood fill operation assigns a unique label to each group of connecting pixels in a binary image, including both the background and foreground pixels, so the foreground object count is equal to the number of labels excluding the background label. Most binary sub-images have an expected object count by inequality (A.5) and are re-incorporated into the segmentation workflow as described in the top row of Figure A.4. However, one sub-image in this study had a higher-than-expected object count due to extreme noise, which is an indicator that the detected threshold by the criterion outlined in Section 2.3.2 is not optimal for that binary sub-image. To control for similar edge cases, we can use information from other sub-images in order to re-threshold the outlier image.

For a cell image made up of K sub-images, let each binary sub-image $\mathbf{B}_k$, $k \in \{1, 2, ..., K\}$, calculated from the predictive mean sub-images have object count S_k. Each binary sub-image was generated by each optimal threshold α^* set by the criterion. The outlier object counts are determined by the following inequality:

$$| S_k - \overline{S} | > 2 \times \sigma_S, \tag{A.5}$$

where $\overline{S}$ is the mean object count of $\{S_1, S_2, ..., S_K\}$, and σ_S is the standard deviation of the object counts. For each binary sub-image $\mathbf{B}_k$ where this inequality is true, the binary threshold of the sub-image predictive mean $\mathbf{F}_k^*$ is reset to the mean threshold $\overline{\alpha}$ (excluding the thresholds of the outlier images), and the binary sub-image $\mathbf{B}_k$ is updated. An example of this process for the edge case is given in the bottom row of Figure A.4. Then, returning to the original workflow, the binary sub-images are restitched to form a single binary image representing the original cell image.

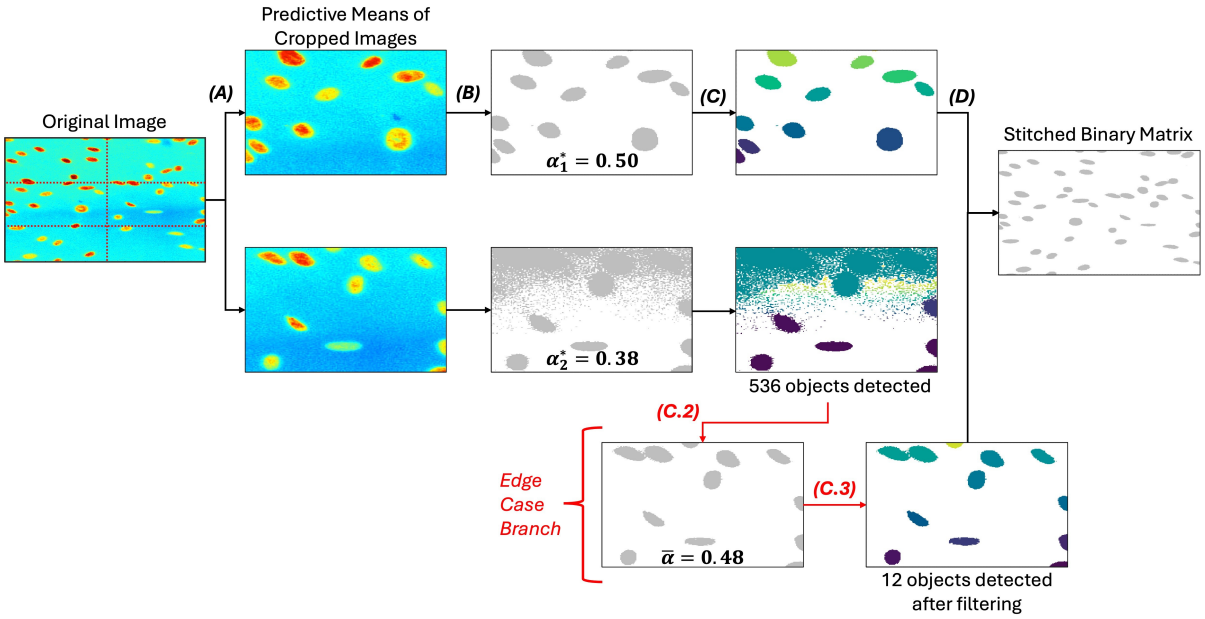


Figure A.4: Workflow of determining the true optimal threshold for selected sub-images. (A) The original image is cropped into six equal parts, and the predictive means are generated via GPs. (B) The initial thresholds α_k^* for the predictive means of two cropped images are determined using the criterion described in Section 2.3.2. (C) The number of objects in the binary sub-images is detected by flood fill. In the bottom row, despite there only being 12 cells, image noise coerces random pixel groups to be detected as foreground, leading to over 500 objects being detected in the binary sub-image. This rare scenario only happened once. (C.2) By inequality in (A.5), this sub-image is determined to have an unusually high object count. The sub-image threshold is reset to the average of all non-outlier sub-image thresholds. (C.3) Twelve objects are detected after watershed and filtering based on the cell sizes discussed in Section 2.3.3. (D) The binary sub-images are stitched, maintaining the spatial continuity of the original image.

A.5 Flood fill algorithm

Distinct objects in a binary image are visually separable, as each cell object is represented with foreground pixels. However, all cell pixels in the binary image are represented by $(x_{i,1}, x_{j,2}) = 1$. To separate these pixel clusters, connecting foreground pixels can be relabeled using the flood fill operation by using the `bwlabel()` function from EBImage package [118]. The flood fill operation is a polygon-filling algorithm that identifies and fills regions of connected pixels with a new specified value. The algorithm checks the adjacent pixels in four directions

(up, down, left, and right) of the starting pixel for matching values. After specifying a starting point, the flood fill algorithm groups the connected pixels of the same value with the same label [117]. This process is repeated until all foreground pixel clusters have a unique identification number.

A.6 Additional figures for watershed simulation

Figure A.5 provides the binary image and negative distance map of an example of two connecting cells utilized in the watershed algorithm introduced in Section 2.3.3. The distance map gives the distance between each pixel and the closest background pixel in the binary image [178]. The negative distance map $-\mathbf{D}$ is the negative of the distance map calculated from the binary image $\mathbf{B}^*$ [118]:

$$-\mathbf{D}(\mathbf{x}_{i,j}) = \begin{cases} -\min\limits_{\{\mathbf{x}_{i',j'}:\mathbf{B}^*(\mathbf{x}_{i',j'})=0\}} \|\mathbf{x}_{i,j} - \mathbf{x}_{i',j'}\|_2, & \text{if } \mathbf{B}^*(\mathbf{x}_{i,j}) = 1, \\ 0, & \text{if } \mathbf{B}^*(\mathbf{x}_{i,j}) = 0. \end{cases}$$

The watershed algorithm is applied to the negative distance map to segment cell objects [118].

A.7 Additional results of simulated studies

Figure A.6 illustrates the ground truth mean, noisy observations, and the predictive mean by fast GPs with the separable Matérn kernel in Equation (1.11) for the Branin function and linear diffusion equation described in Example 1 in Chapter 2. Panels (C) and (F) demonstrate that GPs with separable kernels effectively denoise the lattice data and provide accurate estimates of the mean, even with high levels of noise.

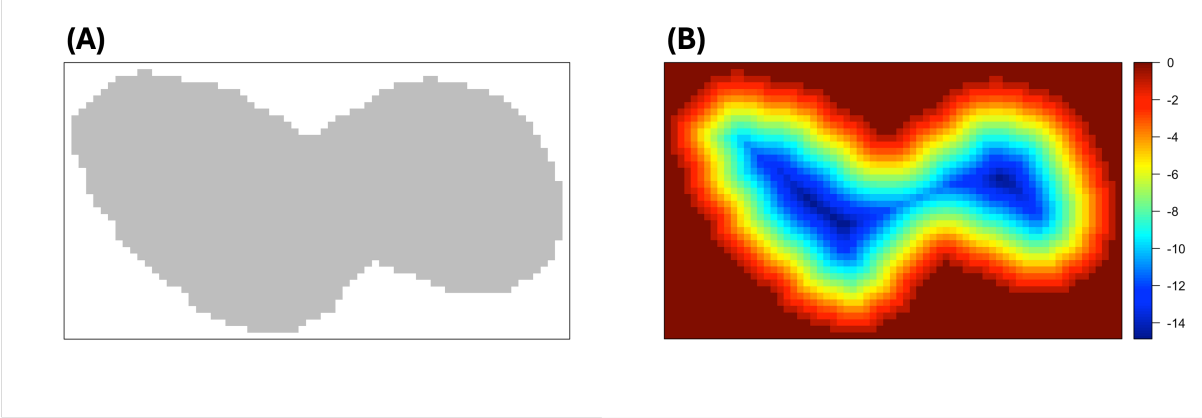


Figure A.5: (A) An example binary image of two connecting cells. (B) Corresponding negative distance map in the watershed algorithm.

A.8 Cell culturing and imaging for real data collection

The cell images used to develop this segmentation technique are of spindle-shaped human dermal fibroblasts (hdFs), and the cells were cultured in Dulbecco's modified eagle medium supplemented with 1% Pen-Strep and 10% fetal bovine serum [1]. Prior to imaging, the cytoplasm was stained with CellTracker deep red, and the nuclei were stained with Hoechst 33342.

At each time point when these microscopy images are acquired, scans of these cell cultures were taken and stitched together, resulting in large composite images. Each scan has two channels: a cytoplasm or whole cell channel and a nuclear channel. We compare the benchmark ImageJ segmentation technique and the proposed GP-based method using both channels. Further details of the images, including image dimensions, names, and channels, can be found in Table A.1.

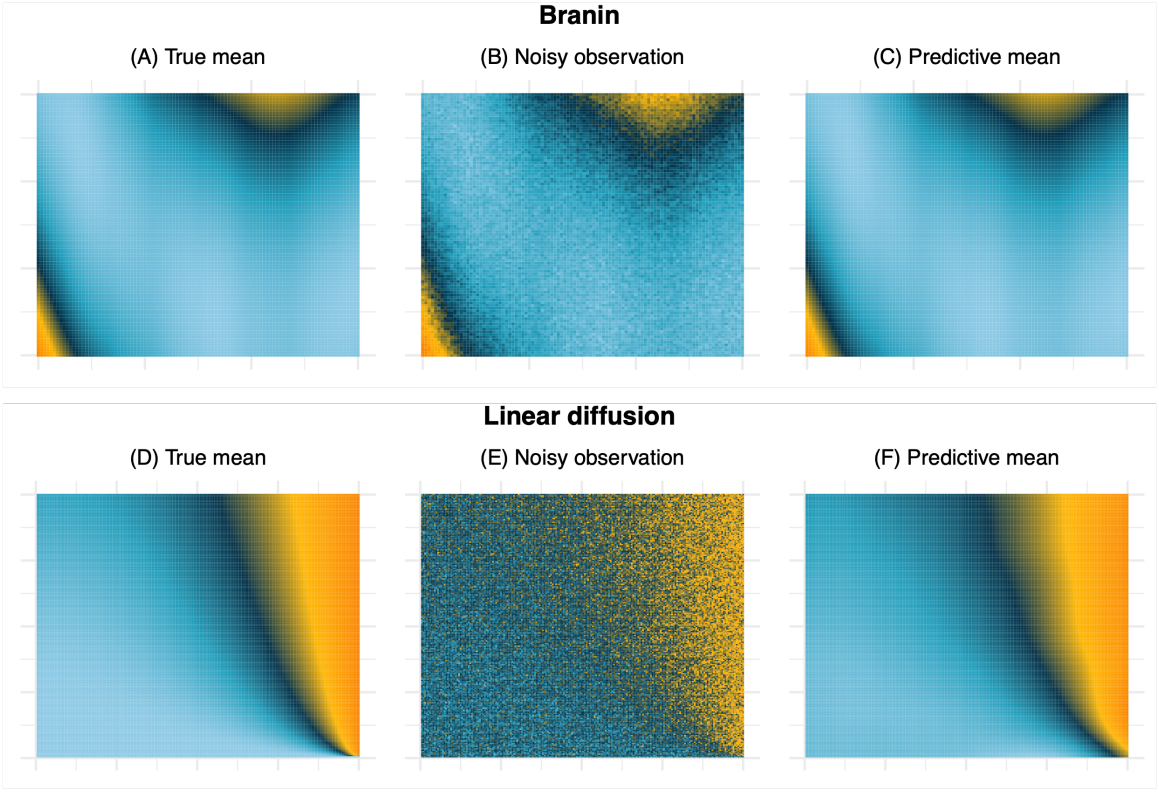


Figure A.6: (A) True mean generated by Branin function. (B) Noisy observations of Branin function ($\sigma_0 = 10$). (C) Predictive mean of Branin function by GPs on lattice data with Matérn kernel. (D) True mean generated by linear diffusion equation. (E) Noisy observations of linear diffusion equation ($\sigma_0 = 0.3$). (F) Predictive mean of linear diffusion equation by GPs on lattice data with Matérn kernel.

A.9 Benchmark ImageJ segmentation steps for real data analysis

ImageJ segmentation and boundary detection methods were utilized to evaluate the performance of the unsupervised detection algorithm. The following image processing steps were followed to generate the cell masks for the microscopy images tested in this article [80]:

1. **Color Channel Splitting.** When necessary, the color channels were split by selecting from the `Image` menu `Color` → `Split Channels`. This operation divides the image into red, green, and blue channels. All images provided were already in grayscale format, so

all channels were the same and matched the original image. For consistency, the blue channel was always selected for further processing.

2. **Brightness and Contrast Adjustment.** The brightness and contrast of the grayscale image were automatically adjusted by selecting the following from the `Image` menu: `Adjust` → `Brightness / Contrast` → `Auto`. With I_{min} being the smallest pixel value in the image and I_{max} being the largest, each pixel $I(x_{i,1}, x_{j,2})$ is reset to $I'(x_{i,1}, x_{j,2}) = \frac{I(x_{i,1},x_{j,2})-I_{min}}{I_{max}-I_{min}} \times 255$. This function linearly rescales the image such that the pixel values span the full range of possible values between 0 and 255, adding more contrast and brightening the foreground pixels [179].

3. **Binary Thresholding.** The image was converted to binary format by selecting from the `Image` menu `Adjust` → `Threshold` → `Auto`. The default method for setting the threshold is a version of the IsoData method. In the IsoData method, after an initial threshold is set, the averages for the pixel intensity values above (T_{above}) and below (T_{below}) the current threshold are taken, and the threshold is set to the composite average ($T_{new} = \frac{T_{above}+T_{below}}{2}$). This process is repeated until convergence [116]. The default thresholding method in ImageJ first identifies and dampens dominant peaks in the pixel intensity histogram values before applying the IsoData, reducing asymmetry in the histogram between the background pixel intensity peak and the foreground pixel intensity peak. [80].

4. **Noise Reduction.** Once the image is thresholded to be binary, a despeckle operation is run to reduce any noise (`Noise` → `Despeckle` from the `Process` menu). Despeckle applies a 3×3 median filter over the image, which replaces each pixel with the median value of the surrounding pixels. This process helps reduce noise and smooth the image [180].

5. **Watershed Segmentation.** To delineate touching cell objects, watershed is applied using `Binary` → `Watershed` from the `Process` menu. Watershed in ImageJ creates an Euclidean distance map to give the distance of each foreground pixel to the nearest background pixel and finds the ultimate eroded point(s) (UEPs) of each binary object using an erode operation on the binary image. Water is simulated to "flood" the image, with the UEPs as the seed points, and an edge is created where the water from two UEPs converges [119].

6. **Cell Mask Generation.** Finally, the object is analyzed to extract the cell masks using `Analyze Particles` from the `Analyze` menu. The parameters were set to detect objects with size 30-`inf` pixels^2 and circularity 0-1.0. To detect cells in the input image, the algorithm scans the image until it reaches the edge of an object (a foreground pixel). It then traces around the edge of the object until the start point is reached [83]. A true binary image can be generated by showing `masks`, and the masks can be uniquely identified by showing `count masks`.

A.10 Ground truth cell mask retrieval for real data analysis

The regions of interest (ROI) detected by the model zoos in the supervised learning algorithm Cellpose were utilized as the starting point for generating the true cell masks for each test image described in Table A.1 [5]. The pretrained nuclei and cyto models were used to generate the initial ROI for the nuclear and whole cell channels, respectively, of the test grayscale images. The estimated cell diameter was kept at the default 30 pixels. The resulting initial ROI were then refined by deleting erroneous and irregular cell masks and adding masks for missed cells using the draw function (right-click) in the Cellpose graphical user interface (GUI) [88],

with the final output being the true masks. The true masks are used to compare the accuracy of the GP-based segmentation method against the benchmark ImageJ segmentation. For clearer illustration in the figures, the true boundaries are detected by finding the outer coordinates of each true mask object. The cell mask and boundary data are made publicly available associated with this article: `https://github.com/UncertaintyQuantification/cell_segmentation`.

A.11 Specifications of microscopy images

Table A.1 contains the image name, dimensions, channel, and relevant figures for the images used in this analysis. Multiple experimental images were cropped and sectioned to generate one or more smaller unique images for method testing, visualization, and analysis, as the original images were too large for time-efficient computation and space-efficient visualization. All images were taken during the experiments described in [1] and are publicly available, and all images can be found in the open-sourced dataset associated with this article: `https://github.com/UncertaintyQuantification/cell_segmentation`.

Table A.1: Name, dimensions, channel utilized, and relevant figures for the microscopy images used in this study.

Image Name	**Dimensions**	**Channel**	**Relevant Figures**
cropped_img_1	488×799	Nuclear	2.1
cropped_img_2	488×799	Nuclear	2.4, A.5 (Cropped section of image shown and analyzed)
cropped_img_4	488×799	Nuclear	2.2, 2.3, A.3 (Cropped section of image shown and analyzed)
cropped_img_5	488×799	Nuclear	A.4
nuclei_2	374×250	Nuclear	2.7
whole_cell_2	602×600	Whole Cell	2.7
nuclei_figure_1/original_fig	962×1128	Nuclear	2.9, A.1, A.2 (Test Image 1)
nuclei_figure_2/original_fig	374×250	Nuclear	2.9, A.1, A.2 (Test Image 2)
nuclei_figure_3/original_fig	510×492	Nuclear	2.9, A.1, A.2 (Test Image 3)
nuclei_figure_4/original_fig	960×608	Nuclear	2.9, A.1, A.2 (Test Image 4)
nuclei_figure_5/original_fig	1024×1024	Nuclear	2.9, A.1, A.2 (Test Image 5)
whole_cell_figure_1/original_fig	602×600	Whole Cell	2.10 (Test Image 1)
whole_cell_figure_2/original_fig	800×800	Whole Cell	2.10 (Test Image 2)
whole_cell_figure_3/original_fig	602×600	Whole Cell	2.10 (Test Image 3)
whole_cell_figure_4/original_fig	1000×1000	Whole Cell	2.10 (Test Image 4)
whole_cell_figure_5/original_fig	1000×1000	Whole Cell	2.10 (Test Image 5)

Appendix B

Appendix of chapter 4

B.1 Diffusion Model Functions

This section provides a list of the functions found in the implemented diffusion model code. The DDPM model was implemented based on the Keras example [149], and Table B.1 provides each function name and utility. The score-based SDE model was implemented based on the corresponding code base to [144], and Table B.2 provides each function name and utility.

B.2 Sensitivity of denoising prediction for score-based generative model timestep selection

The selection of the timestep for DDPM prediction highly influences model output in image denoising. Consider the noisy image generated by adding Gaussian noise with variance $\sigma_0^2 = 0.3^2$ from Figure 4.3. When this image is denoised only using $t = 10$ timesteps (see first row in Figure B.1, the added noise remains in the output image, indicating incomplete prediction. However, when this image is denoised with reference timestep $t = 150$, new image artifacts are erroneously introduced, demonstrating that the expected noise at this timestep is

Table B.1: Function or object class name and utility for Keras DDPM implementation.

Object Name	**Utility**
GaussianDiffusion	Adds noise to input images according to noise variance, as described in Equation (4.5). Also contains functionality for retrieving the diffusion posterior distribution, as described in Equation (4.8).
AttentionBlock	Applies self-attention [147] to residual block.
TimeEmbedding	Embeds diffusion time t with sinusoidal time embedding.
ResidualBlock	Creates a residual block for a feature map and includes group normalization, activation, time embedding, and 2D convolutional operations.
DownSample	Performs convolutional operation $\mathcal{D}_{3\times3}$, with stride 2, to halve spatial resolution.
UpSample	Performs upsampling operation $\mathcal{U}_{2\times2}$ to double spatial resolution followed by convolutional operation $\mathcal{D}_{3\times3}$.
TimeMLP	Passes sinusoidal time embedding through a two-layer MLP with Swish activation [181] to transform embedding into a learnable representation.
build_model	Builds entire U-net architecture with input processing (e.g., time embedding), downsampling, and upsampling to learn $\boldsymbol{\zeta}_\theta(\mathbf{y}_t, t)$.
DiffusionModel	Wrapper for U-net and handles training (Algorithm 1) and image generation (Algorithm 2).

greater than the actual noise introduced. Using $t = 50$ timesteps for denoising yields a similar result to the selected $t = 30$, demonstrating that there is a range of acceptable timesteps for denoising and justifying empirical timestep selection.

Similarly, the timestep selected for denoising using score-based SDE model influences the model output in a similar fashion. Consider the noisy image generated by adding Gaussian noise with variance $\sigma_0^2 = 0.3^2$ from Figure 4.4. When this image is denoised using timestep $t = 0.75$, a significant amount of image information is lost in the predicted result, including smaller or excluded cell objects and a lighter background. Using timestsep $t = 0.6$ also oversmooths cell boundaries and excludes some cell objects, but the structural similarity is better in this result. Using timestep $t = 0.1$ yields a similar prediction to that when using $t = 0.3$,

Table B.2: Function or object class name and utility for the implementation of the score-based SDE model.

Object Name	**Utility**
GaussianFourierProjection	Implements Gaussian random feature embedding for timestep t, as described in Equation (4.21)
Dense	Applies linear transformation to time embedding and reshapes output for broadcasting to feature maps in U-net
ScoreNet	Sets up the architecture for the time-dependent U-net to predict the score function, as described in Equation (4.22)
marginal_prob_std	Calculates the mean and standard deviation of $p_{0t}(\mathbf{y}(t) \mid \mathbf{y}(t))$ to populate distribution given in Equation (4.17)
diffusion_coeff	Computes the vector of diffusion coefficients used in Equations (4.13) and (4.18)
loss_fn	Calculates loss for score-based generative model using Equation (4.20)
ode_sampler	Sampler to generate data from learned score-based model using the ODE described in Equation (4.23)

once again demonstrating that there is a range of acceptable timesteps for denoising.

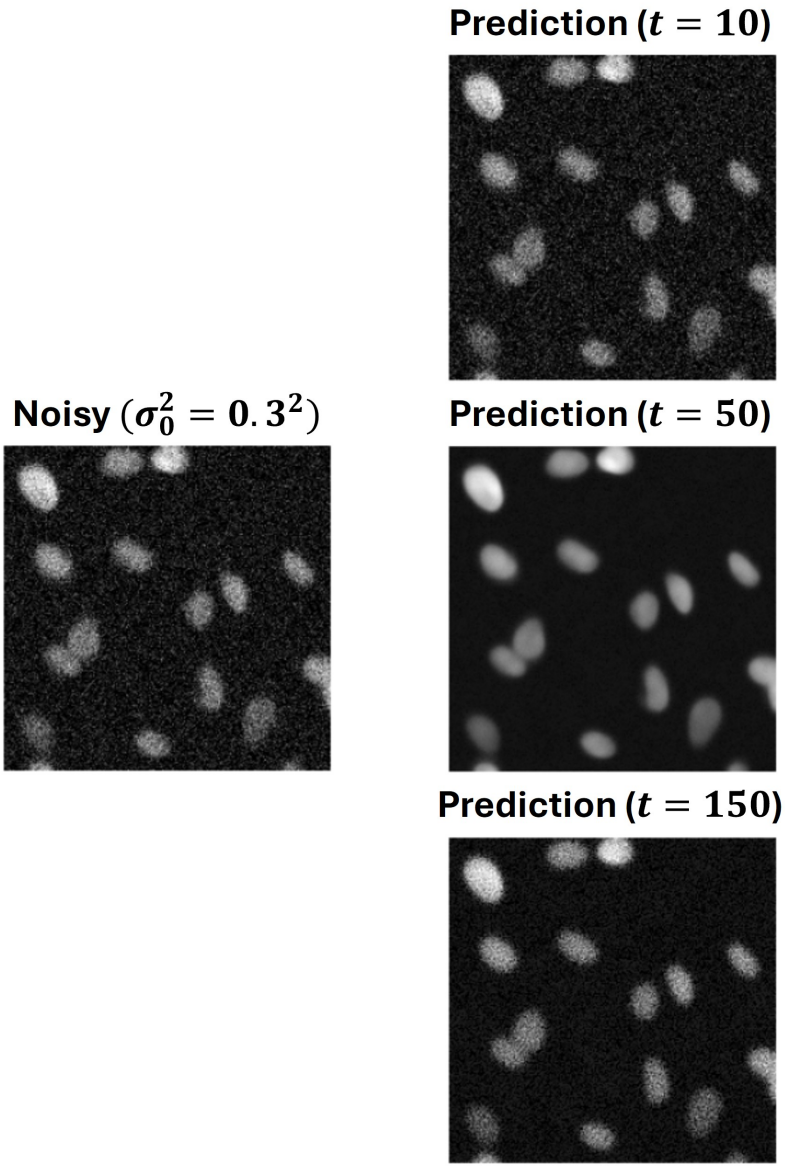


Figure B.1: (*Left*) Noisy image. (*Right*) Predicted output when using timesteps $t = 10$ (*top*), $t = 50$ (*middle*), and $t = 150$ (*bottom*) for denoising with DDPM.

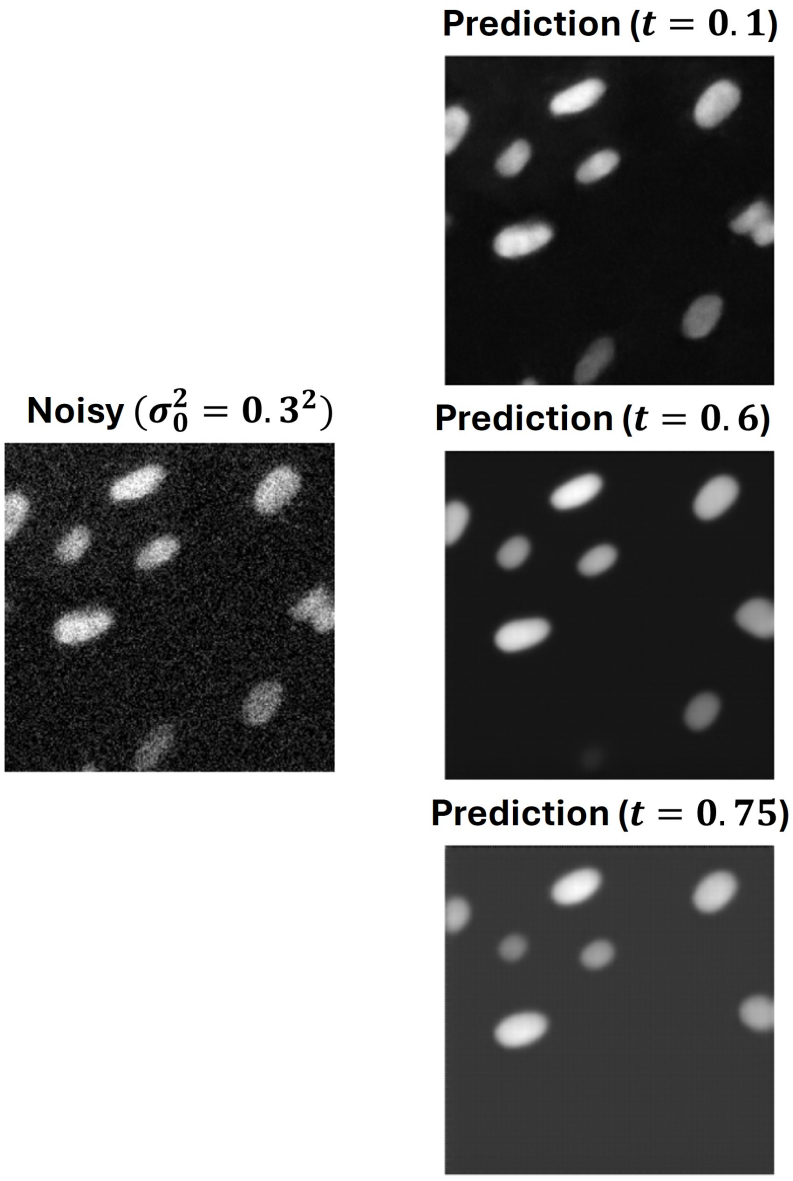


Figure B.2: (*Left*) Noisy image. (*Right*) Predicted output when using timesteps $t = 0.1$ (*top*), $t = 0.6$ (*middle*), and $t = 0.75$ (*bottom*) for denoising with score-based SDE model.

Bibliography

[1] Y. Luo, M. Gu, M. Park, X. Fang, Y. Kwon, J. M. Urueña, J. Read de Alaniz, M. E. Helgeson, C. M. Marchetti, and M. T. Valentine, "Molecular-scale substrate anisotropy, crowding and division drive collective behaviours in cell monolayers," *Journal of the Royal Society Interface*, vol. 20, no. 204, p. 20230160, 2023.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241, Springer International Publishing, 2015.

[3] C. T. Wallace, C. M. St. Croix, and S. C. Watkins, "Data management and archiving in a large microscopy-and-imaging, multi-user facility: Problems and solutions," *Molecular Reproduction and Development*, vol. 82, no. 9, pp. 630–634, 2015.

[4] M. Comes, P. Casti, A. Mencattini, D. Di Giuseppe, F. Mermet-Meillon, A. De Ninno, M. Parrini, L. Businaro, C. Di Natale, and E. Martinelli, "The influence of spatial and temporal resolutions on the analysis of cell-cell interaction: a systematic study for time-lapse microscopy applications," *Scientific Reports*, vol. 9, no. 1, p. 6789, 2019.

[5] C. Stringer, T. Wang, M. Michaelos, and M. Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," *Nature Methods*, vol. 18, no. 1, pp. 100–106, 2021.

[6] T. J. Collins, "ImageJ for microscopy," *Biotechniques*, vol. 43, pp. S25–S30, 2007.

[7] E. Meijering, "Cell segmentation: 50 years down the road," *IEEE Signal Processing Magazine*, vol. 29, no. 5, pp. 140–145, 2012.

[8] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, *et al.*, "Segment anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4015–4026, 2023.

[9] J.-Y. Tinevez, N. Perry, J. Schindelin, G. M. Hoopes, G. D. Reynolds, E. Laplantine, S. Y. Bednarek, S. L. Shorte, and K. W. Eliceiri, "TrackMate: an open and extensible platform for single-particle tracking," *Methods*, vol. 115, pp. 80–90, 2017.

[10] J.-S. Lee, "Digital image smoothing and the sigma filter," *Computer Vision, Graphics, and Image Processing*, vol. 24, no. 2, pp. 255–269, 1983.

[11] L. Baracaldo, B. King, H. Yan, Y. Lin, N. Miolane, and M. Gu, "Unsupervised cell segmentation by fast Gaussian processes," *The New England Journal of Statistics in Data Science*, pp. 1–16, 2026.

[12] R. Cerbino and V. Trappe, "Differential dynamic microscopy: probing wave vector dependent dynamics with a microscope," *Physical review letters*, vol. 100, no. 18, p. 188102, 2008.

[13] M. Gu, Y. He, X. Liu, and Y. Luo, "Ab initio uncertainty quantification in scattering analysis of microscopy," *Physical Review E*, vol. 110, no. 3, p. 034601, 2024.

[14] F. Xing and L. Yang, "Robust nucleus/cell detection and segmentation in digital pathology and microscopy images: a comprehensive review," *IEEE Reviews in Biomedical Engineering*, vol. 9, pp. 234–263, 2016.

[15] J. Schulz, T. Schmidt, O. Ronneberger, H. Burkhardt, T. Pasternak, A. Dovzhenko, and K. Palme, "Fast scalar and vectorial grayscale based invariant features for 3d cell nuclei localization and classification," in *Pattern Recognition*, pp. 182–191, Springer Berlin Heidelberg, 2006.

[16] Y. Dang, X. Lan, Q. Zhang, L. Li, and Y. Huang, "Analysis of grayscale characteristics in images of labeled microtubules from cultured cardiac myocytes," *Microscopy and Microanalysis*, vol. 21, no. 2, pp. 334–342, 2015.

[17] V. Ljosa and A. E. Carpenter, "Introduction to the quantitative analysis of two-dimensional fluorescence microscopy images for cell-based screening," *PLoS Computational Biology*, vol. 5, no. 12, p. e1000603, 2009.

[18] S. Lockett, K. Jacobson, and B. Herman, "Quantitative precision of an automated, fluorescence-based image cytometer," *Analytical and Quantitative Cytology and Histology*, vol. 14, no. 3, pp. 187–202, 1992.

[19] T. E. Gillies and C. Cabernard, "Cell division orientation in animals," *Current Biology*, vol. 21, no. 15, pp. R599–R609, 2011.

[20] C.-Y. Chung, H. Bien, and E. Entcheva, "The role of cardiac tissue alignment in modulating electrical function," *Journal of Cardiovascular Electrophysiology*, vol. 18, no. 12, pp. 1323–1329, 2007.

[21] F. Navaee, N. Khornian, D. Longet, S. Heub, S. Boder-Pasche, G. Weder, A. Kleger, P. Renaud, and T. Braschler, "A three-dimensional engineered cardiac in vitro model: controlled alignment of cardiomyocytes in 3D microphysiological systems," *Cells*, vol. 12, no. 4, p. 576, 2023.

[22] J. M. Lee and W. Y. Yeong, "Engineering macroscale cell alignment through coordinated toolpath design using support-assisted 3D bioprinting," *Journal of the Royal Society Interface*, vol. 17, no. 168, 2020.

[23] J. Huang, J. Chen, and Y. Luo, "Cell-sheet shape transformation by internally-driven, oriented forces," *Advanced Materials*, vol. 37, no. 20, p. 2416624, 2025.

[24] A. Ray, O. Lee, Z. Win, R. M. Edwards, P. W. Alford, D.-H. Kim, and P. P. Provenzano, "Anisotropic forces from spatially constrained focal adhesions mediate contact guidance directed cell migration," *Nature Communications*, vol. 8, p. 14923, 2017.

[25] M. Bachmann, S. Kukkurainen, V. P. Hytönen, and B. Wehrle-Haller, "Cell adhesion by integrins," *Physiological Reviews*, vol. 99, no. 4, pp. 1655–1699, 2019.

[26] M. M. Usaj, C. H. L. Yeung, H. Friesen, C. Boone, and B. J. Andrews, "Single-cell image analysis to explore cell-to-cell heterogeneity in isogenic populations," *Cell Systems*, vol. 12, no. 6, pp. 608–621, 2021.

[27] M. A. Walling and J. R. Shepard, "Cellular heterogeneity and live cell arrays," *Chemical Society Reviews*, vol. 40, no. 7, pp. 4049–4076, 2011.

[28] K. Geiler-Samerotte, C. Bauer, S. Li, N. Ziv, D. Gresham, and M. Siegal, "The details in the distributions: why and how to study phenotypic variability," *Current Opinion in Biotechnology*, vol. 24, no. 4, pp. 752–759, 2013.

[29] M. Boutros, F. Heigwer, and C. Laufer, "Microscopy-based high-content screening," *Cell*, vol. 163, no. 6, pp. 1314–1325, 2015.

[30] S. Gordonov, M. K. Hwang, A. Wells, F. B. Gertler, D. A. Lauffenburger, and M. Bathe,

"Time series modeling of live-cell shape dynamics for image-based phenotypic profiling," *Integrative Biology*, vol. 8, no. 1, pp. 73–90, 2016.

[31] A. D. Coster, C. Wichaidit, S. Rajaram, S. J. Altschuler, and L. F. Wu, "A simple image correction method for high-throughput microscopy," *Nature Methods*, vol. 11, no. 6, pp. 602–602, 2014.

[32] D. Liu and J. Yu, "Otsu method and K-means," in *2009 Ninth International Conference on Hybrid Intelligent Systems*, vol. 1, pp. 344–349, IEEE, 2009.

[33] F. R. Velasco, "Thresholding using the ISODATA clustering algorithm," tech. rep., University of Maryland Computer Science Center, 1979.

[34] L.-K. Huang and M.-J. J. Wang, "Image thresholding by minimizing the measures of fuzziness," *Pattern Recognition*, vol. 28, no. 1, pp. 41–51, 1995.

[35] K. V. Mardia and T. J. Hainsworth, "A spatial thresholding method for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 6, pp. 919–927, 1988.

[36] N. U. Din and J. Yu, "Training a deep learning model for single-cell segmentation without manual annotation," *Scientific Reports*, vol. 11, p. 23995, 2021.

[37] P. Shrestha, N. Kuang, and J. Yu, "Efficient end-to-end learning for cell segmentation with machine generated weak annotations," *Communications Biology*, vol. 6, p. 232, 2023.

[38] B. Kochetov, P. D. Bell, P. S. Garcia, A. S. Shalaby, R. Raphael, B. Raymond, B. J. Leibowitz, K. Schoedel, R. M. Brand, R. E. Brand, *et al.*, "UNSEG: unsupervised segmentation of cells and their nuclei in complex tissue samples," *Communications Biology*, vol. 7, p. 1062, 2024.

[39] E. Moen, D. Bannon, T. Kudo, W. Graf, M. Covert, and D. Van Valen, "Deep learning for cellular image analysis," *Nature Methods*, vol. 16, pp. 1233–1246, 2019.

[40] C.-H. Lai, Y. Song, D. Kim, Y. Mitsufuji, and S. Ermon, "The principles of diffusion models," 2025.

[41] D. N. Lesmes-Leon, A. Dengel, and S. Ahmed, "Systematic review of generative adversarial networks (GANs) in cell microscopy: Trends, practices, and impact on image augmentation," *PloS One*, vol. 20, no. 6, p. e0291217, 2025.

[42] W. Wang, J. Xia, G. Luo, S. Dong, X. Li, J. Wen, and S. Li, "Diffusion model for medical image denoising, reconstruction and translation," *Computerized Medical Imaging and Graphics*, vol. 124, p. 102593, 2025.

[43] M. Gu, Y. Luo, Y. He, M. E. Helgeson, and M. T. Valentine, "Uncertainty quantification and estimation in differential dynamic microscopy," *Physical Review E*, vol. 104, no. 3, p. 034610, 2021.

[44] J. Sacks, S. B. Schiller, and W. J. Welch, "Designs for computer experiments," *Technometrics*, vol. 31, no. 1, pp. 41–47, 1989.

[45] P. Abrahamsen, "A review of Gaussian random fields and correlation functions," 1997.

[46] T. J. Santner, B. J. Williams, W. I. Notz, and B. J. Williams, *The Design and Analysis of Computer Experiments*. Springer, second ed., 2003.

[47] C. K. Williams and C. E. Rasmussen, *Gaussian processes for Machine Learning*. MIT Press, 2006.

[48] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced Lectures on Machine Learning: ML Summer Schools*, pp. 63–71, Springer Berlin Heidelberg, 2004.

[49] E. T. Spiller, M. J. Bayarri, J. O. Berger, E. S. Calder, A. K. Patra, E. B. Pitman, and R. L. Wolpert, "Automating emulator construction for geophysical hazard maps," *SIAM/ASA Journal on Uncertainty Quantification*, vol. 2, no. 1, pp. 126–152, 2014.

[50] M. Gu and J. O. Berger, "Parallel partial Gaussian process emulation for computer models with massive output," *The Annals of Applied Statistics*, vol. 10, no. 3, pp. 1317–1347, 2016.

[51] B. Matérn, *Spatial Variation*. Meddelanden från Statens Skogsforskningsinstitut, second ed., 1960.

[52] O. Roustant, D. Ginsbourger, and Y. Deville, "Dicekriging, DiceOptim: two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization," *Journal of Statistical Software*, vol. 51, no. 1, pp. 1–55, 2012.

[53] M. Gu, J. Palomo, and J. O. Berger, "RobustGaSP: robust Gaussian stochastic process emulation in R," *The R Journal*, vol. 11, no. 1, pp. 112–136, 2019.

[54] M. S. Handcock and M. L. Stein, "A Bayesian analysis of kriging," *Technometrics*, vol. 35, no. 4, pp. 403–410, 1993.

[55] J. O. Berger, V. De Oliveira, and B. Sansó, "Objective Bayesian analysis of spatially correlated data," *Journal of the American Statistical Association*, vol. 96, no. 456, pp. 1361–1374, 2001.

[56] M. Gu, X. Wang, and J. O. Berger, "Robust Gaussian stochastic process emulation," *The Annals of Statistics*, vol. 46, no. 6A, pp. 3038–3066, 2018.

[57] M. Gu, "Jointly robust prior for Gaussian stochastic process in emulation, calibration and variable selection," *Bayesian Analysis*, vol. 14, no. 3, pp. 857–885, 2019.

[58] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Mathematics of Computation*, vol. 35, no. 151, pp. 773–782, 1980.

[59] J. Donnelly, S. Abolfathi, J. Pearson, O. Chatrabgoun, and A. Daneshkhah, "Gaussian process emulation of spatio-temporal outputs of a 2D inland flood model," *Water Research*, vol. 225, p. 119100, 2022.

[60] O. K. Oyebamiji, D. J. Wilkinson, P. G. Jayathilake, T. P. Curtis, S. Rushton, B. Li, and P. Gupta, "Gaussian process emulation of an individual-based model simulation of microbial communities," *Journal of Computational Science*, vol. 22, pp. 69–84, 2017.

[61] G. Camps-Valls, L. Martino, D. H. Svendsen, M. Campos-Taberner, J. Muñoz-Marí, V. Laparra, D. Luengo, and F. J. García-Haro, "Physics-aware Gaussian processes in remote sensing," *Applied Soft Computing*, vol. 68, pp. 69–82, 2018.

[62] S. Conti and A. O'Hagan, "Bayesian emulation of complex multi-output and dynamic computer models," *Journal of Statistical Planning and Inference*, vol. 140, no. 3, pp. 640–651, 2010.

[63] L. Lee, K. Carslaw, K. Pringle, G. Mann, and D. Spracklen, "Emulation of a complex global aerosol model to quantify sensitivity to uncertain parameters," *Atmospheric Chemistry and Physics*, vol. 11, no. 23, pp. 12253–12273, 2011.

[64] L. Lee, K. Carslaw, K. Pringle, and G. Mann, "Mapping the uncertainty in global ccn using emulation," *Atmospheric Chemistry and Physics*, vol. 12, no. 20, pp. 9739–9751, 2012.

[65] K. R. Anderson, I. A. Johanson, M. R. Patrick, M. Gu, P. Segall, M. P. Poland, E. K.

Montgomery-Brown, and A. Miklius, "Magma reservoir failure and the onset of caldera collapse at Kīlauea volcano in 2018," *Science*, vol. 366, no. 6470, 2019.

[66] T. L. Edwards, S. Nowicki, B. Marzeion, R. Hock, H. Goelzer, H. Seroussi, N. C. Jourdain, D. A. Slater, F. E. Turner, C. J. Smith, C. M. McKenna, E. Simon, A. Abe-Ouchi, J. M. Gregory, E. Larour, W. H. Lipscomb, A. J. Payne, A. Shepherd, C. Agosta, P. Alexander, T. Albrecht, B. Anderson, X. Asay-Davis, A. Aschwanden, A. Barthel, A. Bliss, R. Calov, C. Chambers, N. Champollion, Y. Choi, R. Cullather, J. Cuzzone, C. Dumas, D. Felikson, X. Fettweis, K. Fujita, B. K. Galton-Fenzi, R. Gladstone, N. R. Golledge, R. Greve, T. Hattermann, M. J. Hoffman, A. Humbert, M. Huss, P. Huybrechts, W. Immerzeel, T. Kleiner, P. Kraaijenbrink, S. Le clec'h, V. Lee, G. R. Leguy, C. M. Little, D. P. Lowry, J.-H. Malles, D. F. Martin, F. Maussion, M. Morlighem, J. F. O'Neill, I. Nias, F. Pattyn, T. Pelle, S. F. Price, A. Quiquet, V. Radić, R. Reese, D. R. Rounce, M. Rückamp, A. Sakai, C. Shafer, N.-J. Schlegel, S. Shannon, R. S. Smith, F. Straneo, S. Sun, L. Tarasov, L. D. Trusel, J. Van Breedam, R. van de Wal, M. van den Broeke, R. Winkelmann, H. Zekollari, C. Zhao, T. Zhang, and T. Zwinger, "Projected land ice contributions to twenty-first-century sea level rise," *Nature*, vol. 593, no. 7857, pp. 74–82, 2021.

[67] M. Plumlee, T. G. Asher, W. Chang, and M. V. Bilskie, "High-fidelity hurricane surge forecasting using emulation and sequential experiments," *The Annals of Applied Statistics*, vol. 15, no. 1, pp. 460–480, 2021.

[68] H. Zhao, F. Amann, and J. Kowalski, "Emulator-based global sensitivity analysis for flow-like landslide run-out models," *Landslides*, vol. 18, no. 10, pp. 3299–3314, 2021.

[69] S. Cao, Y. Chen, J. Coleman, J. Mulligan, P. Jacobs, R. Soltz, A. Angerami, R. Arora,

S. Bass, L. Cunqueiro, *et al.*, "Determining the jet transport coefficient q from inclusive hadron suppression measurements using Bayesian parameter estimation," *Physical Review C*, vol. 104, no. 2, p. 024905, 2021.

[70] H. Li, M. Zhou, J. Sebastian, J. Wu, and M. Gu, "Efficient force field and energy emulation through partition of permutationally equivalent atoms," *The Journal of Chemical Physics*, vol. 156, no. 18, p. 184304, 2022.

[71] X. Fang, M. Gu, and J. Wu, "Reliable emulation of complex functionals by active learning with error control," *The Journal of Chemical Physics*, vol. 157, no. 21, 2022.

[72] K. Ye, J. Zhao, H. Li, and M. Gu, "A high computationally efficient parallel partial Gaussian process for large-scale power system probabilistic transient stability assessment," *IEEE Transactions on Power Systems*, vol. 39, no. 2, pp. 4650–4660, 2023.

[73] M. Gu, Y. Lin, V. C. Lee, and D. Y. Qiu, "Probabilistic forecast of nonlinear dynamical systems with uncertainty quantification," *Physica D: Nonlinear Phenomena*, vol. 457, p. 133938, 2024.

[74] E. McPhillips, H. Lee, X. Xie, K. Baylis, C. Funk, and M. Gu, "Long-term probabilistic forecast of vegetation conditions using climate attributes in the four corners region," *Remote Sensing*, vol. 18, no. 6, p. 850, 2026.

[75] Y. W. Teh, M. Seeger, and M. I. Jordan, "Semiparametric latent factor models," in *International Workshop on Artificial Intelligence and Statistics*, pp. 333–340, PMLR, 2005.

[76] M. Gu and W. Shen, "Generalized probabilistic principal component analysis of correlated data," *Journal of Machine Learning Research*, vol. 21, no. 13, pp. 1–41, 2020.

[77] M. Gu and Y. Xu, "Fast nonseparable Gaussian stochastic process with application to methylation level interpolation," *Journal of Computational and Graphical Statistics*, vol. 29, no. 2, pp. 250–260, 2020.

[78] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[79] X. Fang and M. Gu, "The inverse Kalman filter," *Biometrika*, vol. 112, no. 4, p. asaf054, 2025.

[80] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri, "NIH image to ImageJ: 25 years of image analysis," *Nature Methods*, vol. 9, no. 7, pp. 671–675, 2012.

[81] A. P. Witkin, "Scale-space filtering," in *Readings in Computer Vision*, pp. 329–332, Morgan Kaufmann, 1987.

[82] L. Nichele, V. Persichetti, M. Lucidi, and G. Cincotti, "Quantitative evaluation of ImageJ thresholding algorithms for microbial cell counting," *OSA Continuum*, vol. 3, no. 6, pp. 1417–1427, 2020.

[83] T. Ferreira and W. Rasband, *ImageJ User Guide – Version 1.44*, Feb 2011.

[84] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. New York: Pearson Education, 4th ed., 2018.

[85] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 207, no. 1167, pp. 187–217, 1980.

[86] K. Jaqaman, D. Loerke, M. Mettlen, H. Kuwata, S. Grinstein, S. L. Schmid, and G. Danuser, "Robust single-particle tracking in live-cell time-lapse sequences," *Nature Methods*, vol. 5, no. 8, pp. 695–702, 2008.

[87] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[88] M. Pachitariu and C. Stringer, "Cellpose 2.0: how to train your own model," *Nature Methods*, vol. 19, no. 12, pp. 1634–1641, 2022.

[89] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, 2012.

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

[91] Y. Lecun, Y. Yere, P. Haffner, Y. Rachmad, and L. Bottou, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, 1998.

[92] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision*, pp. 818–833, Springer, 2014.

[93] S. Eisenbarth, "Dendritic cell subsets in T cell programming: location dictates function," *Nature Reviews Immunology*, vol. 19, no. 2, pp. 89–103, 2019.

[94] A. Prasad and E. Alizadeh, "Cell form and function: interpreting and controlling the shape of adherent cells," *Trends in Biotechnology*, vol. 37, no. 4, pp. 347–357, 2019.

[95] W. F. Liu and C. S. Chen, "Cellular and multicellular form and function," *Advanced Drug Delivery Reviews*, vol. 59, no. 13, pp. 1319–1328, 2007.

[96] G. T. Eisenhoffer, P. D. Loftus, M. Yoshigi, H. Otsuna, C.-B. Chien, P. A. Morcos, and J. Rosenblatt, "Crowding induces live cell extrusion to maintain homeostatic cell numbers in epithelia," *Nature*, vol. 484, pp. 546–549, 2012.

[97] J. Folkman and A. Moscona, "Role of cell shape in growth control," *Nature*, vol. 273, pp. 345–349, 1978.

[98] D. Selmeczi, S. Mosler, P. H. Hagedorn, N. B. Larsen, and H. Flyvbjerg, "Cell motility as persistent random motion: theories from experiments," *Biophysical Journal*, vol. 89, no. 2, pp. 912–931, 2005.

[99] M. Gu, X. Fang, and Y. Luo, "Data-driven model construction for anisotropic dynamics of active matter," *PRX Life*, vol. 1, no. 1, p. 013009, 2023.

[100] A. Khang, A. Barmore, G. Tseropoulos, K. Bera, D. Batan, and K. S. Anseth, "Automated prediction of fibroblast phenotypes using mathematical descriptors of cellular features," *Nature Communications*, vol. 16, pp. 1–17, 2025.

[101] T. Tsukui, K.-H. Sun, J. B. Wetter, J. R. Wilson-Kanamori, L. A. Hazelwood, N. C. Henderson, T. S. Adams, J. C. Schupp, S. D. Poli, I. O. Rosas, *et al.*, "Collagen-producing lung cell atlas identifies multiple subsets with distinct localization and relevance to fibrosis," *Nature Communications*, vol. 11, p. 1920, 2020.

[102] J. Barbazan, C. Pérez-González, M. Gómez-González, M. Dedenon, S. Richon, E. Latorre, M. Serra, P. Mariani, S. Descroix, P. Sens, *et al.*, "Cancer-associated fibroblasts actively compress cancer cells and modulate mechanotransduction," *Nature Communications*, vol. 14, no. 6966, 2023.

[103] M. D. Abràmoff, P. J. Magalhães, and S. J. Ram, "Image processing with ImageJ," *Biophotonics International*, vol. 11, no. 7, pp. 36–42, 2004.

[104] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, *et al.*, "Fiji: an open-source platform for biological-image analysis," *Nature Methods*, vol. 9, no. 7, pp. 676–682, 2012.

[105] M. Gu and H. Li, "Gaussian orthogonal latent factor processes for large incomplete matrices of correlated data," *Bayesian Analysis*, vol. 17, no. 4, pp. 1219 – 1244, 2022.

[106] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," *Advances in Neural Information Processing Systems*, vol. 18, 2005.

[107] F. Lindgren, H. Rue, and J. Lindström, "An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 73, no. 4, pp. 423–498, 2011.

[108] H. Rue, S. Martino, and N. Chopin, "Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 71, no. 2, pp. 319–392, 2009.

[109] A. Datta, S. Banerjee, A. O. Finley, and A. E. Gelfand, "Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets," *Journal of the American Statistical Association*, vol. 111, no. 514, pp. 800–812, 2016.

[110] M. Katzfuss, "A multi-resolution approximation for massive spatial datasets," *Journal of the American Statistical Association*, vol. 112, no. 517, pp. 201–214, 2017.

[111] R. B. Gramacy and D. W. Apley, "Local Gaussian process approximation for large computer experiments," *Journal of Computational and Graphical Statistics*, vol. 24, no. 2, pp. 561–578, 2015.

[112] J. Guinness and M. Fuentes, "Circulant embedding of approximate covariances for inference from Gaussian data on large lattices," *Journal of Computational and Graphical Statistics*, vol. 26, no. 1, pp. 88–97, 2017.

[113] C. G. Kaufman, M. J. Schervish, and D. W. Nychka, "Covariance tapering for likelihood-based estimation in large spatial data sets," *Journal of the American Statistical Association*, vol. 103, no. 484, pp. 1545–1555, 2008.

[114] Y. Zhu, M. Peruzzi, C. Li, and D. B. Dunson, "Radial neighbours for provably accurate scalable approximations of Gaussian processes," *Biometrika*, vol. 111, no. 4, pp. 1151–1167, 2024.

[115] W. Chang, M. Haran, P. Applegate, and D. Pollard, "Calibrating an ice sheet model using high-dimensional binary spatial data," *Journal of the American Statistical Association*, vol. 111, no. 513, pp. 57–72, 2016.

[116] T. W. Ridler, S. Calvard, *et al.*, "Picture thresholding using an iterative selection method," *IEEE Trans. Syst. Man Cybern*, vol. 8, no. 8, pp. 630–632, 1978.

[117] K. Bhawnesh, U. Tiwari, S. Kumar, V. Tomer, and J. Kalra, "Comparison and performance evaluation of boundary fill and flood fill algorithm," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, 08 2020.

[118] G. Pau, F. Fuchs, O. Sklyar, M. Boutros, and W. Huber, "EBImage—an R package for

image processing with applications to cellular phenotypes," *Bioinformatics*, vol. 26, no. 7, pp. 979–981, 2010.

[119] L. Vincent and P. Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, 1991.

[120] G. Berkooz, P. Holmes, and J. L. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows," *Annual Review of Fluid Mechanics*, vol. 25, no. 1, pp. 539–575, 1993.

[121] Y. Lin, X. Liu, P. Segall, and M. Gu, "Fast data inversion for high-dimensional dynamical systems from noisy measurements," *arXiv preprint arXiv:2501.01324*, 2025.

[122] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of Fluid Mechanics*, vol. 656, pp. 5–28, 2010.

[123] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, "On dynamic mode decomposition: theory and applications," *Journal of Computational Dynamics*, vol. 1, no. 2, pp. 391–421, 2014.

[124] V. Picheny, T. Wagner, and D. Ginsbourger, "A benchmark of kriging-based infill criteria for noisy optimization," *Structural and Multidisciplinary Optimization*, vol. 48, no. 3, pp. 607–626, 2013.

[125] H. Lu and D. M. Tartakovsky, "Prediction accuracy of dynamic mode decomposition," *SIAM Journal on Scientific Computing*, vol. 42, no. 3, pp. A1639–A1662, 2020.

[126] K. E. Soetaert, T. Petzoldt, and R. W. Setzer, "Solving differential equations in R: package deSolve," *Journal of Statistical Software*, vol. 33, no. 9, 2010.

[127] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.

[128] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: a metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[129] B. King, H. Yan, L. Baracaldo, and M. Gu, *FastSegmentation: Unsupervised Cell Segmentation by Fast Gaussian Processes*, 2026. R package version 0.0.1.

[130] J. Ooms, *magick: Advanced Graphics and Image-Processing in R*, 2025. R package version 2.9.0.

[131] K. Soetaert, *plot3D: Plotting Multi-Dimensional Data*, 2025. R package version 1.4.2.

[132] M. Sadati, N. T. Qazvini, R. Krishnan, C. Y. Park, and J. J. Fredberg, "Collective migration and cell jamming," *Differentiation*, vol. 86, no. 3, pp. 121–125, 2013.

[133] N. Bagheri, A. E. Carpenter, E. Lundberg, A. L. Plant, and R. Horwitz, "The new era of quantitative cell imaging—challenges and opportunities," *Molecular Cell*, vol. 82, no. 2, pp. 241–247, 2022.

[134] H. R. Petty, "Fluorescence microscopy: established and emerging methods, experimental strategies, and applications in immunology," *Microscopy Research and Technique*, vol. 70, no. 8, pp. 687–709, 2007.

[135] D. M. Benson, J. Bryan, A. L. Plant, A. M. Gotto Jr, and L. C. Smith, "Digital imag-

ing fluorescence microscopy: spatial heterogeneity of photobleaching rate constants in individual cells.," *The Journal of Cell Biology*, vol. 100, no. 4, pp. 1309–1323, 1985.

[136] R. F. Laine, G. Jacquemet, and A. Krull, "Imaging in focus: an introduction to denoising bioimages in the era of deep learning," *The International Journal of Biochemistry & Cell Biology*, vol. 140, p. 106077, 2021.

[137] W. Meiniel, J.-C. Olivo-Marin, and E. D. Angelini, "Denoising of microscopy images: a review of the state-of-the-art, and a new sparsity-based method," *IEEE Transactions on Image Processing*, vol. 27, no. 8, pp. 3842–3856, 2018.

[138] V. Ljosa, K. L. Sokolnicki, and A. E. Carpenter, "Annotated high-throughput microscopy image sets for validation," *Nature Methods*, vol. 9, no. 7, p. 637, 2012.

[139] D. N. Orloff, J. H. Iwasa, M. E. Martone, M. H. Ellisman, and C. M. Kane, "The cell: an image library-CCDB: a curated repository of microscopy data," *Nucleic Acids Research*, vol. 41, no. D1, pp. D1241–D1250, 2012.

[140] C. Luo, "Understanding diffusion models: a unified perspective," *arXiv preprint arXiv:2208.11970*, 2022.

[141] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[142] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[143] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[144] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," *arXiv preprint arXiv:2011.13456*, 2020.

[145] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[146] B. D. Anderson, "Reverse-time diffusion equation models," *Stochastic Processes and their Applications*, vol. 12, no. 3, pp. 313–326, 1982.

[147] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[148] P. Vincent, "A connection between score matching and denoising autoencoders," *Neural Computation*, vol. 23, no. 7, pp. 1661–1674, 2011.

[149] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[150] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz, *et al.*, "Attention U-net: Learning where to look for the pancreas," *arXiv preprint arXiv:1804.03999*, 2018.

[151] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[152] A. Hyvärinen and P. Dayan, "Estimation of non-normalized statistical models by score matching.," *Journal of Machine Learning Research*, vol. 6, no. 4, 2005.

[153] Y. Song, S. Garg, J. Shi, and S. Ermon, "Sliced score matching: A scalable approach

to density and score estimation," in *Uncertainty in Artificial Intelligence*, pp. 574–584, PMLR, 2020.

[154] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7537–7547, 2020.

[155] P. E. Kloeden, "Stochastic differential equations," in *International Encyclopedia of Statistical Science*, pp. 1520–1521, Springer, 2011.

[156] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[157] Z. Zhang, Y. Hua, G. Sun, H. Wang, and S. McLoone, "Training diffusion-based generative models with limited data," in *Forty-second International Conference on Machine Learning*, 2025.

[158] V. Mannam and S. Howard, "Small training dataset convolutional neural networks for application-specific super-resolution microscopy," *Journal of Biomedical Optics*, vol. 28, no. 3, pp. 036501–036501, 2023.

[159] M. Tivnan, S. Yoon, Z. Chen, X. Li, D. Wu, and Q. Li, "Hallucination index: An image quality metric for generative reconstruction models," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 449–458, Springer, 2024.

[160] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment:

from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[161] M. P. Deisenroth, R. D. Turner, M. F. Huber, U. D. Hanebeck, and C. E. Rasmussen, "Robust filtering and smoothing with Gaussian processes," *IEEE Transactions on Automatic Control*, vol. 57, no. 7, pp. 1865–1871, 2011.

[162] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International Conference on Machine Learning*, pp. 8162–8171, PMLR, 2021.

[163] Y. Hoang, J. L. Franklin, Y. S. Dufour, and L. Kroos, "Cell density, alignment, and orientation correlate with C-signal–dependent gene expression during myxococcus xanthus development," *Proceedings of the National Academy of Sciences*, vol. 118, no. 45, p. e2111706118, 2021.

[164] R. Sali, Y. Jiang, A. Attaranzadeh, B. Holmes, and R. Li, "Morphological diversity of cancer cells predicts prognosis across tumor types," *JNCI: Journal of the National Cancer Institute*, vol. 116, no. 4, pp. 555–564, 2024.

[165] X. Xiao, Z. Wang, Y. Kong, and H. Lu, "Deep learning-based morphological feature analysis and the prognostic association study in colon adenocarcinoma histopathological images," *Frontiers in Oncology*, vol. 13, p. 1081529, 2023.

[166] K.-J. Wu, C. Edmund, C. Shang, and Z. Guo, "Nucleation and growth in solution synthesis of nanostructures–from fundamentals to advanced applications," *Progress in Materials Science*, vol. 123, p. 100821, 2022.

[167] G. Lee, G. Leech, M. J. Rust, M. Das, R. J. McGorty, J. L. Ross, and R. M. Robertson-

Anderson, "Myosin-driven actin-microtubule networks exhibit self-organized contractile dynamics," *Science Advances*, vol. 7, no. 6, p. eabe4334, 2021.

[168] F. Schüth, "Nucleation and crystallization of solids from solution," *Current Opinion in Solid State and Materials Science*, vol. 5, no. 5, pp. 389–395, 2001.

[169] D. Kashchiev, *Nucleation*. Elsevier, 2000.

[170] E. Betzig, G. H. Patterson, R. Sougrat, O. W. Lindwasser, S. Olenych, J. S. Bonifacino, M. W. Davidson, J. Lippincott-Schwartz, and H. F. Hess, "Imaging intracellular fluorescent proteins at nanometer resolution," *Science*, vol. 313, no. 5793, pp. 1642–1645, 2006.

[171] P. Dhariwal and A. Nichol, "Diffusion models beat GANs on image synthesis," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8780–8794, 2021.

[172] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*, pp. 2256–2265, PMLR, 2015.

[173] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, "Video diffusion models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8633–8646, 2022.

[174] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," *arXiv preprint arXiv:2210.02747*, 2022.

[175] Y. Lipman, M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. T. Chen, D. Lopez-Paz, H. Ben-Hamu, and I. Gat, "Flow matching guide and code," *arXiv preprint arXiv:2412.06264*, 2024.

[176] M. S. Albergo, M. Goldstein, N. M. Boffi, R. Ranganath, and E. Vanden-Eijnden, "Stochastic interpolants with data-dependent couplings," *arXiv preprint arXiv:2310.03725*, 2023.

[177] G.-H. Liu, A. Vahdat, D.-A. Huang, E. A. Theodorou, W. Nie, and A. Anandkumar, "I^2sb: Image-to-image schrödinger bridge," *arXiv preprint arXiv:2302.05872*, 2023.

[178] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and Image Processing*, vol. 14, no. 3, pp. 227–248, 1980.

[179] B. Jähne, *Digital Image Processing*. Springer Science & Business Media, 2005.

[180] B. Justusson, "Median filtering: statistical properties," *Two-Dimensional Digital Signal Processing II: Transforms and Median Filters*, pp. 161–196, 2006.

[181] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.