

**Exploring Cell Behaviour: A  
Computational Tool Pipeline  
Employing Evolutionary Algorithms  
for Classification and Characterisation**

**Mohammad Iqwhanus Syaffa Amir**

Doctor of Philosophy  
University of York  
Physics, Engineering and Technology  
July 2024

# **Acknowledgements**

I extend my deepest appreciation to my principal supervisor, Professor Stephen Smith, whose guidance was invaluable from the outset of my research journey. His profound mentorship has significantly shaped my doctoral study, and for that, I am profoundly grateful. Additionally, I owe a great deal of thanks to my co-supervisor, Professor Jennifer Southgate, for her insightful suggestions and substantial support, particularly in the realm of biological research. My gratitude also extends to Dr. Dawn Walker for her enlightening comments and steadfast encouragement throughout my study. I am equally thankful to Rosalind Duke for her assistance in preparing the samples and instrumentation crucial for our experiments. Lastly, my heartfelt thanks go to my family, and especially to my wife, for their unwavering support and constant encouragement during my years of study and throughout the process of researching and writing this thesis. This achievement would not have been possible without their endless support. Thank you.

## **Declaration of Authorship**

I declare that this thesis is my original work, and I am the sole author. It has not been submitted for any award at this or any other university. I have acknowledged all sources used as references. Where the thesis includes collaborative work, I have explicitly stated what was contributed by others and what was contributed by myself.

## ABSTRACT

The thesis investigates the intricate behaviour of individual cells within a population and their interactions, which are crucial for elucidating processes such as cell proliferation, differentiation, migration, and apoptosis. Due to the complexity of cellular behaviour, machine learning has emerged as an optimal approach for analysing extensive datasets and detecting nuanced patterns in cellular dynamics. The study describes the development of computational tools designed to extract features from individual cells and quantitatively characterise their behaviour in cultured populations using time-lapse microscopy images. Employing an interpretable "white box" machine learning methodology, the research utilises Recurrent Cartesian Genetic Programming (RCGP) and PySR to classify and delineate the behaviour of individual cells under diverse experimental conditions. The experimental methodology involved conducting time-lapse microscopy on Normal Human Urothelial (NHU) cell cultures over 48 hours under three different conditions: control, transforming growth factor-beta (TGF- $\beta$ ), and SB431542 treatment, the latter serving as a TGF- $\beta$  inhibitor. The objective was to assess cellular responses to these specific interventions. Utilising computer vision tools for segmentation, tracking, and feature extraction, quantitative data were collected on cell growth, migration speed, and angular velocity. These data were subsequently employed to develop RCGP and PySR-based computational models that classify and characterise cell behaviour. Comparison with other machine learning technologies, such as Long Short-Term Memory (LSTM), random forests, and Support Vector Machines (SVMs), demonstrated the effectiveness of RCGP models in integrating pairwise classification and ensemble learning (Bagging) techniques. The methods achieved accuracies of 88.59%. The use of PySR to perform classification based on the combination of symbolic regression and classification achieved an accuracy of 92%. Both approaches facilitated a robust performance and capacity for interpreting the resulting classifiers based on the utilised features. In summary, the study emphasises a novel machine learning approach utilising RCGP and PySR to effectively analyse cell behaviour under diverse conditions. The models and mathematical expressions developed could be applied to other biological systems, offering promising avenues for advancing research in systems biology.

## TABLE OF CONTENTS

**TITLE PAGE**

**ACKNOWLEDGEMENT**

**DECLARATION OF AUTHORSHIP**

**ABSTRACT**

**TABLE OF CONTENTS**

**LIST OF TABLES**

**LIST OF FIGURES**

**LIST OF ABBREVIATIONS**

|                                       |           |
|---------------------------------------|-----------|
| <b>CHAPTER 1 INTRODUCTION</b>         | <b>15</b> |
| 1.1 Background                        | 15        |
| 1.1.1 Cell Cultures                   | 16        |
| 1.1.2 Microscopy Imaging              | 17        |
| 1.1.3 Cell Imaging and Analysis       | 17        |
| 1.1.4 Challenges with current methods | 18        |
| 1.2 Aim                               | 19        |
| 1.3 Thesis Objective                  | 19        |
| 1.4 Thesis Contribution               | 19        |
| 1.5 Thesis Overview                   | 20        |
| <b>CHAPTER 2 LITERATURE REVIEW</b>    | <b>22</b> |
| 2.1 Introduction                      | 22        |
| 2.2 Cell Segmentation                 | 22        |
| 2.2.1 Traditional approaches          | 23        |
| 2.2.2 Machine learning approaches     | 29        |
| 2.3 Cell Tracking                     | 38        |
| 2.3.1 Traditional approaches          | 39        |
| 2.3.2 Machine learning approaches     | 42        |
| 2.4 Cell Characterisation             | 47        |
| 2.4.1 Traditional approaches          | 47        |
| 2.4.2 Machine learning approaches     | 50        |
| 2.5 Conclusion                        | 53        |
| <b>CHAPTER 3 METHODOLOGY</b>          | <b>55</b> |

|     |  |     |
|-----|--|-----|
| 3.1 | Introduction   | 55  |
| 3.2 | Cell culture preparation   | 55  |
| 3.3 | Time-lapse microscopy  | 56  |
| 3.4 | Pre-processing image   | 58  |
|     | 3.4.1 Uneven illumination correction                                   | 60  |
|     | 3.4.2 Image normalisation  | 63  |
| 3.5 | Cell segmentation  | 66  |
|     | 3.5.1 Ilastik  | 67  |
|     | 3.5.1.1 Pixel classification   | 68  |
|     | 3.5.1.1.1 The selection of features                                    | 69  |
|     | 3.5.1.1.2 Pixel classification training                                | 72  |
|     | 3.5.1.1.3 Generating the output file                                   | 75  |
|     | 3.5.1.1.4 Batch processing   | 76  |
| 3.6 | Post-processing  | 77  |
| 3.7 | Cell Tracking  | 78  |
|     | 3.7.1 CellProfiler   | 79  |
|     | 3.7.1.1 Data extraction and cell features                              | 82  |
|     | 3.7.1.1.1 Cell proliferation   | 82  |
|     | 3.7.1.1.1.1 Cell growth curves   | 83  |
|     | 3.7.1.1.1.2 Cell growth rates  | 83  |
|     | 3.7.1.1.2 Cell motility  | 84  |
|     | 3.7.1.1.2.1 Cell migration speed                                       | 84  |
|     | 3.7.1.1.2.2 Cell angular velocity                                      | 84  |
| 3.8 | Cell characterisation  | 85  |
|     | 3.8.1 Genetic programming  | 86  |
|     | 3.8.1.1 Recurrent Cartesian genetic programming                        | 86  |
|     | 3.8.1.1.1 Dataset preparation  | 89  |
|     | 3.8.1.1.2 Dataset Normalisation  | 91  |
|     | 3.8.1.1.3 Dataset Splitting  | 92  |
|     | 3.8.1.1.4 Cross-Validation   | 93  |
|     | 3.8.1.1.5 Pairwise Method Approach for RCGP Symbolic<br>Classification | 95  |
|     | 3.8.1.1.6 Ensemble Methods   | 99  |
|     | 3.8.1.1.6.1 Bagging  | 100 |

|   |            |
|---|------------|
| 3.8.1.1.6.2 Boosting  | 101        |
| 3.8.1.1.6.3 Stacking  | 102        |
| 3.8.1.1.6.4 RCGP with Pairwise and Ensemble Approach  | 103        |
| 3.8.1.2 PySR  | 104        |
| 3.8.1.2.1 Core Principles and Genetic Programming Techniques  | 104        |
| 3.8.1.2.2 PySR Customisation feature  | 110        |
| 3.8.1.2.3 Backend Architecture and Performance  | 111        |
| 3.8.1.2.4 Employing a combination of symbolic regression and classification                                 | 111        |
| 3.8.1.2.5 Dataset Preparation and Splitting   | 112        |
| 3.8.1.2.6 Parameters Value for Symbolic Regression  | 113        |
| 3.8.1.2.7 Parameters Value for Symbolic Classification  | 115        |
| 3.8.1.2.8 Cross-Validation and Model Evaluation   | 117        |
| 3.8.1.2.9 Advantages  | 121        |
| 3.9 Conclusion  | 122        |
| <b>CHAPTER 4 RESULT AND ANALYSIS</b>  | <b>123</b> |
| 4.1 Introduction  | 123        |
| 4.2 Image preprocessing   | 123        |
| 4.2.1 Image Uneven illumination correction  | 123        |
| 4.2.2 Image normalisation   | 128        |
| 4.3 Cell segmentation   | 133        |
| 4.4 Cell tracking   | 139        |
| 4.5 Cell growth curve   | 145        |
| 4.6 Cell growth rate  | 146        |
| 4.7 Cell migration speed  | 148        |
| 4.8 Cell angular velocity   | 150        |
| 4.9 Machine Learning Classification   | 152        |
| 4.9.1 RCGP Pairwise with ensemble classification  | 152        |
| 4.9.1.1 Pairwise Classification 1: Cross-Validation results for dataset with Control and SB431542 Treatment | 153        |

|   |            |
|---|------------|
| 4.9.1.2 Pairwise classification 2: Cross validation result for dataset with control and TGF-Beta treatment  | 154        |
| 4.9.1.3 Pairwise classification 3: Cross validation result for dataset with SB431542 and TGF-Beta Treatment | 155        |
| 4.9.1.4 Test result Pairwise classification   | 156        |
| 4.9.1.5 Ensemble Classification   | 158        |
| 4.9.1.5.1 Bagging   | 158        |
| 4.9.1.5.2 Boosting  | 159        |
| 4.9.1.5.3 Stacking  | 161        |
| 4.9.1.5.4 Test result Ensemble classification   | 162        |
| 4.9.1.6 RCGP pairwise classification model  | 163        |
| 4.9.2 PySR Regression with classification   | 167        |
| 4.9.2.1 Symbolic regression   | 168        |
| 4.9.2.2 Symbolic classification   | 175        |
| 4.9.2.3 Symbolic regression and classification model  | 177        |
| 4.9.3 Comparison of Classification with Different Machine Learning Algorithms                               | 185        |
| 4.9.3.1 Test Result Classification with Different Machine Learning Algorithms                               | 190        |
| 5.0 Conclusion  | 191        |
| <b>CHAPTER 5 CONCLUSIONS AND FUTURE WORK</b>  | <b>193</b> |
| 5.1 Introduction  | 193        |
| 5.2 Summary of Key Findings   | 193        |
| 5.3 Contributions to the Field  | 196        |
| 5.4 Recommendations for Future Research   | 197        |
| <b>REFERENCES</b>   | <b>198</b> |
| <b>APPENDIX A</b>   | <b>228</b> |



## LIST OF TABLES

|   |     |
|---|-----|
| Table 1: The summary of traditional approaches used in cell segmentation.   | 26  |
| Table 2: Summary of machine learning approaches used in cell segmentation.  | 34  |
| Table 3: The summary of traditional approaches used in cell tracking.   | 40  |
| Table 4: The summary of machine learning approaches used in cell tracking.  | 45  |
| Table 5: The summary of traditional approaches used in cell characterisation.                                       | 48  |
| Table 6: The summary of machine learning approaches used in cell characterisation.                                  | 51  |
| Table 7: Type of filters and functionality  | 70  |
| Table 8: Gaussian smoothing at $\sigma = 1, 3.5$ and $10$ .   | 71  |
| Table 9: Laplacian of Gaussian at $\sigma = 1, 3.5$ and $10$ .  | 72  |
| Table 10: Hessian of Gaussian Eigenvalues at $\sigma = 1, 3.5$ and $10$ .   | 72  |
| Table 11: Comparison of Features Selections   | 75  |
| Table 12: Cross-Validation for the dataset in control and SB431542 treatment.                                       | 94  |
| Table 13: Cross-Validation for the dataset in control and TGF-Beta treatment.                                       | 94  |
| Table 14: Cross-Validation for the dataset in SB431542 and TGF-Beta treatment.                                      | 95  |
| Table 15: Mean and Standard Deviation of Coefficient of Variation (%) for Different Correction Methods for Well 0.  | 125 |
| Table 16 : Mean and Standard Deviation of Coefficient of Variation (%) for Different Correction Methods for Well 5  | 125 |
| Table 17: Mean and Standard Deviation of Coefficient of Variation (%) for Different Correction Methods for Well 11  | 126 |
| Table 18: Mean and Standard Deviation of Coefficient of Variation (%) for Different Correction Methods for all well | 128 |
| Table 19: Mean and Standard Deviation of CV (%) for Different Normalisation Methods for Well 0                      | 130 |
| Table 20: Mean and Standard Deviation of CV (%) for Different Normalisation Methods for Well 5                      | 131 |
| Table 21: Mean and Standard Deviation of CV (%) for Different Normalisation Methods for Well 11                     | 132 |
| Table 22: Mean and Standard Deviation of CV (%) for Different Normalisation Methods for all well                    | 133 |
| Table 23: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 0                      | 134 |
| Table 24: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 4                      | 135 |
| Table 25: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 8                      | 136 |
| Table 26: Significance test results for different segmentation algorithms on the control environment dataset.       | 138 |
| Table 27: Significance test results for different segmentation algorithms on the TGF-B treatment dataset.           | 138 |
| Table 28: Significance test results for different segmentation algorithms on the SB431542 treatment dataset.        | 139 |
| Table 29: Mean and SD of MOTA (%) for Different Tracking Method for Well 0  | 141 |
| Table 30: Mean and SD of MOTA (%) for Different Tracking Method for Well 4  | 142 |
| Table 31: Mean and SD of MOTA (%) for Different Tracking Method for Well 8  | 142 |
| Table 32: Significance test results for different cell tracking methods on the control environment dataset.         | 144 |
| Table 33: Significance test results for different cell tracking methods on the TGF-Beta treatment dataset.          | 145 |

|  |     |
|--|-----|
| Table 34: Significance test results for different cell tracking methods on the SB431542 treatment dataset.             | 145 |
| Table 35: Summary Statistics for 5-Fold CV Accuracy (%) - Pairwise Classification 1.                                   | 154 |
| Table 36: Summary Statistics for 5-Fold CV Accuracy (%) - Pairwise Classification 2.                                   | 155 |
| Table 37: Summary Statistics for 5-Fold CV Accuracy (%) - Pairwise Classification 3.                                   | 156 |
| Table 38: Test Accuracy (%) for pairwise classification.   | 156 |
| Table 39: Comparison Test Accuracy (%) for different nodes configuration.  | 157 |
| Table 40: Summary Statistics for 5-Fold CV Accuracy (%) for bagging method.  | 158 |
| Table 41: Summary Statistics for 5-Fold CV Accuracy (%) for boosting method.   | 160 |
| Table 42: Summary Statistics for 5-Fold CV Accuracy (%) for stacking method  | 161 |
| Table 43: Test Accuracy (%) for ensemble classification.   | 162 |
| Table 44: Summary Statistics for 5-Fold CV R <sup>2</sup> Mean for Well 0 Symbolic Regression.                         | 168 |
| Table 45: Summary Statistics for 5-Fold CV R <sup>2</sup> Mean for Well4 Symbolic Regression.                          | 171 |
| Table 46: Summary Statistics for 5-Fold CV R <sup>2</sup> Mean for Well 8 Symbolic Regression.                         | 173 |
| Table 47: Summary Statistics for 5-Fold CV Mean Accuracy (%) for Symbolic Classification.                              | 176 |
| Table 48: R <sup>2</sup> Values for Symbolic Regression Models for All Wells.  | 184 |
| Table 49: Summary Statistics for 5-Fold CV Mean Accuracy of LSTM Classification.                                       | 186 |
| Table 50: Summary Statistics for 5-Fold CV Mean Accuracy of Random Forest Classification.                              | 188 |
| Table 51: Summary Statistics for 5-Fold CV Mean Accuracy of SVM Classification.  | 189 |
| Table 52: Comparison of Test Accuracy for Different Classification Methods (PySR, RCGP, LSTM, Random Forest, and SVM). | 191 |

## LIST OF FIGURES

|   |     |
|---|-----|
| Figure 1: Prokaryote cell taken from [10]   | 16  |
| Figure 2: Eukaryote cell taken from [10]  | 16  |
| Figure 3: Example result of NHU cell segmentation. Left: Input image; Right: Final segmentation.  | 22  |
| Figure 4: Workflow diagram for research methodology.  | 55  |
| Figure 5: Layout of a 12-well microplate used in laboratory experiments.  | 56  |
| Figure 6: cell image comparison between brightfield (a) and phase-contrast (b) microscopy [209].  | 57  |
| Figure 7: Olympus time-lapse microscopy setup.  | 58  |
| Figure 8: User Interface of Fiji (ImageJ) showing the main toolbar with icons for image processing functions.   | 59  |
| Figure 9: A macro script to perform image auto threshold.   | 60  |
| Figure 10: BaSiC is an automatic correction method for dynamic time-lapse data. (a) A time-lapse movie corrupted by both shading in space and photobleaching in time. (b) The BaSiC workflow. (c) BaSiC corrects both spatial shading and background over time. | 62  |
| Figure 11: BaSiC interface for background and shading correction.   | 62  |
| Figure 12: Quantile Based Normalisation tool interface.   | 64  |
| Figure 13: Pixel classification workflow in Ilastik.  | 69  |
| Figure 14: Menu for features selection in Ilastik.  | 69  |
| Figure 15: Image labelling.   | 73  |
| Figure 16: Cell Annotation Areas for Model Training.  | 73  |
| Figure 17: Image after training process   | 74  |
| Figure 18: The prediction export tab in Ilastik displays export options and a preview of the segmented image.   | 75  |
| Figure 19: The Batch Processing tab in ilastik.   | 76  |
| Figure 20: Auto Threshold Settings in ImageJ .  | 77  |
| Figure 21: Cell tracking workflow in CellProfiler.  | 78  |
| Figure 22: CellProfiler Interface.  | 79  |
| Figure 23: CellProfiler TrackObjects module interface.  | 81  |
| Figure 24: Cell Growth Curve [229].   | 83  |
| Figure 25: Example of CGP structure with inputs (0, 1, 2), functional nodes (F0, F2, F1), and output. Shows node reuse and inactive nodes for neutral genetic drift [235].  | 88  |
| Figure 26: Example of an RCGP structure with inputs (0, 1, 2), functional nodes (F2, F0, F1), and output. Shows recurrent connections, including a node connecting to itself [235].   | 88  |
| Figure 27: Pairwise Approach Using Recurrent Cartesian Genetic Programming implementation.  | 96  |
| Figure 28: Flowchart of the RCGP Program.   | 98  |
| Figure 29: Flowchart of the Evolutionary Strategy (ES) Process.   | 100 |
| Figure 30: Bagging process in Ensemble Learning.  | 101 |
| Figure 31: Boosting process in Ensemble Learning.   | 102 |
| Figure 32: Stacking process in Ensemble Learning  | 103 |
| Figure 33: RCGP with pairwise and ensemble methods.   | 104 |
| Figure 34: Block diagram of Symbolic Regression and Classification in the PySR Framework.   | 112 |
| Figure 35: Coefficient of Variation (%) Comparison Across Different Correction Methods for Well 0   | 124 |
| Figure 36: Coefficient of Variation (%) Comparison Across Different Correction Methods for Well 5   | 125 |
| Figure 37: Coefficient of Variation (%) Comparison Across Different Correction Methods for Well 11  | 126 |

|  |     |
|--|-----|
| Figure 38: Coefficient of Variation (%) Comparison Across Different Correction Methods for all well  | 127 |
| Figure 39: Images and histograms for image frame 1 before and after normalization. (a) Image before normalization, (b) Histogram corresponding to the image before normalization, (c) Image after normalization, (d) Histogram corresponding to the image after normalisation    | 129 |
| Figure 40: Images and histograms for image frame 575 before and after normalization. (a) Image before normalization, (b) Histogram corresponding to the image before normalization, (c) Image after normalization, (d) Histogram corresponding to the image after normalisation. | 129 |
| Figure 41: Coefficient of Variation (%) Comparison Across Different Normalisation Methods for Well 0.  | 129 |
| Figure 42: Coefficient of Variation (%) Comparison Across Different Normalisation Methods for Well 5   | 130 |
| Figure 43: Coefficient of Variation (%) Comparison Across Different Normalisation Methods for Well 11  | 131 |
| Figure 44: Coefficient of Variation (%) Comparison Across Different Normalisation Methods for all well   | 132 |
| Figure 45: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 0.  | 134 |
| Figure 46: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 4   | 135 |
| Figure 47: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 8   | 136 |
| Figure 48: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for all well   | 137 |
| Figure 49: MOTA (%) Comparison Across Different Tracking Methods for Well 0.   | 140 |
| Figure 50: MOTA (%) Comparison Across Different Tracking Methods for Well 4  | 141 |
| Figure 51: MOTA (%) Comparison Across Different Tracking Methods for Well 8  | 142 |
| Figure 52: MOTA (%) Comparison Across Different Tracking Methods for all well  | 143 |
| Figure 53: Cell Growth Curve Across Different Treatments   | 146 |
| Figure 54: Cell Growth Rate Comparison Across Different Wells Under Various Treatments.  | 147 |
| Figure 55: Cell Growth Rate Comparison Across Different Treatments with Statistical Significance   | 148 |
| Figure 56: Mean Migration Speed Over Time for Different Treatment.   | 149 |
| Figure 57: Mean Migration Speed Comparison Across Different Treatments with Statistical Significance.  | 149 |
| Figure 58: Mean Angular Velocity Over Time for Different Treatments.   | 151 |
| Figure 59: Mean Angular Velocity Comparison Across Different Treatments with Statistical Significance.   | 152 |
| Figure 60: 5-Fold CV Accuracy (%) for Different Nodes - Pairwise Classification 1.   | 153 |
| Figure 61: 5-Fold CV Accuracy (%) for Different Nodes - Pairwise Classification 2.   | 154 |
| Figure 62: 5-Fold CV Accuracy (%) for Different Nodes - Pairwise Classification 3.   | 156 |
| Figure 63: 5-Fold CV Accuracy (%) for different estimator value - bagging method.  | 158 |
| Figure 64: 5-Fold CV Accuracy (%) for different estimator value - boosting method.   | 159 |
| Figure 65: 5-Fold CV Accuracy (%) for different estimator value - stacking method.   | 161 |
| Figure 66: Model classification for dataset control and SB431542 classification.   | 163 |
| Figure 67: Model classification for dataset control and TGF-Beta classification.   | 165 |
| Figure 68: Model classification for dataset SB431542 and TGF-Beta classification.  | 166 |
| Figure 69: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 0.  | 168 |
| Figure 70: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.870$ ).  | 169 |

|  |     |
|--|-----|
| Figure 71: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 4.  | 170 |
| Figure 72: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.867$ ).  | 171 |
| Figure 73: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 8.  | 172 |
| Figure 74: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.878$ ).  | 174 |
| Figure 75: $R^2$ value distribution for Control, TGF-B, and SB431542 treatments in symbolic regression on the test dataset | 175 |
| Figure 76: Box Plot of classification Model 5-Fold Cross-Validation Results.   | 176 |
| Figure 77: 3D Surface Plot of Predicted Cell Growth vs. Migration Speed and Time. For Well 0                               | 178 |
| Figure 78: 3D Surface Plot of Predicted Cell Growth vs. Migration Speed and Time for Well 4.                               | 180 |
| Figure 79: 3D Plot of Predicted Cell Growth vs. Migration Speed and Time for Well 8.                                       | 182 |
| Figure 80: LSTM Box Plot of Classification Model 5-Fold Cross-Validation Results.  | 186 |
| Figure 81: Random Forest Box Plot of Classification Model 5-Fold Cross-Validation Results.                                 | 187 |
| Figure 82: Support Vector Machine Box Plot of Classification Model 5-Fold Cross-Validation Results.                        | 189 |

## LIST OF ABBREVIATIONS

|            |  |
|------------|--|
| BFGS       | Broyden-Fletcher-Goldfarb-Shanno                 |
| CDP-ODLSSA | Optimal Deep Learning with Salp Swarm Algorithm  |
| CGP        | Cartesian genetic programming                    |
| CLAHE      | Contrast Limited Adaptive Histogram Equalization |
| C-LSTM     | Convolutional Long Short Term Memory             |
| CNN        | Convolution Neural Network                       |
| CSV        | Comma-separated value                            |
| DeLTA      | Deep Learning for Time-lapse Analyse             |
| DMU        | Dynamic memory units                             |
| DNA        | Deoxyribonucleic Acid                            |
| DoG        | Different of Gaussian                            |
| EDT        | Euclidean distance transform                     |
| ES         | Evolutionary strategy                            |
| FFN        | Feedforward network                              |
| FFT        | Fast Fourier Transform                           |
| FFTF       | Fast Fourier Transform Filter                    |
| FN         | False negative                                   |
| FP         | False positive                                   |
| GGM        | Gaussian Gradient Magnitude                      |
| GP         | Genetic programming                              |
| IQR        | Interquartile range                              |
| LAP        | Linear Assignment Problem                        |
| LAP        | Linear Assignment Problem                        |
| LDA        | Linear Discriminant Analysis                     |
| LoG        | Laplacian of Gaussian                            |
| LOOCV      | Leave-One-Out Cross-Validation                   |
| LSTM       | Long Short Term Memory                           |
| LSTM-AE    | Long Short Term Memory-Autoencoder               |
| Mask R-CNN | Mask regional convolutional neural network       |

|              |  |
|--------------|--|
| MCW          | Marker-Controlled Watershed                        |
| MODM         | Multi-object dynamic memory network                |
| MOTA         | Multiple Object Tracking Accuracy                  |
| MPCS         | Membrane Pattern-based Cell Segmentation           |
| MSC          | Mesenchymal stem cell                              |
| NHU          | Normal Human Urothelial                            |
| NLC          | Normalize Local Contrast                           |
| OOB          | Out-of-bag   |
| PFFC         | Pseudo Flat Field Correction                       |
| PR-GLS       | Point Set Registration - Generalized Least Squares |
| PySR         | Python Symbolic Regression                         |
| QBN          | Quantile-Based Normalization                       |
| RCGP         | Recurrent Cartesian Genetic Programming            |
| RCNN         | Region-Based Convolutional Neural Network          |
| RF           | Random Forest                                      |
| SSA          | Salp Swarm Algorithm                               |
| SVM          | Support vector machine                             |
| TGF- $\beta$ | Transforming Growth Factor beta                    |
| TNT          | Titanium dioxide nanotube                          |
| TWS          | Trainable Weka Segmentation                        |

# CHAPTER 1

## INTRODUCTION

### 1.1 Background

Research in cell biology specifically deals with cell interaction, functionality and structures. It is based on the understanding that cells contribute primarily to scientific fields such as genetics, biochemistry, neuroscience, and immunology [1][2]. Furthermore, in order to understand diseases in the medical area, research in cell biology also benefits in diagnosing, treating, and preventing human diseases [3]. This research also includes cell composition, metabolism, communication, and cell cycle (proliferation), and it will give the researcher a better understanding of cell functionality in the organism's development [4]. Additionally, studies in cell biology will benefit the fields of medicine, such as tumours, cancer, and other diseases, as well as regenerative medicine and tissue engineering [5].

Cells are the essential building elements of all living things, serving structural and functional purposes. Prokaryotic [6] and eukaryotic [7] are two primary types of cells, illustrated in Figure 1 and Figure 2, respectively. Bacteria are prokaryotic cells, also known as unicellular, that lack a nucleus and other membrane-bound organelles. In comparison, eukaryotic or multicellular cells have a nucleus and various organelles, such as plants, animals, fungi and protists. Although each cell type has differing functions and structures, in terms of cell membranes, cytoplasm and genetic material, they have similar characteristics [8]. In order to study these characteristics of cells, a tool like microscopy is usually used [9], which is explained below.



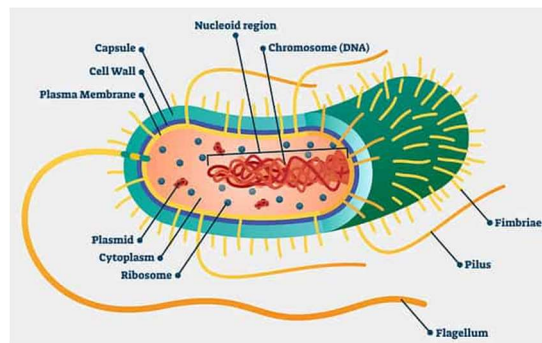


Figure 1: Prokaryote cell taken from [10]

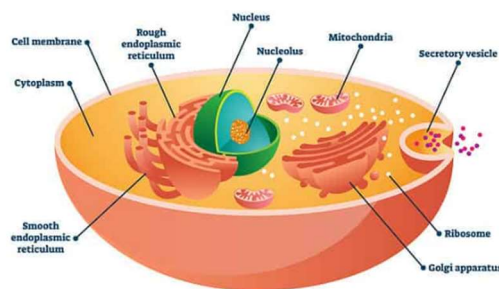


Figure 2: Eukaryote cell taken from [10]

### 1.1.1 Cell Cultures

Cell cultures refer to growing cells under controlled conditions, typically outside their natural environment [11]. This lets scientists see how cells normally work and what happens to them when they get exposed to different conditions, such as medicines or germs. They can do this by studying the cells in isolation, outside of a whole animal or plant [12]. Since the 1900s, growing cell cultures in labs has become important for biomedical research as it helps scientists understand how cells work, their genetics, and their role in disease [13]. Cells can be cultured from many sources, including plant, animal, or human tissues [14] and can be classified as primary or secondary, continuous or immortalised depending on the type and purpose [15]. Over the years, the practice of creating cell cultures has evolved with advances in media formulation, environmental control, and contamination prevention [11]. This has allowed for the maintenance of a homogeneous cell population and facilitated standardised experiments [14]. As cell culture became more prevalent, the need for detailed visualisation of these cultured cells intensified, leading to the integration of microscopy techniques [16].

### **1.1.2 Microscopy Imaging**

A microscopy image [17] is a digital or physical image obtained through the use of a microscope [18], a practical tool that allows scientists to visualise objects at a microscopic level. Microscopy images are useful for many purposes such as studying cellular structures, discovering pathogens, and examining materials at a nanoscale level. In biology, researchers utilise microscopy images to investigate cell morphology, organelle function, and cellular processes [19]. In medicine, these images are used for diagnosing diseases, monitoring treatment effectiveness, and conducting research on numerous health conditions [20]. Different types of microscopies are available for particular applications. For example, in biology, light microscopy [21] is usually used to observe live cells and tissues, while electron microscopy [19] offers high-resolution images of subcellular structures. Additional techniques, such as fluorescence microscopy, enable the visualisation of specific proteins and molecules within cells, and confocal microscopy provides detailed images of thick specimens through optical sectioning. These different microscopy technologies enable researchers to undertake a complementary analysis of cell culture to facilitate their understanding.

### **1.1.3 Cell Imaging and Analysis**

Cell imaging and analysis are the integration of hardware, software tools and biological methods where the software part involves computational tools, whereas the hardware part uses optical technology. For instance, advanced imaging techniques such as fluorescence microscopy allow researchers to tag specific cellular components and observe them in real-time [22]. Coupled with computational tools like the colour segmentation approach, these techniques enable the quantification of cellular responses, understanding signalling pathways, and monitoring cellular health [23]. One of the technologies that implement high-resolution and super-resolution imaging techniques has further expanded cell imaging capabilities, allowing scientists to visualise dynamic behaviours at a very small scale [24]. Additionally, image analysis methods have been developed to determine the type of a cell based on feature parameters extracted from cell images [25]. Imaging and analysis in cell areas have helped study cellular biology, providing valuable insights into the complex processes in living cells. Similarly, in biology and medicine, implementing cell segmentation [26], tracking [27], and characterisation [28][29] is fundamental to understanding how the cell works. Generally, cell segmentation is a process that involves computer vision algorithms to differentiate cells from

other structures in the image and its background [30]. Additionally, cell tracking facilitates scientists to observe and record how a cell moves and changes over time [31]. Imagine marking one object in a crowd and examining the object's direction and pattern. This technique helps understand cell movement [32], growth [33], and how the disease spreads and wounds heal [34].

In the context of cell characterisation, this process involves analysis, such as quantification of the cell's physical [35], structural [36], behaviour [37], biological functions [38], and composition of the molecule [39]. The statistical approach is one of the methods used to characterise the cell, but with the advancement of technology, the implementation of machine learning expands the complexity of cell analysis to uncover new patterns or relationships that traditional methods are not able to detect [40]. All the pipeline processes above benefit the researcher and scientist to explore and recognise how cells behave and change in diseases and respond to treatment. Furthermore, this knowledge is important for the new development of medicines and therapies to treat diseases [41][42][43].

### **1.1.4 Challenges with current methods**

Biological and medical studies have been furthered by major developments in cell imaging and analysis, and have provided new understandings of the behaviour and functionality of cells [44][45]. Nevertheless, these methods also have problems and restrictions. Image data integrity can be jeopardised by factors like sample preparation techniques [46], staining inconsistencies [47], and equipment limitations [48]. Additionally, significant computing power and resources are required for image analysis, particularly when dealing with large datasets [49]. A major technical challenge is accurately segmenting [50][51], tracking [52], and characterising [53][54] cells within intricate biological environments or dynamic developmental stages. This complexity intensifies when data from diverse sources is integrated or captured using different imaging methods [55][56]. Furthermore, Data extraction from high-throughput images needs advanced algorithmic techniques and in-depth knowledge of biology [57][58]. For example, accurate data interpretation is important for discovering new cellular phenomena and understanding biological processes [59]. Data integration also contributes to the challenge [60]. For instance, cohesively analysing data from different modalities of image and external data sources is necessary for producing wide cellular function

and disease studies [61]. This integration is needed to advance personalised medicine and biotechnologies and also involves a broad understanding of how cells react to different conditions [62]. Realising the full potential of cellular research requires addressing these issues and continuing to develop algorithms and computer technologies to improve imaging and analytical techniques [63][64].

## **1.2 Aim**

The aim of the research described in this thesis is to develop a computational tool pipeline that quantifies cell behaviour from time-lapse microscopy images, enabling an objective understanding of cell culture across different conditions and environments.

## **1.3 Thesis Objectives**

This thesis will present a set of objectives that are based on the motivations described and the information presented in the literature review.

1. To establish a pipeline of computer vision tools to reliably segment and track unlabelled cells grown in culture as adherent monolayers using normal human urothelial cells as a representative example.
2. To use a pipeline to describe the variations in growth, migration speed, and angular velocity.
3. To characterise the behaviour leading to changes, the differences in cell populations.
4. To investigate the use of the features within an interpretable “white box” machine learning context to further characterise NHU cell behaviours.

## **1.3. Thesis Contributions**

The contributions of the thesis are as follows:

1. Development of a Computational Tools pipeline: Creating and validating reliable computational tools for accurate cell segmentation and tracking specific to analysing adherent monolayers of unlabelled cells.

2. Segmentation and Tracking Methodology: Introducing practical methods and improvements to existing segmentation techniques for unlabelled cells that enhance the precision and reliability of cell segmentation and tracking.
3. Characterisation of Cell Behaviour Variability: Identifying and analysing features describing variations in cell growth, migration speed, and angular velocity of NHU cells in an objective manner not previously achieved.
4. Insights into Cellular Responses to Treatments: comparative study of cell behaviours under different treatment conditions, such as control, TGF- $\beta$  and SB431542, provides valuable insights for future research on cell responses to compounds.
5. Application of interpretable “white box” machine learning: a novel approach, implementing Genetic Programming to characterise NHU cell behaviours that can reveal patterns and relationships in cell behaviour data that might not be evident through traditional analysis methods.

#### **1.4 Thesis overview**

This thesis consists of five chapters. Chapter 1 has presented a general introduction, background study, aim, objective and contribution. Chapter 2 presents a review of the literature on studies related to cell segmentation, tracking, and characterisation using traditional and machine-learning approaches. Chapter 3 provides the methodology for the research, detailing cell culture preparation, time-lapse microscopy and subsequent image analysis, including pre-processing, the segmentation of cells, and post-processing. Methods for cell tracking, data extraction, cell feature parameters and the implementation of computational intelligence are also considered.

The results and discussions of the overall investigation are presented in Chapter 4. It begins with images and data related to cell segmentation and cell tracking and then considers cell features, including the total number of cells, their growth curve, migration speed, and angular velocity. In the latter part of the discussion, the analysis concentrates on characterising cells using machine learning approaches to perform classification and symbolic regression methods. The chapter also includes comparison results between different algorithms, methods, and machine-learning techniques. The conclusion and future work are addressed in Chapter 5, starting with summaries of work done, drawing conclusions based on the results, and proposing several possible study progressions. Appendix A enhances the main text by including additional

figures and detailed results that expand upon the analyses discussed in Chapter 4. This supplementary material provides a deeper understanding of cell segmentation performance, cell tracking performance, and symbolic regression analyses across various experimental conditions. Figures A.1 to A.9 in Appendix A offer detailed results on cell segmentation, illustrating how cells are differentiated and analysed across different experimental wells. Figures A.10 to A.18 depict cell tracking performance, providing insights into the movement and behaviour of cells under different treatment conditions. Lastly, figures A.19 to A.45 display the outcomes of symbolic regression analyses, which help predict cell growth patterns across the range of experimental setups.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1. Introduction

The aim of this chapter is to review contemporary cell segmentation, tracking and characterisation algorithms used for cell culture in biology applications. The literature review starts with a review of existing segmentation algorithms, which are fundamental for identifying cell morphology in complicated images. The review will then consider cell tracking methods that monitor cell movements over time, followed by an analysis of algorithms that characterise cell features, which are essential for understanding cell behaviour within cell culture. Each of the algorithms will be reviewed based on its application through either traditional or machine-learning approaches.

#### 2.2 Cell segmentation

One of the essential image processing stages is performing image segmentation. The image is separated into areas or objects based on the comparable characteristics that are defined for each class. The aim is to achieve isolation of the cell of interest. The ability to differentiate between the cell of interest and other cells in the background is the prime objective of this procedure. As an example, Figure 3 shows Normal Human Urothelial (NHU) cell segmentation from a brightfield microscopy image, demonstrating the complexity of this task.

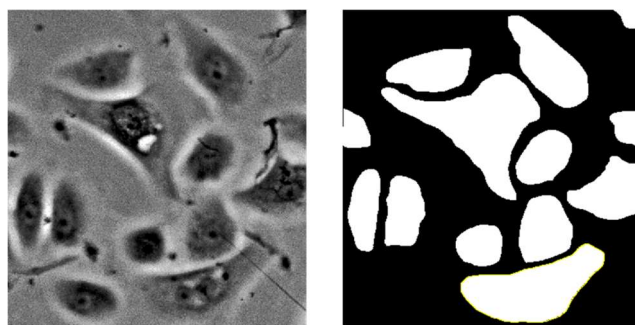


Figure 3: Example result of NHU cell segmentation. Left: Input image; Right: Final segmentation.

### 2.2.1. Traditional approaches

In the review of traditional cell segmentation methods, popular approaches are group according to intensity [65], edge [66] and region-based [67] processing. Intensity in segmentation refers to the brightness levels of pixels in an image. Examples of intensity-based approaches include thresholding [68] and histogram-based [69] methods. Edge-based segmentation focuses on identifying the boundaries between different regions, for example, Canny [70], Sobel [71], and Laplacian of Gaussian algorithms [72]. While region-based segmentation refers to the technique of dividing an image into multiple segments based on similarities between adjacent pixels. This technique focuses on finding homogeneous regions within an image to facilitate accurate object detection and segmentation, such as watershed algorithms [73], region growth [74], and morphological operations [75].

Cell segmentation was undertaken by Chang et al. [76] using a double thresholding technique combined with morphology-based operations to address the challenges in medical image analysis which are often dictated by noisy images with poorly defined structural boundaries. The proposed method employs a two-stage thresholding process—a global threshold to map the original image into a binary matrix representing potential cell locations, followed by an adaptive threshold applied to each connected component to improve the segmentation. This double thresholding approach effectively addresses the variability in cell brightness and contrast. Then, morphological opening processes are used to split cells that are overlapping and make the edges of cells smoother. In areas with low contrast, this method works well to keep cell boundaries intact and to deal with the typical problem of cell merging.

A modified histogram-based thresholding method was used by Kheirkhah et al. [77] to accurately identify sperm cells in microscopic pictures. The picture histogram is first adjusted in a non-linear pre-processing phase to improve the contrast between the sperm cells and the background, increasing the bimodality of the histogram. After pre-processing, a Kittler-based thresholding algorithm is used to the improved histogram. This method minimises the overlap between two Gaussian distributions derived from the histogram data in order to distinguish the foreground (sperm cells) from the background. The approach involves a post-processing step where morphological techniques are used to improve the segmentation after thresholding. In order to provide a cleaner and more accurate segmentation outcome, this stage helps remove noise and artefacts as well as small objects.



The research developed by Ramalho et al. [78] utilised a sophisticated clustering algorithm paired with a Voronoi diagram to segment cell gaps and incorporate with the Canny edge detection algorithm to obtain more precise cell boundaries. The objective of this research is to enhance the automation of cervical cytology analysis by addressing common problems that hinder conventional approaches, such as staining variance and cell occlusion. Through the integration of these computational methodologies, the segmentation accuracy for identifying nuclei and cytoplasm within cell structures was substantially enhanced.

An empirically determined model is used for threshold estimation in the empirical gradient threshold approach reported by Chalfoun et al. [79]. The technique was explained as being applicable to several microscopic modalities, including raw pictures, and working with PC, DIC, brightfield, and fluorescence. The algorithm uses the Sobel operator to get the gradient's absolute value. Next, it finds the threshold based on the percentile, and finally it performs binary morphological operations. Prior to use, three parameters need to be established: minimum cell size (which eliminates small items), minimal hole (which eliminates small holes), and manual fine-tuning (which modifies the predicted threshold). The training pictures' ground true mask was used to calculate the smallest item size for each of these approaches.

A generalised Laplacian of Gaussian (gLoG) filter is the basis of an automated method for nuclei recognition in digitised histology images, as shown by Xu et al. [80]. Initially, a bank of gLoG kernels with various sizes and orientations is produced using the suggested approach. It then obtains a collection of response maps by performing convolution between the candidate image and directional gLoG kernels. A mean-shift technique is utilised to identify and categorise the local maxima of response maps according to their geometrical proximity. The nucleus seed is ultimately chosen from each group based on the maximal response point. According on experimental results on two datasets, the suggested method outperforms current methods for detecting nuclei.

Rahali et al. [81] introduced an improved Marker-Controlled Watershed (MCW) segmentation approach for *Drosophila* (fly) cells. They developed two novel forms of foreground markers: kernels and object-marked point process markers. Kernels produced with the Fiji software were used to identify the core region of each cell, ensuring that each cell was captured clearly. The more advanced object-marked point process, established makers inside the object point process framework, included cell properties such as form, radiometry, and

contrast. This addition considerably improved segmentation accuracy by addressing concerns like intensity heterogeneity and cell clustering, both of which are frequent obstacles in classical watershed segmentation. The efficiency of this approach was evaluated using quantitative measures such as Dice coefficients and F1 scores, which revealed that the MCW with object-marked point process markers outperformed standard methods, particularly in noisy situations.

Stoklasa et al. [82] developed an enhanced approach for automatically segmenting cells in phase-contrast microscopy images, emphasising the difficult task of dealing with crowded cells. The authors provided a novel approach for accurately delineating cell borders that combines superpixel classification with region-growing approaches. This method consists of a multi-phase process that begins with image processing to improve the visibility of internal cell structures, followed by a classification phase to identify cell boundaries, and culminates in an information fusion phase that combines both sets of data to create accurate geometric models of each cell. An advanced morphological image segmentation approach was presented by Zhang et al. [83], specifically designed to address the problem of accurately segmenting circular overlapping cells in biomedical imaging. The method uses morphological processes including erosion, dilatation, opening, and closure to identify division locations between overlapping cells efficiently. It focuses on angles and forms that indicate cell overlap. Often problematic for standard approaches, this strategy works well for mildly to moderately overlapping cells and minimises noise interference and accurately preserves cell structural elements, which are essential for reliable cell analysis and counting in medical research.

An automatic cell segmentation approach, developed by Dimopoulos et al. [84], introduces a novel technique called Membrane Pattern-based Cell Segmentation (MPCS), improving the accuracy of identifying cell boundaries in microscopy images. The method is important for quantitative single-cell biology because it uses distinctive intensity patterns across cell membranes to improve border identification. MPCS solves common problems in microscope image analysis, like cells with irregular shapes, differences in how they look, and a lot of image noise, by using graph cuts and directed cross-correlations. The process begins with the establishment of parameters that are intuitive to biological systems, followed by the identification of potential locations within the cells referred to as "seeds" and the application of a graph cut technique that uses cross-correlation operations. This combination interprets the unique membrane pattern information, ensuring that each cell is separated from the background.

According to Chalfoun et al. [85] , an automatic segmentation method known as FogBank was introduced, which accurately separates cells when confluent and touching each other. The method is based on morphological watershed concepts and has two novel elements to improve accuracy and reduce over-segmentation. At its most basic level, FogBank quantifies pixel intensities by employing histogram binning, which reduces the amount of image noise that results in over-segmentation. In addition, FogBank detects the forms of individual cells using a geodesic distance mask produced from raw images, as opposed to other watershed-like algorithms, which yield more linear cell edges.

Magnusson et al. [86] built the Baxter algorithm tool to perform segmentation of cell biology images by computing the standard deviation of the pixels in a small, square region or “window” around every pixel and then thresholding the region based on the standard deviation of the pixel. In order to fill in the gaps in the binary segmentation mask, morphological erosion is employed to compensate for the spread produced by the large size of the square region, and small items are removed to limit the number of false-positive detections. A watershed transformation to the standard deviation image to split cells into individual cells can then be performed. The watershed transform is restricted to the foreground pixels of the binary segmentation mask, and to minimise over-segmentation, some Gaussian smoothing and H-minima transforms are applied prior to computing the watershed transform. However, Versari et al. [87] introduced CellStar, a set of tools that can do the segmentation process by using a new modification of active rays that makes use of data about the depths of contours. Active rays, also known as polar active contours, are a computational framework for recognising object outlines. The contour extraction problem is described as an energy minimisation problem, and contours are members of a parametric curve’s family. The summary of published cell segmentation methods reviewed utilising traditional techniques is presented in Table 1.

Table 1: The summary of traditional approaches used in cell segmentation.

| Author                      | Method   | Limitation  |
|-----------------------------|--|---|
| Chang et al. , 2018<br>[76] | Two stage thresholding technique combined with morphology-based operations | <ul style="list-style-type: none"> <li>● Over-segmentation [124]</li> <li>● Under-segmentation [125]</li> <li>● Manual parameter tuning. [125]</li> </ul> |

|                               |  |  |
|-------------------------------|--|--|
| Kheirkhah et al. , 2017 [77]  | Histogram-based thresholding   | <ul style="list-style-type: none"> <li>● Inaccurate segmentation in complex images (shapes or overlap) [126]</li> <li>● Sensitivity to intensity distributions [127]</li> <li>● Manual parameter tuning [128]</li> </ul>         |
| Ramalho et al. , 2015 [78]    | Voronoi diagram and Canny edge detection algorithm                           | <ul style="list-style-type: none"> <li>● Sensitivity to intensity distributions. [129]</li> <li>● Lack of adaptability to complex cell shapes. [129]</li> </ul>  |
| Chalfoun et al. , 2015 [79]   | Empirical gradient threshold and Sobel operator                              | <ul style="list-style-type: none"> <li>● Sensitivity to intensity gradients. [79]</li> <li>● Manual parameter tuning. [79]</li> </ul>  |
| Xu et al. , 2017 [80]         | Laplacian of Gaussian (gLoG) and mean-shift technique                        | <ul style="list-style-type: none"> <li>● Lack of adaptability to complex cell shapes. [130]</li> <li>● Sensitivity to noise and artefacts. [130]</li> <li>● Potential for over-segmentation. [130]</li> </ul>                    |
| Rahali et al. , 2022 [81]     | Marker-Controlled Watershed  | <ul style="list-style-type: none"> <li>● Over-segmentation [131]</li> <li>● Sensitivity to markers [131]</li> <li>● Sensitivity to intensity variations [132]</li> </ul>   |
| Stoklasa et al. , 2015 [82]   | Region-growing approaches  | <ul style="list-style-type: none"> <li>● Sensitivity to initial seed points [132]</li> <li>● Potential for over-segmentation [132]</li> </ul>  |
| Zhang et al. , 2022 [83]      | Morphological processes, including erosion, dilatation, opening, and closure | <ul style="list-style-type: none"> <li>● Sensitivity to Structuring Element Size [84]</li> <li>● Limited to Lightly or Moderately Overlapping Cells [83]</li> <li>● Designed for Circular/Elliptical Cell Shapes [83]</li> </ul> |
| Dimopoulos et al. , 2014 [84] | Graph cut technique that uses cross-correlation operations                   | <ul style="list-style-type: none"> <li>● Dependence on Image Quality [133]</li> <li>● Parameter Selection [134]</li> <li>● Separation of Touching Cells [135]</li> </ul>   |

|                                 |  |   |
|---------------------------------|--|---|
|                                 |  | <ul style="list-style-type: none"> <li>● Computational Complexity [136]</li> </ul>  |
| Chalfoun et al. ,<br>2014 [85]  | Employing histogram binning and geodesic distance mask   | <ul style="list-style-type: none"> <li>● Sensitivity to Noise and Artifacts [137]</li> <li>● Resolving Overlapping Cells at Different Depths [138]</li> </ul>                 |
| Magnusson et al. ,<br>2015 [86] | Thresholding, morphological erosion and watershed (Gaussian smoothing and H-minima transforms) | <ul style="list-style-type: none"> <li>● Loss of Important Features [139]</li> <li>● nonuniform intensity [140]</li> <li>● Sensitive to noise [140]</li> </ul>                |
| Versari et al. ,<br>2017 [87]   | Polar active contours  | <ul style="list-style-type: none"> <li>● Handling Irregular Cell Shapes [141]</li> <li>● Sensitivity to Initialization [142]</li> <li>● Tuning of Parameters [143]</li> </ul> |

The summary table 1 outlines various traditional approaches to cell segmentation, each with its own set of strengths and limitations. A critical analysis of these techniques provides insight into which methods are most suitable for the specific requirements of the current study. Thresholding techniques (e.g., Chang et al., 2018; Kheirkhah et al., 2017) are commonly used due to their simplicity and efficiency. However, they suffer from issues such as over-segmentation, under-segmentation, and high sensitivity to intensity variations. Given these drawbacks, especially in cases where cell shapes are complex or overlapping, thresholding techniques may not be ideal for the primary segmentation objectives of this study. Edge detection and gradient-based methods (e.g., Ramalho et al., 2015; Chalifour et al., 2015) offer greater robustness in detecting cell boundaries but are still prone to issues such as sensitivity to intensity gradients and the need for manual parameter tuning. The challenges in handling complex cell shapes and the need for adaptability make these methods less favourable for the specific requirements of the research. Region-growing and watershed approaches (e.g., Stoklasa et al., 2015; Rahali et al., 2022) can effectively segment more complex images, but they tend to over-segment and are highly sensitive to initial conditions or marker selection. These limitations are particularly significant in a study dealing with varied and complex cell shapes, making these approaches less ideal. Morphological processes (e.g., Zhang et al., 2022) combine morphological operations with other methods, offering a more refined segmentation

process. However, their effectiveness is limited to certain types of cell overlaps, and they are sensitive to the size of the structuring element used, which could restrict their applicability in broader scenarios. Advanced techniques (e.g., Dimopoulos et al., 2014; Chalifour et al., 2014; Versari et al., 2017), such as graph cuts and active contours, are more sophisticated and capable of handling complex and irregular cell shapes. However, these methods require careful parameter tuning and are computationally intensive. Despite these challenges, their ability to separate touching cells and handle irregular shapes makes them promising candidates for the current study, provided that the necessary computational resources are available. After carefully analysing these techniques, thresholding methods, despite their limitations, have been chosen for this study but will be used primarily for post-processing the segmented images. Thresholding is selected because it effectively simplifies the segmentation process and enhances the clarity of boundaries when combined with other methods. Although it is not ideal as a standalone technique due to its issues with complex or overlapping cells, its application in post-processing can refine segmentation results by removing noise and ensuring more distinct cell boundaries. This makes it a suitable choice when paired with more robust segmentation methods, thereby addressing the specific challenges of the study.

### **2.2.2. Machine learning approaches**

Machine learning is a field of artificial intelligence that focuses on developing algorithms and statistical models to enable computer systems to learn from data and make decisions or predictions without being explicitly programmed [88]. The main categories of machine learning include supervised learning, unsupervised learning, and reinforcement learning, with additional categories like semi-supervised learning, transduction, and learning to learn [88]. In the context of cell segmentation, machine learning techniques have improved segmentation tools that are capable of learning from previously obtained training datasets and enhancing performance without the need for specific programming or parameter adjustment. FastER (fast segmentation with extremal regions) is a fast and trainable tool for cell segmentation developed by Hilsenbeck et al. [89]. It extracts shape and texture characteristics from cell areas, predicts the cell by implementing a support vector machine (SVM) [90], and estimates an optimal set of non-overlapping cell regions by applying the divide and conquer method. It supports a wide range of cell types and image acquisition modalities.

Arganda-Carreras et al. [91] announced the Trainable Weka Segmentation (TWS) tool, a machine learning technique that uses a small number of manual annotations to train a classifier and automatically segment the remaining data. TWS converts the segmentation problem to a pixel classification problem, in which each pixel is classified as belonging to a specific segment or class. A labelled set of input pixels is represented in a feature space and then used as the training set for a particular classifier. Once trained, the classifier can be used to classify either the remaining input pixels or entirely new image data. Additionally, TWS may provide unsupervised segmentation learning techniques (clustering) and can be adjusted to use custom image features or classifiers. Hernandez et al. [92] employed DeepCell, a Fully Convolutional Network developed for cell segmentation. Each pixel is classified as belonging to one of three categories: the cell interior, the cell boundary, or the image background. Good cell segmentation accuracy is achieved using this network, which is robust enough to adapt to a variety of experimental settings and requires relatively minimum training and manual labelling of ground-truth data.

An interactive tool, Ilastik, developed by Berg et al. [93], aims to make machine learning-based bioimage analysis accessible to those users who do not have extensive computer skills. It includes pre-defined procedures for image segmentation, object classification, and counting. Ilastik performs image segmentation by utilising pixel classification - in other words, it assigns a user-defined class name to each pixel in the image. The pixel classification makes use of image filters as features and a random forest as a classifier. Filters in 2D and 3D comprise a descriptor of pixel colour and intensity, edges, and texture. The pixel classification procedure uses semantic segmentation rather than instance segmentation. Instead of dividing a picture into distinct objects, it divides it into semantic classifications such as foreground vs background. A segmentation technique known as Usiigaci has been developed by Tsai et al. [94] that uses a mask regional convolutional neural network (Mask R-CNN). A pre-trained Mask R-CNN model on the Microsoft COCO dataset was further trained using manually annotated images with single-cell outlines as the classification class. The trained model is used to feed the images into the Mask R-CNN-based segmentation module, which generates highly accurate instance-aware segmented masks as a result of its application. Individual cells in the images have been suitably segmented into identifiers based on their borders.

A method developed by Wang et al. [95] that incorporates both a Convolution Neural Network (CNN) and the standard watershed approach has been developed. Initial training

consisted of using a CNN to learn the Euclidean distance transform (EDT) of the mask corresponding to the input images (deep distance estimator). Subsequently, the authors trained a faster Region-based Convolutional Neural Network (RCNN) to recognise individual cells within the EDT image (deep cell detector). Then, using the outputs from the previous two rounds, the watershed algorithm completed the final segmentation. UNet was utilised by Zhou et al. [96] for primary cell segmentation. Image pixels are classified into cell boundaries, cell interiors, and backgrounds at this step. Training the network is accomplished through the application of the cross-entropy loss function. By using a weight map, more attention is devoted to the boundaries of the cells. The threshold is used to obtain cell segmentation by inferring from cell boundaries and cell interiors. Within segmented cells, it is possible that there are holes, and the flood-fill technique is employed to fill in these internal gaps. At this stage, when several cells are close together, each cell is not separated. As soon as the initial segmentation is completed, cells that are close to one another may segment into a blob, which is a part of the connected region.

Arbelle and Raviv [97] proposed a novel segmentation architecture based on the combination of Convolutional Long Short Term Memory (C-LSTM) and a U-Net encoder-decoder structure. Due to the network's unique architecture, multi-scale, compact, spatio-temporal encoding can be captured in the C-LSTMs memory units. Rather than a pipeline, the innovative design offered here is an interlace structure combining the two notions. Furthermore, their solution is intended for image sequence segmentation, which can be very time-consuming due to the fact that the bidirectional C-LSTM is not computationally viable. Ayanzadeh et al. [98] presented a hybrid deep neural network-based solution for cell segmentation in phase contrast microscopy images that outperformed standard approaches. Specifically, it is intended to reduce the discrepancy between the encoder characteristics and the characteristics that propagate in the decoder of the U-Net architecture. They created an alternative feature extractor by applying the enhanced ResNet18 to the encoder of U-Net and swapping the ordinary blocks with residual blocks in the decoder of the U-Net structure. Furthermore, they used ResNet18 as the backbone of FPN, which has higher performance than traditional techniques. This change improved the model's ability to account for low- and high-level semantics, as well as image details. Moreover, utilising transfer learning in the provided approach boosted training convergence. It enhanced the predictability of the findings, reduced training time, and prevented the model from over-fitting.



Fazeli et al. [99] discussed the usage of deep learning networks, such as StarDist, that frequently need users to train or retrain models using their own images. In comparison, high-quality StarDist pre-trained models are widely available and are likely to underperform when applied to data with varying stains, microscope types, and noise levels. To train StarDist models using the ZeroCostDL4Mic platform, which enables researchers to train (and retrain), evaluate, and deploy deep learning networks. Notably, the ZeroCostDL4Mic StarDist 2D notebook may directly generate a file containing the geometric centre coordinates of all the nuclei (tracking files) used as input for TrackMate. As a result, the proposed pipeline is separated into two sections. To begin, the ZeroCosDL4Mic platform is used to train a StarDist model. This section must be completed once for each type of data. Second, the trained StarDist model is used to segment and generate track files for the object.

DeLTA (Deep Learning for Time-lapse Analyse), a picture processing tool that uses U-Net deep learning models to segment the cells from microscope images, was introduced by Lugagne, Lin and Dunlop [100]. They employed a combination of Ilastik and manual curation to generate segmentation outputs, which were then used as training sets for the Ilastik algorithm. It uses data augmentation operations to ensure that it is robust to experimental variance and imaging condition changes. As outlined in the original U-Net structure, a pixel-wise weighted binary cross-entropy loss function was used to enforce border detection between cells during training the network. Weight maps were constructed prior to data augmentation and training to enhance the connection between two cells. A method for segmenting contacting cells in microscope pictures is presented by Scherr et al. [101]. By utilising a unique representation of cell borders inspired by distance maps, the technique can train on both touching and non-touching cells. Also, this approach is remarkably resistant to annotation errors and demonstrates encouraging results for the segmentation of microscope pictures in the training data that are underrepresented or not included in cell types. The proposed neighbour distances are predicted using a customised U-Net convolutional neural network (CNN) with two decoder paths, which has been trained specifically for this purpose.

Celltrack was invented by Chen et al.[102], which used Mask R-CNN as a backbone network for conventional cell detection and segmentation, as well as for cell tracking. The three original branches for predicting classification scores have been supplemented by adding bounding boxes and segmentation masks. Chamier et al. [103] developed the ZeroCostDL4Mic platform, which allows users to implement two networks, U-Net and StarDist, which perform

state-of-the-art image segmentation on 2D and 3D imaging datasets. The DeepSea model was developed and proposed by Zargari et al. [104] based on a scaled-down version of U-Net architecture and utilised for segmentation. DeepSea makes use of residual blocks in order to increase the depth of the model and improve model performance, as has been shown in Residual Networks. The use of batch normalisation and dropout techniques can increase the speed, performance, and stability of a model. The application of the watershed technique was used to improve the outputs of the models further. As a supplemental approach, the watershed assisted in separating the remaining segmented cell bodies.

Vicar et al. [105] developed four different post-processing pipelines for U-Net-based cell segmentation. To enable non-deep learning transfer of the segmentation network to multiple sample types without the need for annotated training data, the author attempted to construct these post-processing pipelines with only a few customisable parameters. Compared to traditional transfer learning, this strategy eliminates the need for training data and requires computer training of the deep learning model. Additionally, they sought to use specific pretraining methodologies utilising unlabelled images that may be used for self-supervised pre-training to increase the quality of final segmentation. Self-supervised pre-training increased both segmentation performance and transferability to diverse cell types using the proposed methodology.

Wen et al. [106] used a deep network called 3D U-Net to segment cells in 3D images and predict the class labels (cell or non-cell) of individual voxels based on information in neighbouring voxels using a deep network called 3D U-Net. The U-Net can perform precise segmentation under a variety of imaging situations and can be trained using a small number of annotated images. The first volume's pre-processed images are used to train the 3D U-Net, which is subsequently utilised to segment cells in the subsequent volumes. The 3D U-Net can be immediately reused for different datasets taken under comparable optical conditions once trained on one dataset. The cell-like regions discovered by 3D U-Net are clustered into single cells using the watershed approach. Automated mesenchymal stem cell (MSC) segmentation and machine learning-based phenotypic categorisation using morphometric and textural analysis were developed by Mota et al. [107]. The algorithm was trained on micrographs of phase-contrast taken during the early or mid-logarithmic stages of MSC expansion. Localisation of cell regions is accomplished through the use of edge detection, thresholding, and morphological processes. Cell markers are then identified inside each region by employing

the H-minima transform in order to distinguish individual cells from cell clusters. In order to acquire single cells, clusters are segmented using a marker-controlled watershed. Machine learning is used to extract morphometric and textural characteristics from cells to classify them according to their phenotype. The summary of published cell segmentation methods utilising machine learning techniques is presented in Table 2.

Table 2: Summary of machine learning approaches used in cell segmentation.

| Author                              | Method   | Limitations  |
|-------------------------------------|--|--|
| Hilsenbeck et al. , 2017 [89]       | SVM  | <ol style="list-style-type: none"> <li>1) Challenges with Multi-Class and Imbalanced Data [144]</li> <li>2) High Time Complexity in Training [145]</li> </ol>  |
| Arganda-Carreras et al. , 2017 [91] | random forest classifier   | <ol style="list-style-type: none"> <li>1) Requirement for Annotated Training Data [146]</li> <li>2) Generalization to New Cell Types or Imaging Modalities [146]</li> <li>3) Handling Touching or Overlapping Cells [147]</li> </ol> |
| Hernandez et al. , 2018 [92]        | Fully Convolutional Network developed                                | <ol style="list-style-type: none"> <li>1) Requirement of a large amount of annotated training data [148]</li> <li>2) Boundary Sensitivity [149]</li> <li>3) Computational Complexity [150]</li> </ol>                                |
| Berg et al. , 2019 [93]             | use of image filters as features and a random forest as a classifier | <ol style="list-style-type: none"> <li>1) Computational Resources [93]</li> <li>2) Reliance on Local Pixel-Level Features (brightness, colour, and texture) [93]</li> </ol>  |

|   |  |  |
|---|--|--|
| <p>Tsai et al. , 2019 [94],Chen et al. , 2021 [102]</p>   | <p>Mask regional convolutional neural network (Mask R-CNN)</p>                             | <ul style="list-style-type: none"> <li>● Computational demands [151]</li> <li>● Requires a large amount of annotated data, time-consuming and expensive to acquire [152]</li> <li>● Limited Generalisation to Unseen Categories [152]</li> </ul> |
| <p>Wang et al. , 2019 [95]</p>  | <p>Convolution Neural Network (CNN) and the standard watershed approach</p>                | <ul style="list-style-type: none"> <li>● Computational Complexity</li> <li>● Requires a large amount of annotated data, time-consuming and expensive to acquire [152]</li> <li>● Limited Generalisation to Unseen Categories [152]</li> </ul>    |
| <p>Zhou et al. , 2019 [96], Fazeli et al. , 2020 [99], Lugagne, Lin and Dunlop. , 2020 [100], Scherr et al. , 2020 [101], Zargari et al. , 2023 [104] and Vicar et al. , 2021 [105]</p> | <p>UNet</p>  | <ul style="list-style-type: none"> <li>● Computational complexity</li> <li>● Requires a significant amount of accurately annotated [153]</li> <li>● Limited generalisation [153]</li> </ul>  |
| <p>Arbelle and Raviv., 2019 [97]</p>  | <p>Convolutional Long Short Term Memory (C-LSTM) and a U-Net encoder-decoder structure</p> | <ul style="list-style-type: none"> <li>● Complexity and Computational Demand [154]</li> <li>● Difficulty in data training</li> <li>● Increased Parameter Tuning [154]</li> </ul>   |

|                              |   |   |
|------------------------------|---|---|
| Ayanzadeh et al. , 2020 [98] | U-Net and ResNet18  | <ul style="list-style-type: none"> <li>● Complexity and Computational Demand</li> <li>● Requires a large training dataset [155]</li> </ul>  |
| Chamier et al. , 2021 [103]  | U-Net and StarDist  | <ul style="list-style-type: none"> <li>● Complexity and Computational Demand</li> <li>● Requires a large training dataset</li> <li>● Not suitable for non-star-convex shapes [156]</li> </ul>   |
| Wen et al. , 2021 [106]      | 3D U-Net  | <ul style="list-style-type: none"> <li>● Complexity and Computational Demand</li> <li>● Requires a large training dataset</li> </ul>  |
| Mota et al. , 2021 [107]     | Region-based edge detection, marker-based watershed method and logistic regression model. | <ul style="list-style-type: none"> <li>● Sensitive to noise and grayscale unevenness [157]</li> <li>● Require manual intervention [157]</li> <li>● Dependency on Marker Selection [158]</li> <li>● Inadequate for Complex Patterns [159]</li> </ul> |

The summary table 2 outlines various machine learning approaches used in cell segmentation, detailing their methods and associated limitations, and a critical analysis of these techniques helps identify the most suitable options for the specific needs of the current study. Support Vector Machine (SVM) (Hilsenbeck et al., 2017) is known for its effectiveness in handling high-dimensional data and its robustness in separating classes with a clear margin; however, it faces challenges with multi-class and imbalanced data, as well as high time complexity in training, making it less suitable for studies requiring efficient processing of diverse and complex cell images. Similarly, Random Forest Classifiers (Arganda-Carreras et al., 2017) offer flexibility but are constrained by the need for annotated training data, difficulties in generalising to new cell types or imaging modalities, and issues with handling touching or overlapping cells. Fully Convolutional Networks (FCNs) (Hernandez et al., 2018)

provide robust segmentation capabilities but require large amounts of annotated training data, are sensitive to boundaries, and demand significant computational resources, making them challenge to implement unless the necessary data and computational power are readily available. Random Forest Classifiers with Image Filters (Berg et al., 2019) combine feature extraction with classification, yet they require substantial computational resources and rely heavily on local pixel-level features such as brightness, colour, and texture, which might limit their effectiveness in handling the diverse and irregular cell shapes considered in this study. Moreover, Mask Regional Convolutional Neural Networks (Mask R-CNN) (Tsai et al., 2019; Chen et al., 2021) are powerful for segmentation but come with high computational demands and the need for large, annotated datasets that are time-consuming and expensive to acquire; furthermore, they have limited generalisation to unseen categories, which could be a significant drawback in studies where variability is expected. Convolutional Neural Networks (CNNs) and U-Net (Wang et al., 2019; Zhou et al., 2019; and others) are popular due to their ability to perform well on complex segmentation tasks, although they require significant computational resources and a large amount of accurately annotated data; additionally, their generalisation to unseen categories is limited, which may pose challenges depending on the dataset's diversity.

Convolutional Long Short Term Memory (C-LSTM) with U-Net (Arbelle and Raviv, 2019) introduces complexity and increased parameter tuning, making it difficult to train; thus, while it offers enhanced segmentation capabilities, the complexity of the model and its computational demands may not be justified for all studies. Similarly, U-Net with ResNet18 (Ayanzadeh et al., 2020) and U-Net with StarDist (Chamier et al., 2021) are advanced versions of U-Net designed to improve segmentation accuracy; however, both methods require large training datasets and are computationally intensive, and StarDist, in particular, may not be suitable for non-star-convex shapes, limiting its applicability depending on the specific cell structures involved. While 3D U-Net (Wen et al., 2021) offers enhanced three-dimensional segmentation, it does so at the cost of increased computational demand and the need for a large training dataset, making it well-suited for studies requiring 3D analysis but potentially excessive for simpler 2D tasks. Region-Based Methods (Mota et al., 2021) involving edge detection, marker-based watershed, and logistic regression models are more traditional approaches that are sensitive to noise and grayscale unevenness, requiring manual intervention and proving inadequate for handling complex patterns, thereby limiting their effectiveness in more intricate segmentation tasks.

Given these considerations, Image Filters with Random Forest Classifiers using the tool Ilastik has been chosen for this study as Ilastik combines feature extraction with classification and offers several advantages, including a workflow that facilitates pixel classification through interactive segmentation, where users can easily adjust parameters and refine results in real-time. This method benefits from relatively low computational requirements compared to more complex neural networks, making it accessible even in less resource-intensive environments; additionally, Ilastik is particularly suitable for biologists due to its user-friendly interface, which does not require advanced programming skills, and its ability to handle various types of data with intuitive controls. In addition, preprocessing the images before using them in Ilastik is necessary as it can enhance the image quality by reducing noise, improving contrast, and ensuring uniformity across the dataset, which is crucial to maximising the effectiveness of Ilastik's pixel classification by allowing the tool to focus on the most relevant features of the cells, thereby improving the accuracy of the segmentation. Despite the need for substantial computational resources and reliance on local pixel-level features like brightness, colour, and texture, Ilastik's flexibility and ease of use make it a strong candidate for this study, and its workflow, which allows for efficient pixel classification and interactive segmentation, is particularly well-suited for the study's needs; thus, its ability to handle various types of data with relatively low computational demand, combined with its flexibility in adjusting parameters and refining results in real-time, aligns well with the study's objectives.

### **2.3 Cell Tracking**

Live-cell imaging experiments often produce vast quantities of time-lapse images [112] that include much more than a human can analyse through visual inspection alone. A computerised approach to image study allows the opportunity to efficiently and reproducibly obtain the full benefit of available data. In various experiments, a recurring task is the tracking and analysis of large numbers of cells. Over the last decade, many methodologies have been developed with this aim, and software tools based on these methods are continuously becoming feasible. This section will discuss the computational techniques used to perform cell-tracking tasks. The technique will be demonstrated using both traditional and machine learning methods.

### 2.3.1. Traditional approaches

A technique utilising global track linking was implemented by Magnusson et al. [86], to automate the tracking of living cells in microscope image sequences. The algorithm joins cell outlines created by a segmentation algorithm into tracks, which can then be followed automatically. This is accomplished by employing the Viterbi algorithm to determine which tracks provide the most significant conceivable increases to a probabilistically motivated scoring function. In addition, when new tracks are made, a mechanism is introduced that modifies previously created tracks, preventing mistake propagation. The algorithm is capable of dealing with apoptosis, mitosis, and migration, as well as false positives, missed detections, and clusters of cells that have been segmented together. A cell tracking algorithm, Cellstar, developed by Versari et al. [87] employs a multi-criteria optimisation algorithm. It involves the penalisation of relative displacements among neighbours, which gives collective cell motions more resilience.

Hernandez et al. [92] linked cell tracks between binary segmentations in consecutive frames using the Viterbi Algorithm for Cell Tracking. The tracking component of the tracking by detection approach is described below. The sequence of images is represented by this algorithm as a Hidden Markov Model, with individual cell locations serving as hidden states and observed detections; in this case, the segmentation from the neural network, serving as observations. The Viterbi method is then used to find the most likely state changes between each time step. The outcome is accurate tracking as it uses data from the whole image sequence for all the tracks and maximises the probabilistic score function, which only joins tracks if the total score is increased. Nevertheless, Berg et al. [93] developed Ilastik to perform automatic tracking-by-assignment. It tracks numerous pre-detected, potentially separated items over time in both 2D and 3D. In this technique, conservation tracking is used to generate probabilistic graphical models for all identified objects across a wide range of time points at the same time. Fazeli et al. [99] described how TrackMate may employ tracking files to track specified items. A set of images and their accompanying masks are required to train a StarDist model. The most time-consuming element of the analysis workflow shown here is creating a training dataset, which needs hand annotations of the images to be examined. For example, to make the training datasets, each cell or nuclei contour was manually created in Fiji [108] using the freehand selection tool. The generation of a high-quality training dataset is essential because it affects the StarDist model's specificity and performance. The creation of a training dataset, on the other hand, is only necessary once per dataset type. Videos of migrating cells can be efficiently



handled in a batch when a StarDist model has been adequately trained. The author provides a Fiji macro to investigate a folder containing multiple tracking files in addition to the TrackMate graphical interface. Despite this, the batch processing macro will offer basic quantitative information for each track, such as median and maximum speeds. Cell tracking is performed via the changed graph-based cell tracking method, which was developed by Scherr et al. [101]. An estimation of movement is included in the cost function of the customised tracking algorithm, which allows it to re-link tracks having incomplete segmentation masks over a short frames series. The summary of published cell tracking methods utilising traditional techniques is presented in Table 3.

Table 3: The summary of traditional approaches used in cell tracking.

| Author  | Method  | Limitations  |
|---|---|--|
| Magnusson et al. , 2015 [86],Hernandez et al. , 2018 [92] | Viterbi algorithm   | <ul style="list-style-type: none"> <li>• Computational complexity increases with larger data sets. [160]</li> </ul>  |
| Versari et al. , 2017 [87]                                | Employs a multi-criteria optimisation algorithm<br><br>use of a "neighbourhood-preserving criterion" for linking cells between frames during the tracking process | <ul style="list-style-type: none"> <li>• Computationally intensive [172]</li> <li>• Extensive parameter tuning [173]</li> </ul>  |
| Berg et al. , 2019 [93]                                   | Conservation tracking   | <ul style="list-style-type: none"> <li>• Requires accurate target detection and segmentation as a first step [161]</li> <li>• The number of tracking targets must be known a priori or estimated [161]</li> <li>• computationally</li> </ul> |

|                            |  |  |
|----------------------------|--|--|
|                            |  | expensive for very large datasets. [162]   |
| Fazeli et al. , 2020 [99]  | The Linear Assignment Problem (LAP) and Kalman filtering | <ul style="list-style-type: none"> <li>● methods used assume linear motion models [163]</li> </ul>   |
| Scherr et al. , 2020 [101] | Graph-based cell tracking method                         | <ul style="list-style-type: none"> <li>● Computationally intensive when dealing with large datasets. [164]</li> <li>● Manual Parameter Tuning [164]</li> </ul> |
| Tsai et al. , 2019 [94]    | Particle Tracking Velocimetry (PTV) method               | <ul style="list-style-type: none"> <li>● Sensitivity to Parameters [165]</li> <li>● Limited to Blob-Like Features [166]</li> </ul>                             |

The summary table 3 presents various traditional approaches used in cell tracking, each with its own method and associated limitations. A critical examination of these methods provides valuable insights into their suitability for specific research requirements. The Viterbi algorithm (Magnusson et al., 2015; Hernandez et al., 2018) is widely recognised for its ability to find the most likely sequence of hidden states, making it effective in tracking cells over time. However, its computational complexity increases significantly with larger datasets, which can be a limitation when working with extensive or high-resolution time-lapse data. Multi-criteria optimisation algorithms (Versari et al., 2017), which employ a "neighbourhood-preserving criterion" for linking cells between frames during the tracking process, offer a sophisticated approach to ensuring continuity in tracking. While this method is powerful, it is computationally intensive and requires extensive parameter tuning, which could be a challenge in scenarios where computational resources are limited or where quick results are needed. Conservation tracking (Berg et al., 2019) is effective in maintaining the continuity of tracked cells, but it requires accurate target detection and segmentation as a preliminary step. This method is also computationally expensive for very large datasets, and the number of tracking targets must be known a priori or estimated, which adds complexity to its application. The Linear Assignment Problem (LAP) and Kalman filtering (Fazeli et al., 2020) methods are based on the assumption of linear motion models, making them less adaptable to scenarios where cell movements are more erratic or non-linear. While they are useful in certain controlled environments, their assumptions may not hold in more complex or dynamic systems. Graph-

based cell tracking methods (Scherr et al., 2020) offer a flexible framework for cell tracking but are computationally intensive when dealing with large datasets. Additionally, they require manual parameter tuning, which can be time-consuming and may introduce user bias, particularly in less standardised environments. Particle Tracking Velocimetry (PTV) methods (Tsai et al., 2019) are effective for tracking blob-like features, but they are highly sensitive to parameter settings and are limited in their application to more complex cell shapes or movements. This restricts their use in studies where cells exhibit a wide range of morphological features. Considering these factors, the Follow Neighbour method implemented through the tool CellProfiler has been chosen for this study. CellProfiler, which is based on a flow approach, offers several advantages that make it particularly suitable for biologists. Its user-friendly interface does not require advanced programming skills, allowing biologists to perform complex cell tracking tasks with ease. The Follow Neighbour method, specifically, is effective in maintaining continuity in cell tracking by associating cells in subsequent frames based on proximity, which is crucial for accurately following cell movements over time.

### **2.3.2. Machine learning approaches**

Tsai et al.[94] created an A Trackpy-based cell tracker with a graphical user interface which was developed for cell tracking and data verification purposes. Following cell segmentation, each mask has segmented cell out-lines that are labelled with unique identifiers (IDs). The IDs are then used for linking and tracking cells in the tracking software module, which is based on the Trackpy library. The characteristics of an ID, such as its location, equivalent diameter, perimeter, eccentricity, orientation, and shape, are utilised as parameters in Trackpy for cell tracking. In a time-lapse microscopy experiment, the IDs in each successive mask all correspond to the same cell. The investigation was carried out via the Trackpy library, which used the k-dimension tree technique as its default nearest neighbour search option. For multi-cell tracking, Zhou et al. [96] suggested an approach that combines detection and segmentation. The approach involves four divisions: cell centroid identification with multi-frame pictures, primary multi-cell tracker, primary cell segmentation, and fine segmentation. The use of a multi-frame as an input to UNet [109] is suggested, which aids the network in extracting spatio-temporal data. The mitosis [110] detection algorithm improves the detection performance of mitotic cells, which improves the detection performance of mitosis during tracking. For tracking high density small cells, a fine cell segmentation approach is presented.

Cell tracking can be accomplished by combining the main tracking results of cell centroid identification with the findings of primary cell segmentation.

In solving stem cell tracking challenges, Wang, Mao, and Yi [111] suggested a deep learning system with a convolutional structure and multi-output layers. A convolutional structure is used to learn robust cell features via an in-depth feature learning technique applied to huge visual data. This framework tracks the cell's mobility using many output layers and simultaneously detects its mitosis as a helper task, enhancing the model's generalisability and facilitating practical applications in stem cell research. The tracking and detection neural network framework developed here also includes a particle filter-based motion model, a specific cell sampling technique, and a matching model update approach. Its present application to a microscope image collection of human stem cells reveals that it outperforms and is more robust than other commonly used approaches. Lugagne, Lin, and Dunlop [100] track cells from one frame to the next and identify cell divisions using a U-Net architecture. Where, as inputs and outputs, numerous images are used. For every cell point, the author uses four images as inputs: current frame transmitted light images, the previous frame transmitted light images, the previous frame binary mask, and the current frame segmentation mask. This U-Net model is utilised downstream from the segmentation U-Net to complete the pipeline for time-lapse analysis. As training outputs, by using two binary masks, the first defining the location of the seed cell in the next frame and the second defining the location of the prospective daughter cell in the event of a division. Once trained, the tracking pipeline tracks cells and detects cell divisions in time-lapse videos used for evaluation.

CellTrack R-CNN was developed by Chen et al. [102] to perform tracking jointly performed by integrating a Siamese tracking branch with the Mask R-CNN pipeline without extra post-processing techniques. To further enhance tracking performance, spatial information is incorporated into the tracking branch to reach learnable relative position encodings of cell instances, which are then effectively fused with visual features. Li et al. [123] propose a new deep neural network based on dynamic memory and template matching for tracking multiple cells in microscopy images. The network is denoted multi-object dynamic memory network (MODM). MODM consists of multiple dynamic memory units (DMU), which are an extension of the network and use a fully convolutional neural network for feature extraction. An LSTM dynamically updates the template with attention to cope with changing cell appearance, and a deep neural network performs cell detection compared to manual annotation. To improve the

robustness and cope with track drift, the author includes a motion constraint that exploits statistics of cell motion and integrates a mechanism for handling cell mitosis events using a deep neural network for mitotic cell detection. In order to support the Cell Tracker task, Zargari et al. [104] have developed DeepSea Tracking Model employing the U-Net Architecture scaled-down version by using pair data at time  $t$  and time  $t-\tau$  in the segmentation task. To locate the target cell among the segmented cells in the current frame (at time  $t$ ), the tracker model must remember its earlier location (at time  $t-\tau$ ). Because cells move slowly through space, the model can use the cell's past location to predict where it will be in the current frame. The position will be determined by selecting a rectangle search window in the current frame based on the target cell's previous bounding box location (at time  $t-\tau$ ). Using the same search window, clip the previous and current frames and feed them into the convolutional tracker model. The purpose of the tracker model is to localise and segment the current location of the target cell within the search window at time  $t$ .

Wen et al. [106] employed a deep learning technique to combine spatial pattern and local cell region tactics. The feedforward network (FFN) algorithm is used to match temporally adjacent cells based on the distance pattern between each cell and its neighbours. Using the FFN pattern, all cells at time  $t_1$  are compared to all cells at time  $t_2$ , and the cells with the highest degree of similarity are classified as identical cells at time  $t_1$  and time  $t_2$ . The pipeline extracts the cell centre points from the cell areas segmented using 3DU-Net and the watershed approach, and then applies a pre-trained FFN to the cell points to generate the initial matching between volumes  $t$  and  $t+1$ . In order to enhance the initial correlation, a non-rigid point set registration (PR-GLS) method is employed for generating a coherent transformation; that is, identical movements are required for neighbouring cells. Combining FFN and PR-GLS results in more accurate predictions of cell locations, and the predicted positions are accurate due to the use of information from local cell regions in the 3D U-Net output. The summary of published cell tracking methods utilising machine learning techniques is presented in Table 4.

Table 4: The summary of machine learning approaches used in cell tracking.

| Author  | Method  | Limitations   |
|---|---|---|
| Zhou et al. , 2019 [96],<br>Lugagne, Lin, and Dunlop.,<br>2020 [100] and Zargari et al.<br>, 2023 [104] | UNet method   | <ul style="list-style-type: none"> <li>● Difficulty in keeping track of multiple cells across frames.[167]</li> <li>● Manual annotations [168]</li> <li>● Time-consuming [168]</li> </ul> |
| Wang, Mao, and Yi., 2017 [111]  | Convolutional neural network (CNN)  | <ul style="list-style-type: none"> <li>● Computationally intensive [169]</li> <li>● Large training datasets [168]</li> </ul>  |
| Chen et al. , 2021 [102]  | RCNN  | <ul style="list-style-type: none"> <li>● Computationally intensive [170]</li> <li>● Large training datasets</li> </ul>  |
| Li et al. , 2021 [123]  | Fully convolutional neural network,dynamic memory network,template matching and LSTM    | <ul style="list-style-type: none"> <li>● Computationally intensive [171]</li> <li>● Large training datasets [171]</li> </ul>  |
| Wen et al. , 2021 [106]   | Deep learning technique and PR-GLS (Point Set Registration - Generalized Least Squares) | <ul style="list-style-type: none"> <li>● Computationally intensive</li> <li>● Large training datasets</li> </ul>  |

The summary table 4 outlines various machine learning approaches used in cell tracking, highlighting both their methods and associated limitations. Each technique has its strengths but also faces challenges that may affect its suitability for specific research contexts. The UNet method (Zhou et al., 2019; Lugagne, Lin, and Dunlop, 2020; Zargari et al., 2023) is recognised for its effectiveness in segmentation tasks. However, when applied to cell tracking, it struggles to keep track of multiple cells across frames, especially in dynamic environments. Additionally, the requirement for manual annotations makes the process time-consuming, adding to the complexity of using this method in large-scale studies. Convolutional Neural Networks (CNNs) (Wang, Mao, and Yi, 2017) are widely used for their powerful feature

extraction capabilities. Nonetheless, they are computationally intensive and require large training datasets, which can be a significant limitation, particularly in studies with limited computational resources or where acquiring extensive annotated data is challenging. The RCNN method (Chen et al., 2021) builds upon the strengths of CNNs, providing robust performance in tracking tasks. However, it shares similar limitations, being computationally intensive and dependent on large training datasets. These factors can make RCNNs less feasible for projects with tight resource constraints. Fully Convolutional Neural Networks (FCNs) combined with dynamic memory networks, template matching, and LSTM (Li et al., 2021) offer an advanced approach for cell tracking, leveraging deep learning techniques to manage complex tracking scenarios. Despite their advanced capabilities, these methods are also computationally demanding and require large amounts of training data, which can limit their practical application. The approach combining deep learning techniques and PR-GLS (Point Set Registration - Generalized Least Squares) (Wen et al., 2021) is notable for its precision in cell tracking. However, like the other methods, it is computationally intensive and requires large training datasets, making it challenging to implement in resource-limited settings.

While the machine learning approaches outlined in the table, such as UNet, CNNs, RCNN, and others, offer advanced capabilities for cell tracking, they are not being utilised in this study. These methods, although powerful, come with significant limitations such as high computational demands, the need for large training datasets, and extensive manual annotation or parameter tuning. Given the study's specific objectives and available resources, these factors make these machine learning approaches less practical. As previously mentioned, the Follow Neighbour method implemented through CellProfiler has been selected instead. This choice is driven by its suitability for the study's objectives, particularly its lower computational requirements and the absence of a need for extensive training data. CellProfiler's user-friendly interface and flow-based approach also make it particularly accessible for biologists, allowing complex cell tracking tasks to be performed without advanced programming skills. The Follow Neighbour method is effective in maintaining continuity in cell tracking by linking cells based on proximity between frames, making it an ideal solution for accurately tracking cell movements over time. This method aligns well with the study's needs, offering a practical and efficient alternative to the more computationally intensive machine learning techniques.

## **2.4. Cell Characterisation**

In the framework of live-cell imaging studies, cell characterisation refers to a thorough analysis and identification of cellular characteristics across time. These studies produce massive amounts of data that record dynamic cellular activities. These data need to be analysed using modelling techniques to improve accuracy in predicting cellular responses and behaviours and allow simulations of complex biological processes without requiring large experimental setups. Necessary for drug development and disease modelling, these models facilitate understanding complex cellular dynamics and interconnections. This section will examine the wide range of modelling approaches applied to cell characterisation, emphasising how machine learning techniques and traditional approaches work to expedite and simplify the analytical process.

### **2.4.1. Traditional approaches**

Arroyo et al. [113] proposed a multi-agent-based model that is able to describe dynamics in cell populations. The model consists of biological entities (cells) as agents and a biochemical environment. Both are represented by multisets of symbols. The evolution of the environment is regulated by multiset Lindenmayer rules [114] depending on the current state of all agents, while the evolution of each agent, which depends on the environment's current state, is defined by means of multiset patterns. ARCADE, a multi-scale agent-based model, was developed by Yu and Bagheri [115] to investigate the emergent behaviour of heterogeneous cell agents in dynamic microenvironments and to show how emergent dynamics are influenced by the intricacy of intracellular metabolism and signalling. The authors showcase the effectiveness of their approach in obtaining computational and experimental understanding through *in silico* case studies focusing on context, competitiveness, and heterogeneity. There are notable distinctions between emergent behaviour in tissue and colony settings, as well as linear, non-linear, and multimodal effects of parameter modification on competition in simulated co-cultures and varying effects of population and cell heterogeneity on emergent outcomes.

Kihara et al. [116] created a cellular automata model to estimate cell-cell interactions using cultured cells. They used HeLa, human osteosarcoma, rat mesenchymal stem cells, and rat smooth muscle A7r5 cells. The model had five variable parameters: initial cell number, doubling time, motility, cell-cell adhesion, and cell-cell contact inhibition. The simulations



showed that HeLa and HOS cells had low adhesion and weak contact inhibition, while MSCs had high adhesion and positive contact inhibition. This approach is an easy method for evaluating cell-cell interaction properties. Li J. et al. [117] created a model of the cell cycle, focusing on the regulatory network controlling growth and division. They used particle-based computer simulations and continuum theory to analyse this model, focusing on 2D colonies confined in a channel. They found that the profile and speed of these growth fronts are related to substrate friction and cell cycle parameters, offering a method for measuring these parameters in experiments.

Wieczorek [118] presented a hybrid stochastic individual-based model that incorporates chemotaxis in proliferating cells. The model is formulated as a combination of a branching diffusion process and a partial differential equation that represents the concentration of the chemotactic component. It has been demonstrated that in the hydrodynamic limit, as the number of cells approaches infinity, the model converges to the solution of a nonconservative system resembling Patlak-Keller-Segel. A stochastic model with nonlinear mean-field dynamics is established, and it has been demonstrated that the movement of descendants from a single cell in the individual model converges with this mean-field process. The summary of published cell characterisation methods utilising traditional techniques is presented in Table 5.

Table 5: The summary of traditional approaches used in cell characterisation.

| Author                      | Method                        | Limitations  |
|-----------------------------|-------------------------------|--|
| Arroyo et al. , 2019 [113]  | Multi agent based             | <ul style="list-style-type: none"> <li>● Lack of standard analytical methods [174]</li> <li>● Difficulties in comparing [174] different models</li> <li>● Variability in implementation [174]</li> </ul> |
| Yu and Bagheri., 2020 [115] | Multi-scale agent-based model | <ul style="list-style-type: none"> <li>● Lack of standard analytical methods [174]</li> <li>● Difficulties in comparing different models [174]</li> <li>● Variability in implementation [174]</li> </ul> |

|                            |   |   |
|----------------------------|---|---|
| Kihara et al. , 2017 [116] | Cellular automata                               | <ul style="list-style-type: none"> <li>● Oversimplified state transitions [175]</li> <li>● Limitations in capturing stochastic behaviour [176]</li> <li>● Difficulties in simulating complex interacting systems [177]</li> </ul>   |
| Li J. et al. , 2021 [117]  | Continuum theory and particle-based simulations | <ul style="list-style-type: none"> <li>● Computational costs [178]</li> <li>● Lack of structural information [178]</li> <li>● Oversimplification [178]</li> </ul>   |
| Wieczorek., 2023 [118]     | Hybrid stochastic individual-based model        | <ul style="list-style-type: none"> <li>● Complexity in model formulation [179]</li> <li>● Difficulty in parameter calibration [179]</li> <li>● High sensitivity to initial conditions and parameter values[179]</li> <li>● challenges in capturing spatial heterogeneity [180]</li> </ul> |

The summary table 5 outlines various traditional approaches used in cell characterisation, highlighting their methods and associated limitations. Each approach has its strengths, but also faces challenges that may impact its suitability for specific research contexts. Multi-agent based models (Arroyo et al., 2019; Yu and Bagheri, 2020) are recognised for their ability to simulate the interactions between individual agents (cells) within a system. However, these methods suffer from a lack of standard analytical methods, making it difficult to compare results across different models. Additionally, there is significant variability in their implementation, which can lead to inconsistent outcomes and challenges in standardising these models for broader application. Cellular automata (Kihara et al., 2017) provide a simplified framework for modelling cell behaviour, particularly in discrete state transitions. However, their oversimplified state transitions often fail to capture the stochastic nature of biological systems. This method also struggles with simulating complex interacting systems, limiting its applicability to more intricate biological processes. Continuum theory and particle-based

simulations (Li et al., 2021) offer a more detailed approach by incorporating the physical and mechanical properties of cells. However, these methods are based on theoretical models, which can lead to oversimplification when representing the complex, dynamic nature of biological systems. Additionally, these models are not fully based on a data-driven approach, which limits their ability to adapt to real-time or large-scale data variations. They often lack the structural information necessary to fully characterise the systems being studied, making them less effective in capturing the true complexity of cell behaviour. Hybrid stochastic individual-based models (Wieczorek, 2023) attempt to combine the strengths of individual-based models with stochastic elements to better capture the randomness inherent in biological systems. Nevertheless, these models are complex to formulate and calibrate, requiring careful adjustment of parameters. They are also highly sensitive to initial conditions and parameter values, which can introduce variability in the results. Furthermore, these models face challenges in capturing spatial heterogeneity, which is crucial for accurately characterising cell behaviour in a realistic context.

The traditional approaches outlined in the table, such as multi-agent based models, cellular automata, continuum theory, and hybrid stochastic individual-based models, offer various advantages for cell characterisation but are not being utilised in this study. These methods, while valuable, present significant limitations, including complexity in model formulation, difficulties in parameter calibration, and challenges in capturing the stochastic and heterogeneous nature of cell systems. Moreover, their reliance on theoretical models and the fact that they are not fully based on a data-driven approach can sometimes lead to oversimplification of the complex biological phenomena being studied, and limits their adaptability to varying datasets.

#### **2.4.2. Machine learning approaches**

Shen et al. [119] employed cell proliferation on titanium dioxide nanotubes (TNTs) as a case study using machine learning approaches to decode contradicting findings in the literature. The author used the gradient-boosting decision tree model and showed that cell density has a greater influence on cell proliferation than other experimental characteristics. The author also discovers that different TNT diameters can exhibit opposing trends in cell growth depending on how cell density and sterilisation techniques are changed. Based on this finding, they conclude that machine learning helps assess structure-property correlations in biomaterials and improves understanding of complex data in biomedical research. Ma et al. [120] developed

DCell, a visible neural network that simulates cellular growth accurately using 2,526 subsystems. DCell, which has been trained on millions of genotypes, can examine the molecular processes behind genotype-phenotype relationships. It encompasses 80% of growth forecast relevance, with 484 subsystems accounting for 21%. DCell provides a foundation for interpreting the genetics of disease, drug resistance, and synthetic life by capturing complex phenotypes. Balde et al. [121] developed a novel Cell Density Prediction design using the Optimal Deep Learning with Salp Swarm Algorithm (CDP-ODLSSA) technique. This method accurately predicts cell densities in cell suspensions or cultures using the Long Short Term Memory-Autoencoder (LSTM-AE) model and the Salp Swarm Algorithm (SSA). Experimental validation of the CDP-ODLSSA technique was conducted using various simulations, revealing its superiority over other approaches.

CellPhe is a pattern recognition toolkit developed by L. Wiggins et al. [122] for characterising cellular phenotypes in time-lapse videos. It automates cell phenotyping from various imaging modalities and integrates tracking information from different algorithms. CellPhe classifies cellular phenotypes using an ensemble classification approach with multiple machine learning algorithms: Linear Discriminant Analysis (LDA), Random Forest (RF), and Support Vector Machine (SVM) with an RBF kernel. These classifiers use majority voting to determine each cell’s phenotype. The authors specifically aim to classify cellular phenotypes in breast cancer cell lines MDA-MB-231 and MCF-7 to analyse their responses to chemotherapeutic drugs. In addition to classification, CellPhe employs clustering algorithms to analyse cell populations and identify heterogeneous subsets within these populations. Specifically, hierarchical and k-means clustering are used to investigate subgroups within single-class datasets (i.e., treated and untreated cells separately). The summary of published cell characterisation methods utilising machine learning techniques is presented in Table 6.

Table 6: The summary of machine learning approaches used in cell characterisation.

| <b>Author</b>               | <b>Method</b>                         | <b>Limitations</b>  |
|-----------------------------|---------------------------------------|---|
| Shen et al. , 2021<br>[119] | Gradient-boosting decision tree model | <ul style="list-style-type: none"> <li>● Overfitting [181]</li> <li>● Interpretability [182]</li> <li>● Sensitive to noise [183]</li> </ul> |

|                                |   |   |
|--------------------------------|---|---|
| Ma et al. , 2018 [120]         | Visible Neural Network  | <ul style="list-style-type: none"> <li>● Overparameterization [184]</li> <li>● Decreased interpretability [184]</li> </ul>  |
| Balde et al. , 2023 [121]      | Optimal Deep Learning with Salp Swarm Algorithm (CDP-ODLSSA) technique  | <ul style="list-style-type: none"> <li>● Complexity in Training and Tuning [185]</li> <li>● High Computational Resources [185]</li> <li>● Interpretability [185]</li> </ul> |
| L. Wiggins et al. , 2023 [122] | Linear Discriminant Analysis (LDA), Random Forest (RF), Support Vector Machine (SVM), hierarchical and k-means. | <ul style="list-style-type: none"> <li>● Possibly of tie result in ensemble</li> <li>● Interpretability</li> </ul>  |

The summary table 6 outlines various machine learning approaches used in cell characterisation, highlighting their methods and associated limitations. Each method offers distinct advantages, but they also present challenges that may affect their suitability for specific research contexts. The Gradient-boosting decision tree model (Shen et al., 2021) is known for its powerful predictive capabilities, particularly in handling complex datasets. However, it is prone to overfitting, which can lead to poor generalisation of unseen data. Additionally, the model's interpretability is limited, making it challenging to understand the underlying decision-making process. Furthermore, this approach is sensitive to noise in the data, which can impact the accuracy and reliability of the results. The Visible Neural Network (Ma et al., 2018) provides an innovative approach to neural network design, with layers that are more transparent and interpretable. However, this method often suffers from overparameterization, where the model becomes overly complex, leading to decreased interpretability and the risk of overfitting. These factors can make it difficult to apply the Visible Neural Network in contexts where model simplicity and interpretability are crucial. The Optimal Deep Learning with Salp Swarm Algorithm (CDP-ODLSSA) technique (Balde et al., 2023) offers advanced deep learning capabilities, optimised through a metaheuristic algorithm that enhances training and tuning. Despite its potential, this technique is highly complex in terms of both training and tuning, requiring significant computational resources. Moreover, the interpretability of the model remains a challenge, making it less accessible for researchers who require clear insights into the model's functioning. The combined approach of Linear Discriminant Analysis (LDA),

Random Forest (RF), Support Vector Machine (SVM), hierarchical clustering, and k-means (Wiggins et al., 2023) leverages the strengths of multiple algorithms to improve classification and clustering outcomes. However, this ensemble method can lead to ties in results, which complicates decision-making. Additionally, the interpretability of the ensemble model is often reduced, as it combines the outputs of several different methods, each with its own complexities.

The machine learning approaches outlined in the table, including Gradient-boosting decision trees, Visible Neural Networks, CDP-ODLSSA, and ensemble methods like LDA combined with RF, SVM, hierarchical clustering, and k-means, offer advanced capabilities for cell characterisation. However, they are not being utilised in this study due to several significant limitations. These methods tend to require high computational resources, present challenges in interpretability, and can be overly complex to train and tune. Additionally, the risk of overfitting and sensitivity to noise further limits their practical application in certain research settings. The objective here is to use an interpretable "white box" machine learning approach that can produce symbolic expressions in the model of cell characterisation. This type of model is more transparent and allows for easier interpretation of how the inputs are related to the outputs, which is crucial for understanding the underlying biological processes. In this context, an evolutionary algorithm approach has been selected. Evolutionary algorithms are particularly well-suited for this task as they can evolve models that are both interpretable and capable of capturing the essential patterns within the data. This approach aligns well with the study's goals by offering a balance between interpretability and predictive power, making it a practical and effective choice for cell characterisation.

## **2.5. Conclusion**

This chapter has presented cell segmentation, tracking, and characterisation techniques utilising both conventional computer vision and machine learning approaches. These methodologies have demonstrated considerable potential in understanding cellular behaviour, which is fundamental to biomedical research and therapeutic development advancements. However, the review has identified that cell segmentation by traditional methods still needs to improve to extract reliable segmentation results from imperfect microscopy data, especially when dealing with overlapping and partially touching cells [186]. Meanwhile, machine learning approaches could detect and segment cells but involved complex computational processes [187] that often required annotated data [186]. Furthermore, the implementation of

machine learning often requires high computational resources [188] and can be opaque in acquiring knowledge about image segmentation undertaken due to its inherent in explainability, which can discourage biologists from using it in their work [189].

Meanwhile, traditional cell tracking techniques have the challenging task of extending tracking methods to analyse complex cellular behaviours. While the cell cycle phase has been tracked, other complex phenomena have not been resolved, such as the various cellular events leading to cell death or collective cell migration [168]. Recently, using a machine learning approach helped improve accuracy in cell tracking, handling complex data and discovering cellular dynamics [168]. However, it required a large amount of data training and expended significant time, either in the training process or implementation [190]. So, the tool selection not only focuses on accuracy but also needs to consider the user-friendly interface, extensive plugin ecosystem and customisation, high throughput capability, and open source. In this work, based on the criteria above, we will use tools such as ImageJ [191], Ilastik [192], and CellProfiler [193], each selected for their unique capabilities in image processing, segmentation, machine learning-based classification, tracking and quantitative analysis.

In section 2.4, cell characterisation using the traditional method has model limitations such as the influence of environmental factors, estimation of parameters, different scale usage, and lack of comprehensive regulatory models and simplifications [194]. Machine learning approaches can achieve good accuracy in classification and regression models but are not interpretable, sometimes referred to as a “black-box” approach. In biomedical engineering and cellular biology, the interpretable model is important and will benefit understanding in the decision-making process for verifying scientific hypotheses and making clinical decisions. In this study, we will utilise an interpretable “white box” machine learning technique known as evolutionary algorithms (EAs) [195]. Interpretable “white box” models in machine learning are systems wherein the processes and outcomes are comprehensible to human users. These models facilitate transparency, allowing users to ascertain how inputs are systematically transformed into outputs and to understand the rationale underpinning each decision. EAs are notably recognised for their efficacy in generating solutions for various problems based on data-driven approaches, providing a symbolic approach model [196]. This symbolic approach contributes to the interpretability by using understandable mathematical or logical expressions to describe how decisions are derived, making the process more transparent. Such clarity is eminently suitable for applications where trust and a clear understanding of the model's reasoning are paramount.

## CHAPTER 3

### METHODOLOGY

#### 3.1 Introduction

This chapter describes the methodology used to understand and analyse the behaviour of cells in a population. The objective is to establish a pipeline of computer vision tools to reliably segment and track unlabelled cells grown in culture as adherent monolayers using normal human urothelial (NHU) [197] cells as a representative example. Additionally, a pipeline is used to describe the variations in the cell growth curve, migration speed, and angular velocity. Furthermore, to characterise the behaviour leading to changes in the differences in cell populations. Finally, to investigate the use of the features within an interpretable “white box” machine learning context to further characterise NHU cell behaviours. The structure of this chapter is as follows: A comprehensive description of the research design and methods will be provided. This section includes a detailed explanation of the procedure for preparing cell cultures, utilising Time-lapse microscopy for time-lapse image acquisition, and performing cell segmentation and tracking. Furthermore, the implementation of cell characterisation will be elaborated, employing evolutionary algorithms [198], specifically, genetic programming [199], to model cell behaviour and providing a symbolic approach model. Figure 4. Illustrate the process implemented in the study and is explained in details below.

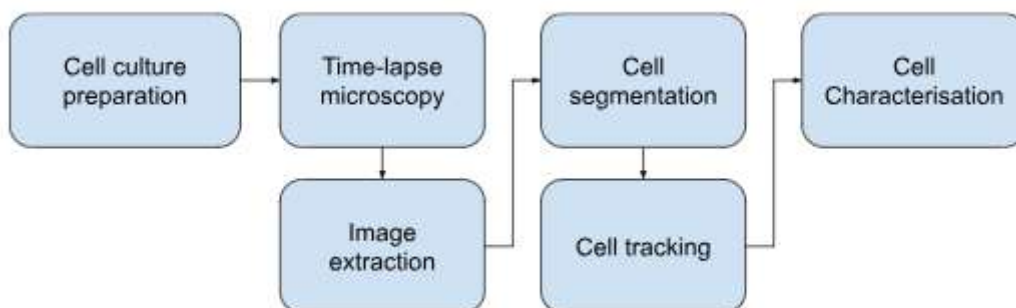


Figure 4: Workflow diagram for research methodology.

#### 3.2 Cell culture preparation

The experimental procedures detailed herein were primarily conducted by Ros Duke at Jack Birch Unit (JBU), with my involvement in assisting in the setup and execution under her



direct supervision and guidance. The cells were seeded in 12 plates [200], separately, with four replicates for each with three different treatments. The culture is seeded in the first four wells with 0.1% DMSO (control), and the subsequent four wells are cultured with 0.1% DMSO and 4ng/mL TGF- $\beta$  [201]. The final four wells were cultured in 0.1% DMSO with 10 $\mu$ M SB431542 as a TGF- $\beta$  inhibitor [202]. The cells were returned to the incubator for 1 hour before being transferred to the time-lapse microscopy chamber to collect data. In Figure 5, the layout of a 12-well microplate is detailed to illustrate the experimental setup used in the laboratory. The wells are organised into three groups based on their treatment conditions to facilitate comparative analysis. Wells 0 through 3 are maintained without any treatment, serving as the control group to provide baseline data. Wells 4 through 7 have been treated with TGF-beta, and the remaining wells, 8 through 11, are treated with SB431542, a selective inhibitor used to study TGF-beta signalling pathways.

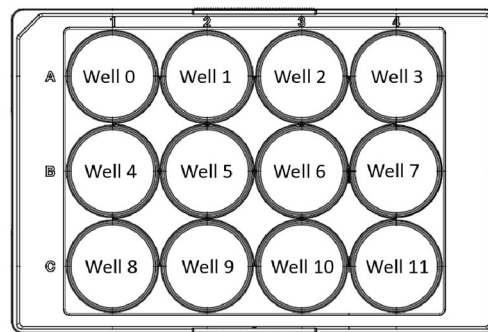


Figure 5: Layout of a 12-well microplate used in laboratory experiments.

### 3.3 Time-lapse microscopy

Time-lapse microscopy [203] is an advanced imaging technique that captures a series of images of a biological specimen at regular intervals, allowing researchers to observe dynamic cellular processes, such as cell growth, differentiation, and migration, over an extended period [204]. This technique has become an indispensable tool in various fields, including cell biology, developmental biology, and microbiology, for understanding the intricate details of cellular and subcellular events [205]. When capturing images over time for multiple wells, it is typically the platform that holds the wells that move, rather than the camera. This movement is often automated to allow the microscope to focus on different regions of interest, take images, and then move to the next position at pre-set time points. This automation is crucial for long-term imaging, as it allows for consistent image capture without manual

intervention. However, this setup can introduce challenges, particularly about illumination consistency. As the platform moves the wells, slight variations in lighting can occur due to changes in the angle of the light sources, the position of the wells relative to the light source, or even fluctuations in light intensity. Uneven illumination can lead to variations in image brightness and contrast, which can complicate the analysis of time-lapse data. The issue of uneven illumination will be explained in Section 3.4: Pre-processing Image. Phase-contrast microscopy, a widely used method in time-lapse microscopy, is based on exploiting differences in refractive indices between the specimen and its surrounding medium [206]. This technique enhances contrast in transparent, unstained biological samples by transforming phase shifts in light waves passing through the specimen into variations in amplitude, which are then visualised as differences in brightness and contrast [207].

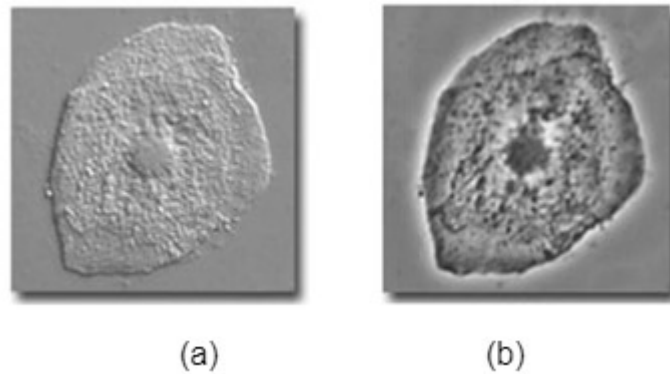


Figure 6: cell image comparison between brightfield (a) and phase-contrast (b) microscopy [209].

The significant advantage of phase-contrast microscopy is its ability to observe living cells without the need for potentially harmful staining or fixation procedures [208]. Figure 6 shows the different images between brightfield and phase contrast. Nevertheless, analysis of the phase-contrast microscopy image by using computer vision algorithms faces two main challenges when it comes to object detection, such as halo formation [210] and optical artefacts [211]. The halo effect, often referred to as the halo-light ring, is caused by the diffraction of light around the boundaries of structures present in the sample, resulting in the obscuring of delicate features. The ring of light created by the unaltered waves is smaller than the phase ring, allowing low-spatial-frequency diffracted light waves from the specimen to pass through the annulus. The light that deviates and passes through the phase ring retains a phase difference of 90 degrees, which prevents any negative interference from occurring. This results in a

reversal of contrast and produces a halo effect at the edges of large objects. Undoubtedly, the presence of a halo surrounding cells contributes to the difficulties of the segmentation process. Besides the halos effect, other optical artefacts, such as shade-off and contrast inversion, could occur. Shade-off is the term used to describe the gradual change in intensity observed in large-phase objects. Contrast inversion occurs when objects with a high refractive index appear brighter rather than darker, making it more challenging to interpret the images. The time-lapse microscopy setup, equipped with an environmental chamber, motorised stage, and focus system (figure. 7), uses an x10 objective to capture detailed images of a cell culture over 48 hours, with an intensity of brightfield light set to 4.1 and 0.5% contrast adjusted for optimal visibility.

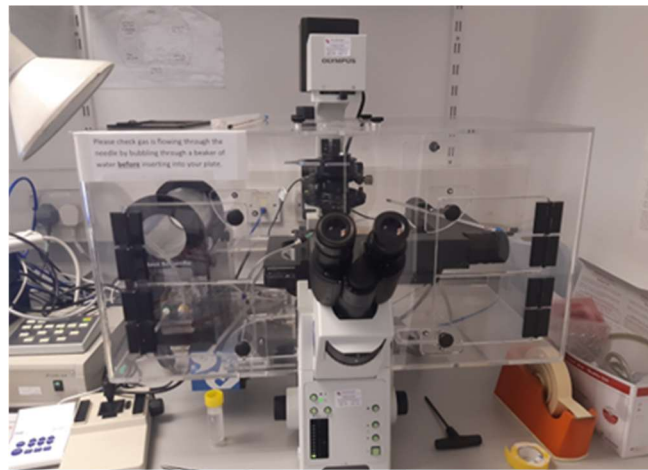


Figure 7: Olympus time-lapse microscopy setup.

### 3.4 Pre-processing image

In the field of microscopy, the presence of uneven illumination [212] in captured images is a prevalent issue. This is due to the background illumination intensity from the microscope light source not being uniform across the field of view, which results from limitations in the design of microscope condensers that cannot provide flat-field illumination. Furthermore, in time-lapse imaging, variations in image intensity may occur due to differences in acquisition conditions, such as lighting [213]. These intensity variations can significantly impact the accuracy and reliability of results, making it challenging to measure changes in cellular phenotypes and behaviours over time. To address these issues, images typically undergo pre-processing correction for these variations before quantification is performed. This process includes a series of operations to improve the quality of the image and simplify the quantification process. ImageJ [214], an open-source software developed by the National

Institutes of Health, is particularly useful in this context. It enables users to visualise, inspect, quantify, and validate scientific image data. Imaging-based methods are crucial in the life sciences, and as new imaging modalities emerge and datasets become more complex, reproducible and reliable methods to interpret biological images are essential. Image analysis allows users to extract information from images reproducibly, ensuring that algorithms and parameters remain open and consistent. Novel imaging modalities offer enhanced resolution, specificity, and coverage, contributing to numerous biological advancements. Modern research requires methods for efficient and robust manipulation, interpretation, and visualisation of advanced, multidimensional imaging data. Image analysis also serves a crucial biomedical role for diagnostic interpretation. As the prevalence of large multidimensional datasets continues to grow, the ability to manually take measurements becomes impractical and the sensitivity, accuracy, objectivity, and reproducibility of doing so can become significantly inhibited. The ImageJ ecosystem has evolved over time, with the first release of ImageJ in 1995, known as Fiji (Fiji is Just ImageJ) [215], introduced in 2007. The developers in Fiji enhanced the main architecture of ImageJ by incorporating an automated updating mechanism, a script editor, and



Figure 8: User Interface of Fiji (ImageJ) showing the main toolbar with icons for image processing functions.

a robust picture data model and also improved the plugin system and extensibility. Fiji enhances the existing structure by providing more libraries. Figure 8 illustrates the user interface of Fiji, an enhanced version of ImageJ, highlighting its extensive toolbar. The toolbar is packed with a range of tools and shortcuts that improve functionality and simplify the image processing process. All fundamental image modification tasks, ranging from simple actions like zooming and cropping to more intricate features such as adjusting thresholds and customising LUTs, may be easily accessed. Additionally, one helpful feature of the ImageJ software is its macro system, which allows for easy programming to automate repetitive processes and expand the capabilities of ImageJ. A macro in ImageJ is a script written in ImageJ's built-in macro language that automates a series of ImageJ commands. There are two ways to create a macro: the first is to write it manually, and the second is to use the Macro Recorder. In the manually writing macro method, the desired script using ImageJ's macro language is used to define specific automated tasks. Once the script has been written, it can be saved for easy access later. The Macro Recorder records the tasks as they are performed by the user and saved for use later, as

for the script macro described above. To run a saved macro, is straightforward within the ImageJ application. An example of an ImageJ macro that opens an image applies an auto threshold, as shown in Figure 9.

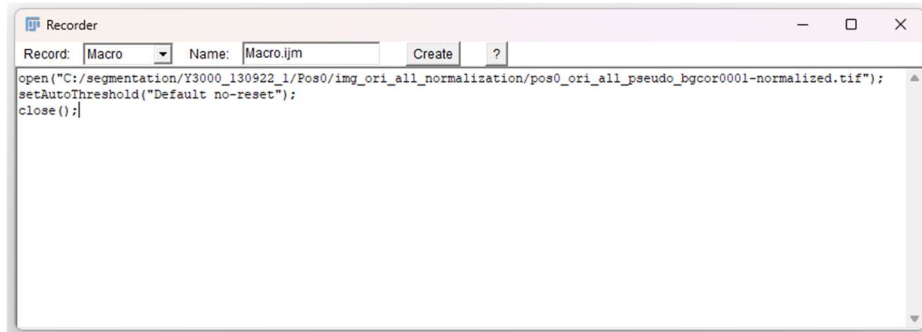


Figure 9: A macro script to perform image auto threshold.

In this image pre-processing phase, two primary steps are undertaken using the ImageJ plugin.

- **Correction of Uneven Illumination:** The initial step involves the utilisation of the BaSiC Plugin to address issues related to uneven illumination. This process is elaborated upon in Section 3.4.1, which focuses on the correction of uneven illumination.
- **Image Normalisation:** Following the correction of illumination, the image progresses to the next stage, which involves normalisation using the Quantile-based Normalisation Plugin. The specifics of this process are detailed in Section 3.4.2, which covers image normalisation.

Upon completion of these two sequential processes, the image is prepared for cell segmentation, which is further discussed in Section 3.5, dedicated to cell segmentation.

### 3.4.1 Uneven illumination correction

The correction of the uneven illumination present in the image was achieved through the implementation of BaSiC [216] ImageJ's plugin. BaSiC is a tool used to correct the background and shading of image sequences. It does this by using a method called sparse and low-rank decomposition. In this method, the shading model in Equation 1 includes both the flat-field component  $S(x)$  and the dark-field component  $D(x)$ .

$$I^{meas}(x) = I^{true}(x) \times S(x) + D(x) \quad (1)$$

Equation 1 represents a linear relationship between a measured image,  $I^{meas}(x)$ , and its uncorrupted true counterpart,  $I^{true}(x)$ .  $S(x)$  denotes the variation in the illumination level across an image, referred to as flat-field and  $D(x)$  known as dark-field. As seen in Fig. 6, BaSiC initially constructs a measurement matrix  $I$  (step I), which is then separated into a low-rank matrix  $I^B$  and a sparse residual matrix  $I^R$  (step II). The highest rank of the low-rank matrix is two. This is because each column is the sum of a scaled version of  $S(x)$  (scaled by  $B_i$ ) and  $D(x)$ , both of which are initialised with zeros in step III. In step IV, the residual matrix's sparsity is improved by promoting a reweighted L1-norm. Furthermore, the sparsity of  $S(x)$  and  $D(x)$  is regulated in the Fourier domain to enforce smooth constraints on both functions. The optimisation problem is resolved by applying the linearised augmented Lagrangian approach. An automated parameter adjustment approach produces the optimal regularisation parameters for  $S(x)$  and  $D(x)$ , which can adapt to various image contents. BaSiC algorithm breaks down the shading-free true image  $I^{true}(x)$  of the  $i$ th frame of a time-lapse microscopy video into two components: a spatially constant baseline signal,  $B_i$ , and a spatially varying foreground signal that represents the biological features of interest. This decomposition aims to enhance the accuracy of single-cell quantification. The model for a time-lapse image is as follows:

$$I^{meas}(x) = (B_i + F_i(x)) \times S(x) + D(x) \quad (2)$$

Due to background bleaching and variable experimental circumstances,  $B_i$  is typically not consistent across frames (Figure. 10a). By using equation (2) in reverse, the estimated  $S(x)$ ,  $D(x)$ , and  $B_i$  are used to adjust the intensity profile of each frame. This correction effectively removes both spatial shading effects and temporal drift, as seen in Fig. 10a versus c.

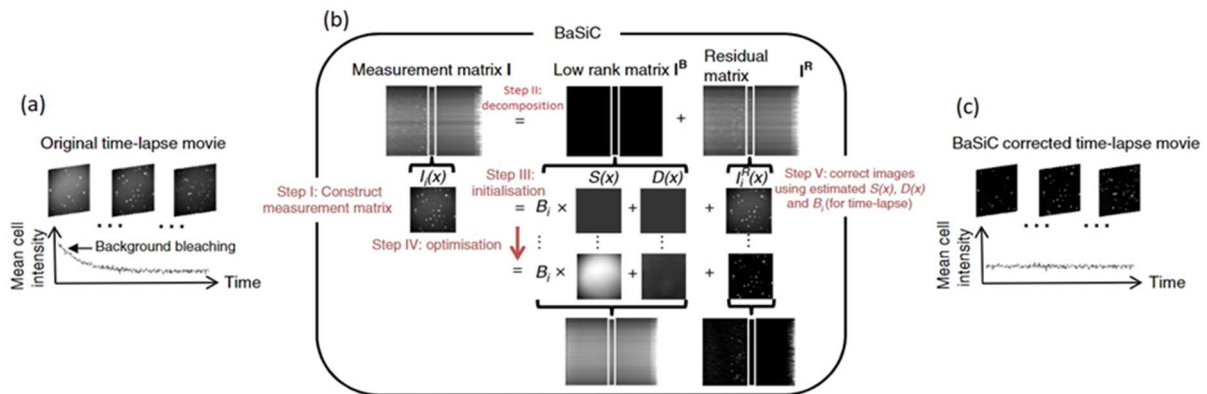


Figure 10: BaSiC is an automatic correction method for dynamic time-lapse data. (a) A time-lapse movie corrupted by both shading in space and photobleaching in time. (b) The BaSiC workflow. (c) BaSiC corrects both spatial shading and background over time.

The BaSiC ImageJ's plugin interface for image background and shading correction is shown in Figure 11. Through the User Interface, users may pick a shading model, define shading estimation profiles, and select a processing stack. Users have control over how baseline drift is handled and may manually or automatically modify regularisation parameters for dark field and flat-field corrections. Furthermore, there are choices for making full shading adjustments or only calculating shading.

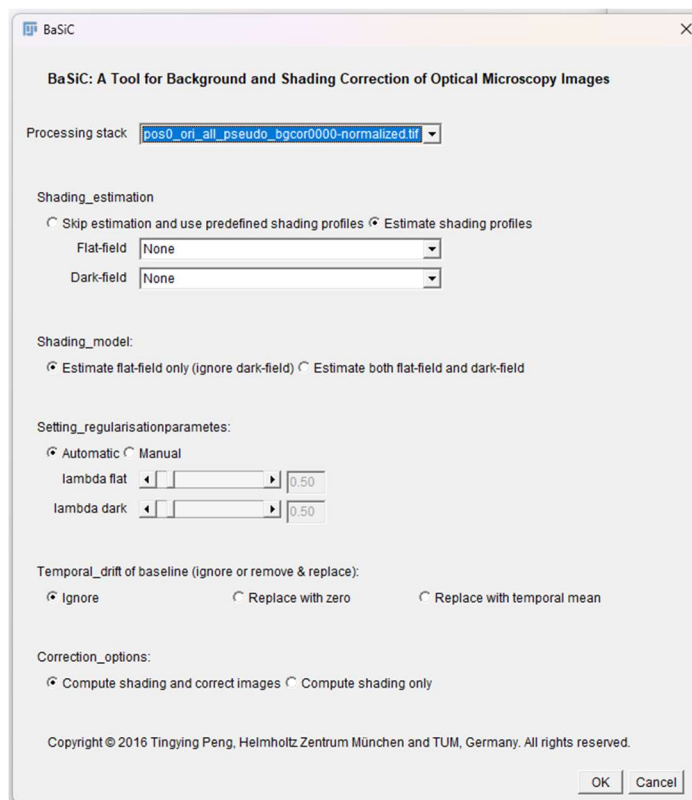


Figure 11: BaSiC interface for background and shading correction.

Systematic implementation of the BaSiC tool began with the use of its default settings. These settings typically included the automatic estimation of both flat-field and dark-field shading profiles based on the input image stack. The tool was initially configured with the "Automatic" option for the regularisation parameters ( $\lambda$  flat and  $\lambda$  dark), which control the smoothness of the estimated shading profiles. These configurations served as a baseline to evaluate the tool's effectiveness in correcting uneven illumination and shading artefacts in microscopy images. After applying the BaSiC correction with these default settings, the tool's performance was assessed using the Coefficient of Variation (CV) within the background regions of the images. The CV was chosen as the primary metric because it quantifies the uniformity of the background after correction, with lower CV values indicating more effective illumination correction. This evaluation provided an initial measure of how well the default settings minimised background intensity variations. If the default settings did not achieve satisfactory results, indicated by a higher-than-acceptable CV, manual adjustments were made to further optimise the correction. Specifically, the regularisation parameters ( $\lambda$  flat and  $\lambda$  dark) were manually fine-tuned. These parameters are crucial as they influence the smoothness of the flat-field and dark-field estimations. The manual adjustment process involved iteratively modifying these parameters and re-evaluating the CV after each change, continuing until optimal uniformity in the background was achieved. The parameter adjustment process was considered complete once the CV indicated that background uniformity had reached an optimal level. The final settings were those that produced the lowest CV within the background regions while ensuring the overall integrity of the signal in the image. These optimal settings represented the most effective configuration of the BaSiC tool for the specific microscopy images being analysed.

### **3.4.2 Image normalisation**

After correcting uneven illumination, image normalisation using quantile-based normalisation [217] is implemented using an ImageJ plugin to normalise the distribution of values in multiple images. The objective of the quantile approach is to equalise the distribution of probe intensities across all arrays in a given set. The approach is based on the basis that a quantile-quantile plot indicates that the distribution of two data vectors is identical if the plot forms a straight diagonal line and different if it deviates from a diagonal line. The principle is expanded to  $n$  dimensions so that if all  $n$  data vectors follow the same distribution, plotting the



quantiles in  $n$  dimensions results in a straight line along the unit vector  $\left(\frac{1}{\sqrt{n}}, \dots, \frac{1}{\sqrt{n}}\right)$ . This suggests that a set of data can be given the same distribution by projecting the points of an  $n$  dimensional quantile plot onto the diagonal. Let  $q_k = (q_{k1}, \dots, q_{kn})$  for  $k = 1, \dots, p$  represents the vector of the  $k$ th quantiles for all  $n$  arrays  $q_k = (q_{k1}, \dots, q_{kn})$ . Additionally, let  $d = \left(\frac{1}{\sqrt{n}}, \dots, \frac{1}{\sqrt{n}}\right)$  be the unit diagonal. To achieve the alignment of all quantiles along the diagonal, one might consider projecting  $q$  onto  $d$ .

$$\text{Proj}_d q_k = \left(\frac{1}{n} \sum_{j=1}^n q_{kj}, \dots, \frac{1}{n} \sum_{j=1}^n q_{kj}\right) \quad (3)$$

This implies that each array can be given the same distribution by taking the mean quantile and using it to replace the value of the data item in the original dataset. This concept motivates the following algorithm for normalising a set of data vectors to share the same distribution: Given  $n$  arrays of length  $p$ , first form matrix  $X$  of dimensions  $p \times n$ , where each array is a column. Next, sort each column of  $X$  to generate  $X_{\text{sort}}$ . Then, calculate the row-wise means of  $X_{\text{sort}}$  and assign this mean to each element in that row, resulting in  $X'_{\text{sort}}$ . Finally, obtain  $X_{\text{normalised}}$  by rearranging each column of  $X'_{\text{sort}}$  to match the original ordering in  $X$ . The quantile

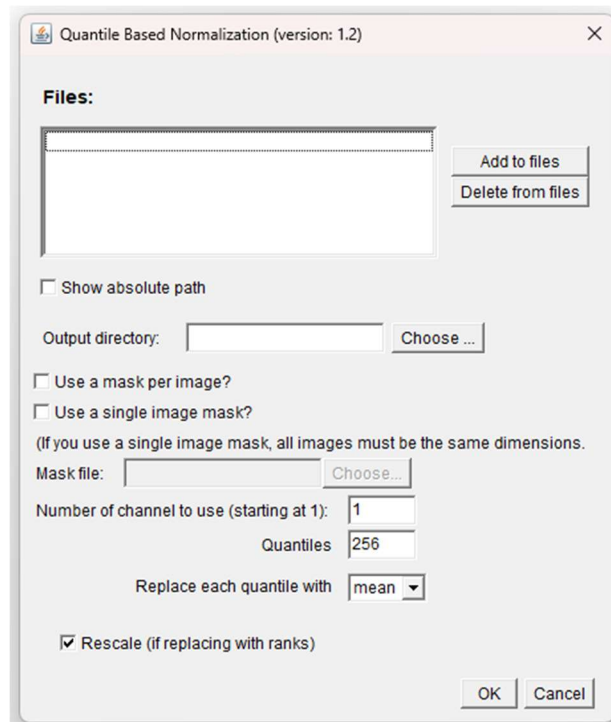


Figure 12: Quantile Based Normalisation tool interface.

normalisation approach is a particular instance of the transformation  $x'_i = F^{-1}(G(x_i))$ , where  $G$  is estimated using the empirical distribution of each array and  $F$  is estimated using the empirical distribution of the averaged sample quantiles. Further enhancements to the approach might be made by improving the estimation of  $F^{-1}$  and  $G$  to provide smoother results. The application of quantile normalisation to image stacks can be performed with the same technique. In essence, the concept entails categorising all the values present in every image and distributing them into a number of quantiles. Subsequently, the mean value across all images is substituted for each value inside a certain quantile. Consequently, every image should have highly similar value distributions when analysing their histograms. Figure 12 shows the Quantile Based Normalisation tool interface, allowing users to add or delete files for processing. The panel includes options to specify an output directory, select a masking method, set the number of quantiles and the replacement value, and enable rank-based rescaling if needed.

The process of systematically implementing the Quantile-Based Normalisation tool began with the use of its default settings to establish a baseline for performance evaluation. The tool was initially configured with the default settings, which included setting the number of quantiles to 256 and replacing each quantile with the mean value. The "Rescale (if replacing with ranks)" option was also enabled by default. These settings were chosen to standardise the intensity distribution across the images or channels to be processed. After applying the default settings, the performance of the normalisation process was assessed using the Coefficient of Variation (CV) across the images. The CV provided a straightforward metric to evaluate how well the default settings achieved consistent normalisation. If the CV indicated that the default settings did not produce satisfactory results, such as when the intensity distribution remained uneven or biologically relevant features were distorted, manual adjustments were made. For example, the number of quantiles was adjusted to a different value if the default 256 quantiles did not provide the desired level of granularity in the normalisation process. Additionally, the option to replace each quantile with a value other than the mean, such as the median, was considered if the mean replacement introduced unwanted bias. The manual adjustment process was iterative, with parameters being tweaked and the CV re-evaluated after each adjustment. This process continued until the normalisation achieved the desired uniformity across images while preserving the integrity of the biological data. The parameter adjustment process was considered complete when the CV indicated that the images were evenly normalised, achieving a balance between uniformity and biological relevance. The final settings were those that

produced the lowest CV across the images, ensuring that the normalisation was both effective and reliable.

In order to evaluate the effectiveness of image correction, the Coefficient of Variation (CV) was employed as a key metric for assessing uniformity, specifically within the background regions of the image. The CV is calculated using the following equation:

$$CV = \left( \frac{\text{Standard Deviation}}{\text{Mean}} \right) \times 100 \quad (4)$$

This equation quantifies the degree of variation remaining in the background after correction, with a lower CV indicating more effective background correction. To obtain the necessary pixel intensity data for calculating the CV in the background regions, the Plot Profile tool within ImageJ was used. After applying background correction, regions expected to represent the background were identified and selected by drawing lines across these areas using ImageJ's Line Tool or Freehand Line Tool. The Plot Profile tool then generated a profile of pixel intensities along these selected lines, providing a detailed view of the intensity variations within the background. The pixel intensity data from the Plot Profile was then used to calculate the mean and standard deviation of the intensities within these background regions. This calculation could be performed directly within ImageJ using its measurement functions or alternatively in a spreadsheet program like Excel, where standard statistical functions were applied. With the mean and standard deviation values derived from the background pixels, the CV was calculated to assess the uniformity of the corrected background. A lower CV in these regions indicated that the background correction had successfully minimised variations, achieving a more uniform and consistent background.

### **3.5 Cell segmentation**

Cell segmentation is the process of separating an image of a cell or group of cells from the background and other cells. The task of segmentation involves identifying the boundaries of individual cells and dividing the image into distinct regions, each representing a single cell [218]. The aim of cell segmentation is to accurately and reproducibly identify individual cells in an image, which is an important stage in biological and medical applications such as cell

tracking, cell counting, and cell morphometry. The most difficult parts of cell segmentation are the different shapes, sizes, and intensities of the cells and the noise and cells that overlap. To overcome the challenges, various methods are used to separate cells depending on the quality and complexity of the images. These methods can range from simple thresholding techniques to more complex machine-learning algorithms. In this study, the combination of Ilastik and ImageJ is employed, where Ilastik is used to convert the image that has undergone the preprocessing pipeline into image probabilities, and ImageJ is used to convert these image probabilities into a binary image for object detection.

### **3.5.1 Ilastik**

Ilastik was initially released in 2011 by researchers at the University of Heidelberg [219], and then it was further maintained and developed by Anna Kreshuk's group at the European Molecular Biology Laboratory [220]. Ilastik is user-friendly, free, open-source, and available for Windows, Mac and Linux operating systems. The software usage did not require previous experience in image processing and offered a simple user interface with pixel-level and object-classification capabilities for image segmentation and classification [219]. The implementation of Ilastik for cell segmentation involved a combination of automatic and manual approaches to achieve optimal results. Ilastik uses a supervised machine learning approach, where various image features such as intensity, texture, and edges are employed to classify and segment different regions within an image. The Random Forest algorithm, which serves as the classifier in Ilastik, was set automatically by the tool. The automatic configuration of the Random Forest included the number of trees and other relevant parameters, ensuring a robust starting point for classification. However, the selection of image features and the process of image annotation were handled manually. Features such as intensity, edge, and texture filters were manually chosen based on their relevance to the specific characteristics of the images being analysed. The performance of the classifier was evaluated using the Out-of-Bag (OOB) error, which is generated during the training of the Random Forest. The OOB error served as a metric for assessing the generalisation performance of the classifier based on the selected features and image annotations. A lower OOB error indicated that the selected features and annotations were effectively capturing the necessary information for accurate classification. The details about the selection of features and image annotation will be discussed further in the next section.

If the OOB error indicated that the initial settings did not produce satisfactory results, further manual adjustments were made. This involved refining the selection of image features by incorporating more complex texture filters, Gaussian smoothing, or edge detectors to enhance the classifier's ability to distinguish between different regions. Additionally, more detailed image annotations were provided to improve the training process, thereby reducing the OOB error. Once a satisfactory OOB error was achieved, the accuracy of the cell segmentation was further evaluated using the Sørensen-Dice coefficient. The Dice coefficient, which measures the overlap between the predicted segmentation and the ground truth, provided a quantitative metric for segmentation accuracy. A higher Dice coefficient indicated better segmentation performance, with values closer to 1 representing near-perfect segmentation. The manual adjustment process was iterative, with features being added or refined and image annotations being improved. After each adjustment, the OOB error and Dice coefficient were re-evaluated. This iterative process continued until both metrics indicated satisfactory performance, specifically a low OOB error and a high Dice coefficient. The parameter adjustment process was complete when the OOB error indicated strong classification performance and the Dice coefficient confirmed accurate cell segmentation. The final settings, with the Random Forest algorithm automatically configured and the features and annotations manually optimised, achieved an optimal balance between segmentation accuracy and computational efficiency. This ensured that Ilastik was well-configured for the specific microscopy images being analysed.

### **3.5.1.1 Pixel classification**

In addition to cell segmentation, Ilastik provides the pixel classification workflow to assign a label or class to each pixel in an image based on its characteristics, such as colour, texture, and shape. The workflow provides generic pixel features such as smoothed pixel intensity, edge filters, and texture descriptors. It is used in image analysis tasks that involve semantic segmentation of time-lapse microscopy images [219]. Figure 13 illustrates the steps involved in the pixel classification process, starting from input data through feature selection, image annotation, and training, to generating output data and enabling batch processing.

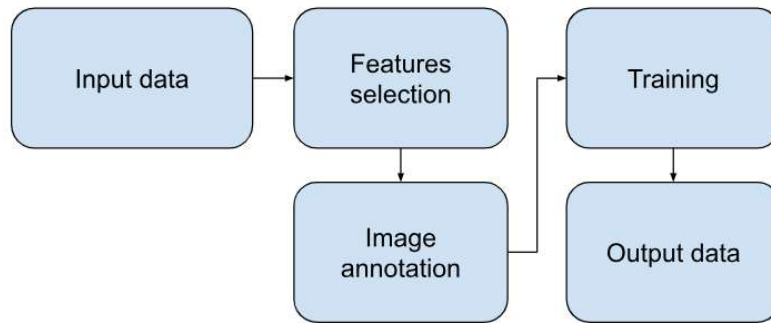


Figure 13: Pixel classification workflow in Ilastik.

### 3.5.1.1.1 The selection of features

After preprocessing the RAW image, it was used as the workflow's input image. After loading the image, proceed to the subsequent applet Feature Selection. Here, the user will select

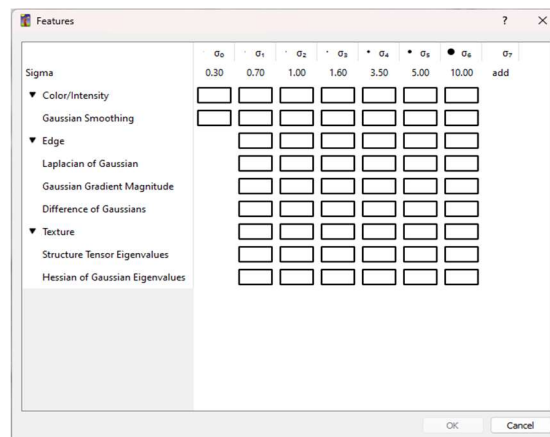


Figure 14: Menu for features selection in Ilastik.

the pixel features and their scales, which will be used in the next step to differentiate between the various pixel classes. The selection of features in Ilastik is the process of deciding which pixel characteristics to use as input for the classification process. These characteristics include colour/intensity, edge, and texture. The selection of features is crucial because it can substantially affect the classification's precision. Figure 14 depicts an example of the menu feature selection, and Table 7 provides details about the function of each feature.

The sigma parameter of the Gaussian function is used to perform image smoothing before applying the filter. It acts as a representation of the many scales that users can select for all features. Larger sigma filters can gather information from more prominent neighbourhoods but may compromise finer details. The effects of various sigma values on the performance of each filter are displayed in tables 8, 9, and 10. At a low sigma, the filter has the ability to see finer details, whereas at a high sigma, it is limited to perceiving bigger and more generalised

cell shapes. Ilastik provides a user-friendly interface for selecting and combining multiple features, with accompanying visualisations designed to impart a deeper understanding of the consequences of feature choices on the classification results. The selected features for this study were sigma values of 3.50, 5.00, and 10.00 for colour/intensity, edge, and texture. These values were chosen for several reasons. Firstly, the values capture a broader context because the selected sigma values cause the Gaussian filters to consider a larger neighbourhood around each pixel. This allows the classifier to take into account more spatial context when making decisions about pixel labels. Secondly, the values better distinguish between classes, as the sigma values help the classifier separate different classes, especially if the classes have similar local features but differ in larger-scale texture or shape. The additional context provided by a larger sigma aids the classifier in distinguishing the classes. Lastly, the higher sigma values facilitate the segmentation of whole structures. A higher sigma is needed to capture the relevant features for images with larger objects or structures. For example, to segment large cells, a too small sigma may only detect the edges and fail to capture the full cell shape.

Table 7 : Type of filters and functionality

| Feature         | Filter                            | Function  |
|-----------------|-----------------------------------|---|
| Color/intensity | Gaussian smoothing                | The technique used to smooth out intensity images represents the light intensity at each point in the image. This technique helps remove noise from the image. A larger sigma value will result in more smoothing, while a smaller value will result in less smoothing. |
| Edge            | Laplacian of Gaussian (LoG)       | The method used to identify edges in an image is by applying a Gaussian blur and a Laplacian operator to calculate the second derivative of the image.  |
|                 | Gaussian Gradient Magnitude (GGM) | The method used to identify edges in an image by applying a Gaussian blur and a gradient operator to calculate the magnitude of the image gradient.   |

|         |                                 |  |
|---------|---------------------------------|--|
|         | Different of Gaussian (DoG)     | The method consists of subtracting two Gaussians, where a kernel has a standard deviation smaller than the previous one. The convolution between the resulting kernel subtraction and the input image leads to edge detection in the image.  |
| Texture | Structure Tensor Eigenvalues    | The structure tensor encodes an image's local gradient information, and its eigenvalues can determine the dominant gradient orientation at a given spot. This information can help identify images' edges, corners, and recurring patterns. Structure tensor eigenvalues determine texture anisotropy, which can differentiate between textures. |
|         | Hessian of Gaussian Eigenvalues | The Hessian of Gaussian is a second-order derivative of a Gaussian function. These methods compute eigenvalues used to determine the dominant orientation of the image gradient at a given point.  |

Table 8: Gaussian smoothing at sigma = 1, 3.5 and 10.

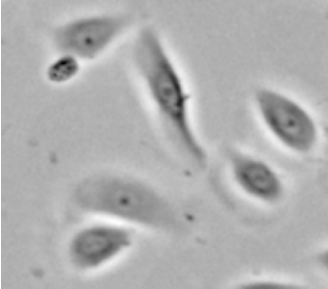
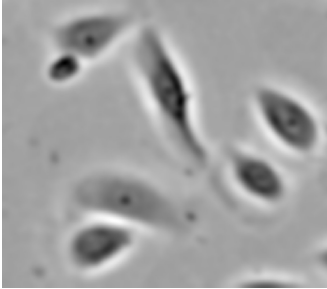

| Gaussian smoothing  |   |   |
|---|---|---|
| Sigma, $\sigma=1$   | $\sigma=3.5$  | $\sigma=10$   |
|  |  |  |



Table 9: Laplacian of Gaussian at sigma = 1, 3.5 and 10.

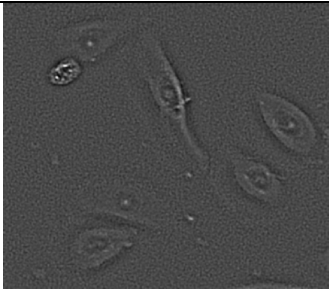
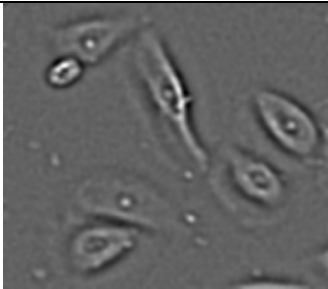

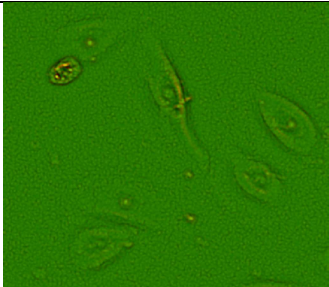
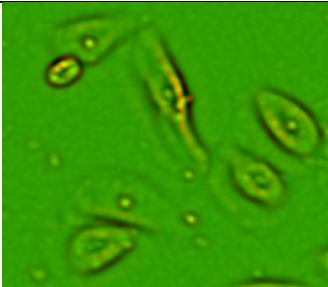
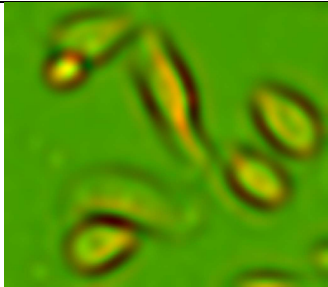
| Laplacian of Gaussian   |   |   |
|---|---|---|
| $\sigma=1$  | $\sigma=3.5$  | $\sigma=10$   |
|  |  |  |

Table 10: Hessian of Gaussian Eigenvalues at sigma = 1,3.5 and 10.

| Hessian of Gaussian Eigenvalues  |  |  |
|--|--|--|
| $\sigma=1$   | $\sigma=3.5$   | $\sigma=10$  |
|  |  |  |

### 3.5.1.1.2 Pixel classification training

The next step in pixel classification involves training a classifier to differentiate object classes. This process is accomplished through iterative annotation, evaluation of interactive prediction, and correction of errors through further annotation. Users must perform input labelling in order to begin classifier training. Each label added should represent a pixel class to be separated. This study utilised two labels: yellow for the cell and blue for the foreground. Figure 15 shows the image labelling for the cell and background. The cell annotation approach used in this study focuses on three scopes: the selection of images, annotation areas, and cell selection. The image selection is made by choosing one in the middle of the dataset to ensure it is representative of the overall data distribution and captures the average characteristics of the image dataset, such as cell shape. For the annotation area selection, the task involves annotating two areas at the top, one in the middle, and two at the bottom of the image. This method ensures the annotation covers the entire image, accounting for any spatial variability

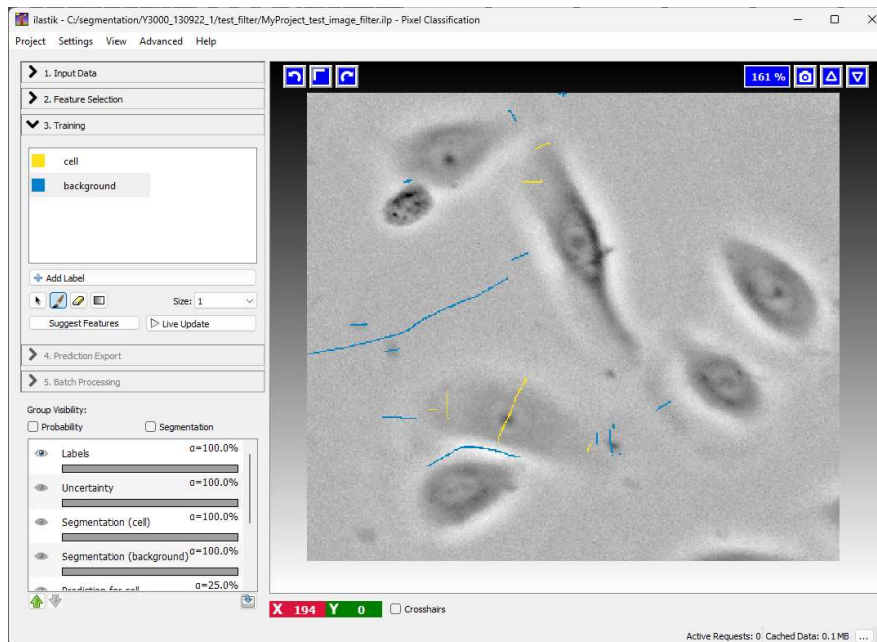


Figure 15: Image labelling.

in the appearance of cells, avoiding biasing the model towards any particular region, and helping the model generalise across different regions. In terms of cell selection, three to five cells in each area will be annotated based on their clear appearance and background. Focusing on cells with clear backgrounds and appearances makes sure that the annotations are clear and makes it easier to train a model that can tell the difference between cells and background noise, making it better at finding cells in new images.

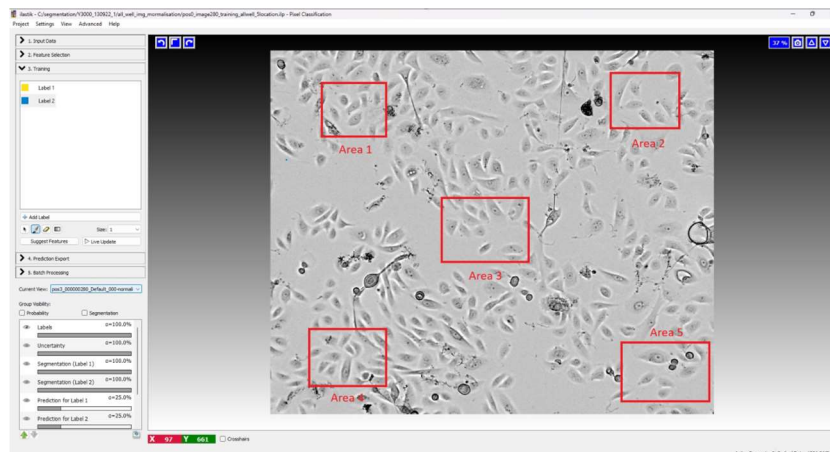


Figure 16: Cell Annotation Areas for Model Training.

Figure 16 shows five annotation areas in the image, distributed to ensure comprehensive coverage and account for spatial variability. To initiate classifier training and view predictions, press the "Live Update" button. The Random Forest classifier will start the training process based on the image annotations, and the trained classifier will predict the classes when completed. The predictions will be shown as an overlay on the image. Figure 17 illustrates the

predicted image for three classes, with yellow representing cells, blue for the background, and light blue indicating uncertainty. To improve the image annotation, the regions of uncertainty require relabelling to minimise the area of uncertainty. A well-trained classifier has minimum uncertainty within class regions, such as between cells and backgrounds. Generally, in the Random Forest model, the out-of-bag (OOB) error value will indicate the uncertainty region.

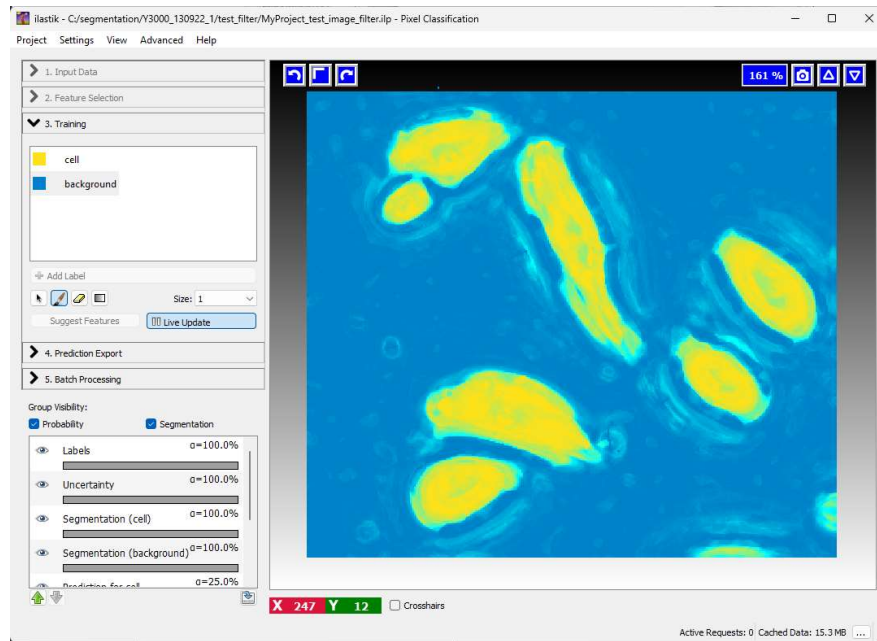


Figure 17: Image after training process

After completing the training process, the Out-of-bag (OOB) error value was used to assess the accuracy of the random forest model. The OOB error measures the generalisation performance of a random forest and is calculated as the proportion of OOB samples misclassified by the model. These samples are the ones in the dataset that are not used in the training of a particular decision tree within the random forest. A lower OOB error rate generally indicates better generalisation performance of the model, meaning that it can classify samples it has never seen before correctly. In contrast, a higher OOB error rate may indicate that the model is overfitting to the training data and is not generalising well to unseen samples. Therefore, the lower the OOB error, the more accurate the classification.

Table 11 illustrates the comparison of different feature selection configurations and their corresponding Out-Of-Bag (OOB) error values. Configuration No. 1 was selected as the optimal setting because it resulted in the lowest OOB error value of 0.008. This configuration provided the best balance among the selected features and scales ( $\sigma$ ), leading to more accurate predictions. In contrast, Configuration No. 2, which had an OOB error of 0.046, was automatically generated by the Ilastik tool. This setting did not perform as well, likely due to

less tailored feature selection. Configuration No. 3, which involved selecting all feature values across all scales ( $\sigma$ ), resulted in the highest OOB error of 0.058, indicating that including all features indiscriminately may have introduced noise or overfitting, reducing the overall model performance.

Table 11: Comparison of Features Selections

| No. | Features              |                       |                             |                        |                              |                                  | OOB error |
|-----|-----------------------|-----------------------|-----------------------------|------------------------|------------------------------|----------------------------------|-----------|
|     | Gaussian Smoothing    | Laplacian of Gaussian | Gaussian Gradient Magnitude | Different of Gaussians | Structure Tensor Eigenvalues | Hessian of Gaussians Eigenvalues |           |
| 1   | $\sigma=3.5,5,10$     | $\sigma=3.5,5,10$     | $\sigma=3.5,5,10$           | $\sigma=3.5,5,10$      | $\sigma=3.5,5,10$            | $\sigma=3.5,5,10$                | 0.008     |
| 2   | $\sigma=1.6,5,10$     | $\sigma=1.6,3.5,5,10$ | $\sigma=1.6,3.5,5,10$       | $\sigma=1.6,3.5,5,10$  | $\sigma=1.6,3.5,5,10$        | $\sigma=1.6,3.5,5,10$            | 0.046     |
| 3   | $\sigma = \text{all}$ | $\sigma = \text{all}$ | $\sigma = \text{all}$       | $\sigma = \text{all}$  | $\sigma = \text{all}$        | $\sigma = \text{all}$            | 0.058     |

### 3.5.1.1.3 Generating the output file

The output-generated file can be performed by following the workflow named “Predicted Export” tab. Using this tab, the probability image can be generated by clicking the “export all” button. The probability images can then be post-processed in ImageJ using custom

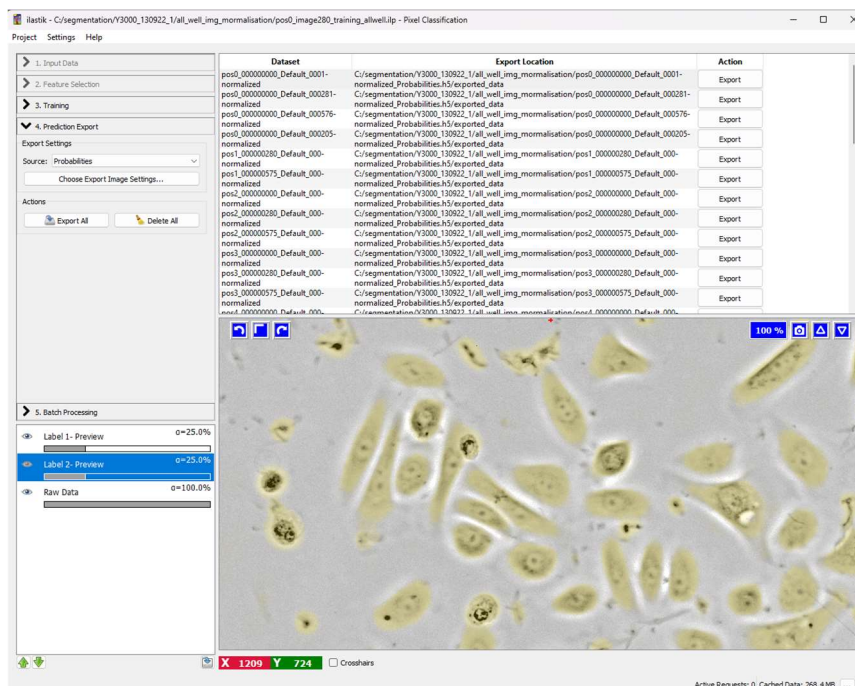


Figure 18: The prediction export tab in Ilastik displays export options and a preview of the segmented image.

thresholding, removing small objects, and converting the image to binary images. The example of the prediction export tab is shown in Figure 18.

### 3.5.1.1.4 Batch processing

Batch processing in Ilastik allows users to apply a trained classifier to multiple images efficiently. Image files for batch processing can be added by dragging and dropping them into the file browser or using the "Select Raw Data Files" button. This method allows single or multiple files to be selected from the file selection. After setting up the export settings and adding the files, click on the "Process all files" button. Ilastik will then begin batch processing all images and write the resulting classification results to the specified output files. For environments without graphical capabilities or for automation, Ilastik can be run in headless mode. This requires using the command line with specific flags and arguments: `--headless` to trigger headless operation, `--project` to specify the path to the project file, and `--input_axes` to specify the meaning of the axes if the default guess is incorrect. Figure 19 illustrates the Batch Processing tab in Ilastik, displaying the selected raw data files for processing and the option to initiate the process.

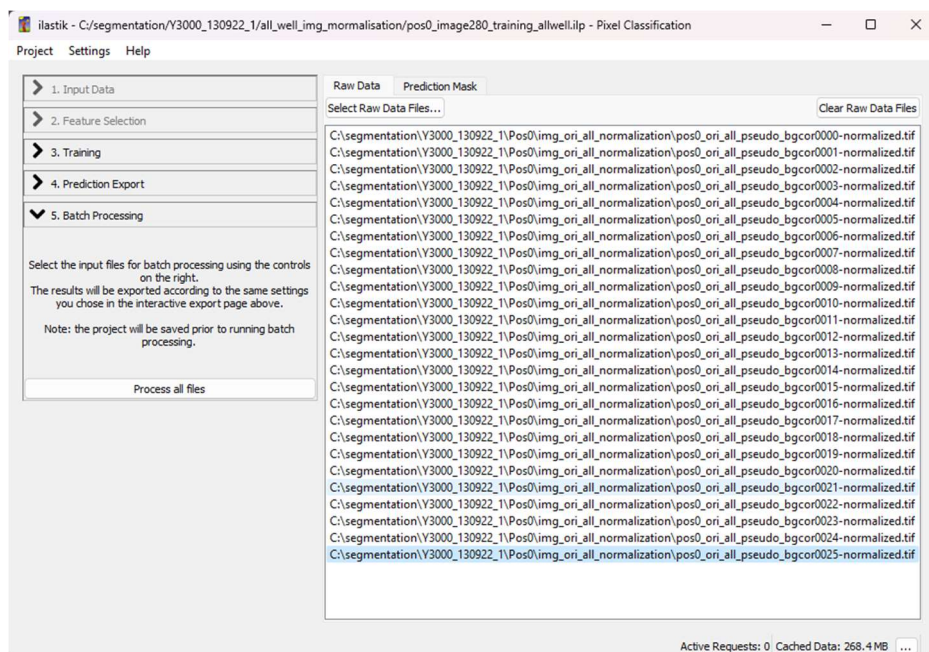


Figure 19: The Batch Processing tab in Ilastik.

### 3.6 Post-processing

At this stage, the post-processing for the image probabilities from Ilastik is done using ImageJ auto threshold. The step of auto threshold here is to convert the image to a binary image, making it easier to identify objects. Specifically, the IsoData auto threshold method is used. The method automatically determines the threshold value by iteratively adjusting it until the average of the foreground and background pixels converges. The algorithm begins with an initial threshold value,  $t$ , typically set to the midpoint of the range of pixel values in the image. Based on this initial threshold, the image is divided into two groups: one consisting of pixel values less than or equal to  $t$ , and the other consisting of pixel values greater than  $t$ . The mean of all pixel values in each group is then calculated, denoted as  $m_L$  for the lower group and  $m_H$  for the higher group. A new threshold value,  $t_{new}$ , is then calculated as the average of these two means, using the equation 5. This new threshold becomes the current threshold for the next iteration. This iterative process continues, with the threshold being updated each time, until the change in threshold value between successive iterations is smaller than a predefined tolerance level, indicating convergence.

$$t_{new} = \frac{m_L + m_H}{2} \quad (5)$$

Once the image is converted to binary, small objects can be removed to reduce noise, and further processing steps can be applied to enhance the segmentation and ensure accurate identification of the objects of interest. Figure. 20 illustrates the settings window for the Auto Threshold plugin showing various options for thresholding images.

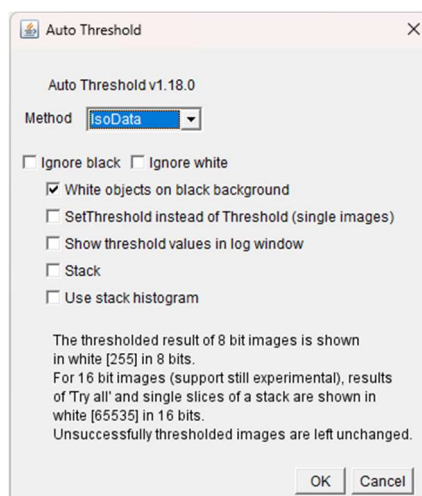


Figure 20: Auto Threshold Settings in ImageJ .

The metric used to evaluate cell segmentation in this study is the Sørensen-Dice coefficient. The Sørensen-Dice coefficient is a statistical tool used to measure the similarity between two sets. It was independently developed by the botanists Thorvald Sørensen and Lee Raymond Dice in the 1940s [221][222]. The Sørensen-Dice coefficient is calculated using the following equation 6:

$$D(A, B) = 2 \times (|A \cap B|) / (|A| + |B|) \quad (6)$$

In the equation  $|A|$  and  $|B|$  are the cardinalities (number of elements) of set  $A$  and  $B$ , respectively. Where  $|A \cap B|$  is the number of elements common to both sets.

In the context of image segmentation, sets  $A$  and  $B$  represent the sets of pixels in the segmented image and the ground truth image, respectively. The coefficient ranges between 0 and 1, where 0 indicates no overlap between the sets, and 1 indicates perfect agreement. The coefficient is particularly valuable for quantifying the performance of cell segmentation algorithms by comparing the automated segmentation results with the manually annotated ground truth, thereby providing a metric for the accuracy and reliability of the segmentation process.

### 3.7 Cell Tracking

Cell tracking refers to the process of monitoring and analysing the movement and behaviour of living cells over time. This involves the visual depiction, characterisation, and quantification of biological processes at the cellular and subcellular levels within intact living organisms. The primary goal is to follow the cells' trajectories in both space and time. The benefit of cell tracking is to study various biological processes such as cell motility, division, and interaction with other cells, providing insights into embryologic development, wound

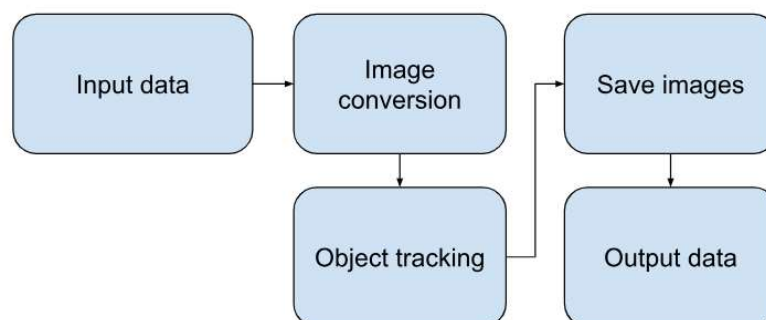


Figure 21: Cell tracking workflow in CellProfiler.

healing, tissue maintenance, cancer metastasis, and immune responses [223]. In general, cell tracking is a crucial technique in biological research and medical diagnostics, providing an in-depth understanding of cellular activities and facilitating progress in scientific knowledge and practical uses. The task for cell tracking in this study is using CellProfiler, which analyses binary images and identifies cells as objects to extract quantitative data. Figure. 21 illustrates the cell tracking workflow in CellProfiler. The process starts with input data, which undergoes image conversion. The converted images are then used for object tracking. The tracked objects and processed images are saved, resulting in the final output data.

### 3.7.1 CellProfiler

CellProfiler was first developed in 2005 and officially released in 2006 [224], where the tool is used for quantifying and examining images of cells. This tool allows biologists who do not have expertise in computer vision or programming to accurately quantify characteristics in a large number of images. CellProfiler offers advanced computational methods for image analysis, which are structured as separate modules that can be combined in order to create a pipeline. This pipeline detects and quantifies cells or other biological entities and their morphological characteristics [225]. CellProfiler is specifically designed to generate comprehensive data for every individual cell or item of interest in each image and perform the same unbiased analysis on a large scale, such as across hundreds or millions of photographs. Regarding flexible feature extraction, separate modules assess conventional morphological

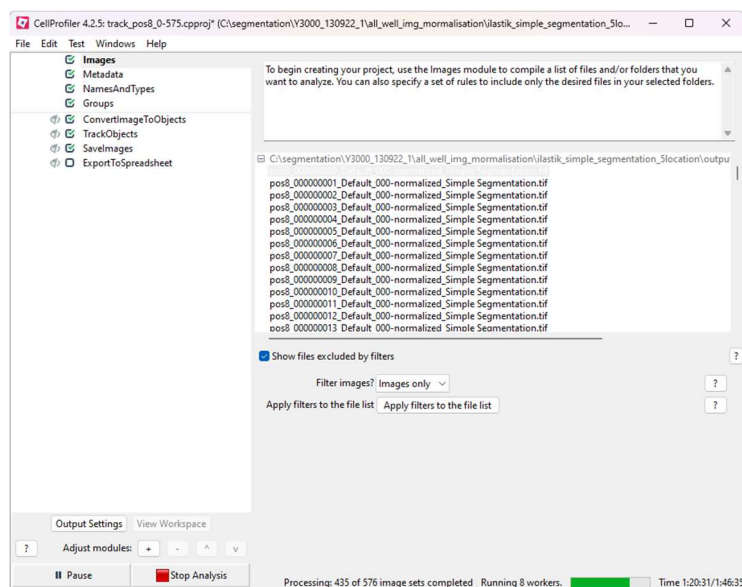


Figure 22: CellProfiler Interface.



characteristics such as dimensions, form, brightness, and texture. Complex information may be extracted using customised combinations of components. The advantage of CellProfiler is that it is user-friendly and that each module includes expertly written documentation from the imaging and biology fields. This ensures that image processing is made more accessible and comprehensible for scientists. Moreover, each configuration is elucidated in pragmatic terms to assist researchers in applying it. Figure 22 illustrates an example of the CellProfiler interface pipeline. The initial step to performing cell tracking is to include the postprocessing image as input data. Then the image will be processed in a module called ConvertImageToObjects. The ConvertImageToObjects module converts images into identifiable objects, making it particularly useful for importing previously segmented or labelled images while preserving their integer labels. The functionality of this module also offers the ability to convert grayscale images to binary before converting them into objects. In the binary conversion process, connected components of the image are assigned to the same object, which is beneficial for clearly distinguishing objects using a threshold. When using this module with grayscale images, note that they will be converted to binary images where pixel intensities below or equal to 50% of the input's full intensity range are assigned to the background (value 0), and pixel intensities above 50% are assigned to the foreground (value 1). The following pipeline uses the TrackObjects module, which enables tracking objects across sequential frames in a series of images, ensuring that each object maintains a unique identity throughout the output measurements. This module must be placed downstream of an object-identification module, such as IdentifyPrimaryObjects. TrackObjects link each object to its corresponding object in preceding and subsequent frames, facilitating the study of object lineages and the timing and characteristics of dynamic events in time-lapse images. The settings of the TrackObjects module are shown in Figure 23. The module is configured to use the "Follow Neighbors" method for tracking objects. Results are displayed with both colour and number, and the output image is named "TrackedCells."

The module provides several key measurements for each tracked object, ensuring a comprehensive analysis of object behaviour and dynamics. Each tracked object is assigned a unique identifier called a label. If an object splits or merges, the resulting child objects inherit the label of their ancestor. Additionally, the module records the ImageNumber and ObjectNumber of the parent object in the previous frame. For splits, each child object carries the label of the original object from which it was split. In the case of a merge, the child object inherits the label of the closest parent. Motion measurements include the direction of an object's

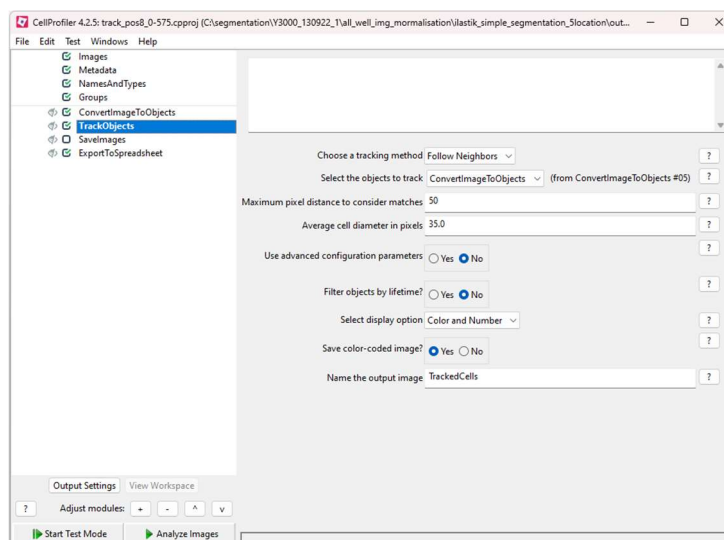


Figure 23: CellProfiler TrackObjects module interface.

motion in the x and y coordinates, known as TrajectoryX and TrajectoryY, from the previous frame to the current frame. The DistanceTraveled measurement represents the distance an object has moved from the previous frame to the current frame, calculated as the magnitude of the trajectory vectors. Displacement measures the shortest distance an object has travelled from its initial starting position to its current position, essentially a straight-line path between these points. IntegratedDistance represents the total distance an object has travelled throughout its lifetime. Linearity measures the linearity of the object's trajectory over its lifetime, calculated as the ratio of displacement (from the initial to the final location) to the integrated distance travelled, with the value ranging from 0 to 1. Lifetime measurements indicate the number of frames an object has existed, starting at 1 when the object first appears and incrementing with each frame it persists. The lifetimes of all remaining objects are reported in the final frame of the image set or movie. FinalAge is similar to Lifetime but is only reported at the final frame of an object's life or the end of the movie, whichever comes first. At this point, the final age of the object is outputted, with no values stored for earlier frames. These measurements provide detailed insights into the behaviour and characteristics of each tracked object, enabling robust analysis of dynamic events and object trajectories in time-lapse imaging.

The cell tracking method applied in this study is the Follow Neighbour method, which uses a probabilistic approach combined with graph-based optimisation techniques to ensure that the tracked cell movements are consistent and realistic. The core of the method involves maximum weighted bipartite graph matching to find the best cell pairings between consecutive frames. Additionally, an object's movement direction is more likely to be in agreement with the movement directions of its "neighbours," promoting coherent and natural cell tracking [226].

The cell tracking accuracy assessment was strategised based on a systematic approach to ensure a comprehensive and representative evaluation of the tracking algorithm. First, the frames were sampled by selecting 20 frames from the beginning, 20 from the middle, and 20 from the end of the sequence. This approach ensures that different stages of the cell population dynamics are captured, reflecting potential variations in cell density and behaviour over time. Next, ten cells were randomly chosen to track each selected frame. These cells were selected from the middle of the images, providing a representative sample of the cell population within the frame. Finally, the tracking accuracy was evaluated using the Multiple Object Tracking Accuracy (MOTA) metric [227]. This metric provides a robust measure of the tracking performance, accounting for false positives, false negatives, and identity switches, thus giving a comprehensive assessment of the tracking algorithm's effectiveness. The definition of Multiple Object Tracking Accuracy (MOTA) is as follows:

$$MOTA = 1 - \frac{\sum_t(FN_t + FP_t + IDS_t)}{\sum_t GT_t} \quad (7)$$

The variable  $t$  is the frame index, and  $FN_t$  and  $FP_t$  are the false negative and false positive counts at frame  $t$ . Furthermore, the variable  $IDS_t$  is the number of identity switches at frame  $t$ , while the number of ground-truth objects at frame  $t$  represents the variable  $GT_t$ .

### 3.7.1.1 Data extraction and cell features

All the extracted data is obtained using the ExportToSpreadsheet module in CellProfiler, where raw measurement output files are in comma-separated value (CSV) format. The data included in the CSV file are the number of cells, cell location (x and y), cell label, image number, distance, and lifespan. The data provided in the CSV files will be used to explore features such as cell proliferation and cell motility. These features offer detailed insights into cell movement and behaviour, which are essential for understanding the effects of various treatments in time-lapse images. Python will be used to calculate data automatically.

#### 3.7.1.1.1 Cell proliferation

Cell proliferation is the process of cell division and growth. This basic biological mechanism promotes multicellular organisms' growth and development. Cell proliferation is typically measured by counting the number of cells over time, assessing DNA synthesis, or using cell division markers. Cell proliferation is directly reflected in the cell growth curve and the cell growth rate. During the log phase of the growth curve, the high proliferation rate results

in a steep slope, indicating rapid cell division. Conversely, during the lag and stationary phases, the proliferation rate is lower, leading to a shallower slope on the growth curve. By analysing the cell growth curve, researchers can infer the growth rate and understand cell proliferation dynamics under various conditions.

### 3.7.1.1.1 Cell growth curves

Cell growth curves [228] are graphical representations depicting the growth of a cell population over time. These curves illustrate various phases of cell growth in culture, typically plotted as the number of cells against time. In this study, the number of cells can be found in the CSV file extracted from cell tracking using CellProfiler. Figure 24 shows the cell growth curve with four distinct phases: lag phase, log (exponential) phase, stationary phase, and death phase. The y-axis represents the number of cells (logarithmic scale), and the x-axis represents time (in days). The lag phase is characterised by minimal cell division, the log phase by rapid exponential growth, the stationary phase by a balance between cell division and cell death, and the death phase by a decline in the number of cells.

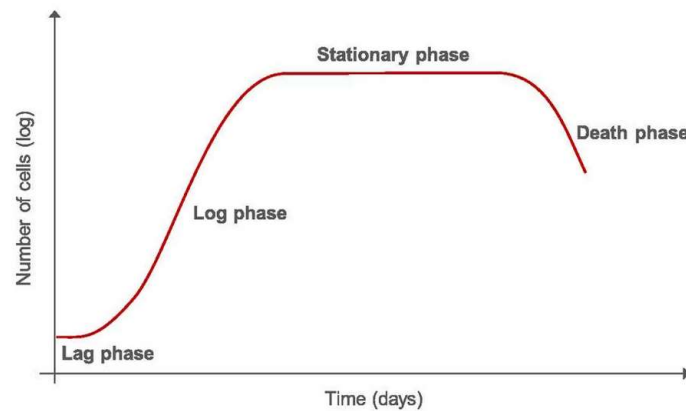


Figure 24: Cell Growth Curve [229].

### 3.7.1.1.2 Cell growth rates

Cell growth rates [228] refer to the speed at which a population of cells increases in number over a given period of time. Calculating this metric is critical for quantifying how quickly cells are proliferating under specific conditions. Equation 8 is used to calculate the cell growth rate.

$$\mu = \frac{\ln (t)-\ln N_0}{t-t_0} \quad (8)$$

where  $N(t)$  is the number of cells at time  $t$ ,  $N_0$  is the initial number of cells at the starting time  $t_0$ ,  $t$  is the final time, and  $t_0$  is the initial time.

### 3.7.1.1.2 Cell motility

Cell motility is the capacity of cells to move actively [230]. Fundamentally, migration is a component of cell motility and is an essential process for growth and survival [231]. Cells undergo transitions between stationary and mobile states in an effort to minimise a quasipotential; this phenomenon can be conceptualised as an active phase transition problem [232]. Two relevant features related to cell motility that distinguish cell behaviour under different treatments are cell migration speed and angular velocity.

#### 3.7.1.1.2.1 Cell migration speed

The cell migration speed provides information about the rate at which a cell moves from one location to another by quantifying the rate of cell relocation during migration. It is typically calculated based on the total distance a cell travels divided by the time elapsed. The speed of cell migration can be calculated using the following equation:

$$\text{Migration speed} = \frac{d}{t} \mu/\text{min} \quad (9)$$

To calculate cell migration speed, start by measuring the initial and final positions of the cell. Determine the initial position  $(x_0, y_0)$  of the cell at time  $t_0$ . Then, determine the final position  $(x_f, y_f)$  of the cell at time  $t_f$ . Next, calculate the distance travelled by using the Euclidean distance formula. The distance  $d$  is given by:

$$d = \sqrt{(x_f - x_0)^2 + (y_f - y_0)^2} \quad (10)$$

Finally, determine the time interval by calculating the difference between the final and initial times. The time interval  $t$  is given by:

$$t = t_f - t_0 \quad (11)$$

#### 3.7.1.1.2.2 Cell angular velocity

Cell angular velocity measures the rate of change in a cell's direction or orientation during migration. It quantifies the rotational motion and is calculated based on the change in

direction between sequential positions. The angular velocity of cell migration can be calculated using the following method: To calculate cell angular velocity, start by measuring the initial and subsequent positions of the cell. For example, consider the cell moving through three positions. The coordinates of these positions are: Position 1,  $(x_1, y_1)$ ; Position 2,  $(x_2, y_2)$ ; and Position 3,  $(x_3, y_3)$ . Next, calculate the direction angles of the cell's movement between these positions using the arctangent function:

1. The angle between Position 1 and Position 2:

$$\theta_1 = \tan^{-1} \left( \frac{(y_2 - y_1)}{(x_2 - x_1)} \right) \quad (12)$$

2. The angle between Position 2 and Position 3:

$$\theta_2 = \tan^{-1} \left( \frac{(y_3 - y_2)}{(x_3 - x_2)} \right) \quad (13)$$

Finally, calculate the angular velocity as the absolute difference between these two angles:

$$\text{Angular velocity} = |\theta_2 - \theta_1| \text{radians/frame} \quad (14)$$

### 3.8 Cell characterisation

In live-cell imaging studies, the comprehensive analysis of cellular behaviour over time is crucial for understanding complex cellular processes. Modelling techniques are essential to enhance predictive capabilities and simulate intricate cell behaviour accurately. Utilising interpretable "white box" machine learning, such as Genetic Programming, can reveal patterns and relationships in cell population behaviour under various treatments. This approach enables the extraction of valuable insights from dynamic cellular activity data, aiding in the prediction and simulation of the complexity of cell behaviour. By employing such techniques, researchers

can gain a deeper understanding of cellular processes' dynamics and improve cell behaviour's characterisation in response to different stimuli.

### **3.8.1 Genetic programming**

Genetic Programming (GP) is an automated method for creating a working computer program from a high-level problem statement [233]. It is a type of evolutionary algorithm, a subset of machine learning, that is inspired by biological evolution and its fundamental mechanisms [234]. Genetic Programming implements an algorithm that uses random mutation, crossover, a fitness function, and multiple generations of evolution to resolve a user-defined task [234]. The process starts with a population of randomly created computer programs, which is progressively evolved over a series of generations using the Darwinian natural selection principle (survival of the fittest) and analogues of various naturally occurring operations, including crossover, mutation, gene duplication, and gene deletion [233]. The study explores the use of Recurrent Cartesian Genetic Programming (RCGP) [235] in combination with pairwise [236] and ensemble [237] methods for symbolic classification [238]. Additionally, the study utilises PySR [239], a genetic programming technique that integrates symbolic regression [240] with symbolic classification.

#### **3.8.1.1 Recurrent Cartesian genetic programming**

Recurrent Cartesian Genetic Programming (RCGP) is an extension of Cartesian Genetic Programming (CGP) that allows the creation of recurrent or cyclic graphs. RCGP can learn from feedback (storing internal state), which makes it useful for tasks with limited information. The parameter known as recurrent connection probability determines the possibility of mutations resulting in the creation of recurrent connections, hence controlling the recurrent connections. Prior to exploring RCGP in depth, it is essential to have an overview of Cartesian Genetic Programming (CGP). CGP originated from a method developed by Miller et al. in 1997 for evolving digital circuits [241]. The term "Cartesian Genetic Programming" first appeared in 1999 [242] and was introduced as a general form of genetic programming in 2000 [243]. The name "Cartesian" is derived from its use of a two-dimensional grid of nodes to represent a program. CGP [244] is a variant of GP [245] that primarily generates computational structures consisting of nodes (graphs) arranged in a non-cyclic manner, with each node

identified by its Cartesian coordinates. CGP is not affected by bloat [246], which is a disadvantage commonly found in many other GP approaches. CGP chromosomes possess non-functional genes, which facilitate neutral genetic drift in the course of evolution [247]. CGP commonly employs point or probabilistic mutation without crossover and utilises a  $(1 + \lambda)$ -ES. While CGP chromosomes maintain a constant size, the number of active nodes can change during evolution, resulting in phenotypes of varying lengths. The user defines a maximum number of nodes from which only a fraction will be active. Previous studies have demonstrated that overestimating the number of nodes can greatly facilitate evolution [248]. This is believed to enhance neutral genetic drift and could possibly offset any bias towards longer sequences [249]. Each CGP chromosome has function genes ( $F_i$ ), connection genes ( $C_i$ ), and output genes ( $O_i$ ). The function genes serve as indices in a function look-up table, describing the functionality of each node. The connection genes specify where each node gets its inputs. In regular acyclic CGP, connection genes can connect a node to any prior program node or program inputs. The output genes address any program input or internal node, determining which are utilised as program outputs.

CGP programs were organised with nodes arranged in rows (nodes per layer) and columns (layers), with each node indexed by its row and column. The arrangement of nodes into rows and columns is optional. A configuration using just one row with multiple columns can be equally effective as long as the total number of nodes remains the same. The key capability of CGP lies in its ability to evolve the connections between nodes rather than being constrained by their physical arrangement in rows and columns. Equation 15 represents a generic CGP chromosome with a single row structure. In this equation,  $\alpha$  (alpha) is the arity of each node, signifying the number of inputs it can take. Additionally,  $n$  represents the total number of nodes in the program, and  $m$  signifies the number of outputs the program produces.

$$F_0 C_{0,0} \dots C_{0,\alpha} F_1 C_{1,0} \dots C_{1,\alpha} \dots F_n C_{n,0} \dots C_{n,\alpha} O_0 \dots O_m \quad (15)$$

Figure 25 illustrates a Cartesian Genetic Programming (CGP) example and its chromosome, showing that nodes connect to earlier nodes or inputs, and not all inputs are necessarily used. This allows evolution to determine important inputs. CGP's advantage over tree-based genetic programming is that node outputs can be reused, avoiding redundant recalculations. Additionally, not every node contributes to the final output, allowing for inactive nodes that support neutral genetic drift and variable-length phenotypes.



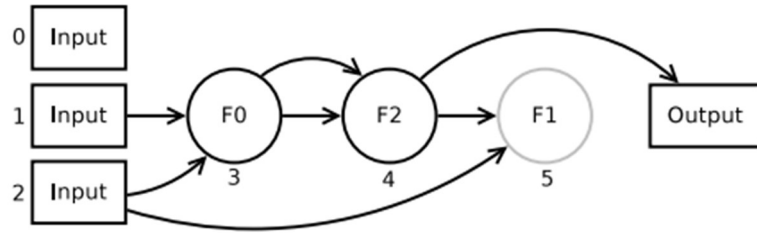


Figure 25: Example of CGP structure with inputs (0, 1, 2), functional nodes (F0, F2, F1), and output. Shows node reuse and inactive nodes for neutral genetic drift [235].

Unlike Cartesian Genetic Programming (CGP), which restricts connections to prevent loops, RCGP allows any connection within the program. This includes connections between any nodes, even allowing a node to connect to itself or to the program's inputs. Figure 26 demonstrates this flexibility with an example program generated using RCGP and its corresponding chromosome.

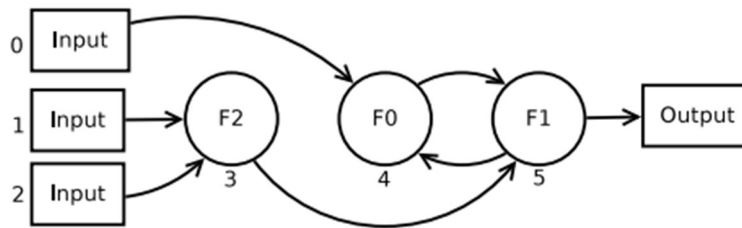


Figure 26: Example of an RCGP structure with inputs (0, 1, 2), functional nodes (F2, F0, F1), and output. Shows recurrent connections, including a node connecting to itself [235].

RCGP phenotypes operate similarly to CGP phenotypes. The process begins with the active node nearest to the inputs, where each node computes its output value based on its inputs. After all active nodes have been updated, the program outputs are recorded. However, recurrent connections can pose a challenge, as a node's output may be needed before it has been computed. To address this, all nodes are initially set to output zero until they calculate their own values. When executing a RCGP phenotype, the process unfolds as follows: First, all active nodes are set to output zero. Next, the subsequent set of program inputs is applied. After this, all active nodes are updated once, transitioning their state from program inputs to program outputs. Then, the program outputs are read. This cycle, starting from applying the next set of program inputs, is repeated until all program input sets have been applied. In RCGP, the

"recurrent connection probability" parameter determines the chance that mutations will produce recurrent connections. For instance, setting this probability at 10% means that 10% of the mutations in connection genes will create cyclic connections, whereas a setting of 0% maintains a traditional CGP setup with only acyclic connections. While RCGP chromosomes function in a manner similar to traditional CGP chromosomes, they differ primarily in how they handle program outputs.

### 3.8.1.1.1 Dataset preparation

A cell culture dataset was generated from a 12-well plate, with wells 0, 1, 2, and 3 under control conditions (no treatment). Wells 4, 5, 6, and 7 were treated with 4 ng/mL TGF $\beta$ , while wells 8, 9, 10, and 11 were treated with 10  $\mu$ M SB431542. Data were collected from all wells every 5 minutes for a duration of 48 hours. Features such as cell growth (number of cells), cell migration, and cell angular velocity were extracted from each well and calculated for every time point. However, the data for cell migration and angular velocity were averaged over the interval frames. This averaging method [250] [251] [252] [253] is a helpful approach used to analyse the overall behaviour of the cell population. The benefit of using this average interval frame is that it captures the changes over time and the patterns of variation in the datasets at various time intervals, offering more insightful measurements than a single overall average. This method minimises the intricacy of unprocessed time series data to a controlled amount while still preserving important information. Utilising standardised time intervals enables unbiased comparisons between treatments by eliminating the influence of non-uniform time points. It also facilitates trend analysis, providing a detailed understanding of treatment effects over time. Average interval frames smooth out short-term fluctuation and noise, resulting in more stable and robust measures less influenced by outliers. Equation 16 represents the calculation of the average value over a frame interval.

$$Average_j = \frac{1}{k} \sum_{i=(j-1)k+1}^{jk} x_i \quad (16)$$

The variable  $j$  represents the interval index, ranging from 1 to  $M$ , where  $M$  is the total number of intervals. The variable  $k$  denotes the number of frames in each interval. The term  $x_i$  refers to the value of the feature (such as cell migration speed) at frame  $i$ . The expression  $(j-1)k+1$  indicates the starting frame of the  $j$ -th interval, while  $jk$  signifies the ending frame of the  $j$ -th interval.

Listing 1 shows the pseudocode for calculating the average of cell migration speed and angular velocity over specified intervals. The function `calculate_interval_averages` accepts a DataFrame `df` and an interval length `interval_length`. It computes the number of intervals and iteratively calculates the mean for each interval, appending these means to a list. The list is then converted into a DataFrame and returned. The main script reads data from a CSV file, sets the interval length, calls the function to calculate interval averages, and subsequently prints and saves the resulting averages to a new CSV file

Listing 1: Average operation interval frame.

```
BEGIN
FUNCTION calculate_interval_averages(df, interval_length)
  SET num_intervals = number of rows in df DIVIDED BY interval_length
  INITIALIZE empty list interval_averages
  FOR i FROM 0 TO num_intervals - 1
    SET start_idx = i * interval_length
    SET end_idx = (i + 1) * interval_length
    SET interval_avg = mean of rows from start_idx to end_idx in df
    APPEND interval_avg to interval_averages
  CONVERT interval_averages to DataFrame interval_avg_df
  RETURN interval_avg_df
READ data from 'data.csv' into DataFrame df
SET interval_length to 10
SET interval_averages_df = calculate_interval_averages(df[['migration_speed',
'angular_velocity']], interval_length)
PRINT interval_averages_df
SAVE interval_averages_df to 'interval_averages.csv'
END
```

### 3.8.1.1.2 Dataset Normalisation

After completing the data averaging process, the next task is to perform normalisation of the datasets using Min-max normalisation. Normalisation is a technique used to scale the features of a dataset to a specific range, typically [0, 1]. This process is essential when the features have different ranges or units, as it ensures that each feature contributes equally to the analysis. In this study, the features being normalised are cell growth, cell migration speed, and angular velocity. Min-max normalisation [254], also known as min-max scaling, is done by subtracting the minimum value of a feature from each of its values and then dividing the result by the range of the feature (the difference between the maximum and minimum values). Equation 17 illustrates the min-max scaling formula.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (17)$$

Where  $X_{norm}$  represents the normalised feature value.  $X$  denotes the original feature value.  $X_{min}$  is the minimum value of the feature.  $X_{max}$  is the maximum value of the feature.

Listing 2 shows pseudocode for normalising data using MinMaxScaler. This algorithm begins by importing the necessary libraries, specifically pandas for data manipulation and MinMaxScaler from sklearn.preprocessing for normalisation. It reads the data from a CSV file into a DataFrame, initialises the MinMaxScaler with a feature range of (0, 1), and selects the columns to be normalised (growth, migration\_speed, angular\_velocity). The algorithm then fits and transforms these columns using the scaler, prints the first few rows of the normalised data, verifies the minimum and maximum values of the normalised columns, and finally saves the normalised data to a new CSV file. This process ensures that the data is scaled within a specific range, facilitating better performance and comparability in subsequent analysis or machine learning tasks.

Listing 2: Normalisation using MinMaxScaler.

```
BEGIN
  IMPORT pandas AS pd
  IMPORT MinMaxScaler FROM sklearn.preprocessing
```

```

# Load the data from a CSV file
df <- pd.read_csv('data.csv')

# Initialize MinMaxScaler
scaler <- MinMaxScaler(feature_range=(0, 1))

# Select the columns to be normalised
columns_to_normalise <- ['growth', 'migration_speed', 'angular_velocity']

# Fit and transform the data
df[columns_to_normalise] <- scaler.fit_transform(df[columns_to_normalise])
# Display the first few rows of the normalised data
PRINT df.head()

# Verify the min and max values
PRINT df[columns_to_normalise].min()
PRINT df[columns_to_normalise].max()

# Save the normalised data to a new CSV file
df.to_csv('normalised_data.csv', index=False)
END

```

### 3.8.1.1.3 Dataset Splitting

Dataset splitting is necessary for machine learning, ensuring that models are trained, validated, and tested on separate data subsets. The task approach helps evaluate the model's performance accurately and prevents overfitting [255]. The splitting involves dividing the dataset into training, validation, and testing sets by selecting data points based on the number of wells for each treatment. The training set is used to construct the model, in which the algorithm acquires knowledge from the data. The validation set is utilised to fine-tune hyperparameters and mitigate overfitting, offering an unbiased assessment during the training process. The testing set provides an unbiased evaluation of the finished model, ensuring that

the performance measures accurately represent the model's capacity to apply to new data. As mentioned earlier in this chapter, there are 12 well datasets, where datasets for Wells 0 through 3 are maintained without any treatment (control). Datasets from Wells 4 through 7 have been treated with TGF-beta, while the remaining wells, 8 through 11, are treated with SB431542. The data-splitting approach is based on the number of wells for each treatment. For example, for the control group, datasets from Wells 0 and 1 are used for training, the dataset from Well 2 for validation, and the dataset from Well 3 for testing. The same approach is also applied to datasets from wells treated with TGF-beta and SB431542. The implementation of dataset validation is based on K-fold cross-validation, where cross-validation is another method where the dataset is divided into k subsets, and the model is trained and validated k times, each time using a different subset as the validation set and the remaining k-1 subsets for training. This technique maximises the available data for training and validation, providing a more robust estimate of model performance [256].

#### **3.8.1.1.4 Cross-Validation**

Cross-validation is a widely used statistical method in machine learning for evaluating the performance of models and algorithms [257] [258]. It is beneficial when working with limited amounts of data or when assessing the model's ability to generalise to unseen data. The primary goal of cross-validation is to partition the dataset into multiple subsets, training and testing the model on each subset to obtain a more accurate estimate of its performance. Cross-validation helps mitigate the risk of overfitting by better estimating the model's performance on unseen data [259]. Using different training and testing sets during each iteration ensures that the model's ability to generalise is considered. There are several variations of cross-validation, such as k-Fold Cross-Validation, Stratified k-Fold Cross-Validation, Leave-One-Out Cross-Validation (LOOCV), Time-Series Cross-Validation, and Grouped k-Fold Cross-Validation. Each of these variations is tailored to different types of datasets and model evaluation requirements [260]. The cross-validation method helps researchers obtain a reliable estimate of their machine-learning model's performance and benefits model selection and improved predictions on new data [261]. In order to enhance the model evaluation, this study applied the K-fold cross-validation method. K-fold cross-validation divides the training set into K subsets, using each subset once as a validation set and the remaining subsets for training. This process is repeated K times, and the performance metrics are averaged over the iterations to provide a

more reliable estimate of the model's generalisation performance. Tables 12, 13, and 14 present the configuration for cross-validation applied to the dataset in the control, TGF-Beta and SB431542 treatment scenarios, demonstrating the allocation of different wells into training, validation, and testing sets across five folds. This setup ensures that each well is systematically rotated through each phase of the model evaluation process. Specifically, wells are assigned to the training set in combinations that include every well at least once across the folds, while the validation and test sets are populated to complement the training data effectively. The distribution helps in obtaining a reliable estimate of the model's generalisation performance by averaging the results across multiple configurations and treatments.

Table 12: Cross-Validation for the dataset in control and SB431542 treatment.

| Fold | Training Well | Validation Well | Test Well |
|------|---------------|-----------------|-----------|
| 1    | 0,1,9,10      | 2,8             | 3,11      |
| 2    | 0,2,8,10      | 1,9             |           |
| 3    | 1,2,8,9       | 0,10            |           |
| 4    | 0,1,8,10      | 2,9             |           |
| 5    | 0,2,9,10      | 1,8             |           |

Table 13: Cross-Validation for the dataset in control and TGF-Beta treatment.

| Fold | Training Well | Validation Well | Test Well |
|------|---------------|-----------------|-----------|
| 1    | 0,1,4,5       | 2,6             | 3,7       |
| 2    | 0,2,4,6       | 1,5             |           |
| 3    | 1,2,5,6       | 0,4             |           |
| 4    | 0,1,5,6       | 2,4             |           |
| 5    | 0,2,4,5       | 1,6             |           |

Table 14: Cross-Validation for the dataset in SB431542 and TGF-Beta treatment.

| Fold | Training Well | Validation Well | Test Well |
|------|---------------|-----------------|-----------|
| 1    | 4,5,8,9       | 6,10            | 7,11      |
| 2    | 4,6,8,10      | 5,9             |           |
| 3    | 5,6,9,10      | 4,8             |           |
| 4    | 4,5,9,10      | 6,8             |           |
| 5    | 4,6,8,9       | 5,10            |           |

### 3.8.1.1.5 Pairwise Method Approach for RCGP Symbolic Classification

Pairwise classification [262] is a method employed in machine learning, specifically in ranking and preference learning. The process involves converting a multi-class problem into a series of two-class problems, one for each pair of classes, and each combination of classes is examined individually. The advantages of using the pairwise method include its ability to be easily parallelised, as it breaks down into independent subproblems [263]. This method also provides redundancy by learning multiple classifiers for each pair of classes, which can enhance generalisation performance [264]. Additionally, it requires less training time [265]. The classification in this study aimed to determine which datasets belong to Control, TGF-Beta, and SB431542. In a pairwise approach, the classification is done in pairs, resulting in three binary classifiers for the pairs (Control, SB431542), (Control, TGF-Beta), and (SB431542, TGF-Beta). Each classifier will learn to distinguish between the two classes in its pair. To classify a new instance, each of these classifiers makes a prediction, and these predictions are then aggregated to decide the final class label for the instance using an ensemble method. Figure 27 showcases the implementation of pairwise classification using Recurrent Cartesian Genetic Programming (RCGP), utilising three cell features: Cell Growth, Cell Migration Speed, and Cell Angular Velocity, labeled as Input 0, Input 1, and Input 2. Each input is employed across models for comprehensive pairwise comparisons where Pairwise 1 distinguishes between Control and SB431542 treatments, Pairwise 2 differentiates between Control and TGF-beta treatments, and Pairwise 3 compares SB431542 to TGF-beta treatments. Each model's output is depicted as continuous values to enhance clarity and understanding,



with the figure legend explicitly defining each comparison. All three models utilise the same inputs but are trained to distinguish between different pairs of treatments, and their outputs collectively determine the final classification of the treatment type.

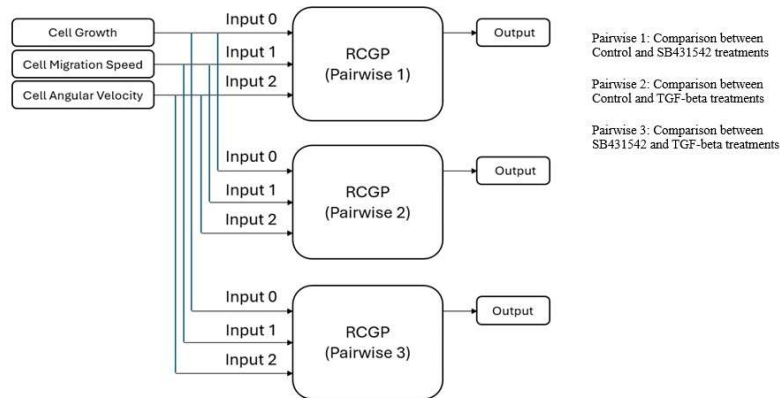


Figure 27: Pairwise Approach Using Recurrent Cartesian Genetic Programming implementation.

Before proceeding with the training process, the datasets need to be structured as shown in Listing 3. The listing presents a sample dataset formatted for Recurrent Cartesian Genetic Programming (RCGP). The first line of the dataset specifies that each sample contains 3 input values and 2 output values, and there are a total of 60 samples. Each subsequent line represents an individual sample, with the input values listed first and the output values separated by commas. For example, the first sample consists of input values 0.583, 0.121, and 0.532, followed by output values 0 and 1. This format is consistently applied across all samples in the dataset.

Listing 3: Example Dataset Format.

```
3,2,60
0.583,0.121,0.532,0,1
0.583,0.285,0.156,0,1
0.583,0.036,0.029,0,1
0.583,0.201,0.338,0,1
0.583,0.109,0.446,0,1
0.583,0.125,0.395,0,1
0.583,0.145,0.656,0,1
0.583,0.114,0.038,0,1
```

The setting of parameters for RCGP is shown in listing 4. The listing outlines the configuration parameters for the Recurrent Cartesian Genetic Programming (RCGP) model. The model uses a (1+4)-ES evolutionary strategy, meaning one parent generates four offspring each generation. The random seed is set to 10 to ensure reproducibility of results. The model takes two input values and has 30 computational nodes. It produces three output values, and each node can have up to two input connections. The probability of mutation is set to 5%, and the probability of connections being recurrent (feedback loops) is set to 10%. A supervised learning fitness function is used to evaluate the model, with a target fitness value of 0.1. This means the sum of the absolute differences between the predicted and target outputs should be 0.1 or lower. The selection scheme used is to select the fittest individuals, and offspring are produced by mutating a randomly chosen parent. The model's parameters are updated every 5 generations. The function set available for nodes includes addition (add), subtraction (sub), multiplication (mul), division (div), sigmoid (sig), and rectified linear unit (relu), with six functions in total. The model will run for a total of 8000 generations (numGens = 8000).

Listing 4: Configuration Parameters for RCGP Model.

| Parameters                        |                              |
|-----------------------------------|------------------------------|
| Evolutionary Strategy:            | (1+4)-ES                     |
| Random Seed:                      | 10                           |
| Inputs:                           | 2                            |
| Nodes:                            | 30                           |
| Outputs:                          | 3                            |
| Node Arity:                       | 2                            |
| Mutation rate:                    | 0.050000                     |
| Recurrent Connection Probability: | 0.100000                     |
| Fitness Function:                 | supervisedLearning           |
| Target Fitness:                   | 0.100000                     |
| Selection scheme:                 | selectFittest                |
| Reproduction scheme:              | mutateRandomParent           |
| Update frequency:                 | 5                            |
| Function Set:                     | add sub mul div sig relu (6) |

The implementation of the pairwise approach for RCGP symbolic classification was accomplished by applying the 12 configurations of cross-validation for each pair. The flow of the RCGP program, as illustrated in Figure 28, begins with the initialisation of variables. Pointers for various parameters, datasets, and results are declared, along with integer and double variables for settings such as the number of inputs, nodes, outputs, and various control parameters, including mutation rates and update frequencies. Once the variables are declared, specific values are assigned to each, configuring the operational settings of the RCGP system. Following the setup of these variables, the RCGP parameters are initialised. This phase involves setting up the genetic programming environment by defining the types of node functions that can be used in the chromosomes and setting system-wide parameters such as the update frequency, the random number seed, and the probability of recurrent connections within the genetic structures. The parameters are then displayed to verify the correct setup.

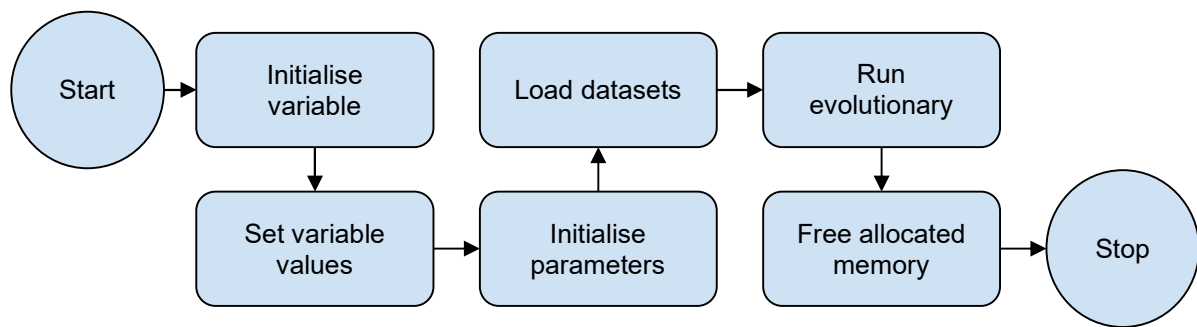


Figure 28: Flowchart of the RCGP Program.

The next step involves loading the datasets. These datasets, which consist of training, validation, and test data, are loaded from specified file paths. The files contain the data necessary to evolve and evaluate genetic programs. Once the datasets are prepared, the evolutionary strategy is executed. This core phase of the process involves running the RCGP algorithm, which iteratively evolves a population of chromosomes over several generations. The objective is to optimise the chromosomes to solve a specific problem, with fitness evaluations guiding the evolutionary process towards better solutions. After completing the evolutionary strategy, the system proceeds to free-allocate memory. This crucial cleanup step involves relocating all the memory used for parameters, datasets, and results to prevent memory leaks. It ensures that all resources are properly managed and released after their use. Finally, the process concludes, marking the end of the RCGP execution. Metrics used for assessing model performance (e.g., accuracy) are then reviewed to evaluate the effectiveness of the

model. An evolutionary strategy (ES) can be summarised in a series of steps typically followed in its implementation. Initially, an initial population of potential solutions is created, where each individual in the population is represented by a set of parameters (often real-valued) that define a possible solution to the problem at hand. The next step is the evaluation of the fitness of each individual in the population. The fitness function measures how well each individual solves the problem. Following this, selection occurs, where individuals from the current population are chosen based on their fitness. Individuals with higher fitness are more likely to be selected to pass their genes to the next generation. Recombination, or crossover, then takes place. This step involves combining pairs of selected individuals to create offspring by exchanging parts of the parents' parameter sets to produce new individuals. Mutation follows, wherein random changes are applied to the offspring's parameters. This introduces variability and allows the algorithm to explore new areas of the search space. In ES, mutation parameters can self-adapt, meaning they can change over time based on the success of previous mutations. Replacement is the next phase, replacing the old population with the new one. This can be done in various ways, such as replacing the entire population or only the least fit individuals. Following replacement, the process checks if the termination criteria are met. Typical criteria include reaching a maximum number of generations, achieving a satisfactory fitness level, or observing no significant improvement over several generations. Finally, the output is determined: the best individual in the final population is considered the solution to the problem. The flowchart of the Evolutionary Strategy (ES) process shown in Figure 29 illustrates how the algorithm iteratively improves the population of solutions until an optimal or near-optimal solution is found.

#### **3.8.1.1.6 Ensemble Methods**

Ensemble learning algorithms are a powerful machine learning technique that combines robust classifiers for enhanced performance. By running multiple classification algorithms simultaneously, this method yields better model performance and more accurate solutions than using a single classifier. The fundamental principle behind ensemble models is to merge a group of weak learners into a strong learner, thereby increasing the model's accuracy [266]. Errors in learning typically stem from noise, bias, and variance. Combining multiple classifiers helps reduce these factors, resulting in a more reliable classification than what can be achieved with a single classifier. The types of ensemble methods include Bagging, Boosting, and Stacking.

Five-fold cross-validation is used in this ensemble learning approach, and the metric to evaluate the performance is based on accuracy.

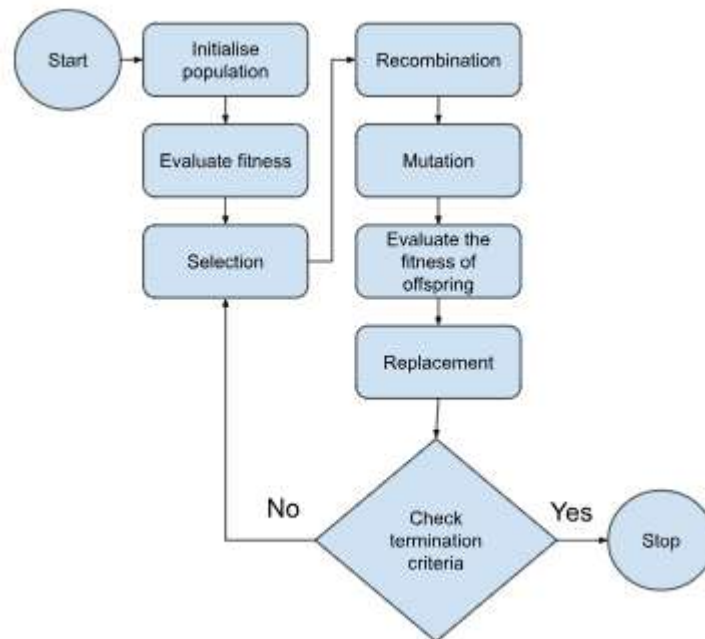


Figure 29: Flowchart of the Evolutionary Strategy (ES) Process.

### 3.8.1.1.6.1 Bagging

The Bagging method, short for Bootstrap Aggregating, is a machine learning technique that improves model accuracy by combining the predictions of multiple base models trained on different subsets of the training data. These subsets are created using a process called bootstrap sampling, where data is randomly sampled with replacement. The final prediction is typically made by averaging the predictions for regression tasks or taking a majority vote for classification tasks. Figure 30 illustrates the Bagging process in ensemble learning, showcasing the three main steps involved. Initially, the training data is divided into multiple subsets (Subset 1, Subset 2, Subset 3) using Bootstrap Sampling. This involves randomly selecting samples from the dataset with replacements, creating diverse subsets of the original data. Each subset is then used to train separate Random Forest Classifiers as base models (Base Model 1, Base Model 2, Base Model 3), with the diagram showing arrows connecting Subset 1, Subset 2, and Subset 3 to their respective base models. This indicates that these are instances of the same model trained on different subsets of data. Once all the base models are trained, their predictions are combined in the aggregation step to produce the final prediction (Output). The

aggregation block demonstrates this process, where the predictions from Base Model 1, Base Model 2, and Base Model 3 are merged. For classification tasks, this is typically done by taking a majority vote, ensuring a more reliable and accurate final output. This method enhances the model's performance by reducing variance and leveraging the strengths of multiple classifiers, as indicated in the final output block.

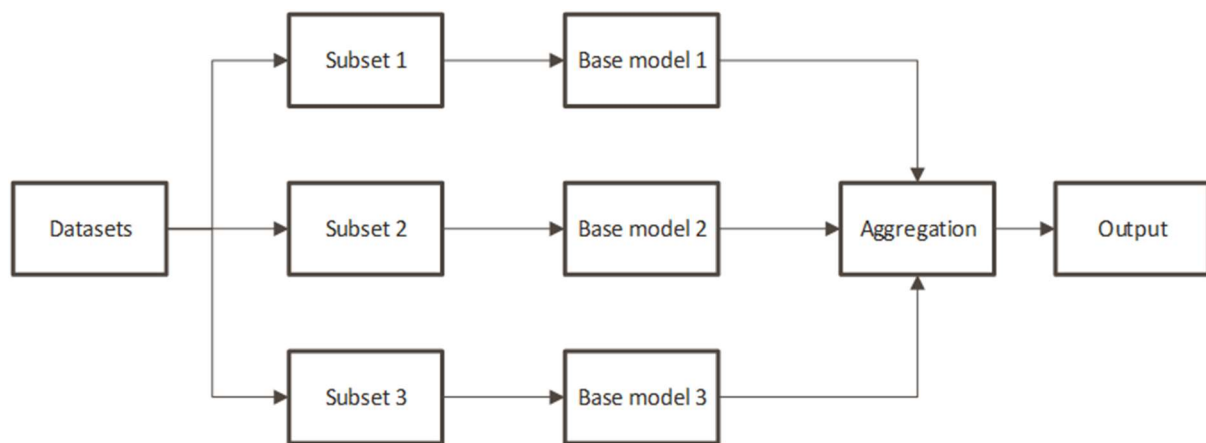


Figure 30: Bagging process in Ensemble Learning.

### 3.8.1.1.6.2 Boosting

Boosting is an ensemble learning method that improves model accuracy by training multiple models sequentially. Each new model specifically focuses on correcting the errors made by the previous models, giving more weight to misclassified instances. The final prediction is made by combining the predictions of all the models, often using a weighted vote for classification tasks. Figure 31 illustrates the Boosting process in ensemble learning, specifically employing the XGBoost algorithm, which aims to enhance performance through sequential model training. The process begins with an initial model trained on the entire dataset  $(x, y)$ . After this initial training, the model's predictions are assessed to identify instances that were misclassified  $(e_1)$ , which are then assigned increased weight, resulting in a weighted dataset  $(x, y, + e_1)$  used for training the next model.

The training sequence continues with Model 2, which operates on this newly weighted data to produce new predictions and further identify misclassifications  $(e_2)$ . The weight adjustment process is repeated, generating another weighted dataset  $(x, y, + e_2)$ . This cycle proceeds until a predetermined number of models (Model N) are trained or until the error rate ceases to improve. Each subsequent model increasingly focuses on the instances that were

previously misclassified, aiming to correct these errors and reduce bias. Ultimately, the trained models' predictions are aggregated to produce the final output. For classification tasks, this aggregation is typically executed through weighted voting, where the combined predictions from each model are tallied, and the majority vote determines the final prediction.

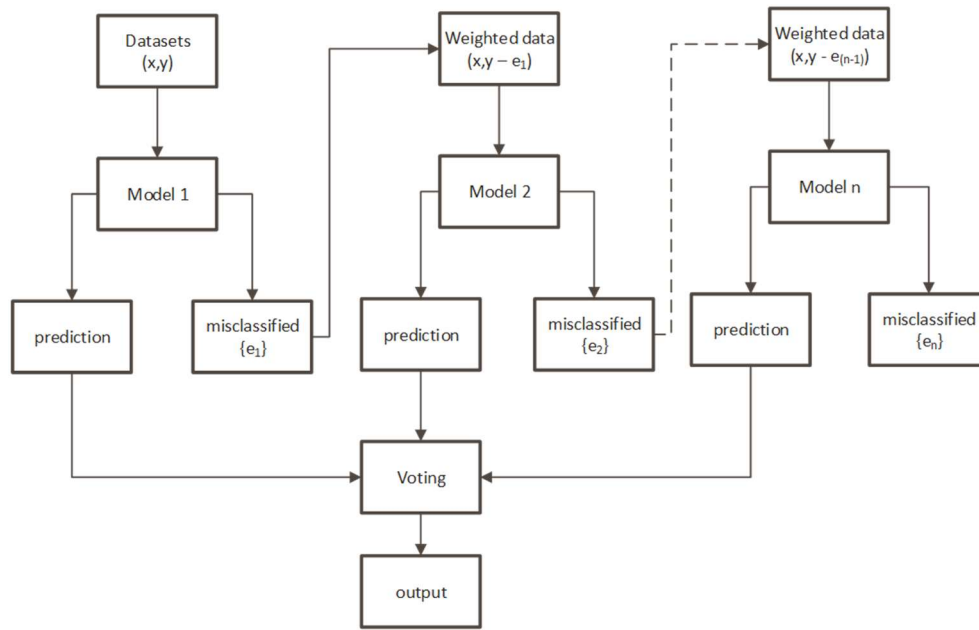


Figure 31: Boosting process in Ensemble Learning.

### 3.8.1.1.6.3 Stacking

Stacking ensemble is a machine learning technique that combines multiple models to improve predictive performance by using a meta-model to aggregate the predictions of base models. The base models are first trained on the original data, and their predictions are then used as input features for the meta-model, which makes the final prediction. Figure 32 illustrates the Stacking process in ensemble learning, also known as stacked generalisation. This method combines multiple classification models via a meta-learner to enhance predictive performance. In this example, the base models consist of Logistic Regression (Base Model 1), Decision Tree Classifier (Base Model 2), and Random Forest Classifier (Base Model 3), while the meta-model is a Logistic Regression model. The process begins with training the individual models on the entire dataset. Each base model (Base Model 1, Base Model 2, and Base Model 3) makes predictions on the validation set, and these predictions are collected and used as input features for the meta-model. The diagram shows the flow from the datasets to the base models and from the base models to the meta-model, with arrows indicating the transfer of predictions

as features. The individual models are trained on the complete training set. After training, the predictions of each base model on the validation set are used as new input features for the meta-learner. The meta-learner, in this case, a Logistic Regression model, is then trained on these predictions to make the final prediction. When new input data is provided, it is first passed through the individual base models, whose predictions are combined and fed into the meta-learner. The meta-learner uses these combined predictions to generate the final output. This method leverages the strengths of each base model and the logistic regression meta-learner to improve overall predictive performance. This sequential process of training, predicting, and combining enhances model accuracy, as depicted in the final output block.

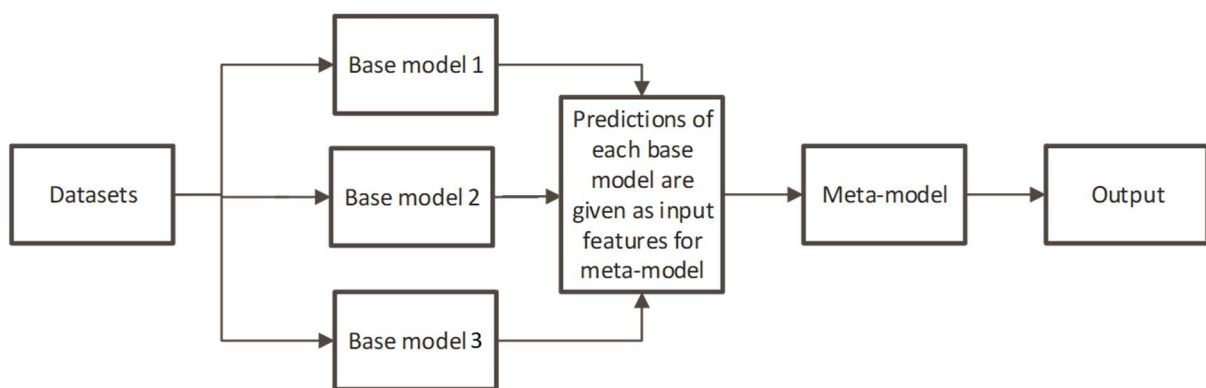


Figure 32: Stacking process in Ensemble Learning

#### 3.8.1.1.6.4 RCGP with Pairwise and Ensemble Approach

The implementation of RCGP models in conjunction with pairwise and ensemble approach is a novel method for improving classification accuracy. Figure 33 depicts an ensemble classification system utilising multiple RCGP models. It begins with the system taking three types of input data: Cell Growth, Cell Migration Speed, and Cell Angular Velocity. These inputs are fed into three separate RCGP models, each performing pairwise comparisons or analyses, labelled as RCGP (Pairwise 1), RCGP (Pairwise 2), and RCGP (Pairwise 3). These three RCGP models' outputs are combined in an ensemble module. This ensemble method aggregates the results from the different models to enhance the overall classification performance. Finally, the ensemble module produces a final classification output based on the integrated information from the three RCGP models. This figure shows a novel symbolic classification system that leverages the strengths of multiple RCGP models by integrating pairwise and ensemble learning to achieve a more accurate and robust classification outcome.



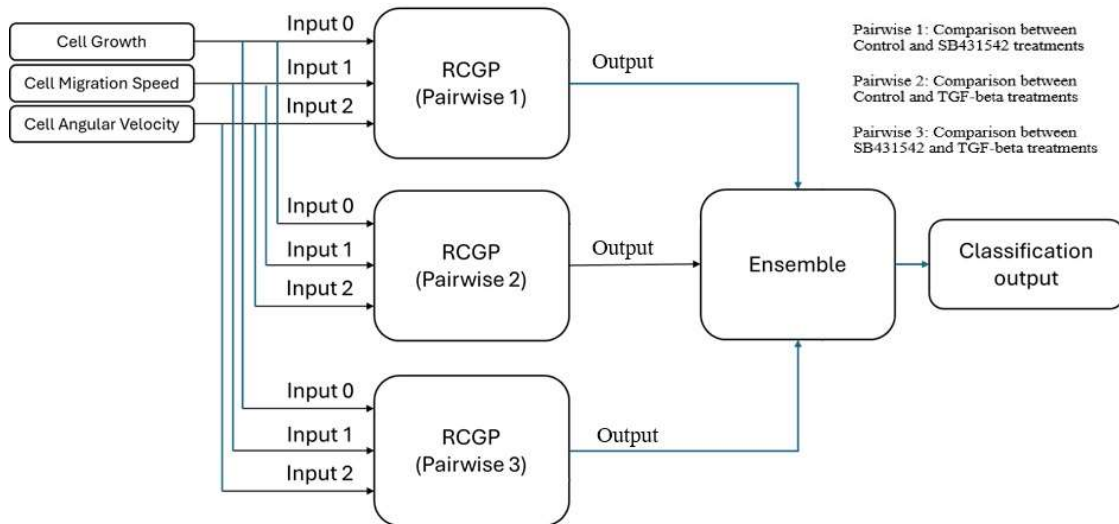


Figure 33: RCGP with pairwise and ensemble methods.

### 3.8.1.2 PySR

PySR [267] is an open-source library based on genetic programming techniques to perform symbolic regression and symbolic classification, a machine learning task that aims to discover interpretable symbolic expressions that model a given dataset.

#### 3.8.1.2.1 Core Principles and Genetic Programming Techniques

PySR was first released in 2020 [239]. It operates with multiple populations evolving independently but asynchronously, allowing different populations to progress at their own rates and interact periodically. At its core, PySR's process is based on a classical evolutionary algorithm using tournament selection [268] for individual selection, involving several mutations and crossovers to generate new individuals. In the tournament selection process, a subset of individuals is randomly selected from the population, their fitness is evaluated, and the fittest individual is chosen as the winner with a certain probability. If the fittest is not selected, the process repeats until an individual is chosen. The selected individual then undergoes a mutation, creating a new individual that replaces a population member. One significant modification in PySR is age-regularised evolution [269], where the eldest individual (determined by the Unix creation time) is replaced rather than the least fit one. This approach prevents premature convergence and maintains diversity within the population, avoiding a

condition where individuals become too similar and focus narrowly within the search space. PySR's algorithm is computationally efficient and supports parallelisation, with different groups of individuals evolving in parallel and asynchronously migrating between groups to enhance diversity and exploration capabilities. The evolutionary algorithm in PySR involves initialising a population, applying tournament selection to subsets of individuals, selecting and mutating the fittest individuals, and replacing the eldest members to maintain diversity.

Three main modifications enhance PySR's classic evolutionary algorithm. The first modification is the application of simulated annealing [270] to the mutation acceptance process, introducing a temperature parameter that affects the probability of mutation rejection based on fitness changes. This allows the algorithm to alternate between phases of high and low temperatures, balancing the exploration and refinement of individuals. Higher temperatures increase diversity, while lower temperatures optimise the fittest individuals, significantly speeding up the search process. The second modification is embedding the genetic algorithm within an evolve-simplify-optimize loop. The "evolve" phase involves applying tournament selection-based evolution for a set number of mutations. The "simplify" phase reduces equations to their simplest equivalent forms using algebraic equivalencies, and the "optimize" phase uses classical optimisation techniques, such as the BFGS algorithm [271], to fine-tune numerical constants within the equations. This loop allows the exploration of necessary intermediate states and significantly enhances the discovery of useful equations, especially those containing real constants, which is essential for scientific applications.

The third modification involves using an adaptive parsimony metric. The number of nodes defines complexity in PySR in an expression tree, and this definition is user-configurable. Traditionally, complexity is penalised with a constant parsimony factor, but PySR adaptively adjusts the complexity penalty based on the frequency and recency of expressions with similar complexity in the population. This adaptive penalty encourages exploring a balanced range of simple and complex expressions, preventing premature convergence to suboptimal functional forms and improving the chances of discovering more accurate and interpretable models. Listing 5 shows the pseudocode of the PySR outer loop program, which involves initialising multiple populations with random expressions, evolving these populations through a process of selection, mutation, simplification, and optimisation, and periodically migrating the best expressions between populations. The algorithm operates in parallel over multiple iterations to find the best mathematical expressions for a given dataset. The main steps

include evolving expressions, simplifying and optimising them, updating the best expressions, and migrating these best expressions across populations to enhance diversity and exploration.

Listing 5: PySR [267].

```
input : X , the dataset to find expressions for
output : the best expressions at each complexity
param : np , the number of populations ( =40)
param : L, the number of expressions in each population ( =1000)
param :  $\alpha H$  , replacement fraction using expressions from H (=0.05)
param :  $\alpha M_i$  , replacement fraction using expressions from  $U_i M_i$  (=0.05)

1 function pysr( X )
2   for i in range( np ) // [code]
3     create P i containing L random expressions of complexity 3
4     // e.g., (3.2 + x1) has size 3
5     create empty set M i // will store best expressions seen in P i
6   end
7   create empty set H // will store best expressions overall
8   for n in range( niter ) // [code]
9     // the following loop is parallelized over workers:
10    for i in range( np )
11      // evolve-simplify-optimize:
12      P i = evolve( P i , X )
13      for E in P i
14        simplify E
15        optimize constants in E
16        store updated E in P i
17      end
18      M i  $\leftarrow$  most accurate expression in P i at each complexity
19      // (In actuality, M i is updated throughout evolve )
20      H  $\leftarrow$  most accurate expression in M i  $\cup$  H at each complexity
21    end
22    // migration:
23    for E in P i
24      if rand() <  $\alpha H$ 
25        replace E in P i with a random expression from H
26      end
27      if rand() <  $\alpha M$ 
28        replace E in P i with a random expression from  $U_{j \neq i} M_j$ 
29      end
30    end
31  end
32  return H
33 end
```

Listing 6 illustrates the evolve-simplify-optimise loop in the PySR algorithm, detailing how a set of expressions (P) is evolved using a dataset (X). The process involves multiple iterations where either mutation or crossover operations are applied to the expressions. For mutations, an expression is selected through a tournament and then mutated based on an annealing temperature that decreases over time. This mutated expression replaces the oldest expression in the set. For crossovers, two expressions are selected, combined, and the resulting new expressions replace the two oldest ones, provided they meet specific constraints. The process repeats for a set number of iterations to evolve the set of expressions towards better solutions.

Listing 6: Evolution [267].

```

input : P, a set of expressions
input : X as in listing 1
output : P, the evolved set of expressions
param : nc, the number of mutations per evolve() call (=300000)
param : pcross, the probability of crossover (=0.01)

1 function evolve(P, X )
2   for k in range(nc) // [code]
3     if rand() > pcross // [code]
4       // mutation
5       E ← tournament(P, X )
6       T ← 1 - k / nc // annealing temperature
7       E* ← mutate(E, T )
8       replace oldest expression in P with E*
9     else
10      do
11        // crossover
12        E1 ← tournament(P, X )
13        E2 ← tournament(P, X )
14        E1*, E2* ← crossover(E1, E2 )
15        replace oldest two expressions in P with E1* and E2*
16        until satisfies_constraints(E1*) and satisfies_constraints(E2*)
17      end
18    end
19  return P
20 end

```

Listing 7 shows the set of mutations supplemented with simulated annealing. The mutation function modifies an expression (E) based on a type of mutation selected randomly but weighted by predefined probabilities. The mutation types include mutating constants, replacing operators, appending or inserting nodes, deleting subtrees, simplifying the expression, or

generating a completely new tree. After applying the mutation, the new expression ( $E^*$ ) is evaluated for constraints and compared to the original for complexity and accuracy. The acceptance of the mutated expression is determined by a probability that incorporates the annealing temperature and a parsimony factor, which balances the trade-off between exploration and exploitation. If the mutated expression is accepted, it replaces the original; otherwise, the original expression is retained.

Listing 7: Mutation [267].

Algorithm 3: Mutations.

```

input : E, an expression
input : T, the annealing temperature  $\in [0, 1]$ 
output : mutated version of E
param :  $m_i$ , for  $i = 1, \dots, 8$ , the probability weight of mutation type  $i$ 
param :  $f$ , the constant perturbation scale (=1)
param :  $c$ , the minimum perturbation (=0.01)
param :  $\alpha$ , the temperature scale (=0.1)

1 function mutate(E, T) // [code]
2   adjust weights  $w_1, \dots, w_8$  based on constraints
3    $i \leftarrow$  random choice of 1, ..., 8 weighted by  $w_1, \dots, w_8$ 
4    $E^* \leftarrow$  copy(E)
5   switch  $i$ 
6     case 1 // mutate constant
7        $a \leftarrow (1 + f \times T + c)^{(2 \times \text{rand}())} - 1$ 
8       if  $\text{rand}() < 0.5$ 
9          $a \leftarrow -a$ 
10      end
11      multiply random constant in  $E^*$  by  $a$ 
12     case 2 // mutate operator
13       randomly replace an operator in  $E^*$  with an operator of the same degree
14     case 3 // append/prepend node
15       insert random node to root or leaf of  $E^*$ 
16     case 4 // insert node
17       insert a random node inside  $E^*$ 
18     case 5 // delete subtree
19       replace a node and its children from  $E^*$  with a constant or variable
20     case 6 // simplify tree
21       simplify  $E^*$ 
22     case 7 // new tree entirely
23        $E^* \leftarrow$  random expression
24     case 8 // no mutation
25       do nothing
26   end

```

```

27  until satisfies_constraints(E*)
28  C, C* ← complexity of expressions E, E*, respectively
29  L, L* ← accuracy of expressions E, E*, respectively
30   $\delta_{anneal} \leftarrow \exp((L - L^*) / \alpha T)$ 
31  C* ← complexity of expressions E
32   $\theta_{parsimony} \leftarrow \text{frecency}[C]$ 
33  if rand() <  $\delta_{anneal} \cdot \theta_{parsimony}$ 
34    return E*
35  else
36    return E
37  end
38 end

```

The last process outline in listing 8 describes the tournament selection strategy. The tournament function selects a single expression from a population by randomly choosing a subset and iteratively finding the fittest expression within that subset. The probability of selecting the fittest expression is determined by a predefined parameter. If the fittest expression is not selected, it is removed from the subset, and the process repeats until one expression remains. The get fittest function calculates the fittest expression based on a combination of complexity and accuracy, adjusted for parsimony, to ensure the most suitable candidate is chosen. This selected expression is then returned as the winner of the tournament.

Listing 8: Tournament selection [267].

Algorithm 4: Tournament selection.

```

input : P, a population of expressions
input : X as in algorithm 1
output : a single expression (the winner of the tournament)
param : ns, the tournament size (=12)
param : ptournament, the probability of selecting the fittest individual (=0.9)

1  function tournament(P, X) // [code]
2    Q ← a random subset of size ns of P
3    while length(Q) > 1
4      E ← get_fittest(Q)
5      if rand() < ptournament
6        break
7      end
8      remove E from Q
9    end
10   return E
11 end

```

```

12 function get_fittest(P) // [code]
13   ℓbest ← ∞
14   for E in P
15     C ← complexity of E
16     ℓ ← accuracy of E
17     // include adaptive parsimony:
18     ℓ ← ℓ × exp(frequency[C])
19     if ℓ < ℓbest
20       ℓbest ← ℓ
21       E* ← E
22   end
23 end
24 return E*
25 end

```

### 3.8.1.2.2 PySR Customisation feature

PySR supports additional features such as custom loss functions, which can be specified as a string or a function object. These functions take scalar predictions and targets, optionally include a scalar weight, and return a real number greater than zero. The per-point losses are then summed over the dataset, allowing users to define custom regression objectives, custom likelihoods (such as log-likelihoods for individual data points), classification-based losses, and implicit objectives tailored to specific needs. Another benefit option of PySR is its ability to use custom operators by allowing users to define their own unary or binary functions, which can be integrated into the symbolic regression process. These custom operators can be any function and PySR treats them the same as built-in operators, making no distinction apart from simplification strategies. Additionally, PySR also supports various hard constraints that can be enforced to discover expressions in order to control their complexity and structure. These constraints are checked at every mutation step, and any mutation that violates a constraint is rejected. Users can specify constraints such as the maximum size of an expression, which limits the number of instances of operators, variables, and constants to bound the search space. The maximum depth of an expression can also be controlled to limit how deeply nested the resultant expressions are. Moreover, users can set the maximum size of a subexpression within a specific operator's arguments, reducing the complexity of discovered expressions. Another constraint allows for limiting the number of nests of a particular operator combination, such as allowing

$\sin(\cos(x))$  but not  $\sin(\cos(\cos(x)))$  or  $\sin(\sin(x))$ . These constraints help ensure that the generated expressions are both interpretable and relevant to the specific needs of the domain.

### 3.8.1.2.3 Backend Architecture and Performance

PySR leverages the high-performance SymbolicRegression.jl library in Julia [272] as its backend, enabling fast evaluation of expressions and distributed parallel computing. The search algorithm underlying PySR, described in pseudocode in listings 3 to 6, is written in pure Julia, which combines a high-level interface with performance comparable to languages such as C++ and Rust [273]. The key advantage of using Julia is its just-in-time (JIT) compilation, allowing PySR to perform optimisations not possible with statically compiled libraries. For instance, it can compile user-defined operators and losses into the core functions, and fuse operators into single compiled units. This automatic fusion of operators, even for user-defined ones, significantly speeds up the expression evaluation, which is the main bottleneck in PySR. Despite the backend being in Julia, PySR offers a simple Python frontend API, adopting the popular style of the scikit-learn machine learning library [274]. This makes symbolic regression and classification accessible to Python users while benefiting from Julia's speed and efficiency. PySR is parallelised by dispatching populations of expressions to workers (either threads for single-node or processes for multi-node setups), where they evolve independently over many iterations. After a batch of iterations, populations are returned to the head worker, which records the best expressions in a global "hall of fame," tracks current expressions for potential migration, and randomly migrates expressions from other populations and the hall of fame. This asynchronous parallelisation allows different populations to evolve at different speeds without slowing down the process. The inclusion of a global, permanent hall of fame, as opposed to migrating only current population members, can significantly speed up the search by reintroducing top expressions.

### 3.8.1.2.4 Employing a combination of symbolic regression and classification

The novel approach used in this study involves integrating symbolic regression and classification. Figure 34 describes modelling cell population behaviour through a combination of symbolic regression and classification using the PySR framework. The workflow begins



with each well receiving three input features: Time, Average Cell Migration Speed, and Average Cell Angular Velocity. For each well, labelled Well 0, Well 1, and so on up to Well n, symbolic regression models are employed to model the correlation between these input features and cell growth. These models are specific to each well, taking the provided inputs and outputting a prediction for cell growth. All wells' predicted cell growth values are then fed into a symbolic classification model. This classification model covers the outputs of the individual symbolic regression models from each well and uses this aggregated data to predict multiple outputs labelled  $y_0$ ,  $y_1$ , and  $y_2$ , representing different treatment conditions. Specifically, an output of  $y_0 = 1$ ,  $y_1 = 0$ , and  $y_2 = 0$  corresponds to the Control treatment,  $y_0 = 0$ ,  $y_1 = 1$ , and  $y_2 = 0$  corresponds to the TGF-Beta treatment, and  $y_0 = 0$ ,  $y_1 = 0$ , and  $y_2 = 1$  corresponds to the SB431542 treatment. The figure visually represents the data flow from the input features through the symbolic regression models for each well, culminating in a combined symbolic classification model that outputs the final predictions for treatment conditions. This approach leverages the strengths of both symbolic regressions for detailed prediction of cell growth and symbolic classification for integrating these predictions to determine the specific treatment associated with the observed cell behaviour.

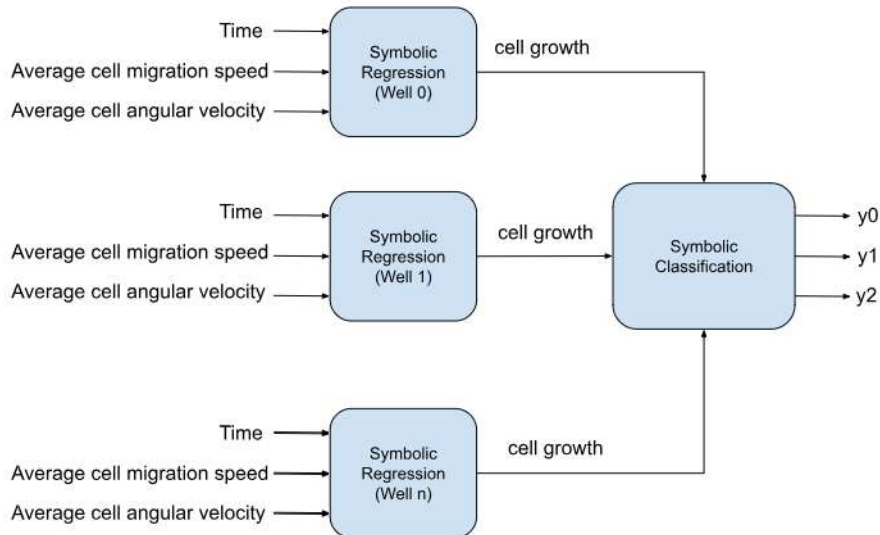


Figure 34: Block diagram of Symbolic Regression and Classification in the PySR Framework.

### 3.8.1.2.5 Dataset Preparation and Splitting

The data preparation involved features such as time, average cell migration speed, angular velocity, cell growth, and multi-class labels. In terms of splitting the data into training,

validation, and test sets, the `train_test_split` function from Python's `sklearn.model_selection` was used. This study split the data into a training set (60% of the total data) and a remaining set. The remaining data was then equally divided into validation and test sets. Thus, the final distribution of data is 60% for training, 20% for validation, and 20% for testing. Listing 9 illustrates the pseudocode to perform the data-splitting task in Python. The arrangement of the data will then be as shown in Figure 34.

Listing 9: Pseudocode for Splitting Dataset into Training, Validation, and Test Sets.

```
BEGIN
// Import the train_test_split function from the sklearn.model_selection library
Import train_test_split from sklearn.model_selection

// Assume X is the array of input features and y is the array of corresponding labels

// Step 1: Split the full dataset into a training dataset and a remaining dataset
// Define the proportion of the dataset to include in the train split
Set train_size to 0.7

// Perform the initial split
// X_train and y_train will contain 70% of the original data
// X_rem and y_rem will contain the remaining 30% of the original data
X_train, X_rem, y_train, y_rem = train_test_split(X, y, train_size)

// Step 2: Split the remaining data into validation and test datasets
// Define the proportion of the remaining dataset to include in the validation split
Set remaining_train_size to 0.5

// Perform the secondary split
// X_val and y_val will contain 50% of the remaining data (15% of the original dataset)
// X_test and y_test will contain the other 50% of the remaining data (15% of the original dataset)
X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, remaining_train_size)

// Output the datasets for further processing or analysis
Output X_train, y_train, X_val, y_val, X_test, y_test
END
```

### 3.8.1.2.6 Parameters Value for Symbolic Regression

Listing 10 shows the settings for the PySRRegressor model for symbolic regression are configured as follows: The number of processors to use is set to 12, enabling the model to perform parallel computations and thereby speeding up the process significantly by distributing the workload across multiple processors. The number of populations in the evolutionary algorithm is set to 18, which helps enhance the genetic diversity of the solutions. Having multiple populations increases the chances of finding a globally optimal solution by allowing

different sub-populations to explore different parts of the solution space simultaneously. The size of each population is set to 1000, meaning each population will contain 1000 individuals. A larger population size can improve the robustness of the search but may also increase the computational time required for each iteration. The number of cycles per iteration is set to 500, controlling how many evolutionary cycles are performed in each iteration. More cycles per iteration allow the model to refine solutions incrementally and improve the quality of the solutions found in each iteration. The total number of iterations is set to 200, indicating the total number of iterations the algorithm will run. More iterations allow the model to explore a larger portion of the solution space and increase the likelihood of finding better solutions. The binary operators used include addition, subtraction, and multiplication. These operators are the fundamental building blocks used to construct the mathematical expressions in the symbolic regression model. The precision of the calculations is set to 64-bit, meaning the calculations will be performed with double precision. Higher precision can lead to more accurate results but may also increase computational demands. The maximum size of the expressions is limited to 40, which helps manage the complexity and computational load, ensuring the expressions remain interpretable and manageable. The maximum depth of the expressions is limited to 6, preventing the generated expressions from becoming overly complex and difficult to interpret.

Listing 10: PySR Configuration Parameters for symbolic regression.

```
model = PySRRegressor(  
    # Number of processors to use  
    procs=12,  
  
    # Number of populations in the evolutionary algorithm  
    populations=18,  
  
    # Size of each population  
    population_size=1000,  
  
    # Number of cycles per iteration  
    ncycles_per_iteration=500,  
  
    # Total number of iterations  
    niterations=200,  
  
    # List of binary operators to use  
    binary_operators=["+", "-", "*"],
```

```
# Precision of the calculations
precision=64,

# Maximum size of the expressions
maxsize=40,

# Maximum depth of the expressions
maxdepth=6,
)
```

### 3.8.1.2.7 Parameters Value for Symbolic Classification

The settings for the PySRRegressor model for symbolic classification are configured, as shown in listing 11. The number of processors is set to 12, which allows the model to perform parallel computations. This significantly speeds up the process by distributing the workload across multiple processors. The number of separate populations in the evolutionary algorithm is set to 18. This helps enhance the genetic diversity of the solutions and improves the chances of finding a globally optimal solution by allowing different sub-populations to explore different parts of the solution space simultaneously. Each population consists of 1000 individuals. A larger population size improves the robustness of the search, although it may also increase the computational time required for each iteration. The number of cycles per iteration is set to 500, which controls how many evolutionary cycles are performed in each iteration. More cycles per iteration allow the model to refine solutions incrementally and potentially improve the quality of the solutions found in each iteration. The total number of iterations the algorithm will run is set to 200, enabling the model to explore a larger portion of the solution space and potentially find better solutions. The binary operators used to build expressions include addition (+), subtraction (-), and multiplication (\*). These operators are the fundamental building blocks of the mathematical expressions generated by the symbolic classification model. Additionally, a logistic function is used as a unary operator, defined as  $\text{logistic}(x) = 1 / (1 + \exp(-x))$ . This function is also mapped in the `extra_sympy_mappings` to ensure it can be used correctly during the symbolic classification process.

The precision of the calculations is set to 64-bit, meaning the calculations will be performed with double precision. This higher precision can lead to more accurate results but may also increase computational demands. The maximum size of the generated expressions is limited to 40, which helps manage the complexity and computational load, ensuring the

expressions remain interpretable and manageable. The maximum depth of the expressions is limited to 6, preventing the generated expressions from becoming overly complex and difficult to interpret. These settings are designed to optimise the symbolic classification process, allowing the PySRRegressor model to effectively handle and optimise complex relationships within the data while ensuring the generated expressions are interpretable and computationally feasible.

Listing 11: PySR Configuration Parameters for symbolic classification.

```
model = PySRRegressor(  
    # Number of processors to use  
    procs=12,  
    # Number of populations in the evolutionary algorithm  
    populations=18,  
    # Size of each population  
    population_size=1000,  
    # Number of cycles per iteration  
    ncycles_per_iteration=500,  
    # Total number of iterations  
    niterations=200,  
    # List of binary operators to use  
    binary_operators=["+", "-", "*"],  
    # List of unary operators to use  
    unary_operators=["logistic(x) = 1 / (1 + exp(-x))"],  
    # Custom mapping for the sigmoid function  
    extra_sympy_mappings={"logistic": lambda x: 1 / (1 + exp(-x))},  
    # Precision of the calculations  
    precision=64,  
    # Maximum size of the expressions  
    maxsize=40, # Maximum size of the expressions  
    # Maximum depth of the expressions  
    maxdepth=6,  
)
```

### 3.8.1.2.8 Cross-Validation and Model Evaluation

A comprehensive approach was adopted to implement cross-validation and model training effectively for symbolic regression and classification. The training set was divided into five subsets or folds to ensure that each fold served as a validation set once, while the remaining four were used for training. This method ensured that every dataset segment was utilised for validation precisely once, and the folds were representative of the entire dataset, containing a mix of all data features, including time, average cell migration speed, angular velocity, cell growth, and multi-class labels. For symbolic regression, the input features are time, average cell migration speed, and angular velocity, and the target value is cell growth. The process began with setting up cross-validation by splitting the training set into five folds and initialising lists to store performance metrics for each fold. For each fold, the data was split into training and validation sets. The symbolic regression model was then trained on the training set using the specified input features and target value. The model was validated on the remaining fold and mean squared error (MSE) and R-squared performance metrics were recorded. This process was repeated for each of the five folds to maintain consistency across the dataset. After completing the 5-fold cross-validation, the average performance metrics were calculated to understand the stability and reliability of the regression model.

Hyperparameter tuning was performed using grid search during cross-validation to find the best model parameters. A hyperparameter grid for the symbolic regression model was defined, and for each combination of hyperparameters in the grid, the training and validation steps were repeated, and performance metrics were recorded. The hyperparameter combination that yielded the best performance metrics was selected. The final symbolic regression model was then trained on the entire training dataset using the selected optimal hyperparameters, and the model was evaluated on the previously held-out test set to obtain an unbiased estimate of its performance. Pseudocode for 5-Fold Cross-Validation and Grid Search for Symbolic Regression is shown in Listing 12.

The input feature for symbolic classification is the actual cell growth values, and the target is the multi-class labels. The process also began with setting up cross-validation by splitting the training set into five folds and initialising lists to store performance metrics for each fold. For each fold, the data was split into training and validation sets. The actual cell growth values were used as input features for symbolic classification. The symbolic classification model was trained on the training set using these input features and the multi-

class labels. The model was validated on the remaining fold, and the accuracy of the performance metrics was recorded. This process was repeated for each of the five folds to ensure consistency across the dataset. After completing the 5-fold cross-validation, the average performance metrics were calculated to understand the stability and reliability of the classification model. Hyperparameter tuning was also performed for symbolic classification using grid search during cross-validation to find the best model parameters. A hyperparameter grid for the symbolic classification model was defined, and for each combination of hyperparameters in the grid, the training and validation steps were repeated, and performance metrics were recorded. The hyperparameter combination that yielded the best performance metrics was selected. The final symbolic classification model was then trained on the entire training dataset using the selected optimal hyperparameters, and the model was evaluated on the previously held-out test set to obtain an unbiased estimate of its performance. Listing 13 illustrates the pseudocode responsible for 5-Fold Cross-Validation and Grid Search in the context of symbolic classification.

Listing 12: Pseudocode for 5-Fold Cross-Validation and Grid Search for Symbolic Regression.

```
BEGIN
// Import necessary functions
Import train_test_split from sklearn.model_selection
Import KFold from sklearn.model_selection
Import GridSearchCV from sklearn.model_selection
Import symbolic_regression_model_library

// Assume X is the array of input features and y is the array of corresponding labels

// Step 1: Split the full dataset into a training dataset and a test dataset
// Define the proportion of the dataset to include in the train split
Set train_size to 0.7

// Perform the initial split
// X_train and y_train will contain 70% of the original data
// X_test and y_test will contain the remaining 30% of the original data
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size)

// Step 2: Initialize the 5-fold cross-validation
kf = KFold(n_splits=5)

// Step 3: Define the hyperparameter grid for tuning
```

```

hyperparameter_grid = {
    'param1': [value1, value2, value3],
    'param2': [value1, value2, value3],
    ...
}

// Step 4: Initialize variables to store the best parameters and best score
best_params = None
best_score = float('inf')

// Step 5: Perform the grid search with 5-fold cross-validation
FOR each combination of hyperparameters in hyperparameter_grid:
    current_params = combination of hyperparameters
    model = your_symbolic_regression_model_library.Model(**current_params)

    scores = []

    FOR train_index, val_index IN kf.split(X_train):
        X_fold_train, X_fold_val = X_train[train_index], X_train[val_index]
        y_fold_train, y_fold_val = y_train[train_index], y_train[val_index]

        // Train the model
        model.fit(X_fold_train, y_fold_train)

        // Validate the model
        predictions = model.predict(X_fold_val)
        score = calculate_score(y_fold_val, predictions) // Use MSE, R-squared, etc.
        scores.append(score)

    // Calculate the average score across all folds
    average_score = mean(scores)

    // Update the best parameters if the current score is better
    IF average_score < best_score:
        best_score = average_score
        best_params = current_params

// Step 6: Train the final model on the entire training dataset with the best parameters
final_model = your_symbolic_regression_model_library.Model(**best_params)
final_model.fit(X_train, y_train)

// Step 7: Evaluate the final model on the test dataset
final_predictions = final_model.predict(X_test)
final_score = calculate_score(y_test, final_predictions)

// Output the best parameters and the final performance metrics
Output best_params, final_score
END

```



Listing 13: Pseudocode for 5-Fold Cross-Validation and Grid Search for Symbolic Classification.

```
BEGIN
// Import necessary functions
Import train_test_split from sklearn.model_selection
Import KFold from sklearn.model_selection
Import GridSearchCV from sklearn.model_selection
Import your_symbolic_classification_model_library

// Assume X is the array of actual cell growth values (features) and y is the array of
corresponding multi-class labels

// Step 1: Split the full dataset into a training dataset and a test dataset
// Define the proportion of the dataset to include in the train split
Set train_size to 0.7

// Perform the initial split
// X_train and y_train will contain 70% of the original data
// X_test and y_test will contain the remaining 30% of the original data
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size)

// Step 2: Initialize the 5-fold cross-validation
kf = KFold(n_splits=5)

// Step 3: Define the hyperparameter grid for tuning
hyperparameter_grid = {
    'param1': [value1, value2, value3],
    'param2': [value1, value2, value3],
    ...
}

// Step 4: Initialize variables to store the best parameters and best score
best_params = None
best_score = float('-inf')

// Step 5: Perform the grid search with 5-fold cross-validation
FOR each combination of hyperparameters in hyperparameter_grid:
    current_params = combination of hyperparameters
    model = your_symbolic_classification_model_library.Model(**current_params)

    scores = []

    FOR train_index, val_index IN kf.split(X_train):
        X_fold_train, X_fold_val = X_train[train_index], X_train[val_index]
        y_fold_train, y_fold_val = y_train[train_index], y_train[val_index]

        // Train the model
        model.fit(X_fold_train, y_fold_train)
```

```

// Validate the model
predictions = model.predict(X_fold_val)
score = calculate_score(y_fold_val, predictions) // Use accuracy
scores.append(score)

// Calculate the average score across all folds
average_score = mean(scores)

// Update the best parameters if the current score is better
IF average_score > best_score:
    best_score = average_score
    best_params = current_params

// Step 6: Train the final model on the entire training dataset with the best parameters
final_model = your_symbolic_classification_model_library.Model(**best_params)
final_model.fit(X_train, y_train)

// Step 7: Evaluate the final model on the test dataset
final_predictions = final_model.predict(X_test)
final_score = calculate_score(y_test, final_predictions)

// Output the best parameters and the final performance metrics
Output best_params, final_score
END

```

### 3.8.1.2.9 Advantages

Symbolic regression and classification provide significant benefits by offering flexibility, clarity, and the capacity to reveal complex data relationships. Symbolic regression identifies both linear and non-linear relationships to make accurate predictions; specifically, this study finds the correlation between feature time, cell migration speed, and cell angular velocity to predict cell growth. Symbolic classification establishes clear, understandable rules based on target outputs, where the classification input is cell growth and the target output is multi-class labels. Using these techniques together allows practitioners to develop effective models that offer deep insights into data processes, supporting improved decision-making and understanding.

### 3.9 Conclusion

In conclusion, the methodology of this study was designed to outline the sequential and interdependent processes crucial for comprehensive cell analysis. It commenced with cell culture preparation across twelve plates, each containing four replicates, across three different treatments: 0.1% DMSO as the control, 0.1% DMSO with 4 ng/mL TGF $\beta$ , and 0.1% DMSO with 10  $\mu$ M SB431542 as a TGF $\beta$  inhibitor. This setup was crucial for creating a controlled environment that facilitated detailed observations of cell growth and responses under varying conditions, addressing key research hypotheses about cellular behaviour. Time-lapse microscopy, employing a phase-contrast x10 objective, captured detailed images over 48 hours, with adjustments for optimal visibility. The BaSiC plugin was used to correct uneven illumination, followed by image normalisation using a quantile-based ImageJ plugin, ensuring uniform data quality and reliability, which is vital for subsequent analyses. Cell segmentation involved Ilastik, which transformed pre-processed images into image probabilities via pixel classification, and ImageJ, which converted these probabilities into binary images for precise object detection. Cell tracking was performed using CellProfiler, utilising the Follow Neighbour method to identify cells and extract detailed quantitative data. These methods were chosen for their accuracy and reliability, crucial for analysing cell populations. Cell characterisation utilised RCGP and PySR for modelling, chosen for their interpretability and capability to simulate complex cellular behaviours and interactions. These tools were instrumental in uncovering novel relationships between cellular features, providing significant insights that align with the study's hypotheses regarding cell dynamics under different treatments. These methodologies have limitations, despite their strengths. Potential biases in image-based analysis and the need for broader data sets for model validation suggest areas for future methodological improvement. Future research could explore integrating multi-modal imaging and enhanced computational techniques to deepen our understanding and expand its applicability in biological research. This study's results deepen our understanding of cellular dynamics and highlight how these techniques can impact broader biological and medical research areas.

## CHAPTER 4

### RESULTS AND ANALYSIS

#### 4.1 Introduction

The chapter discusses results and analysis, beginning with the initial phase of image preprocessing outlined in Section 4.2. The process involves correcting uneven illumination and normalising images to enhance quality for accurate analysis. Following preprocessing, Sections 4.3 and 4.4 detail the segmentation and tracking of cells. These steps are essential for monitoring cell behaviour over time and under different treatment conditions. Sections 4.5 through 4.8 focus on analysing cell growth curves, growth rates, migration speed, and angular velocity. These analyses provide comprehensive insights into cellular dynamics. The latter part of the chapter, starting from Section 4.9, focuses on applying machine learning techniques. Included is Recurrent Cartesian Genetic Programming (RCGP) for pairwise and ensemble classifications, detailed in Sections 4.9.1, which analyse the effectiveness of various models in distinguishing between control and treated samples. Section 4.9.2 introduces PySR for regression and classification tasks across multiple wells, offering a nuanced approach to modelling cell behaviour based on datasets. Section 4.9.3 and its subsection 4.9.3.1 compare the efficacy of different machine learning algorithms in classifying cell data, emphasising the versatility and robustness of the approaches used. The chapter concludes with a reflection on the findings and their implications for understanding cellular responses under varying conditions, solidifying the role of advanced computational techniques in biological research.

#### 4.2 Image preprocessing

##### 4.2.1 Image Uneven illumination correction

The results of the Image Uneven Illumination Correction illustrate the variation in grey values across pixels using different methods in image processing. The coefficient of variation (CV) is a statistical measure that quantifies the relative variability of grey values across pixels. In the context of uneven illumination correction, a lower CV is desirable because it indicates less variation in grey values, suggesting more uniform and consistent image quality after

correction, while a higher CV suggests persistent variation and incomplete correction. In the image time-lapse study, images were captured from 12 wells, each requiring uneven illumination correction to ensure accurate analysis. While the correction process was applied to all wells in practice, this thesis shows the results only for Wells 0, 5, and 11. Well 0 represents the dataset under control conditions, Well 5 corresponds to the dataset treated with TGF-B, and Well 11 is associated with the dataset treated with SB431542. The selected wells are positioned diagonally across the 12 wells, providing a representative sample of different experimental conditions and spatial positions within the plate. Although the results for the other wells are not shown here, it is important to note that the correction method was applied across all 12 wells. The decision to highlight only Wells 0, 5, and 11 was made to provide clear examples of the algorithm’s effectiveness. In actual implementation, however, the correction method is intended to be universally applied to all wells, ensuring uniformity across the entire dataset. Figures 35, 36, and 37 and Tables 15, 16, and 17 provide comparative analyses of Wells 0, 5, and 11, respectively, each employing three different processing methods: Fast Fourier Transform Filter (FFTF) [275], Pseudo Flat Field Correction (PFFC) [276], and BaSiC. The Fast Fourier Transform (FFT) filter is a digital image processing technique that removes unwanted spatial frequencies from images, such as those for uneven illumination correction in microscopy images. It transforms the image into its frequency components, selectively removes or attenuates specific frequency ranges associated with illumination artefacts, and then converts the filtered result back to the spatial domain using the inverse Fourier transform.

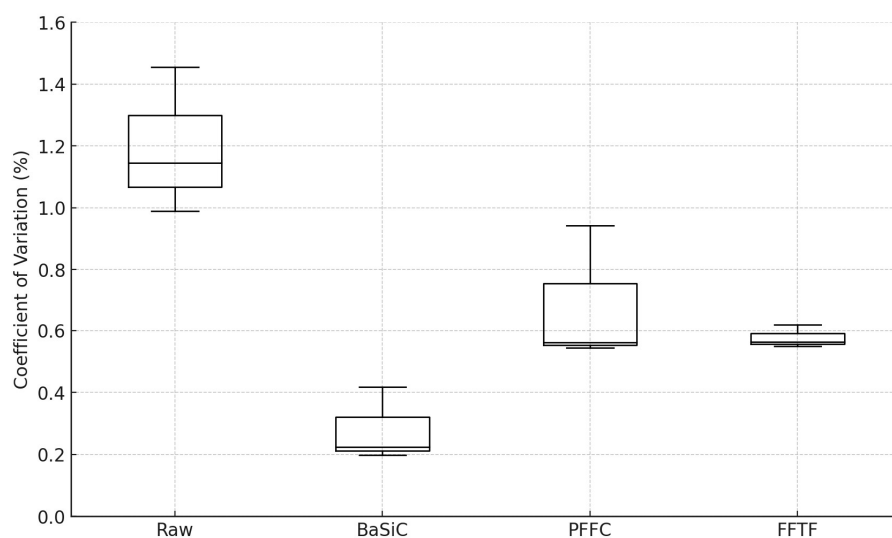


Figure 35: Coefficient of Variation (%) Comparison Across Different Correction Methods for Well 0

Table 15: Mean and Standard Deviation of Coefficient of Variation (%) for Different Correction Methods for Well 0.

| Method | Mean   | Standard deviation |
|--------|--------|--------------------|
| RAW    | 1.1962 | 0.1937             |
| BaSiC  | 0.2799 | 0.0982             |
| PFFC   | 0.6828 | 0.1840             |
| FFTF   | 0.5772 | 0.0298             |

By adjusting the filter parameters, this method can manage gradual shading effects and localised illumination problems, resulting in a more uniformly illuminated image. Meanwhile, Pseudo Flat Field Correction (PFFC) is a technique used to correct uneven illumination or background in images when a true flat field reference image is not available.

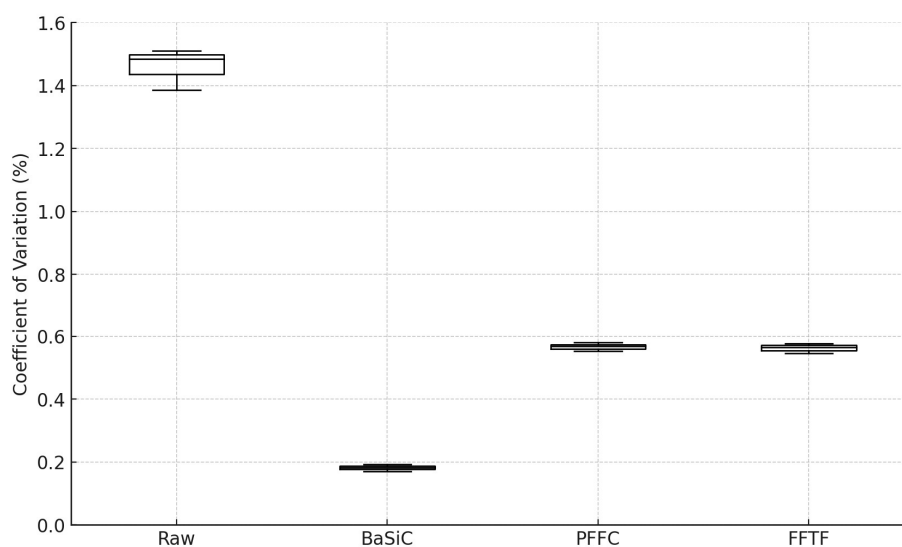


Figure 36: Coefficient of Variation (%) Comparison Across Different Correction Methods for Well 5

Table 16 : Mean and Standard Deviation of Coefficient of Variation (%) for Different Correction Methods for Well 5

| Method | Mean   | Standard deviation |
|--------|--------|--------------------|
| RAW    | 1.4606 | 0.0543             |

|       |        |        |
|-------|--------|--------|
| BaSiC | 0.1820 | 0.0088 |
| PFFC  | 0.5666 | 0.0115 |
| FFTF  | 0.5629 | 0.0129 |

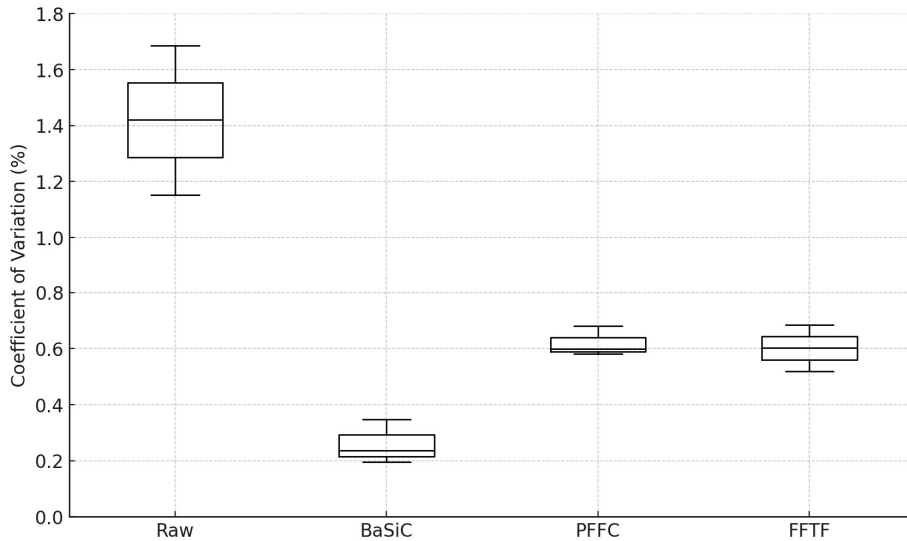


Figure 37: Coefficient of Variation (%) Comparison Across Different Correction Methods for Well 11

Table 17: Mean and Standard Deviation of Coefficient of Variation (%) for Different Correction Methods for Well 11

| Method | Mean   | Standard deviation |
|--------|--------|--------------------|
| RAW    | 1.4194 | 0.2180             |
| BaSiC  | 0.2596 | 0.0649             |
| PFFC   | 0.6199 | 0.0437             |
| FFTF   | 0.6017 | 0.0682             |

This process involves duplicating the original image, applying a Gaussian blur to the duplicate to remove objects and retain only the background illumination pattern, and then dividing the original image by the blurred version to flatten out the background. This approximates the effect of using a true flat field image for correction without the need for a separate reference image.

The comparative analysis of the coefficient of variation for Wells 0, 5, and 11 is effectively depicted through the boxplots and corresponding tables in Figures 35, 36, and 37. These visual and statistical representations illustrate the performance of various image-processing methods across the wells. The RAW data consistently displays the highest coefficient of variation across all wells, highlighting significant variability due to uneven illumination. In contrast, the BaSiC method shows the lowest variability, evidenced by its minimal presence on the boxplots and significantly lower mean and standard deviation values: 0.2799 and 0.0982 for Well 0, 0.1820 and 0.0088 for Well 5, and 0.2596 and 0.0649 for Well 11. Meanwhile, the PFFC and FFTF methods present medium levels of variation, with mean values of 0.6828 and 0.5772 and standard deviations of 0.1840 and 0.0298 for Well 0; 0.5666 and 0.5629 with standard deviations of 0.0115 and 0.0129 for Well 5; and 0.6199 and 0.6017 with standard deviations of 0.0437 and 0.0682 for Well 11, respectively. These detailed statistics provide insights into the variability and distribution characteristics of each method, elucidating how they manage illumination inconsistencies in processed images. Figures 38 and Table 18 present a thorough evaluation of the coefficient of variation across all wells, as depicted in the accompanying boxplot. The BaSiC approach has minimal variability, with a remarkably low mean of 0.2405 and a standard deviation of 0.0725, establishing it as the most stable and consistent method in terms of coefficient of variation (CV). The PFFC and FFTF approaches, which have higher CV values of 0.6231 and 0.5806, respectively, along with standard deviations of 0.1060 and 0.0410, show considerable variability compared to BaSiC. The low coefficient of variation (CV) values of the BaSiC approach highlight its efficiency in

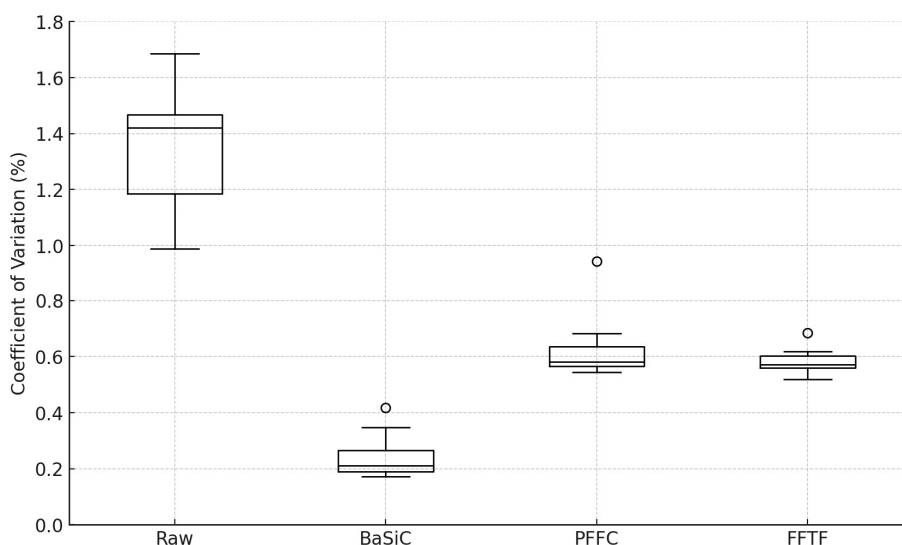


Figure 38: Coefficient of Variation (%) Comparison Across Different Correction Methods for all well



rectifying uneven illumination. These findings validate BaSiC as the optimal option for image processing in different wells, efficiently handling variability and enhancing consistency in image processing results.

Table 18: Mean and Standard Deviation of Coefficient of Variation (%) for Different Correction Methods for all well

| Method | Mean   | Standard deviation |
|--------|--------|--------------------|
| RAW    | 1.3588 | 0.1884             |
| BaSiC  | 0.2405 | 0.0725             |
| PFFC   | 0.6231 | 0.1060             |
| FFTF   | 0.5806 | 0.0410             |

#### 4.2.2 Image normalisation

The results of the image normalisation illustrate the variation in grey values across pixels in multiple frames using different methods of image processing. To illustrate the outcomes of image normalisation, Figures 39 and 40 provide a comparative analysis of images and their corresponding histograms before and after normalisation for image frames 1 and 575, respectively. In both figures, the images on the left side (a and b) represent the state before normalisation, while the images on the right side (c and d) show the state after normalisation. In Figure 39, image (a) displays the image after illumination correction, with histogram (b) illustrating the distribution of pixel intensities before normalisation, alongside key statistics such as the mean and standard deviation. After normalisation, image (c) shows the normalised image of frame 1, with histogram (d) reflecting the changes in pixel intensity distribution, indicating a more consistent and clear representation.

Similarly, Figure 40 focuses on image frame 575. Image (a) shows the image after illumination correction, with histogram (b) depicting the initial pixel intensity distribution before normalisation. Post-normalization, image (c) presents the normalised image of frame 575, and histogram (d) demonstrates the effects of normalisation on the pixel intensities, showing a more refined and consistent image. These examples clearly illustrate the impact of normalisation on image quality, enhancing clarity and uniformity across the frames.

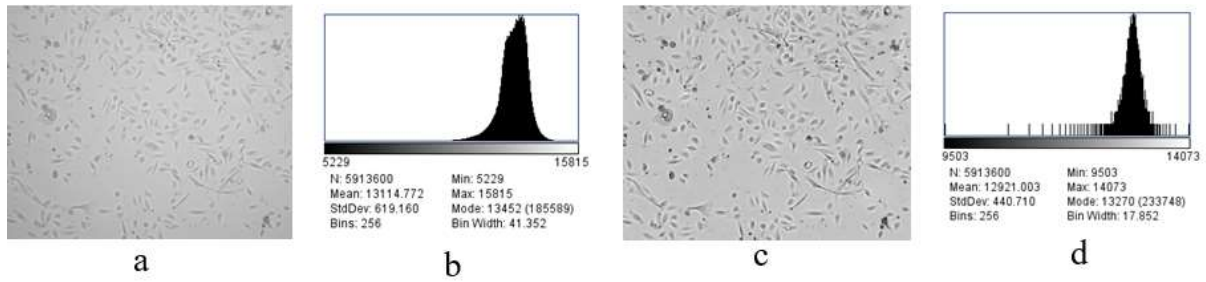


Figure 41: Images and histograms for image frame 1 before and after normalization. (a) Image before normalization, (b) Histogram corresponding to the image before normalization, (c) Image after normalization, (d) Histogram corresponding to the image after normalisation

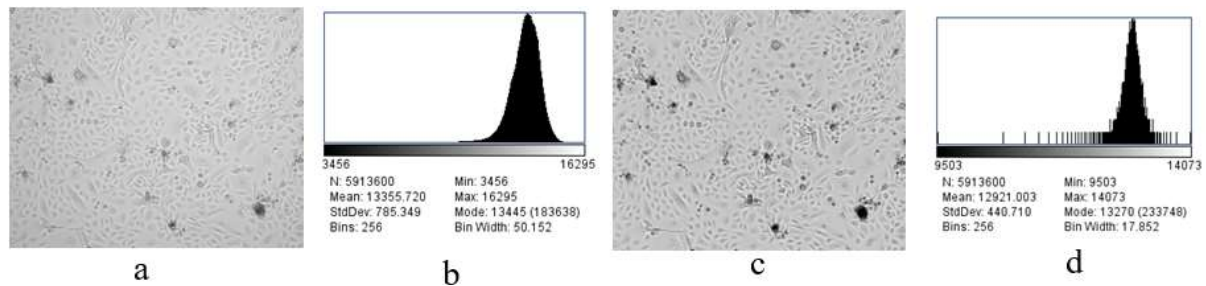


Figure 40: Images and histograms for image frame 575 before and after normalization. (a) Image before normalization, (b) Histogram corresponding to the image before normalization, (c) Image after normalization, (d) Histogram corresponding to the image after normalisation.

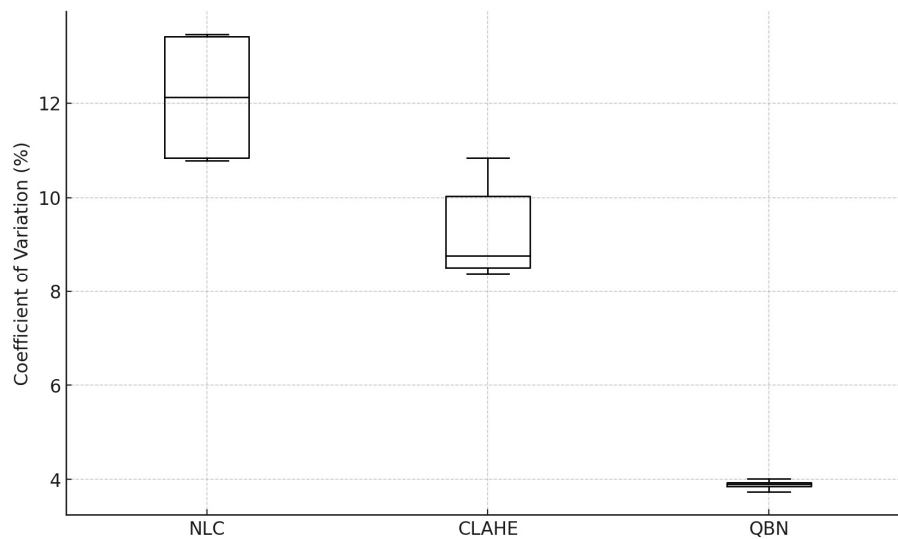


Figure 39: Coefficient of Variation (%) Comparison Across Different Normalisation Methods for Well 0.

Figures 41, 42, and 43 and Tables 19, 20, and 21 provide comparative analyses of Wells 0, 5, and 11, respectively, each employing three different processing methods: Normalize Local Contrast (NLC) [277], Contrast Limited Adaptive Histogram Equalization (CLAHE) [278], and Quantile-Based Normalization (QBN). CLAHE is an image processing technique that enhances image contrast by adaptively equalising the histogram of pixel intensities within small regions or tiles of the image. CLAHE works locally to improve contrast and reveal details

in both bright and dark areas. The image is divided into smaller, non-overlapping tiles, and a histogram is computed for each tile to represent the intensity distribution within that region. Histogram equalisation is applied independently to each tile to spread the intensity values more evenly. To prevent noise amplification in homogeneous regions, the histogram is clipped at a predefined limit, and the excess part is redistributed among all histogram bins. In order to create a consistent overall image, the contrast-enhanced tiles are finally combined using bilinear interpolation, which provides uniform transitions between neighbouring tiles.

Table 19: Mean and Standard Deviation of CV (%) for Different Normalisation Methods for Well 0

| Method | Mean    | Standard deviation |
|--------|---------|--------------------|
| NLC    | 12.1217 | 1.0993             |
| CLAHE  | 9.2524  | 0.8909             |
| QBN    | 3.8851  | 0.0775             |

Besides, Normalise Local Contrast is a method that changes pixel intensity levels within predefined neighbourhoods to improve local contrast. For the purpose of determining the size of the local region around each pixel, the method comprises picking block radii along the x and y axes. Within these small locations, the degree of contrast enhancement is controlled by the standard deviation parameter. The approach computes the local mean and standard deviation

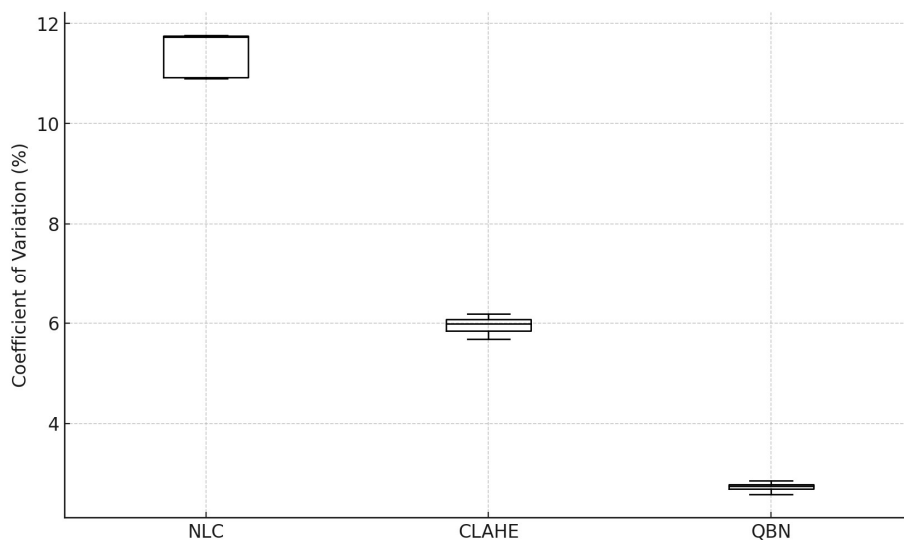


Figure 42: Coefficient of Variation (%) Comparison Across Different Normalisation Methods for Well 5

for every block and normalises the pixel values based on these computations to attain a uniform distribution. This normalisation improves detail visibility and lessens the impact of non-uniform light. The comparative analysis of the CV for Wells 0, 5, and 11 is visually and statistically represented through the boxplots and corresponding tables in Figures 41, 42, and 43, illustrating the performance of different normalisation methods across the wells. In Well 0, the boxplot shows NLC with the highest coefficient of variation, followed by CLAHE and QBN with the least.

Table 20: Mean and Standard Deviation of CV (%) for Different Normalisation Methods for Well 5

| Method | Mean    | Standard deviation |
|--------|---------|--------------------|
| NLC    | 11.4656 | 0.4036             |
| CLAHE  | 5.9609  | 0.1614             |
| QBN    | 2.7343  | 0.0782             |

The summary statistics (Table 19) detail NLC having the highest mean of 12.1217 and a standard deviation of 1.0993, CLAHE with a mean of 9.2524 and a standard deviation of 0.8909, and QBN with the lowest mean of 3.8851 and a minimal standard deviation of 0.0775. For Well 5 (Table 20), the NLC method exhibits the highest variability in CV, with a mean of 11.4656 and a standard deviation of 0.4036. Following NLC, CLAHE and QBN show lower

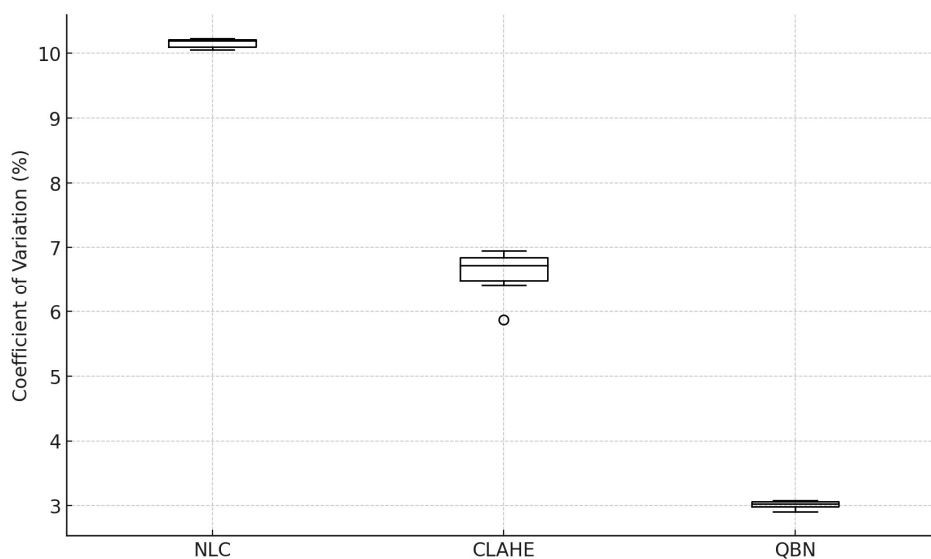


Figure 43: Coefficient of Variation (%) Comparison Across Different Normalisation Methods for Well 11

variability. CLAHE has a mean of 5.9609 and a standard deviation of 0.1614, while QBN displays a mean of 2.7343 and a standard deviation of 0.0782. In Well 11 (Table 21), QBN exhibits the lowest variability, with a mean of 3.0225 and a standard deviation of 0.0524. Next, the NLC method shows the highest coefficient of variation, with a mean of 10.1634 and a standard deviation of 0.0624. Lastly, CLAHE presents lower variability compared to NLC, with a mean of 6.6540 and a standard deviation of 0.2766.

Table 21: Mean and Standard Deviation of CV (%) for Different Normalisation Methods for Well 11

| Method | Mean    | Standard deviation |
|--------|---------|--------------------|
| NLC    | 10.1634 | 0.0624             |
| CLAHE  | 6.6540  | 0.2766             |
| QBN    | 3.0225  | 0.0524             |

The coefficient of variation (CV) across all wells was analysed, with results presented in Figures 44 and Table 22 focusing on image normalisation. The QBN method showed the best performance in image normalisation, with minimal variability (mean CV = 3.2286, SD = 0.5035). Such a result establishes QBN as the most stable and consistent approach for image normalisation. NLC and CLAHE displayed higher variability, with NLC yielding a mean CV

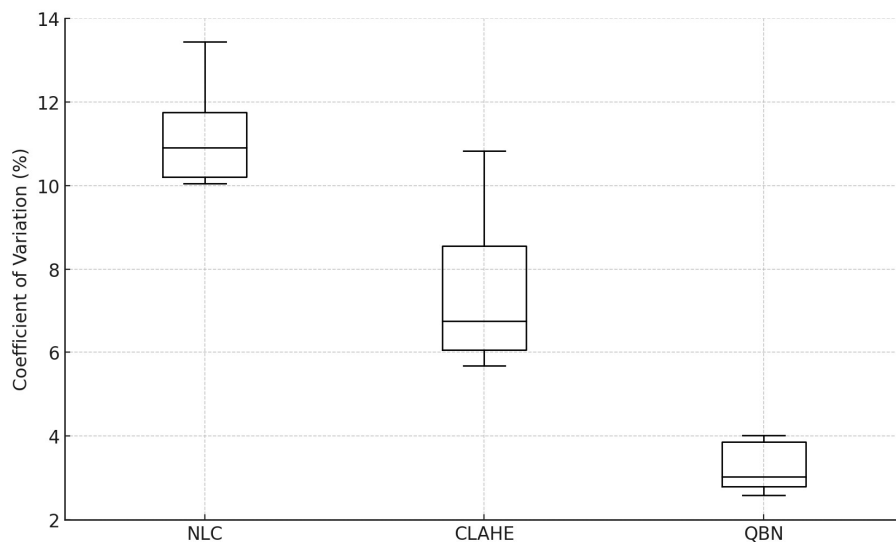


Figure 44: Coefficient of Variation (%) Comparison Across Different Normalisation Methods for all well

of 11.2692 (SD = 1.0520) and CLAHE producing a mean CV of 7.3318 (SD = 1.5395). The low CV values of the QBN approach demonstrate its effectiveness in generating uniform image enhancements across different well conditions. These findings support QBN as the optimal image processing technique for applications requiring consistent results in image normalisation across multiple wells.

Table 22: Mean and Standard Deviation of CV (%) for Different Normalisation Methods for all well

| Method | Mean    | Standard deviation |
|--------|---------|--------------------|
| NLC    | 11.2692 | 1.0520             |
| CLAHE  | 7.3318  | 1.5395             |
| QBN    | 3.2286  | 0.5035             |

### 4.3 Cell segmentation

In this analysis, the performance of three segmentation tools, Ilastik, Weka [279], and Labkit [280], are evaluated using datasets from the control, TGF\_beta, and SB431542 treatments. Ilastik, detailed in Chapter 3, is the primary tool and is compared against Weka and Labkit to show the most effective tool for cell segmentation. The Weka segmentation tool, known as Trainable Weka Segmentation, is an advanced image analysis tool that combines machine learning algorithms with image processing techniques to perform automated segmentation of complex images. It utilises Weka data mining and machine learning software to train a classifier based on user-defined examples of different image regions or features. This classifier then applies the learned patterns to segment entire images or image stacks, distinguishing between various structures or objects of interest. Labkit is a Fiji plugin that segments large microscopy image data. It provides memory-efficient and fast random forest-based pixel classification for automated image segmentation. Additionally, Labkit offers a user-friendly interface for the manual annotation of images.

This section provides a summary of cell segmentation performance. Data from well 0, representing the control group, well 4, associated with TGF-B treatment, and well 8, corresponding to SB431542 treatment, are presented. Detailed information for the remaining wells can be found in Appendix A (Figure A.1 to Figure A.9). For the data from well 0 (Figure

45, Table 23), Ilastik exhibited a mean Dice coefficient of 0.9076 with a standard deviation of 0.0162. In comparison, Weka and Labkit displayed lower mean coefficients of 0.7995 and 0.6345, with standard deviations of 0.0180 and 0.0485, respectively. The corresponding boxplot highlights Ilastik's high median and narrow interquartile range (IQR), indicating reliable performance. The cell segmentation performance for the TGF-Beta treatment dataset is depicted in the figures and tables provided. In well 4 (Figure 46, Table 24), Ilastik showed a mean Dice coefficient of 0.8713 with a standard deviation of 0.0225. Weka and Labkit recorded lower mean coefficients of 0.7859 and 0.7945, with standard deviations of 0.0109 and 0.0078, respectively. The boxplot demonstrates Ilastik's superior performance, characterised by a higher median and narrower interquartile range (IQR), which indicates more consistent results compared to Weka and Labkit.

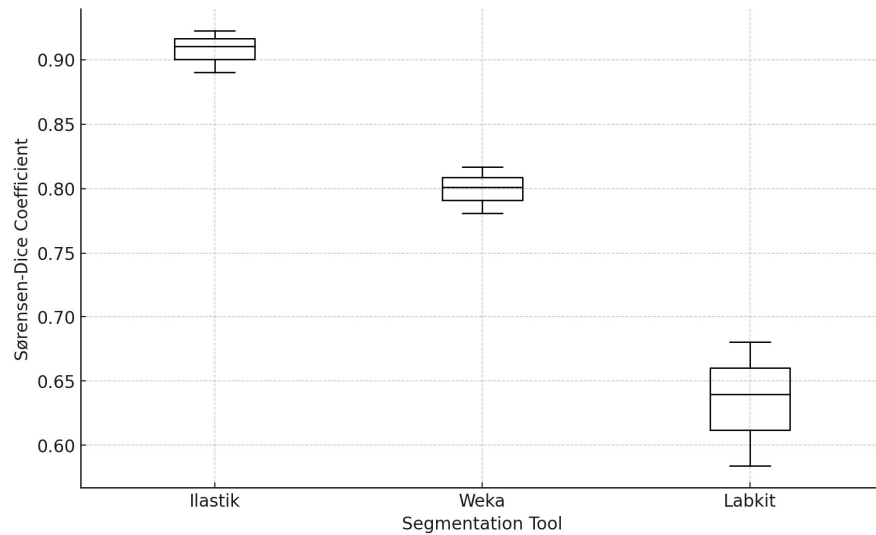


Figure 45: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 0.

Table 23: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 0

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.9076 | 0.0162             |
| Weka              | 0.7995 | 0.0180             |
| Labkit            | 0.6345 | 0.0485             |

The figures and tables provided details on the cell segmentation performance of the SB431542 treatment dataset. In well 8 (Figure 47, Table 25), Ilastik achieved a mean Dice coefficient of 0.8942 with a standard deviation of 0.0095. In contrast, Weka and Labkit exhibited lower mean coefficients of 0.8003 and 0.7150, with standard deviations of 0.0210 and 0.0133, respectively. The boxplot highlights Ilastik's superior performance, which is evident from its higher median and narrower interquartile range (IQR), indicating more consistent results than its counterparts.

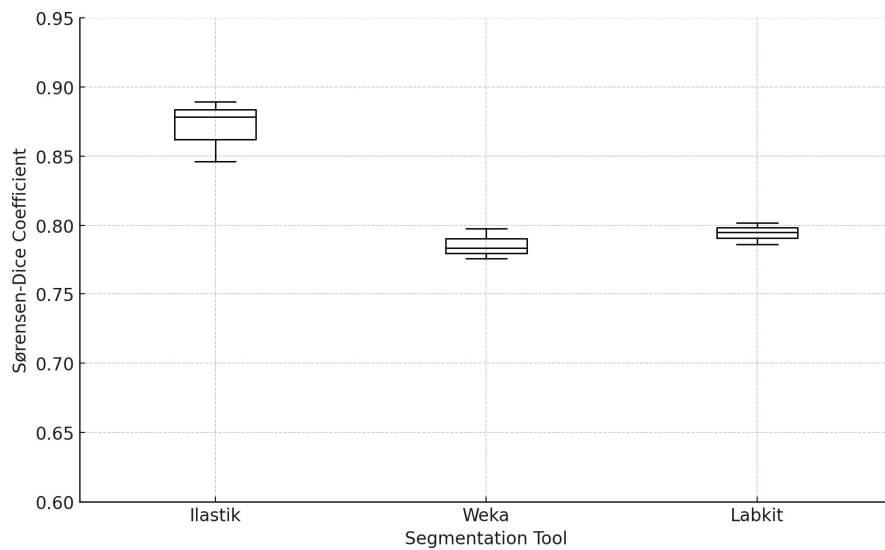


Figure 46: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 4

Table 24: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 4

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.8713 | 0.0225             |
| Weka              | 0.7859 | 0.0109             |
| Labkit            | 0.7945 | 0.0078             |



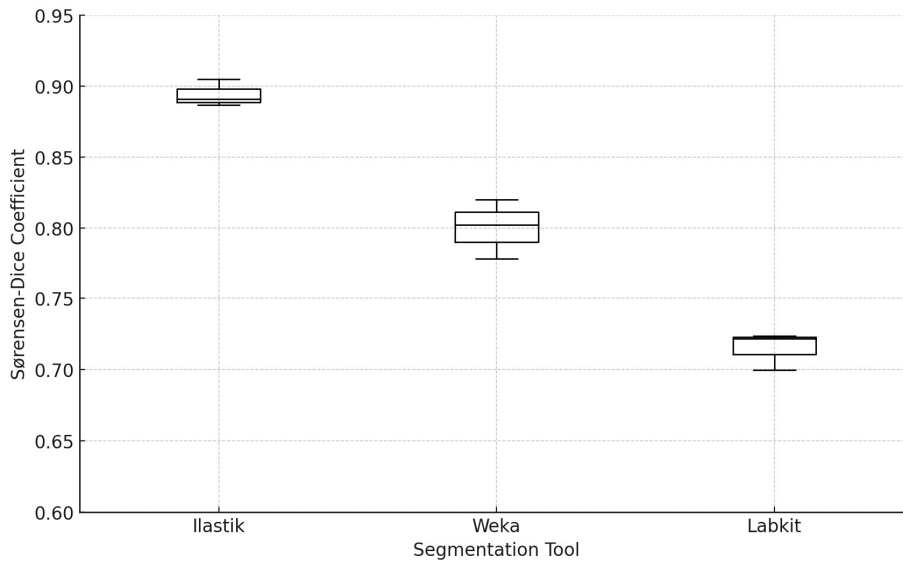


Figure 47: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 8

Table 25: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 8

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.8942 | 0.0095             |
| Weka              | 0.8003 | 0.0210             |
| Labkit            | 0.7150 | 0.0133             |

The overall performance of cell segmentation across different treatments using various tools is illustrated in Figure 48. This analysis encompasses datasets from control, TGF-Beta, and SB431542 treatments. In Figure 48, the boxplots show the Sørensen-Dice Coefficient for Ilastik, Weka, and Labkit across all treatments. Ilastik consistently achieved the highest median Dice coefficients across the treatments, indicating its effectiveness in segmentation. Weka displayed moderate performance with greater variability, while Labkit showed the lowest and most variable results across the treatments. The summary statistics for the Sørensen-Dice Coefficient using different segmentation tools for all treatments. Ilastik demonstrated the highest mean Dice coefficient for the control treatment (0.9121, SD = 0.0166), followed by TGF-Beta (0.8745, SD = 0.0262), and SB431542 (0.8771, SD = 0.0175). Weka showed lower mean Dice coefficients, with 0.8059 (SD = 0.0142) for the control, 0.7776 (SD = 0.0323) for

TGF-Beta, and 0.7771 (SD = 0.0290) for SB431542. Labkit exhibited the lowest mean Dice coefficients, with 0.7599 (SD = 0.0797) for the control, 0.7234 (SD = 0.0549) for TGF-Beta, and 0.7283 (SD = 0.0531) for SB431542

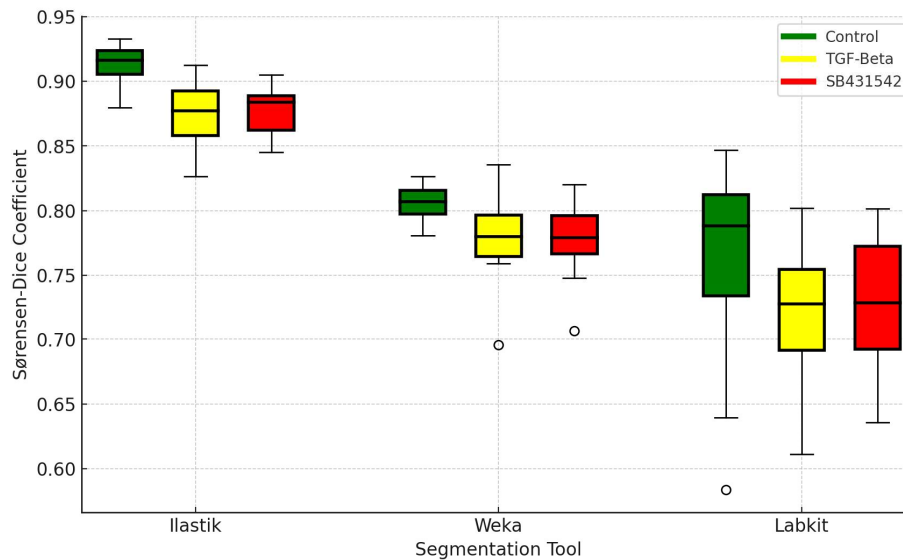


Figure 48: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for all well

These results highlight Ilastik's effectiveness in cell segmentation across all treatments, as evidenced by its highest mean Dice coefficients and relatively low standard deviations. This consistency is particularly important in biological research, where treatments like TGF-Beta and SB431542 can lead to significant morphological changes in cells. Ilastik's ability to maintain high accuracy across diverse conditions ensures that these treatment-related changes are captured accurately, leading to more reliable interpretations of experimental outcomes. Weka, while showing moderate segmentation performance, displayed higher variability. This variability could complicate the interpretation of results in biological studies, potentially leading to less reliable conclusions about treatment effects. Labkit consistently performed the most lacking, with the lowest mean Dice coefficients and higher standard deviations. This suggests that Labkit may struggle to accurately segment cells, particularly when treatments cause significant morphological changes, making it less suitable for studies requiring precise cell segmentation. In summary, a higher Sørensen-Dice Coefficient value signifies enhanced performance of the segmentation algorithm. A coefficient approaching 1 indicates that the predicted segmentation closely aligns with the ground truth, demonstrating greater accuracy.

This level of precision is crucial in biological research, where accurate segmentation is necessary for downstream analyses such as cell tracking and treatment efficacy assessment. As a result, Ilastik stands out as the most efficient tool for achieving precise and dependable cell segmentation across different treatment conditions.

Table 26 presents the results of the significance test conducted using the Wilcoxon Signed-Rank Test, which demonstrates that Ilastik exhibits a statistically significant difference in performance compared to both Weka and Labkit when applied to the dataset from the control environment ( $p\text{-value} < 0.05$ ). Furthermore, the test results indicate that there is no statistically significant difference in performance between Weka and Labkit, as the  $p\text{-value}$  exceeds 0.05. This implies that these two algorithms perform similarly in terms of the segmentation task.

Table 26: Significance test results for different segmentation algorithms on the control environment dataset.

| Comparison        | P-Value | Significant Different |
|-------------------|---------|-----------------------|
| Ilastik vs Weka   | 0.00024 | Yes                   |
| Ilastik vs Labkit | 0.00023 | Yes                   |
| Weka vs Labkit    | 0.08    | No                    |

Table 27 presents the results of the significance test using the same method as in Table 26, demonstrating that Ilastik, Weka, and Labkit each perform differently when applied to the TGF-B treatment dataset. The  $p\text{-values}$  are all less than 0.05, indicating statistically significant differences between each pair of algorithms in terms of segmentation task performance.

Table 27: Significance test results for different segmentation algorithms on the TGF-B treatment dataset.

| Comparison        | P-Value | Significant Different |
|-------------------|---------|-----------------------|
| Ilastik vs Weka   | 0.00023 | Yes                   |
| Ilastik vs Labkit | 0.00022 | Yes                   |
| Weka vs Labkit    | 0.03    | Yes                   |

Table 28 shows that Ilastik, Weka, and Labkit exhibit different performance levels when applied to the SB431542 treatment dataset. The p-values for each pairwise comparison between Ilastik and Weka, Ilastik and Labkit, and Weka and Labkit are all below 0.05, confirming statistically significant differences in segmentation task performance among the three algorithms.

Table 28: Significance test results for different segmentation algorithms on the SB431542 treatment dataset.

| Comparison        | P-Value | Significant Different |
|-------------------|---------|-----------------------|
| Ilastik vs Weka   | 0.00022 | Yes                   |
| Ilastik vs Labkit | 0.00021 | Yes                   |
| Weka vs Labkit    | 0.0061  | Yes                   |

#### 4.4 Cell tracking

In this analysis, the performance of three cell tracking methods, Follow Neighbour (FN), Overlap [281], and Linear Assignment Problem (LAP) [282], is evaluated using datasets from the control, TGF-beta, and SB431542 treatments. Follow Neighbour (FN), detailed in Chapter 3, serves as the primary tracking method and is compared against the Overlap and Linear Assignment Problem (LAP) methods to determine the most effective approach for cell tracking. Overlap-based cell tracking is developed to follow the movement and behaviour of cells over time in a sequence of time-lapse images. It relies on measuring the overlap between cellular regions in consecutive frames to determine cell position and identity. The core concept is that a cell in one frame will occupy a similar position in the next frame, allowing the tracking algorithm to match cells based on their overlapping areas. The method is highly automated, requiring minimal user intervention, making it suitable for handling large datasets efficiently. It is adaptable to various imaging conditions and cell types, optimised for speed, enabling the rapid processing of large volumes of images. The Linear Assignment Problem (LAP) is a combinatorial optimisation challenge that assigns agents to tasks to minimise total costs. This optimisation is realised by establishing an optimal assignment that relies on a cost matrix, where each matrix element quantifies the cost of assigning a specific agent to a particular task. In cell tracking, LAP is employed to monitor cell movements across consecutive frames in time-lapse imaging. The Hungarian algorithm is utilised to solve the LAP efficiently. It

functions by subtracting the smallest element from each row and column of the cost matrix, covering all resulting zeros with the minimal number of lines, and iteratively adjusting the matrix until an optimal assignment is achieved. In cell tracking, LAP encompasses two principal steps. The initial step, frame-to-frame linking, involves connecting detected cells between consecutive frames using a cost matrix typically derived from the distances between cell positions, aiming to minimise these cost sums. The subsequent step, track segment linking, connects these initial track segments to forge complete trajectories. The metric MOTA is used to quantify the accuracy of the cell tracking performance, where a MOTA close to 100% is preferred because it indicates that the tracking algorithm is highly accurate, with minimal errors in tracking multiple objects. This section provides the cell tracking performance. Data from well 0, representing the control group, well 4, associated with TGF-B treatment, and well 8, corresponding to SB431542 treatment, are presented. Detailed information for the remaining wells can be found in Appendix A (Figure A.10 to Figure A.18). The Follow Neighbour method, as shown in Figure 49 and Table 29 for well 0, demonstrated the highest performance, with a mean MOTA (Multiple Object Tracking Accuracy) of 90.17 and a standard deviation of 3.08. The Overlap method followed with a mean of 81.72 and a standard deviation of 7.22. The Linear Assignment Problem (LAP) method showed the lowest performance, with a mean of 62.03 and a standard deviation of 2.88. The corresponding boxplot illustrates that the Follow Neighbour method had the highest median MOTA and a narrow interquartile range (IQR), indicating consistent performance.

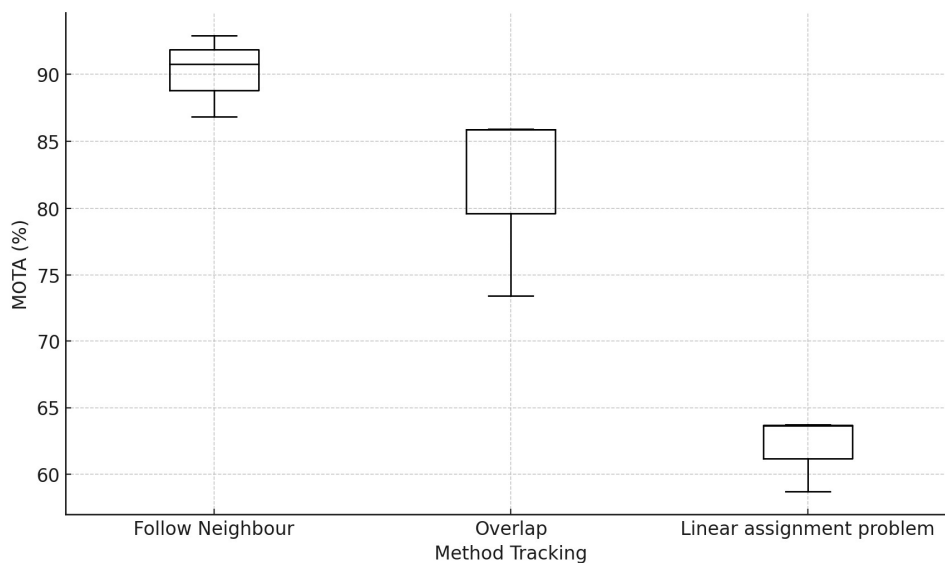


Figure 49: MOTA (%) Comparison Across Different Tracking Methods for Well 0.

Table 29: Mean and SD of MOTA (%) for Different Tracking Method for Well 0

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 90.17 | 3.08               |
| Overlap                   | 81.72 | 7.22               |
| Linear assignment problem | 62.03 | 2.88               |

As shown in Figure 50 and Table 30 for well 4, the Follow Neighbour method demonstrated the highest performance, with a mean MOTA (Multiple Object Tracking Accuracy) of 85.22 and a standard deviation of 3.14. The Overlap method followed with a mean of 74.64 and a standard deviation of 3.99. In contrast, the Linear Assignment Problem (LAP) method exhibited the lowest performance, with a mean of 53.35 and a standard deviation of 4.62. In well 5 (Figure 61, Table 40), the Follow Neighbour method achieved the highest mean MOTA at 83.80, with a standard deviation of 4.02. The Overlap method recorded a mean of 71.50 and a standard deviation of 1.96. In contrast, the LAP method's performance was significantly lower, with a mean of 54.37 and a standard deviation of 1.58.

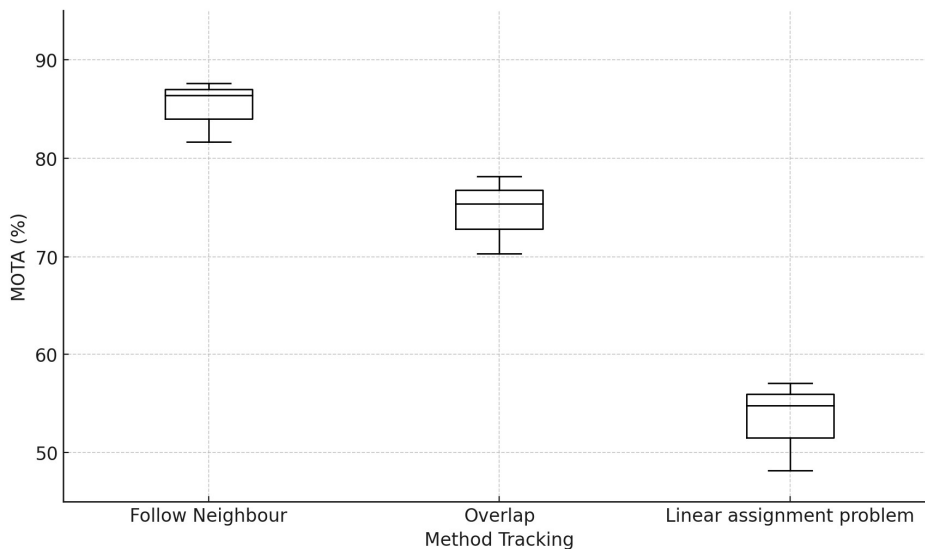


Figure 50: MOTA (%) Comparison Across Different Tracking Methods for Well 4

Table 30: Mean and SD of MOTA (%) for Different Tracking Method for Well 4

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 85.22 | 3.14               |
| Overlap                   | 74.64 | 3.99               |
| Linear assignment problem | 53.35 | 4.62               |

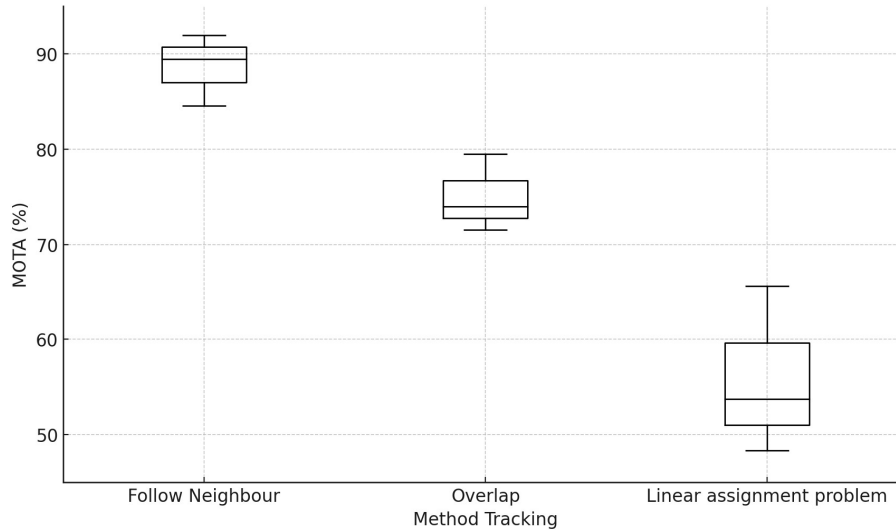


Figure 51: MOTA (%) Comparison Across Different Tracking Methods for Well 8

The dataset for Well 8 is illustrated in Figure 51 and summarised in Table 31. The Follow Neighbour method is the most effective, with a mean score of 88.66 and a standard deviation of 3.76, indicating stable results. The Overlap Method Tracking achieves a moderate mean of 75.03 and a standard deviation of 4.06. In contrast, the Linear Assignment Problem method records the lowest mean at 55.88 and the highest variability with a standard deviation of 8.81. The box plot in Figure 51 demonstrates these findings, showing higher median values and tighter interquartile ranges for the Follow Neighbour method compared to the others.

Table 31: Mean and SD of MOTA (%) for Different Tracking Method for Well 8

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 88.66 | 3.76               |
| Overlap                   | 75.03 | 4.06               |
| Linear assignment problem | 55.88 | 8.81               |

The overall performance of cell tracking across different treatments using various methods is illustrated in Figure 52. This analysis encompasses datasets from control, TGF-Beta, and SB431542 treatments. In Figure 52, the boxplots display the Multi-Object Tracking Accuracy (MOTA) for the Follow Neighbour, Overlap, and Linear Assignment Problem methods across all treatments. The Follow Neighbour method consistently achieved the highest median MOTA percentages across the treatments, indicating its effectiveness in tracking. The Overlap method displayed moderate performance with greater variability, while the Linear Assignment Problem method exhibited the least favourable results with the most variability across the treatments. The summary statistics for MOTA using different tracking methods for all treatments. The Follow Neighbour method demonstrated the highest mean MOTA for the control treatment (88.23, SD = 3.30), followed by TGF-Beta (86.31, SD = 4.12), and SB431542 (87.35, SD = 4.55). The Overlap method showed lower mean MOTA percentages, with 75.44 (SD = 8.37) for the control, 74.56 (SD = 4.74) for TGF-Beta, and 74.64 (SD = 4.03) for SB431542. The Linear Assignment Problem method exhibited the lowest mean MOTA percentages, with 55.58 (SD = 7.97) for the control, 56.53 (SD = 5.12) for TGF-Beta, and 56.35 (SD = 5.52) for SB431542. These results highlight the effectiveness of the Follow Neighbour method in cell tracking across all treatments, evidenced by its highest mean MOTA percentages and relatively low standard deviations. The Overlap method, while showing moderate tracking performance, displayed higher variability. The Linear Assignment Problem method consistently exhibited the lowest mean MOTA percentages and higher standard deviations. In

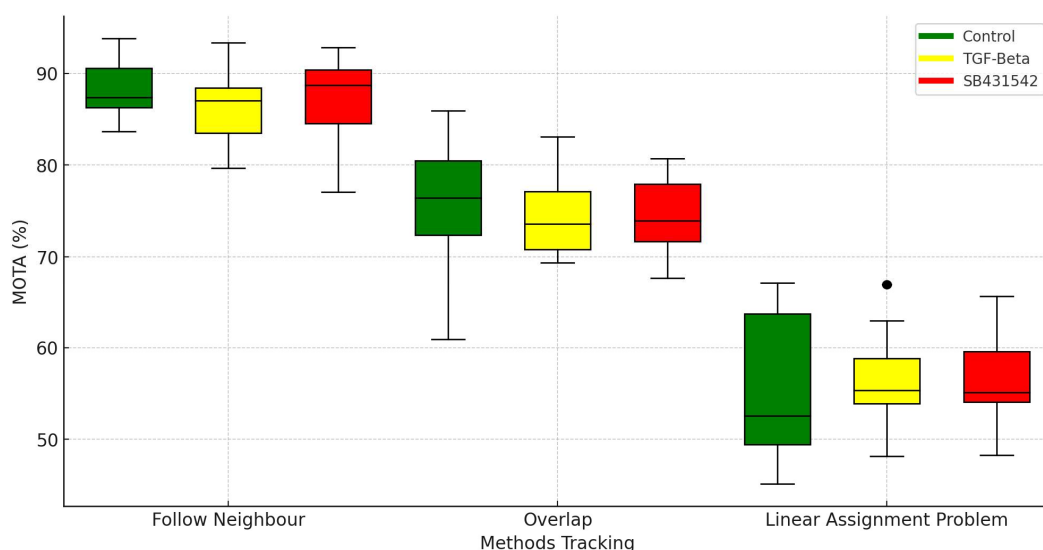


Figure 52: MOTA (%) Comparison Across Different Tracking Methods for all well



summary, a higher MOTA value signifies enhanced performance of the tracking method. A higher percentage indicates that the predicted tracking closely aligns with the ground truth, demonstrating greater accuracy. As a result, the Follow Neighbour method stands out as the most efficient method for achieving precise and dependable cell tracking across different treatment conditions.

Table 32 presents the results of the significance test conducted using the Wilcoxon Signed-Rank Test, demonstrating that the different cell tracking methods exhibit statistically significant differences in performance when applied to the dataset from the control environment ( $p\text{-value} < 0.05$ ). The results indicate that there are significant performance differences between Follow Neighbour and Overlap and Linear Assignment Problem, as well as Overlap and Linear Assignment Problem. These findings suggest that the algorithms perform differently in terms of the cell tracking task under the control environment.

Table 32: Significance test results for different cell tracking methods on the control environment dataset.

| Comparison                           | P-Value | Significant Different |
|--------------------------------------|---------|-----------------------|
| Follow Neighbour vs Overlap          | 0.00048 | Yes                   |
| Ilastik vs Linear Assignment Problem | 0.00052 | Yes                   |
| Overlap vs Linear Assignment Problem | 0.00047 | Yes                   |

Table 33 presents the results of the significance test using the same method as in the previous analysis, demonstrating that each of the cell tracking methods (Follow Neighbour, Overlap and Linear Assignment Problem) perform significantly differently when applied to the TGF-Beta treatment dataset. The p-values for all comparisons are less than 0.05, indicating statistically significant differences between each pair of methods in terms of their cell tracking performance.

Table 33: Significance test results for different cell tracking methods on the TGF-Beta treatment dataset.

| Comparison                           | P-Value | Significant Different |
|--------------------------------------|---------|-----------------------|
| Follow Neighbour vs Overlap          | 0.00041 | Yes                   |
| Ilastik vs Linear Assignmnet Problem | 0.00049 | Yes                   |
| Overlap vs Linear Assignmnet Problem | 0.00043 | Yes                   |

Table 34 demonstrates the significance of test results using the same test method as in previous tables, focusing on the cell tracking methods applied to the SB431542 treatment dataset. The table shows that Follow Neighbour, Overlap and Linear Assignment Problem exhibit statistically significant performance differences, with all p-values well below 0.05. These results confirm significant cell tracking performance distinctions among the tested methods.

Table 34: Significance test results for different cell tracking methods on the SB431542 treatment dataset.

| Comparison                           | P-Value | Significant Different |
|--------------------------------------|---------|-----------------------|
| Follow Neighbour vs Overlap          | 0.00049 | Yes                   |
| Ilastik vs Linear Assignmnet Problem | 0.00054 | Yes                   |
| Overlap vs Linear Assignmnet Problem | 0.00045 | Yes                   |

#### 4.5 Cell growth curve

Figure 53 illustrates the growth curves of cells over time, with data points collected at 5-minute intervals and the number of cells recorded for each well. The wells are categorised based on the treatments applied: green lines represent the control treatment (Wells 0, 1, 2, 3), red lines represent wells treated with SB431542, an inhibitor of the TGF-beta signalling pathway (Wells 8, 9, 10, 11), and yellow lines represent wells treated with TGF-Beta, a growth

factor (Wells 4, 5, 6, 7). At the beginning of the line graph, the initial cell counts for each treatment and well did not start at the same number of cells. This variation is due to factors such as the differing settings of fields of view for each well, which were based on the centre of the well and cell visibility, prioritising good lighting and minimal debris. Additionally, the movement of cells in and out of the microscope's field of view contributed to this variation. The green lines for Wells 0-3, representing the control treatment, show a steady increase in cell growth over time. Such a pattern indicates normal cell proliferation without any interference. The growth curves for the control wells are relatively similar, suggesting consistency in cell behaviour under standard conditions. In the case of the TGF-beta treatment, the treated wells show a slow increase in cell count over time, suggesting that TGF-beta may have an inhibitory effect on cell proliferation, thereby slowing down the growth rate compared to the control and SB431542 treatments. For the wells treated with SB431542, there is an increase in cell count, but they exhibit a moderate level of cell growth compared to the control treatment.

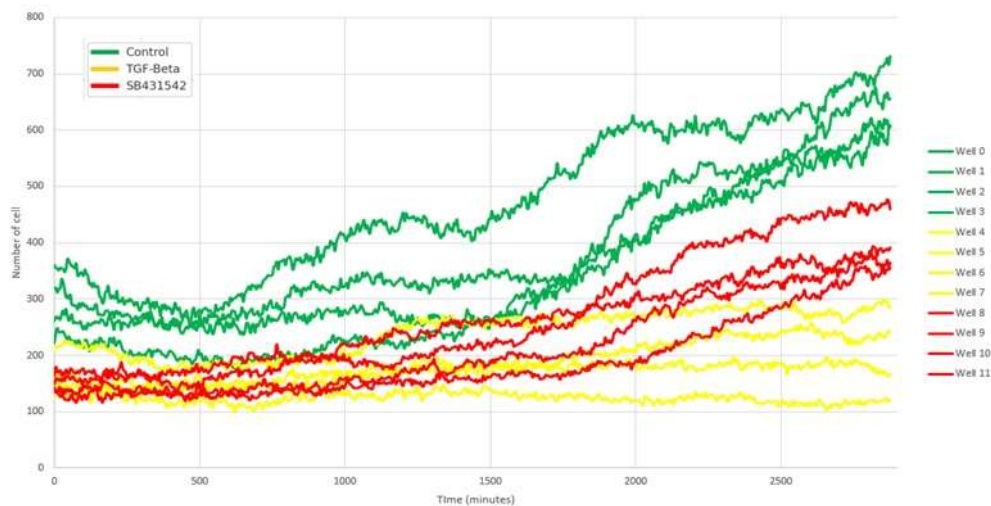


Figure 53: Cell Growth Curve Across Different Treatments

#### 4.6 Cell growth rate

Figure 54 illustrates the growth rate for different wells under control, TGF-beta, and SB431542 treatments. The colours indicate the specific treatments, with green bars representing the control (Wells 0-3), yellow bars for TGF-beta (Wells 4-7), and red bars for SB431542 (Wells 8-11). The wells under the control treatment show high growth rates, ranging from approximately  $6.0 \times 10^{-4}$  to  $8.0 \times 10^{-4} \text{ min}^{-1}$ , indicating normal cell proliferation under standard conditions without any treatment interference. The control wells display consistent

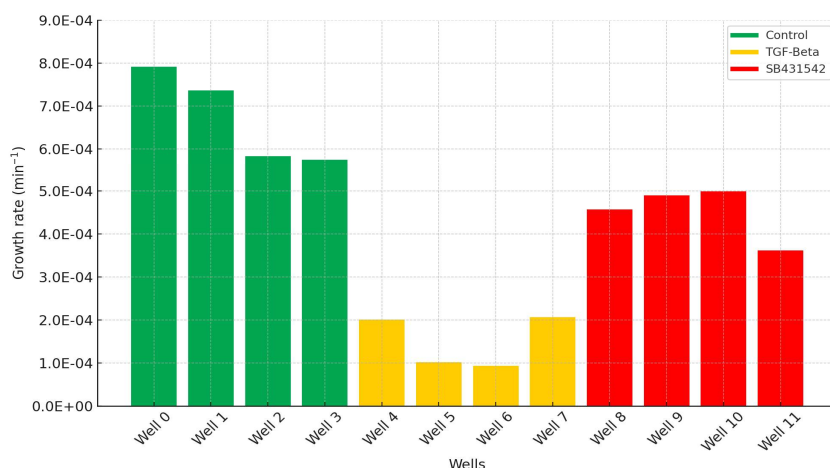


Figure 54: Cell Growth Rate Comparison Across Different Wells Under Various Treatments.

and high growth rates, serving as a baseline for normal cell proliferation. In contrast, the TGF-beta treatment results in significantly lower growth rates in Wells 4-7, mostly around  $1.0 \times 10^{-4}$  to  $3.0 \times 10^{-4} \text{ min}^{-1}$ , suggesting that TGF-beta has a substantial inhibitory effect on cell proliferation. The TGF-beta treated wells exhibit markedly lower growth rates than the control, confirming the inhibitory effect of TGF-beta on cell proliferation. The SB431542 treatment results in moderate to high growth rates in Wells 8-11, ranging from approximately  $4.0 \times 10^{-4}$  to  $7.0 \times 10^{-4} \text{ min}^{-1}$ . These rates are lower than the control but significantly higher than the TGF-beta treated wells. The wells treated with SB431542 show growth rates between the control and TGF-beta treated wells, promoting cell proliferation but not to the level of untreated cells. SB431542 is known to inhibit TGF-beta signalling, which explains the observed increase in growth rate compared to the TGF-beta treatment alone. Figure 55 illustrates the overall growth rates for the control, TGF-beta, and SB431542 treatments. The control group exhibits the highest growth rates, indicating consistent cell proliferation under standard conditions. The TGF-beta treatment group shows significantly lower growth rates, suggesting that TGF-beta substantially inhibits cell proliferation. The SB431542 treatment group demonstrates intermediate growth rates compared to the control and TGF-beta treatments.

The significance test used in Figure 55 to derive the p-values was the independent t-test, which is appropriate for comparing the means of two independent groups, particularly because the data is small and normally distributed, making this parametric test well-suited for evaluating differences in mean cell growth rates across the experimental conditions. Based on the significance test results, the p-value between the control group and the TGF-Beta group is 0.000427, which is significantly less than the typical significance level of 0.05, indicating a statistically significant difference in growth rate between the control and the TGF-Beta

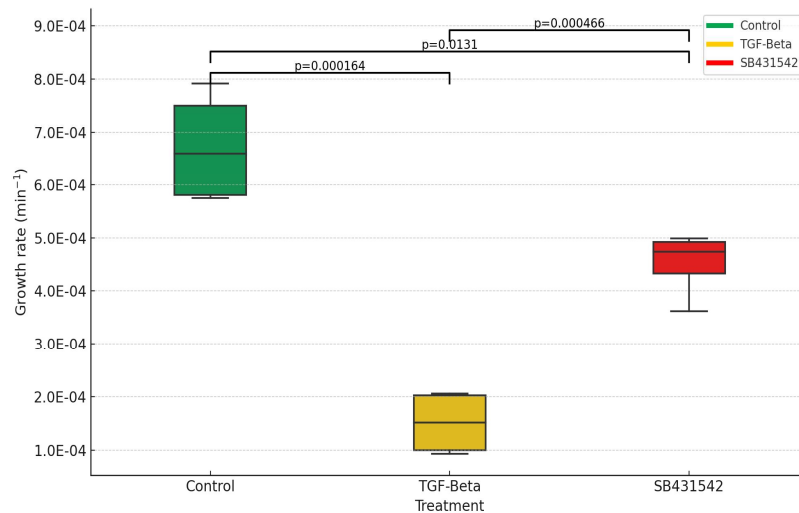


Figure 55: Cell Growth Rate Comparison Across Different Treatments with Statistical Significance

treatment. Similarly, the p-value between the control and the SB431542 treatment is 0.032384, also below the 0.05 threshold, confirming a statistically significant difference in growth rate between these two groups. Additionally, the p-value between the TGF-Beta and the SB431542 treatment is 0.000889, well below 0.05, again indicating a statistically significant difference in growth rate between these two treatment groups. Finally, the significance tests show that all pairwise comparisons between the Control, TGF-Beta, and SB431542 groups reveal statistically significant differences in growth rate. These findings suggest that the treatments, TGF-Beta and SB431542, significantly impact the growth rate compared to the control and each other.

#### 4.7 Cell migration speed

Figure. 56 illustrates the mean migration speed of cells over time under three different treatments: Control, TGF-beta, and SB431542. The migration speeds are plotted for multiple wells, with each treatment group represented by a specific colour. Green lines (Wells 0-3) represent the control treatment, yellow lines (Wells 4-7) represent the TGF-beta treatment, and red lines (Wells 8-11) represent the SB431542 treatment. The cells in the control group start with a high migration speed that decreases and stabilises at a lower rate. The same pattern is observed in the TGF-beta and SB431542 treated groups, but with notably higher stabilised speeds. The general decrease in migration speed observed across all treatments over time can be attributed to several biological factors, notably the effects of increasing cell density. As the cell density within the culture increases, it leads to crowding [283] and contact inhibition,

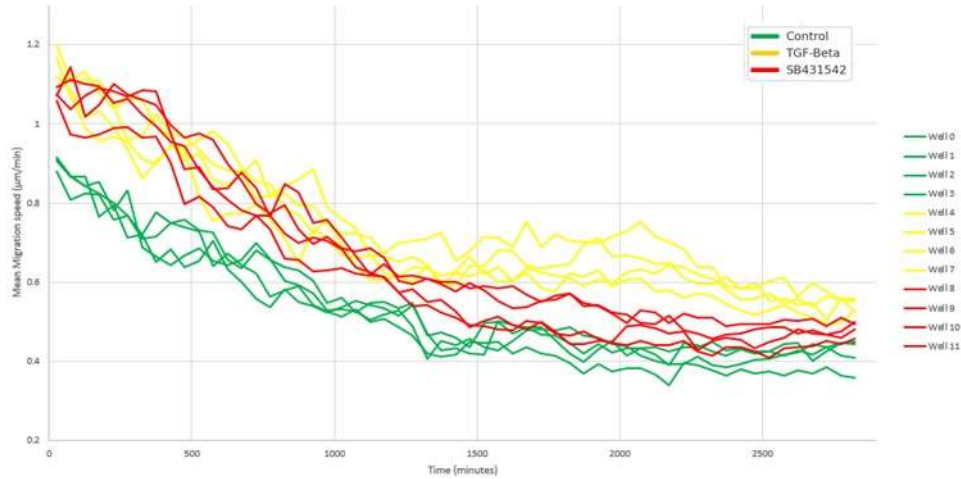


Figure 56: Mean Migration Speed Over Time for Different Treatment.

which physically restricts cell movements. Increased cell density results in more crowded conditions that trigger several phenomena affecting cell behaviour. These include increased mechanical forces exerted between cells, changes in the strain fields surrounding them, and alterations in the patterns of collective cell movement. Collectively, these factors contribute to a gradual decrease in migration speed as the cell population becomes denser, demonstrating how environmental and spatial constraints can significantly influence cellular dynamics.

Figure 57 presents the overall mean migration speeds for the Control, TGF-Beta, and SB431542 treatments, as depicted through boxplots summarising the data. The Control group exhibits the lowest migration speeds, representing baseline cell motility under standard,

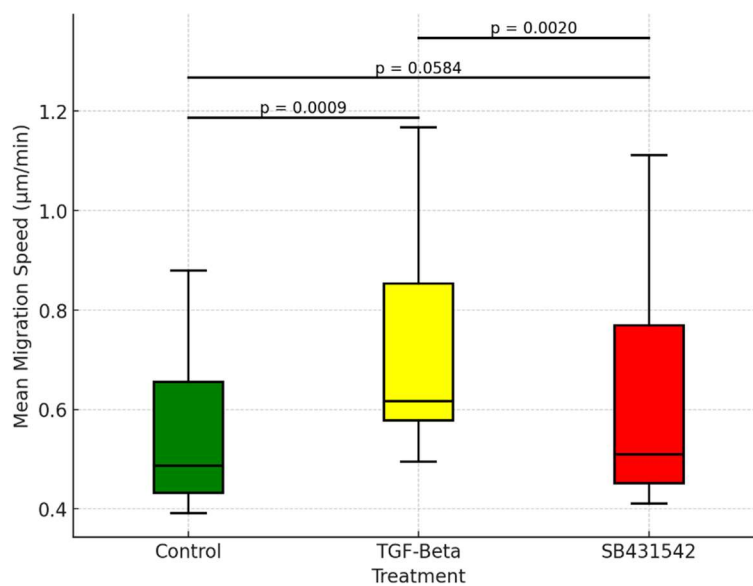


Figure 57: Mean Migration Speed Comparison Across Different Treatments with Statistical Significance.

untreated conditions. In contrast, the TGF-Beta treatment group demonstrates significantly higher migration speeds, indicating that TGF-Beta enhances cell motility. The SB431542 treatment group shows slightly lower migration speeds than the TGF-Beta group, although still higher than those observed in the Control group. The p-values displayed in the figure, derived from the Mann-Whitney U test, indicate the differences in migration speeds between the groups, some of which are statistically significant. Several conclusions can be drawn from the Mann-Whitney U test's statistical results regarding the effects of the treatments on mean migration speed. The comparison between the Control and TGF-Beta treatments reveals a significant difference in mean migration speed, with a p-value of  $p=0.0009$ . Given that this p-value is below the commonly accepted significance threshold of 0.05, it indicates that the TGF-Beta treatment significantly enhances mean migration speed compared to the Control, consistent with its established role in promoting cellular motility. In contrast, the comparison between the Control and SB431542 treatments yields a p-value of  $p=0.0584$ , which exceeds the significance threshold of 0.05. This suggests that the difference in mean migration speed between these two treatments is not statistically significant, implying that SB431542 has a more moderate effect on cell motility than TGF-Beta. Furthermore, the comparison between the TGF-Beta and SB431542 treatments demonstrates a significant difference in mean migration speed, with a p-value of  $p=0.0020$ . This finding indicates that TGF-Beta and SB431542 exert differential effects on cell motility. Overall, TGF-Beta significantly increases mean migration speed compared to both the Control and SB431542 treatments, while SB431542 exhibits a less pronounced but still elevated effect compared to the Control. These results underscore the potent role of TGF-Beta in promoting cell migration.

#### **4.8 Cell angular velocity**

Figure 58 displays the mean angular velocity of cells over time under three treatments: Control, TGF-beta, and SB431542. The angular velocities are plotted for multiple wells, with each treatment group represented by a specific colour. Green lines (Wells 0-3) represent the control treatment, yellow lines (Wells 4-7) represent the TGF-beta treatment, and red lines (Wells 8-11) represent the SB431542 treatment. At the initial state, the control treatment displays a slightly higher mean angular velocity than the TGF-beta and SB431542 treatments, which start with almost similar angular velocity values. Over time, cells in the control treatment show a slight increase in mean angular velocity, but these values remain lower than those

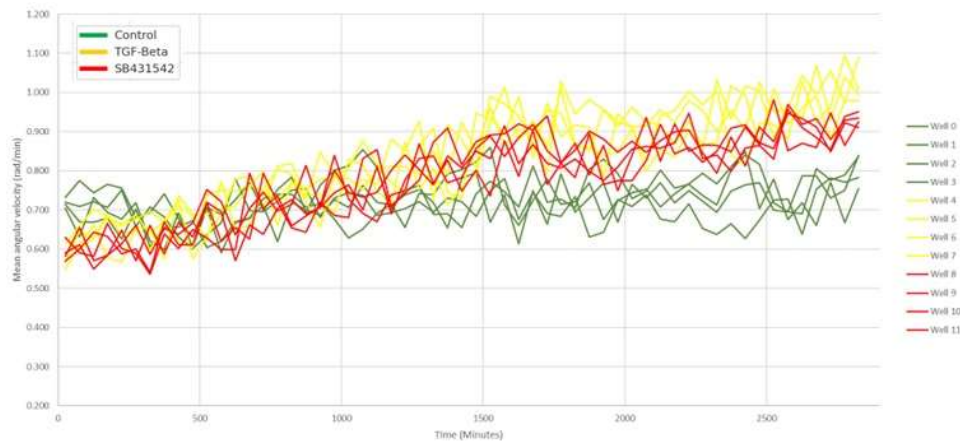


Figure 58: Mean Angular Velocity Over Time for Different Treatments.

observed in the TGF-beta and SB431542 treatments. However, cells treated with TGF-beta show a more pronounced increasing trend in angular velocity, stabilising at a higher level than the control. Similarly, cells treated with SB431542 also demonstrate an upward trend in angular velocity over time, with values higher than the control but slightly lower than the TGF-beta treated treatment. The observed increase in angular velocity over time across all treatments can be attributed to several biological factors, particularly the effects of increasing cell density in culture. As cell density rises, higher intercellular torque significantly affects the regulation of angular velocity in rotating cells [284]. With increasing crowding, the intercellular forces and torques change, leading to a higher mean angular velocity. These observations suggest that physical interactions between cells, influenced by their density, play a crucial role in modulating their movement dynamics.

Figure 59 illustrates the mean angular velocity of cells under three different treatments: Control, TGF-beta, and SB431542. The control treatment exhibits the lowest angular velocity, indicating more stable or uniform cell rotation under standard conditions. In contrast, the TGF-beta treatment group shows significantly higher angular velocity, suggesting that TGF-beta enhances the motility and dynamic behaviour of the cells. The SB431542 treatment group demonstrates angular velocities higher than the control but slightly lower than the TGF-beta treatment, indicating a moderated effect on cell motility compared to TGF-beta. The results of the significance tests between the Control and TGF-Beta treatments reveal a notable difference in mean angular velocity, with a p-value of 0.00025, which is well below the conventional significance threshold of 0.05. However, the results of the significance test between the Control and SB431542 treatments indicate no statistically significant difference, with a p-value of 0.065, which is above the 0.05 threshold. Additionally, the difference in mean angular velocity



between the TGF-Beta and SB431542 treatments is significant, with a p-value of 0.018, confirming that this difference is statistically meaningful since it is below the 0.05 threshold. These tests highlight significant differences in mean angular velocities between the Control and TGF-Beta, as well as between the TGF-Beta and SB431542 groups. However, the Control and SB431542 groups do not show a statistically significant difference. This underscores the unique influence of TGF-Beta on mean angular velocity compared to the other treatments. Results from all the above figures indicate that cells under control conditions served as a baseline. Cells treated with TGF-beta demonstrated suppressed proliferation [285], increased cell migration [286], and enhanced angular velocity. Conversely, cells treated with SB431542 exhibited promoted cell growth [287]. Experimental variables, such as the selection of the field of view in time-lapse imaging, influenced the average cell growth observed with the SB431542 treatment. Additionally, studies have shown that higher concentrations of SB431542 might lead to off-target effects that could inhibit cell growth. Furthermore, it has been demonstrated that SB431542 can exert dose-dependent effects on various cell types [288] and inhibit cell migration and angular velocity [289].

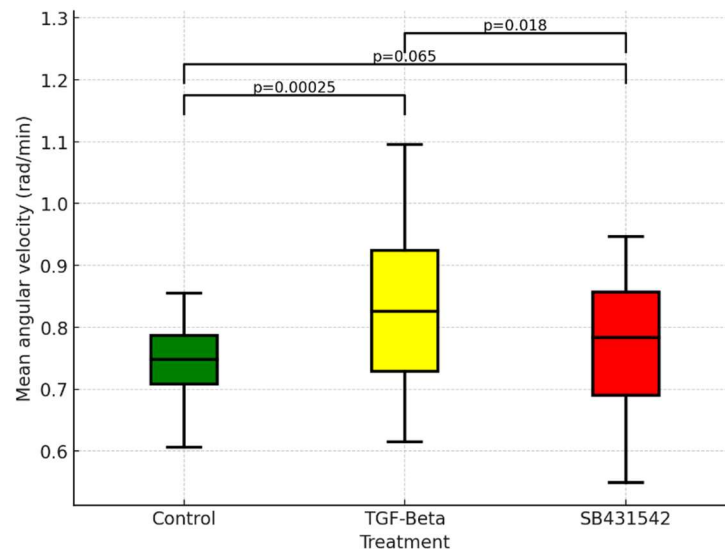


Figure 59: Mean Angular Velocity Comparison Across Different Treatments with Statistical Significance.

## 4.9 Machine Learning Classification

### 4.9.1 RCGP Pairwise with ensemble classification

This section presents the results for three pairwise classifications using distinct treatment datasets. The data from wells 0, 1, 2, and 3 represent the datasets for control

treatment, whilst the data from wells 4, 5, 6, and 7 are used for the TGF-Beta dataset and the data from wells 8, 9, 10, and 11 for the SB431542 dataset. The pairwise approach involves comparing two datasets to identify their differences or similarities. In this study, three pairwise classifications were conducted. The first classification, Pairwise 1, compares the dataset from the control condition with the dataset from the SB431542 treatment. The second classification, Pairwise 2, compares the control condition dataset with the TGF-Beta treatment dataset. Finally, the third classification, Pairwise 3, examines the dataset from the SB431542 treatment alongside the dataset from the TGF-Beta treatment.. Additionally, the results from the three pairwise comparisons (Pairwise 1, Pairwise 2, and Pairwise 3) were combined using an ensemble approach. In this approach, ensemble techniques such as Bagging, Boosting, and Stacking were applied to enhance the overall classification performance. The selection of the final ensemble method was determined based on which technique achieved the highest test accuracy. The detailed architecture of how the pairwise classifications were implemented and how the ensemble method was applied is illustrated in Figure 33.

#### 4.9.1.1 Pairwise Classification 1: Cross-Validation results for dataset with Control and SB431542 Treatment

The results for Pairwise Classification 1, utilising a dataset subjected to control and SB431542 treatment. Data were analysed across three different configurations: 5 nodes (node 5), 15 nodes (node 15), and 30 nodes (node 30). The boxplot and the summary table, labelled as Figure 60 and Table 35, respectively, provide a thorough analysis of the 5-fold cross-validation accuracy for each node configuration. The boxplot for the 5-node configuration

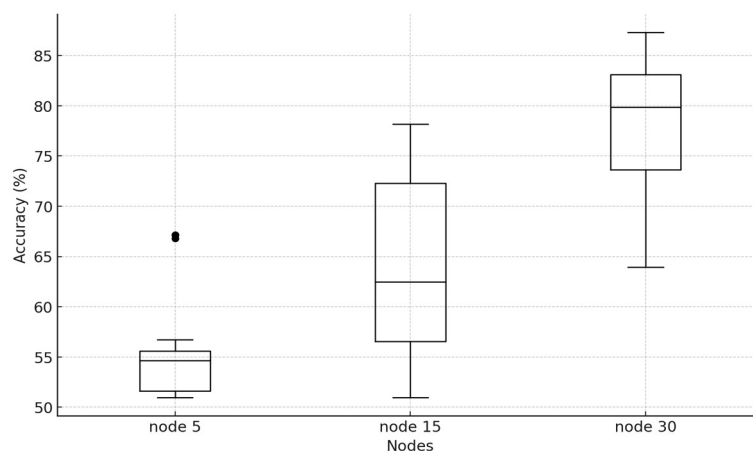


Figure 60: 5-Fold CV Accuracy (%) for Different Nodes - Pairwise Classification 1.

reveals the lowest median accuracy and the narrowest interquartile range (IQR). The findings echo the summary table, which indicates a mean accuracy of 55.07% and a standard deviation of 4.47. Conversely, the 15-node configuration displays a higher median accuracy and a wider IQR, with the table documenting a mean accuracy of 63.79% and a standard deviation of 8.91. The boxplot for the 30-node configuration illustrates the highest median accuracy, supported by the table, which reports the highest mean accuracy of 78.17% and a standard deviation of 6.54. The observed trend of increasing accuracy with a more significant number of nodes suggests that the model's predictive capabilities are enhanced by expanding the node count. The enhancement, particularly evident in the 30-node configuration, likely reflects advancements in the model's parameters and optimised settings, resulting in the most robust performance.

Table 35: Summary Statistics for 5-Fold CV Accuracy (%) - Pairwise Classification 1.

| Statistic          | Node 5 | Node 15 | Node 30 |
|--------------------|--------|---------|---------|
| Mean accuracy (%)  | 55.07  | 63.79   | 78.17   |
| Standard deviation | 4.47   | 8.91    | 6.54    |

#### 4.9.1.2 Pairwise classification 2: Cross validation result for dataset with control and TGF-Beta treatment

The findings from Pairwise Classification 2, which involved datasets from control and TGF-Beta treatment, were assessed using three different node configurations: 5 nodes (node

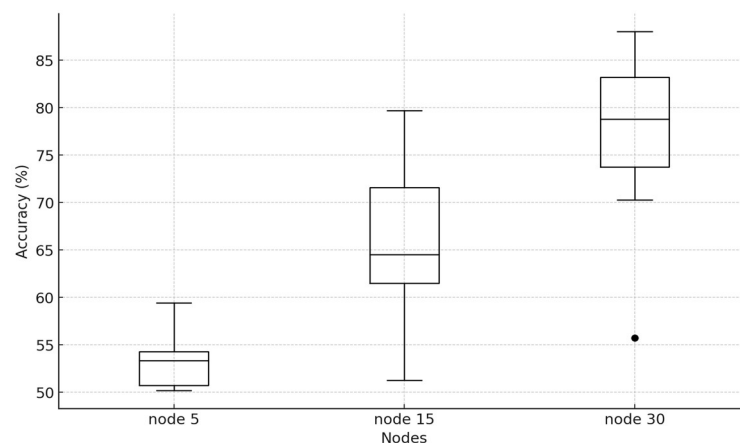


Figure 61: 5-Fold CV Accuracy (%) for Different Nodes - Pairwise Classification 2.

5), 15 nodes (node 15), and 30 nodes (node 30). Both the boxplot, shown as Figure 61, and the summary table, noted as Table 36, detailed the 5-fold cross-validation accuracy for each configuration. For the configuration with 5 nodes, the boxplot revealed the lowest median accuracy alongside the narrowest interquartile range (IQR), a detail that aligns with the summary table showing a mean accuracy of 52.92% and a standard deviation of 2.23. The configuration with 15 nodes showed a notable increase in median accuracy and a broader IQR, with reported mean accuracy and standard deviation of 66.46% and 7.54, respectively. The configuration encompassing 30 nodes displayed the highest median accuracy, which the summary table supports, noting the highest mean accuracy at 78.28% and a standard deviation of 6.78. This increasing trend in accuracy with adding more nodes suggests an enhancement in the model’s predictive efficiency. The most notable improvements seen in the 30-node configuration are likely due to refined model parameters and optimised settings, contributing to better performance.

Table 36: Summary Statistics for 5-Fold CV Accuracy (%) - Pairwise Classification 2.

| Statistic          | Node 5 | Node 15 | Node 30 |
|--------------------|--------|---------|---------|
| Mean accuracy (%)  | 52.92  | 66.46   | 78.28   |
| Standard deviation | 2.23   | 7.54    | 6.78    |

#### 4.9.1.3 Pairwise classification 3: Cross validation result for dataset with SB431542 and TGF-Beta Treatment

The results from Pairwise Classification 3, involving datasets treated with SB431542 and TGF-Beta, were evaluated using three different node configurations: 5 nodes (node 5), 15 nodes (node 15), and 30 nodes (node 30). Both the boxplot, displayed as Figure 62, and the summary table, referred to as Table 37, outlined the 5-fold cross-validation accuracy for each configuration. For the 5-node configuration, the boxplot showed the lowest median accuracy along with the narrowest interquartile range (IQR), corresponding with the summary table, which recorded a mean accuracy of 52.77% and a standard deviation of 1.21. The 15-node configuration indicated a moderate increase in median accuracy and a broader IQR, with a mean accuracy of 60.64% and a standard deviation of 5.55 reported. The 30-node configuration exhibited the highest median accuracy, which was supported by the summary table, showing the highest mean accuracy of 67.97% and a standard deviation of 5.54. The increasing accuracy with more nodes suggests the model becomes more efficient as it grows. The significant

improvements seen in the 30-node configuration are likely due to refined model parameters and optimised settings, leading to better performance.

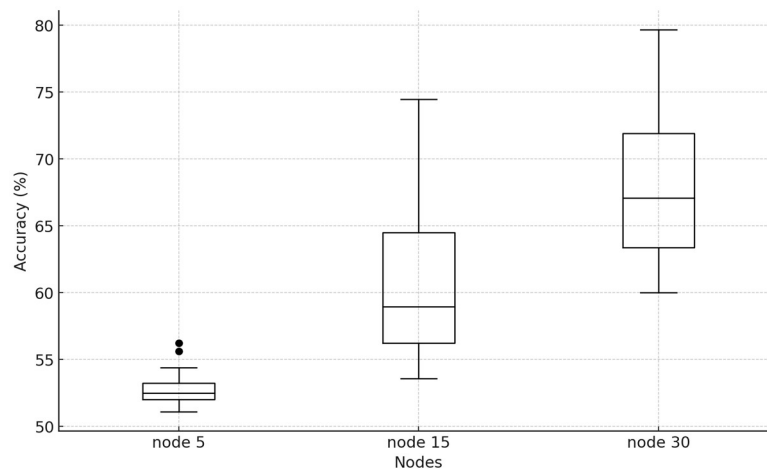


Figure 62:5-Fold CV Accuracy (%) for Different Nodes - Pairwise Classification 3.

Table 37: Summary Statistics for 5-Fold CV Accuracy (%) - Pairwise Classification 3.

| Statistic          | Node 5 | Node 15 | Node 30 |
|--------------------|--------|---------|---------|
| Mean accuracy (%)  | 52.77  | 60.64   | 67.97   |
| Standard deviation | 1.21   | 5.55    | 5.54    |

#### 4.9.1.4 Test result Pairwise classification

The test accuracy percentages outlined in Table 38 for three distinct pairwise classifications showcase the performance across different datasets and node configurations.

Table 38: Test Accuracy (%) for pairwise classification.

| Pairwise classification | Test accuracy (%) |
|-------------------------|-------------------|
| Pairwise 1              | 80.76             |
| Pairwise 2              | 79.17             |
| Pairwise 3              | 73.46             |

The table provides a comprehensive analysis of the results from each classification, emphasising the node configurations that produced the best outcomes. In Pairwise 1, the dataset

treated with control and SB431542 was analysed using a 30-node configuration, which proved the most effective for this set. This classification achieved the highest test accuracy of 80.76%. For Pairwise 2, the dataset involving treatments with control and TGF-Beta was also analysed under a 30-node configuration. It recorded a test accuracy of 79.17%, slightly lower than that of Pairwise 1. Pairwise 3, which included datasets treated with both SB431542 and TGF-Beta, used the same 30-node configuration, resulting in the lowest test accuracy at 73.46%. The reduced accuracy in this classification, compared to the others, suggests an increased complexity arising from the combination of both treatments. This complexity might have led to increased variability or overlapping features within the dataset, making accurate classification more challenging. These findings indicate that while a 30-node configuration was consistently the best choice across the classifications, the varying levels of accuracy highlight the importance of aligning the model configuration with each dataset's specific challenges and characteristics to optimise outcomes.

The selection of a 30-node limit for the RCGP architecture was strategically determined based on accuracy metrics derived from test datasets and the structural efficacy of the final network model.

Table 39: Comparison Test Accuracy (%) for different node configurations.

| RCGP architecture | Test Accuracy | Computational time | Network diagram model |
|-------------------|---------------|--------------------|-----------------------|
| 30 nodes          | 80.76         | ≈12 hours          | 19 nodes              |
| 60 nodes          | 81.52         | ≈24 hours          | 39 nodes              |

The data in Table 39 indicate that the 30-node configuration of the Recurrent Cartesian Genetic Programming (RCGP) architecture provides an optimal balance between computational efficiency and accuracy. The test accuracy for the 30-node model, derived from pairwise 1 classification, is 80.76%, slightly lower than the 81.52% achieved with 60 nodes. However, this marginal improvement in accuracy is offset by the significant increase in computational time, which doubles from approximately 12 hours to 24 hours. Furthermore, the complexity of the network diagram increases from 19 nodes in the 30-node model to 39 nodes in the 60-node model, further contributing to the higher computational demands. Therefore, the 30-node

configuration is preferable, offering a more efficient and manageable solution without compromising significantly on accuracy.

#### 4.9.1.5 Ensemble Classification

##### 4.9.1.5.1 Bagging

The provided visual data and table, labelled as Figure 63 and Table 40, respectively, illustrate the performance metrics of a bagging ensemble method using Random Forest for

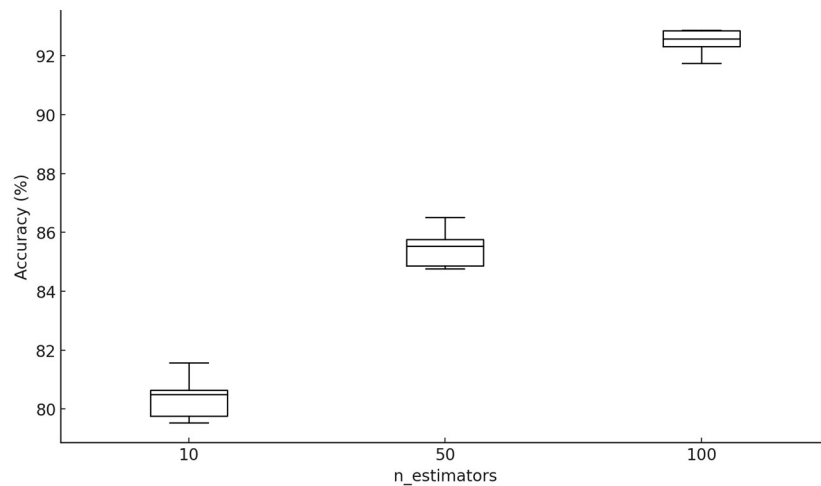


Figure 63:5-Fold CV Accuracy (%) for different estimator value - bagging method.

classification tasks across three settings for the number of estimators: 10, 50, and 100. These components detail how classification accuracy improves as the number of estimators increases. In Figure 63, the Random Forest ensemble with 10 estimators displays the lowest accuracy, with the median just above 80% and a narrow spread, indicative of consistent but modest performance. Further substantiation comes from Table 40, which shows a mean accuracy of 80.41% with a standard deviation of 0.82 for this setting.

Table 40:Summary Statistics for 5-Fold CV Accuracy (%) for bagging method.

| Parameter setting | Mean accuracy | Standard deviation |
|-------------------|---------------|--------------------|
| n_estimators=10   | 80.41         | 0.82               |
| n_estimators=50   | 85.90         | 0.72               |
| n_estimators=100  | 92.48         | 0.45               |

As the number of estimators increases to 50, there is a noticeable rise in median accuracy to around 85%, accompanied by a slightly reduced interquartile range (IQR), suggesting improved performance with less variability. The table confirms this with a mean accuracy of 85.90% and a reduced standard deviation of 0.72. The configuration with 100 estimators exhibits the highest accuracy, with the median nearing 93% and the tightest IQR, highlighting the best performance and greatest consistency among trials. Correspondingly, Table 40 shows that 100 estimators achieve the highest mean accuracy of 92.48% with the lowest standard deviation of 0.45, firmly confirming optimal performance and stability observed in the boxplot. Data from the study demonstrates a positive correlation between the number of Random Forest bagging ensemble estimators and classification accuracy. Increasing the number of estimators enhances the model's generalisation capabilities. Decreasing standard deviation values as the number of estimators increases suggests that more estimators not only boost accuracy but also enhance the stability and consistency of the model's predictions across different subsamples of the data. The observed trend underscores the effectiveness of investing in more complex models with additional estimators, which can yield substantial improvements in predictive accuracy. For practical applications, opting for a higher number of estimators in Random Forest configurations can significantly enhance outcomes, especially in scenarios that demand high reliability and precision in classification tasks.

#### 4.9.1.5.2 Boosting

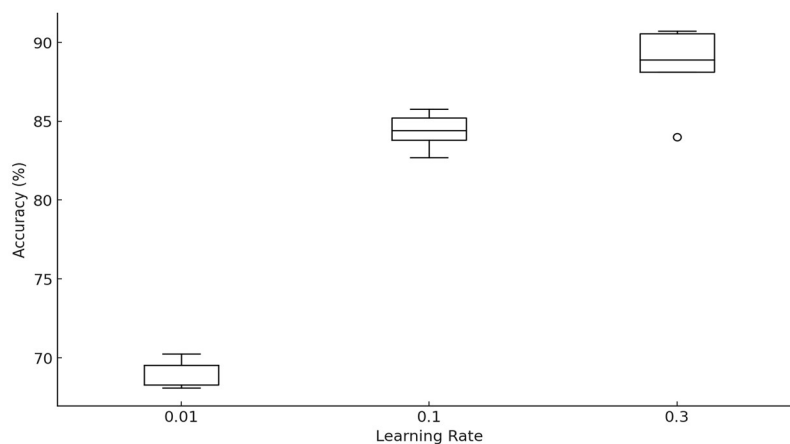


Figure 64:5-Fold CV Accuracy (%) for different estimator value - boosting method.



Table 41: Summary Statistics for 5-Fold CV Accuracy (%) for boosting method.

| Parameter setting  | Mean accuracy | Standard deviation |
|--------------------|---------------|--------------------|
| Learning rate=0.01 | 69.53         | 0.74               |
| Learning rate=0.1  | 84.38         | 1.18               |
| Learning rate=0.3  | 88.85         | 2.83               |

The visual data and table, labelled as Figure 64 and Table 41, respectively, showcase the performance metrics of an XGBoost boosting ensemble method evaluated at different learning rates (0.01, 0.1, and 0.3) based on 5-fold cross-validation accuracy. Figure 64 boxplot at a learning rate of 0.01 shows the lowest accuracy, with the median just above 70% and a narrow interquartile range (IQR), indicative of consistent yet modest performance. Table 41 records a mean accuracy of 69.53% with a standard deviation of 0.74 at this learning rate, aligning with the boxplot's indicated performance. As the learning rate increases to 0.1, the median accuracy in the boxplot rises significantly to around 85%, and the IQR broadens slightly, suggesting improved performance with a moderate increase in variability; the table confirms this with a mean accuracy of 84.38% and a standard deviation of 1.18. At the highest learning rate of 0.3, the boxplot reveals the best performance, with the median accuracy nearing 90% and the widest IQR, reflecting the highest variability and the strongest overall accuracy. Table 41 supports the findings, indicating that the mean accuracy at this learning rate peaks at 88.85% with a standard deviation of 2.83, confirming optimal performance and more significant variability. Data show that higher learning rates in the XGBoost method correlate positively with increased classification accuracy. However, such increases in accuracy also lead to greater variability in results, as evidenced by expanding interquartile ranges and rising standard deviations with higher learning rates. This pattern illustrates a clear trade-off between learning rate and performance in XGBoost boosting methods. While higher learning rates enhance model accuracy, they also introduce more variability in the model's predictions. Thus, carefully managing learning rates is crucial, particularly in scenarios that require consistent and robust performance. Adjusting the learning rate to match the specific demands and tolerance for the variability of the task can significantly enhance outcomes in predictive tasks utilising XGBoost.

### 4.9.1.5.3 Stacking

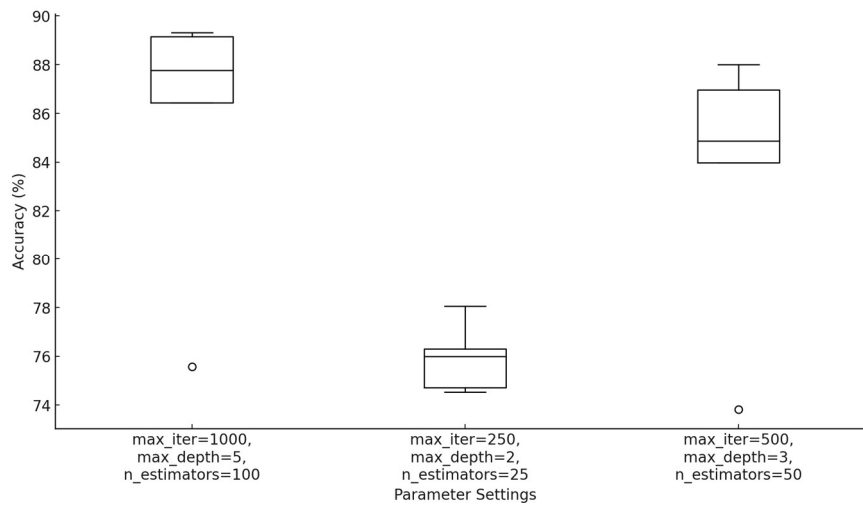


Figure 65: 5-Fold CV Accuracy (%) for different estimator value - stacking method.

Table 42: Summary Statistics for 5-Fold CV Accuracy (%) for stacking method

| Parameter setting                            | Mean accuracy | Standard deviation |
|--|---------------|--------------------|
| max_iter=250, max_depth=2, n_estimators=25   | 76.56         | 1.59               |
| max_iter=500, max_depth=3, n_estimators=50   | 84.26         | 5.94               |
| max_iter=1000, max_depth=5, n_estimators=100 | 85.65         | 5.70               |

The provided visual data and table, labelled as Figure 65 and Table 42, illustrate the performance metrics of a stacking ensemble method that integrates three different algorithms: Logistic Regression, Decision Tree Classifier, and Random Forest Classifier. This method evaluates the impact of tuning specific parameters—max\_iter for Logistic Regression, max\_depth for Decision Tree Classifier, and n\_estimators for Random Forest Classifier—on 5-fold cross-validation accuracy. The boxplot in Figure 65 displays accuracy results for three configurations. The first configuration with max\_iter=250, combined with max\_depth=2 and n\_estimators=25, shows the lowest median accuracy slightly below 80% and a narrow interquartile range (IQR), indicative of consistent but modest performance. Increasing the parameters to max\_iter=500, max\_depth=3, and n\_estimators=50 results in a median accuracy of about 85% and a broader IQR, reflecting improved performance with a moderate increase

in variability. The third configuration maximizes the parameters to  $\text{max\_iter}=1000$ ,  $\text{max\_depth}=5$ , and  $\text{n\_estimators}=100$ , achieving the highest median accuracy around 86% with an IQR similar to the second setting, suggesting the best and most consistent performance.

Corresponding numerical data in Table 42 supports these observations: the initial parameter set ( $\text{max\_iter}=250$ ,  $\text{max\_depth}=2$ ,  $\text{n\_estimators}=25$ ) achieves a mean accuracy of 76.56% with a standard deviation of 1.59, reflecting the lowest performance. The second set ( $\text{max\_iter}=500$ ,  $\text{max\_depth}=3$ ,  $\text{n\_estimators}=50$ ) shows an increased mean accuracy of 84.26% and a higher standard deviation of 5.94, indicating enhanced performance. The most advanced parameter configuration ( $\text{max\_iter}=1000$ ,  $\text{max\_depth}=5$ ,  $\text{n\_estimators}=100$ ) records the highest mean accuracy at 85.65% with a standard deviation of 5.70, confirming the optimal performance indicated in the boxplot. These findings suggest that enhancing the parameters of  $\text{max\_iter}$  for Logistic Regression,  $\text{max\_depth}$  for Decision Tree Classifier, and  $\text{n\_estimators}$  for Random Forest Classifier within a stacking ensemble method can significantly improve classification accuracy. As these parameters increase, the models more effectively capture complex patterns in the data, leading to higher accuracy. However, this also introduces variability in the results, as seen in the expanding IQRs and rising standard deviations.

#### 4.9.1.5.4 Test result Ensemble classification

The test accuracy percentages presented in Table 43 for three distinct ensemble classification methods, Bagging, Boosting, and Stacking, illustrate their performance across various ensemble strategies. The table provides a comprehensive analysis of each method's results, emphasising each's effectiveness in handling classification tasks.

Table 43: Test Accuracy (%) for ensemble classification.

| Ensemble classification | Test accuracy (%) |
|-------------------------|-------------------|
| Bagging                 | 88.59             |
| Boosting                | 85.34             |
| Stacking                | 82.15             |

Bagging classification, employing Random Forests techniques to aggregate multiple decision trees, achieved the highest test accuracy at 88.59%. The performance highlights Bagging's strength in reducing variance and delivering robust predictions by averaging several deep and possibly overfit trees. Boosting classification, specifically using the XGBoost algorithm, recorded a test accuracy of 85.34%. Although slightly lower than Bagging, XGBoost excels at addressing challenging cases by incrementally building the ensemble and adjusting the weight of instances, thus enhancing the ensemble's accuracy with each iteration. Stacking classification involves a combination of Logistic Regression, Decision Tree Classifier, and Random Forest Classifier to enhance the final prediction accuracy and yield the lowest test accuracy at 82.15%. The outcome may indicate challenges in effectively blending the outputs of diverse classifiers or fine-tuning the meta-classifier that integrates these outputs, potentially leading to reduced efficacy compared to other ensemble strategies. Overall, Bagging was the most effective strategy in this analysis, providing the highest classification accuracy.

#### 4.9.1.6 RCGP pairwise classification model

The computational network graph, shown in Figure 66, displays a model with three input data points on cell growth, migration speed, and angular velocity to produce two binary outputs, "Output 0" and "Output 1." These outputs classify biological conditions into two categories: Control and SB431542, based on specific combinations of the binary results:

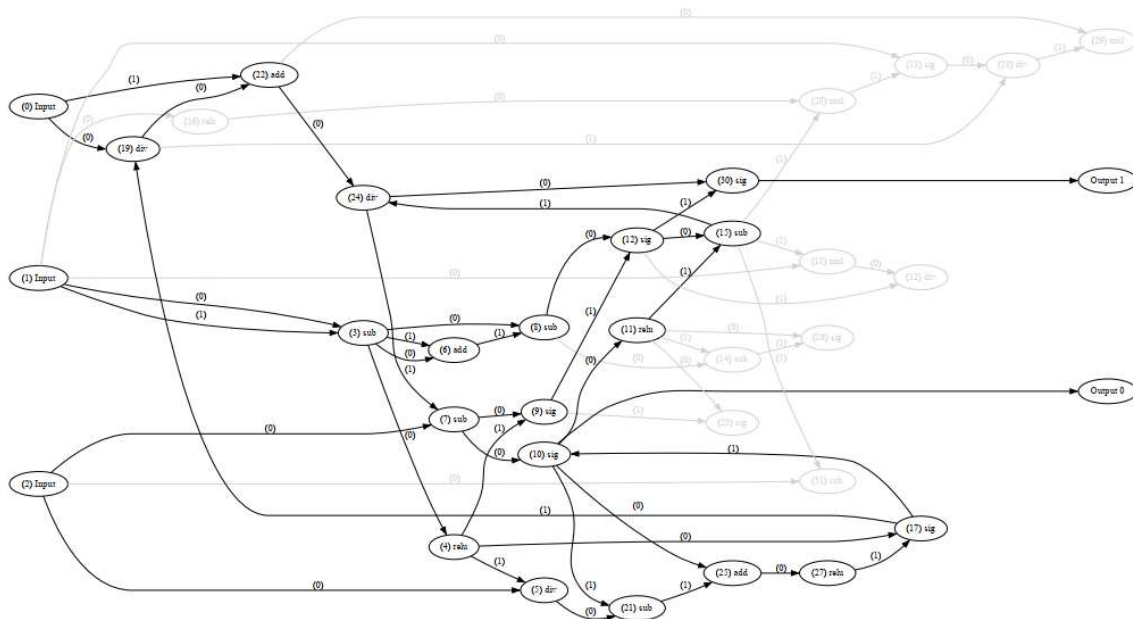


Figure 66: Model classification for dataset control and SB431542 classification.

Control (0,1) and SB431542 (1,0). The model consists of 19 nodes, not including the three primary input nodes and two output nodes. These nodes are engaged in performing a variety of mathematical operations that manipulate the data as it moves through the model. Inactive components not used in the model are shown in grey, while the active parts of the optimised model are depicted in black. This distinction helps identify which elements of the computational graph actively contribute to the outputs, illustrating an efficient data flow through key pathways. Data processing begins with three specific inputs: cell growth ((0) input), migration speed ((1) input), and angular velocity ((2) input). Each input undergoes addition, subtraction, multiplication, division, and is processed by sigmoid functions, which crucially integrate the inputs to reflect underlying biological processes or interactions. In binary classification, the sigmoid functions are employed to map any real-valued number into the 0 to 1 range. This function is vital for generating probabilities and determining whether a specific condition or classification is met. The output configurations provide clear classifications: "Control" is indicated by an output combination of (0,1), suggesting typical or expected behaviour under normal conditions, while the combination of (1,0) indicates conditions influenced by SB431542, signifying a specific cellular response or behaviour due to the treatment. The model offers a robust framework for conducting detailed biological analyses and classifying cellular behaviours under control or SB431542 treatment by methodically mapping complex inputs through calculated operations and using sigmoid functions for precise classification.

Figure 67 presents a computational network graph incorporating three input variables: cell growth, migration speed, and angular velocity. This model generates two binary outputs, "Output 0" and "Output 1," which are utilised to categorise biological conditions into Control and TGF-beta. The binary combinations of Control (0,1) and TGF-beta (1,0) determine the classifications. The model's architecture includes 14 nodes, excluding the three primary input and two output nodes. These nodes execute various mathematical operations that transform and manipulate the data throughout the model. Inactive parts of the model are coloured grey, highlighting non-utilised components, whereas active sections that are part of the optimised model are coloured black. This colour coding is essential for delineating which components of the computational graph are integral to the outputs, showcasing an effective data flow through essential pathways. The process begins with the three inputs mentioned above: cell growth ((0)

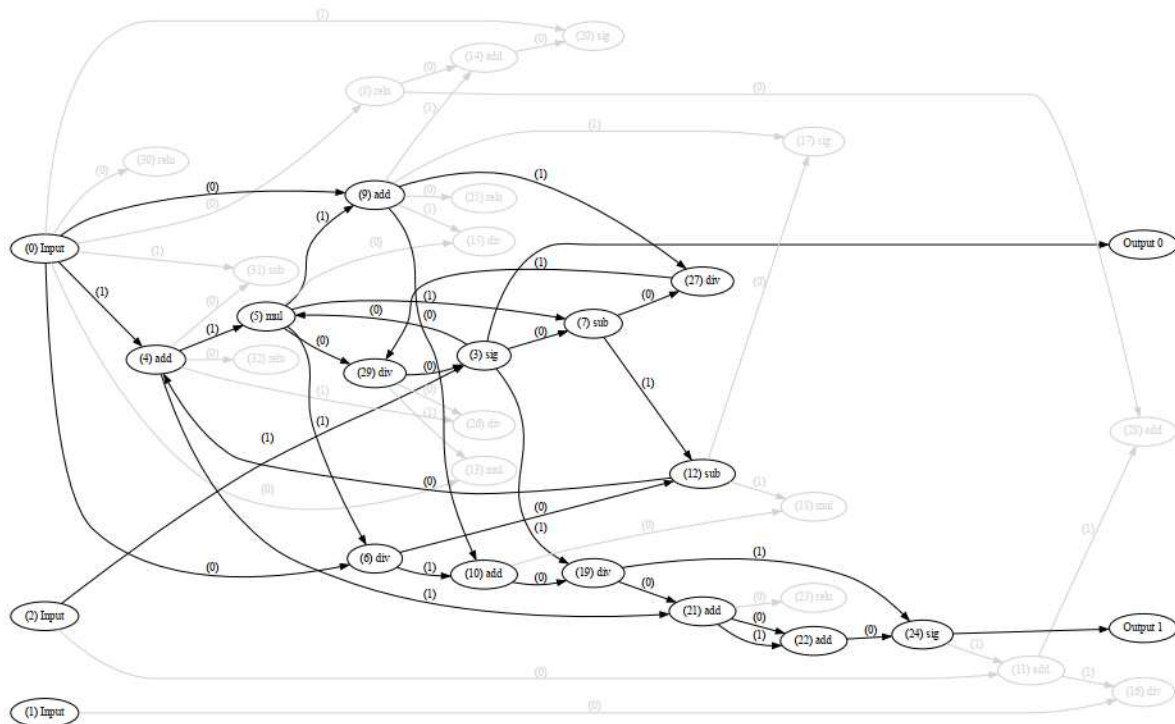


Figure 67: Model classification for dataset control and TGF-Beta classification.

input), migration speed ((1) input), and angular velocity ((2) input). Operations, including addition, subtraction, multiplication, division, and processing through sigmoid functions, are applied to these inputs. These functions are critical as they integrate the data to mirror underlying biological processes or interactions. The sigmoid functions are essential in binary classification, where they convert any real-valued number into a 0 to 1 range, crucial for calculating probabilities and establishing whether a specific condition or classification is achieved. The outputs are distinctly defined: "Control" is signified by an output combination of (0,1), indicating typical or expected behaviour under normal conditions. Conversely, the combination of (1,0) represents conditions affected by TGF-beta, denoting a particular cellular response or behaviour resulting from the treatment. This model provides a solid framework for classifying cellular behaviours under conditions of Control or TGF-beta influence by methodically organising complex inputs through strategic operations and utilising sigmoid functions for accurate classification.

Figure 68 presents a computational network graph incorporating three input variables: cell growth, migration speed, and angular velocity. This model generates two binary outputs, "Output 0" and "Output 1," which are utilised to categorise biological conditions into SB431542 and TGF-beta. The binary combinations SB431542 (0,1) and TGF-beta (1,0) determine the classifications. The model's architecture includes 22 nodes, excluding the three

primary input and two output nodes. These nodes execute various mathematical operations that transform and manipulate the data throughout the model. Inactive parts of the model are coloured grey, highlighting non-utilised components, whereas active sections that are part of the optimised model are coloured black. The process begins with the three mentioned inputs: cell growth ((0) input), migration speed ((1) input), and angular velocity ((2) input). Operations, including addition, subtraction, multiplication, division, and processing through the sigmoid and ReLU functions, are applied to these inputs. The inclusion of ReLU functions, which are used to introduce non-linearity without affecting the scale of the input, is critical as they further refine the data. Meanwhile, sigmoid functions are essential in binary classification, where they convert any real-valued number into a 0 to 1 range, crucial for calculating probabilities and establishing whether a specific condition or classification is achieved. The outputs are distinctly defined: "SB431542" is signified by an output combination of (0,1), indicating typical or expected behaviour under conditions influenced by SB431542. Conversely, the combination of (1,0) represents conditions affected by TGF-beta, denoting a specific cellular response or behaviour resulting from the treatment. This model provides a robust framework for classifying cellular behaviours under conditions influenced by SB431542 or TGF-beta by methodically organising complex inputs through strategic operations and utilising sigmoid and ReLU functions for accurate classification.

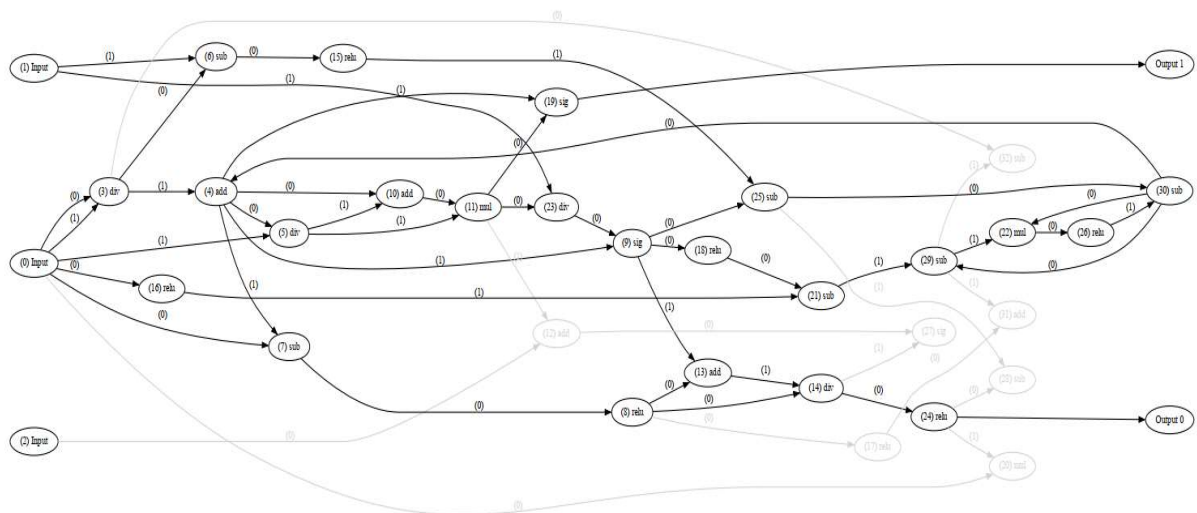


Figure 68: Model classification for dataset SB431542 and TGF-Beta classification.

#### 4.9.2 PySR Regression with classification

This section presents the results of symbolic regression and classification using the PySR genetic programming approach across distinct treatment datasets. The datasets for the control treatment are represented by data from wells 0, 1, 2, and 3. Data from wells 4, 5, 6, and 7 are used for the TGF-Beta dataset, and data from wells 8, 9, 10, and 11 for the SB431542 dataset. Symbolic regression will be performed for each well, acknowledging the inherent heterogeneity within biological systems. Each well in a multi-well plate may experience slight variations in conditions such as cell density, microenvironment, and reagent exposure, leading to distinct behaviours or responses. By analysing each well separately, symbolic regression captures these unique characteristics, resulting in more accurate models that reflect the specific conditions of each well. This approach also facilitates the discovery of mathematical models that describe the relationships between variables. Independent analysis of each well uncovers specific patterns that might be lost in aggregated data, providing a more detailed understanding of the underlying biological processes. This level of detail is crucial for accurate modelling in complex systems.

Biologically, variability is a fundamental aspect of experiments, arising from factors such as biological noise and differences in cell populations. Treating each well as an independent micro-experiment allows us to understand how these variables uniquely influence the system, offering insights into the biological variability and robustness of observed phenomena. Moreover, it enables the discovery of well-specific phenomena that might be obscured in a pooled analysis, potentially leading to new hypotheses for further investigation. Analysing individual wells preserves important information that might be lost through data aggregation. Aggregating data can mask subtle yet significant differences between wells. By analysing each well independently, we maintain the integrity and richness of the data, leading to more tailored models that better capture the system's dynamics and enhance our understanding of the biological processes involved. In this analysis, the input features for each symbolic regression include time, migration speed, and angular velocity. The output of each symbolic regression, representing cell growth, will be combined into a symbolic classification model that provides the final predictions for treatment conditions. Symbolic classification will then be applied to all treatments, ensuring that the diverse responses to treatments are accurately captured and modelled, ultimately allowing for a more precise prediction of treatment outcomes across different conditions.



### 4.9.2.1 Symbolic regression

This section presents the results and analysis of a regression model designed to predict cell growth. The analysis includes symbolic regression for well 0, which represents the control group; well 4, associated with TGF- $\beta$  treatment; and well 8, corresponding to the SB431542 treatment. Detailed results for additional wells are provided in Appendix A (Figure A.19 to Figure A.36). The effectiveness of the regression model is further illustrated in Figures 69 and 70, and Table 44. Specifically, Figure 85 features a series of box plots that demonstrate the coefficient of determination ( $R^2$ ) across various parameter settings in a 5-fold cross-validation setup for Well 0. The settings range from 5 populations with 30 iterations to 15 populations with 150 iterations, demonstrating that increased model complexity generally improves predictive accuracy, as evidenced by rising median  $R^2$  values.

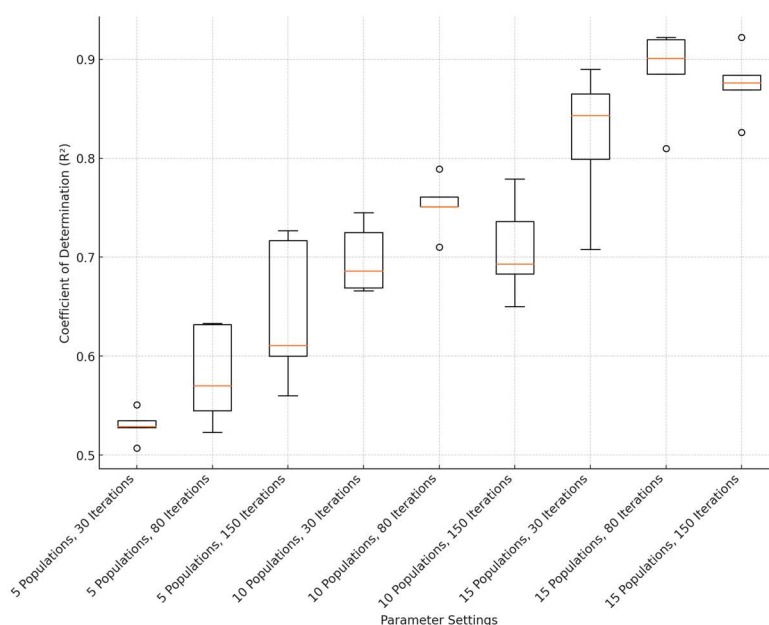


Figure 69: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 0.

Table 44: Summary Statistics for 5-Fold CV  $R^2$  Mean for Well 0 Symbolic Regression.

| Parameter setting             | $R^2$ Mean | Standard deviation |
|-------------------------------|------------|--------------------|
| 5 Populations, 30 Iterations  | 0.529      | 0.0022             |
| 5 Populations, 80 Iterations  | 0.583      | 0.0174             |
| 5 Populations, 150 Iterations | 0.654      | 0.0206             |

| Parameter setting              | R <sup>2</sup> Mean | Standard deviation |
|--------------------------------|---------------------|--------------------|
| 5 Populations, 30 Iterations   | 0.529               | 0.0022             |
| 10 Populations, 30 Iterations  | 0.695               | 0.0088             |
| 10 Populations, 80 Iterations  | 0.757               | 0.0024             |
| 10 Populations, 150 Iterations | 0.705               | 0.0078             |
| 15 Populations, 30 Iterations  | 0.835               | 0.0131             |
| 15 Populations, 80 Iterations  | 0.889               | 0.0118             |
| 15 Populations, 150 Iterations | 0.884               | 0.0045             |

Table 44 provides a numerical complement to the visual data, offering detailed summaries of the mean R<sup>2</sup> values and their standard deviations for the parameter configurations. The table confirms the trends observed in the box plots and identifies the optimal setting of 15 populations and 80 iterations as having the highest mean R<sup>2</sup> value of 0.889 and standard

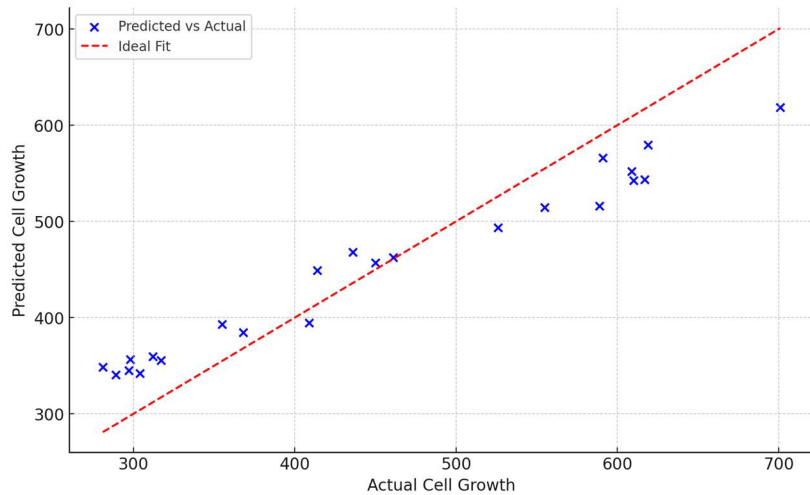


Figure 70: Actual Vs Predicted Growth On Test Data (R<sup>2</sup> = 0.870).

deviation of 0.0118. Such findings suggest that this configuration offers the best accuracy. In Figure 70, a scatter plot is presented that compares actual versus predicted cell growth using a model configured with 15 populations and 80 iterations. The application of this model to the test dataset yields an R<sup>2</sup> value of 0.870, demonstrating robust predictive performance.

The box plot (Figure 71) and summary statistics table (Table 45) provide a detailed analysis of the coefficient of determination (R<sup>2</sup>) for various parameter settings in a symbolic regression model using 5-fold cross-validation with a focus on data from Well 4. An upward

trend in  $R^2$  values is observed as the number of populations and iterations increases, reflecting enhanced model performance. For instance, increasing from 5 populations and 30 iterations ( $R^2=0.493$ ) to 15 populations and 150 iterations ( $R^2=0.880$ ) shows a significant improvement. The decrease in variability, represented by the size of the boxes and the length of the whiskers, suggests more consistent performance with higher populations and iterations. Minimal outliers in the box plot further indicate consistent cross-validation results. As shown in the summary statistics table, standard deviation values generally decrease with higher populations and iterations, indicating increased consistency in performance. The model exhibits the lowest performance at 5 populations and 30 iterations with a mean  $R^2$  of 0.493 and a standard deviation of 0.0194. Performance improves with 80 and 150 iterations, reaching mean  $R^2$  values of 0.533 and 0.569, respectively. For 10 populations and 30 iterations, performance increases to  $R^2=0.643$ . With 80 and 150 iterations, mean  $R^2$  values are 0.699 and 0.746, respectively. The standard deviation remains low, indicating consistent model performance. At 15 populations and 30 iterations, the mean  $R^2$  is 0.819. Increasing iterations to 80 and 150 further improves the mean  $R^2$  values to 0.843 and 0.880, respectively, with low standard deviation indicating high consistency.

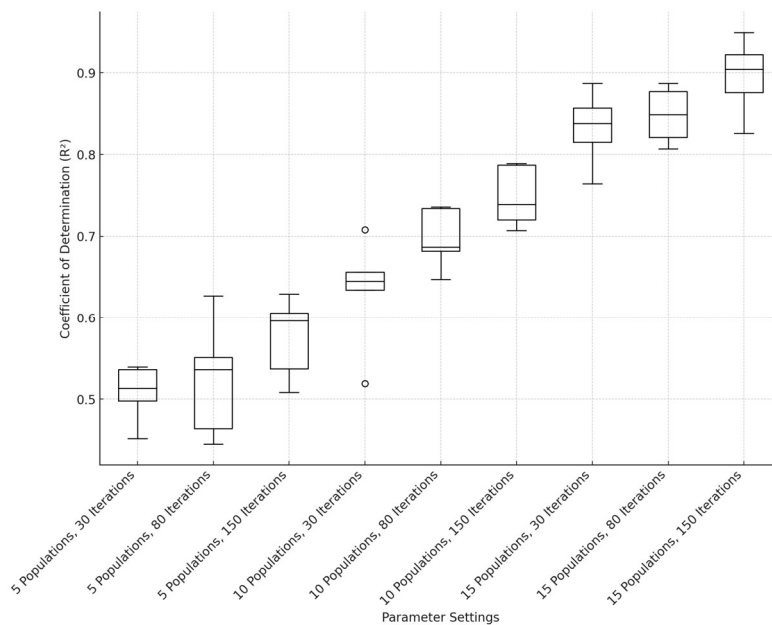


Figure 71: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 4.

Table 45: Summary Statistics for 5-Fold CV R<sup>2</sup> Mean for Well4 Symbolic Regression.

| Parameter setting              | R <sup>2</sup> Mean | Standard deviation |
|--------------------------------|---------------------|--------------------|
| 5 Populations, 30 Iterations   | 0.493               | 0.0194             |
| 5 Populations, 80 Iterations   | 0.533               | 0.0384             |
| 5 Populations, 150 Iterations  | 0.569               | 0.0367             |
| 10 Populations, 30 Iterations  | 0.643               | 0.0092             |
| 10 Populations, 80 Iterations  | 0.699               | 0.0198             |
| 10 Populations, 150 Iterations | 0.746               | 0.0174             |
| 15 Populations, 30 Iterations  | 0.819               | 0.0264             |
| 15 Populations, 80 Iterations  | 0.843               | 0.0157             |
| 15 Populations, 150 Iterations | 0.880               | 0.0241             |

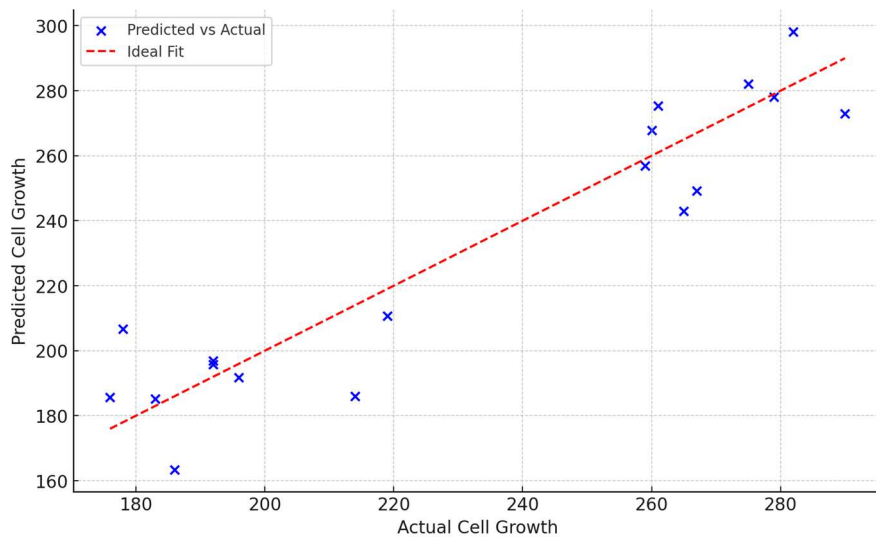


Figure 72: Actual Vs Predicted Growth On Test Data (R<sup>2</sup> = 0.867).

Figure 72 presents the actual versus predicted cell growth on test data, with an R<sup>2</sup> value of 0.867. The scatter plot shows predicted values plotted against actual values, with the ideal

fit line (dashed red) representing perfect predictions. Most data points closely align with the ideal fit line, indicating a strong correlation between predicted and actual values. The high  $R^2$  value of 0.867 on test data confirms the cross-validation findings, affirming the model's reliability and accuracy. The test data scatter plot demonstrates that the model performs well in predicting cell growth. Points closely follow the ideal fit line, indicating accurate predictions. Optimal parameter settings, identified as 15 populations and 150 iterations, result in the best performance, yielding the highest mean  $R^2$  value of 0.880 and a low standard deviation. The test data results, with an  $R^2$  value of 0.867, validate the model's effectiveness, assuring its suitability for applicable applications.

The performance of a symbolic regression model under various parameter configurations was evaluated using 5-fold cross-validation, with results for well 8 displayed in Figure 73 (box plot) and Table 46 (summary statistics). The coefficient of determination ( $R^2$ ) exhibits a positive correlation with increasing populations and iterations, suggesting enhanced model performance. Specifically,  $R^2$  values improve from 0.511 to 0.910 as populations increase from 5 to 15 and iterations from 30 to 150. The box plot reveals decreasing variability in  $R^2$  values, evidenced by smaller boxes and shorter whiskers, as population and iteration numbers increase. This trend indicates more consistent performance across folds.

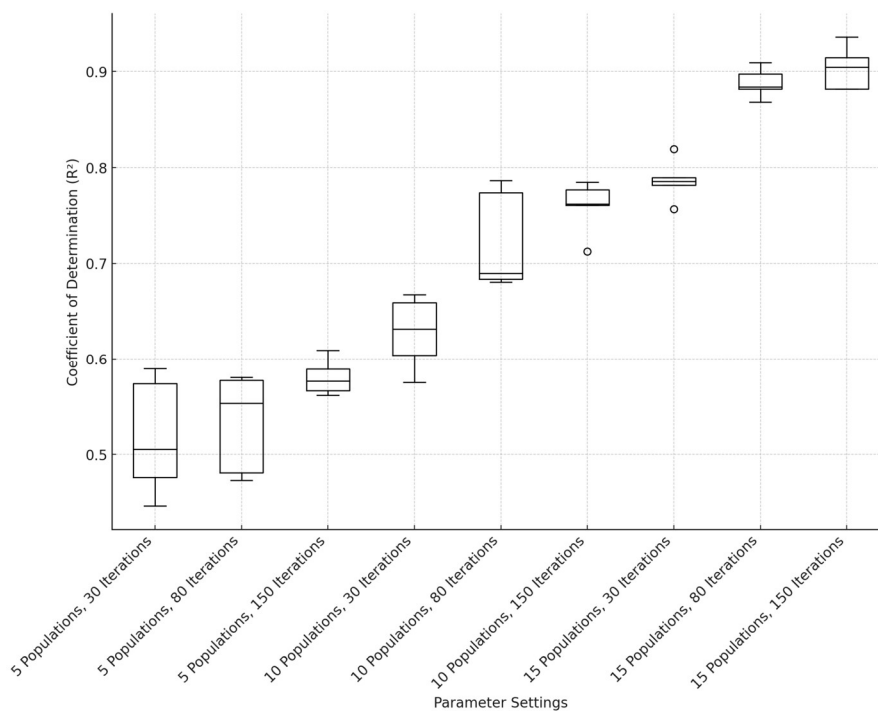


Figure 73: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 8.

Table 46: Summary Statistics for 5-Fold CV R<sup>2</sup> Mean for Well 8 Symbolic Regression.

| Parameter setting              | R <sup>2</sup> Mean | Standard deviation |
|--------------------------------|---------------------|--------------------|
| 5 Populations, 30 Iterations   | 0.511               | 0.0314             |
| 5 Populations, 80 Iterations   | 0.527               | 0.0210             |
| 5 Populations, 150 Iterations  | 0.591               | 0.0099             |
| 10 Populations, 30 Iterations  | 0.629               | 0.0180             |
| 10 Populations, 80 Iterations  | 0.729               | 0.0319             |
| 10 Populations, 150 Iterations | 0.775               | 0.0031             |
| 15 Populations, 30 Iterations  | 0.784               | 0.0074             |
| 15 Populations, 80 Iterations  | 0.894               | 0.0105             |
| 15 Populations, 150 Iterations | 0.910               | 0.0129             |

The summary statistics table demonstrates a decreasing trend in standard deviation values with increasing population and iteration numbers, further supporting improved consistency in performance. Performance analysis reveals that the model achieves its lowest at 5 populations and 30 iterations (mean R<sup>2</sup> = 0.511, SD = 0.0314). Modest improvements are observed at 80 and 150 iterations (mean R<sup>2</sup> = 0.527 and 0.591, respectively). With 10 populations, performance improves across iterations: R<sup>2</sup> = 0.629 (30 iterations), R<sup>2</sup> = 0.729 (80 iterations), and R<sup>2</sup> = 0.775 (150 iterations), maintaining low standard deviations. Optimal performance is achieved with 15 populations: R<sup>2</sup> = 0.784 (30 iterations), R<sup>2</sup> = 0.894 (80 iterations), and R<sup>2</sup> = 0.910 (150 iterations), with minimal variability. Figure 74 presents a scatter plot of predicted versus actual cell growth on test data, yielding an R<sup>2</sup> of 0.878. Most data points closely align with the ideal fit line, validating the model's predictive accuracy and corroborating cross-validation results. Minor discrepancies are observed at lower growth values. The optimal configuration (15 populations, 150 iterations) demonstrates superior performance (mean R<sup>2</sup> =

0.910, low variability). The model's performance on test data, as indicated by an  $R^2$  value of 0.878, demonstrates its practical utility. This slight reduction in performance compared to the cross-validation results ( $R^2 = 0.910$ ) is consistent with typical patterns observed when models are applied to novel datasets, reflecting the challenges inherent in generalising unseen data.

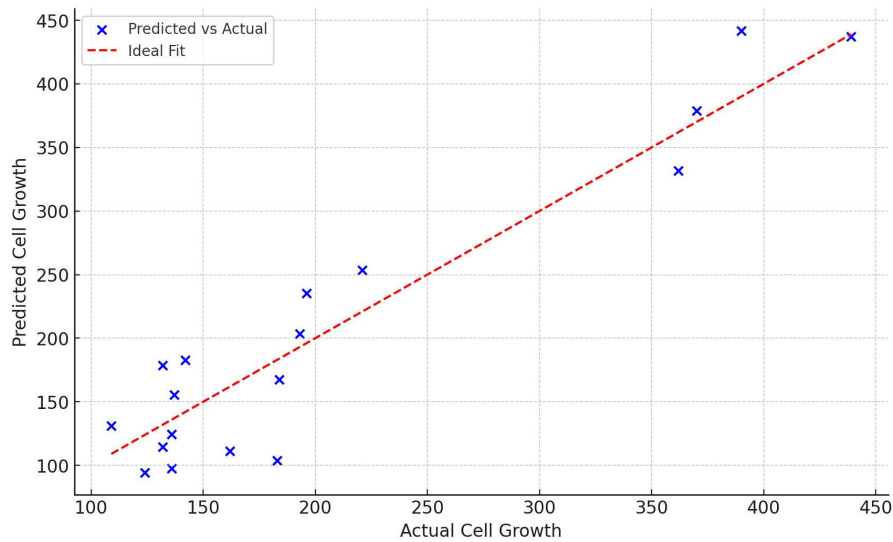


Figure 74: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.878$ ).

Figure 75 illustrates the distribution of  $R^2$  values for the Control, TGF-B, and SB431542 treatments, based on symbolic regression applied to the test dataset. The boxplot provides a concise visual summary, effectively highlighting the consistency of the model's performance across the different treatments. To determine whether there are statistically significant differences in the  $R^2$  values among the three treatment groups (Control, TGF-B, and SB431542), a Kruskal-Wallis test was conducted. The test resulted in a p-value of 0.789. As this p-value is greater than the commonly used significance level of 0.05, the null hypothesis cannot be rejected. This result indicates that there is no statistically significant difference in the  $R^2$  values among the Control, TGF-B, and SB431542 treatment groups. Consequently, the lack of significant difference suggests that the performance of the model is consistent across the different treatments. The key biological insight here is that the symbolic regression model, which predicts cell growth based on inputs such as time, migration speed, and angular velocity, demonstrates consistent performance across different treatment conditions. The consistent  $R^2$  values across the Control, TGF-B, and SB431542 groups suggest that the model reliably predicts cell growth. This robustness is crucial for understanding the impact of these treatments on cell behaviour and supports the model's applicability across diverse experimental scenarios.

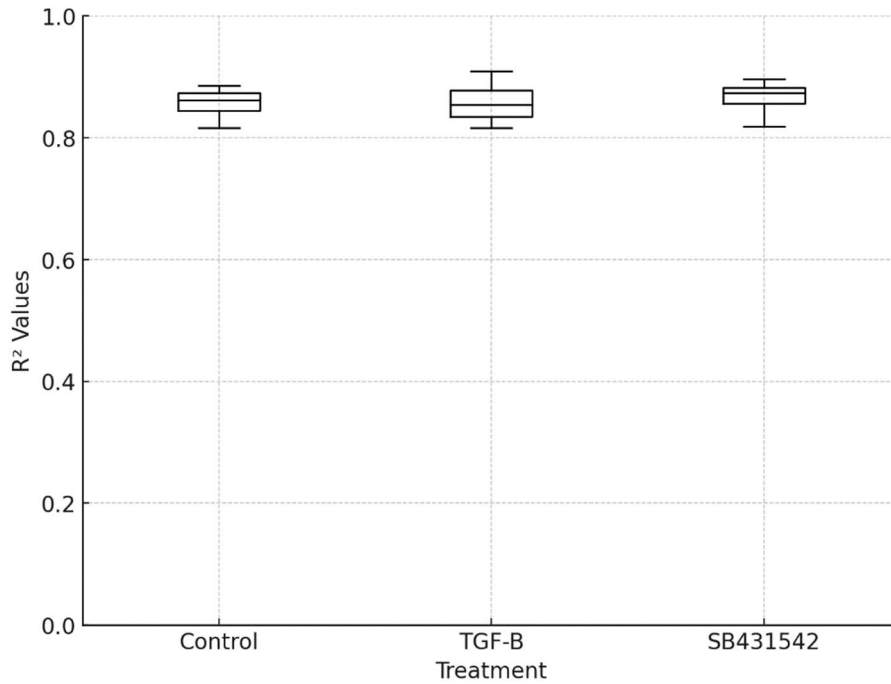


Figure 75:  $R^2$  value distribution for Control, TGF-B, and SB431542 treatments in symbolic regression on the test dataset

#### 4.9.2.2 Symbolic classification

The classification model's performance was evaluated using a dataset from control (well 0, 1, 2, and 3), TGF-Beta (well 4, 5, 6, and 7), and SB431542 (well 8, 9, 10, and 11) treatment. The model's accuracy was assessed through a 5-fold cross-validation process, and the results are summarised in a box plot and a table of mean accuracies with standard deviations. Figure 76 illustrates the box plot of the classification model's 5-fold cross-validation results. The x-axis represents various parameter settings, specifically the number of populations (3, 9, and 18) and the number of iterations (50, 100, and 200). The y-axis indicates the accuracy achieved by the model. A clear trend of increasing accuracy can be observed as the number of populations and iterations increase. For instance, the accuracy for the configuration with 3 populations and 50 iterations starts around 41.80%, while the configuration with 18 populations and 200 iterations achieves an accuracy close to 87.10%. Table 47 presents the summary statistics for the 5-fold cross-validation mean accuracy, including the standard deviation for each parameter setting. Configurations with 3 populations show a mean accuracy ranging from 41.80% to 58.10%, with increasing iterations resulting in higher accuracy. Configurations with 9 populations demonstrate a mean accuracy from 64.70% to 73.90%. The most notable improvements were made with 18 populations, where the mean



accuracy ranged from 79.30% to 87.10%. The highest accuracy observed is 87.10%, achieved with 18 populations and 200 iterations, indicating the model's capability to classify the dataset accurately.

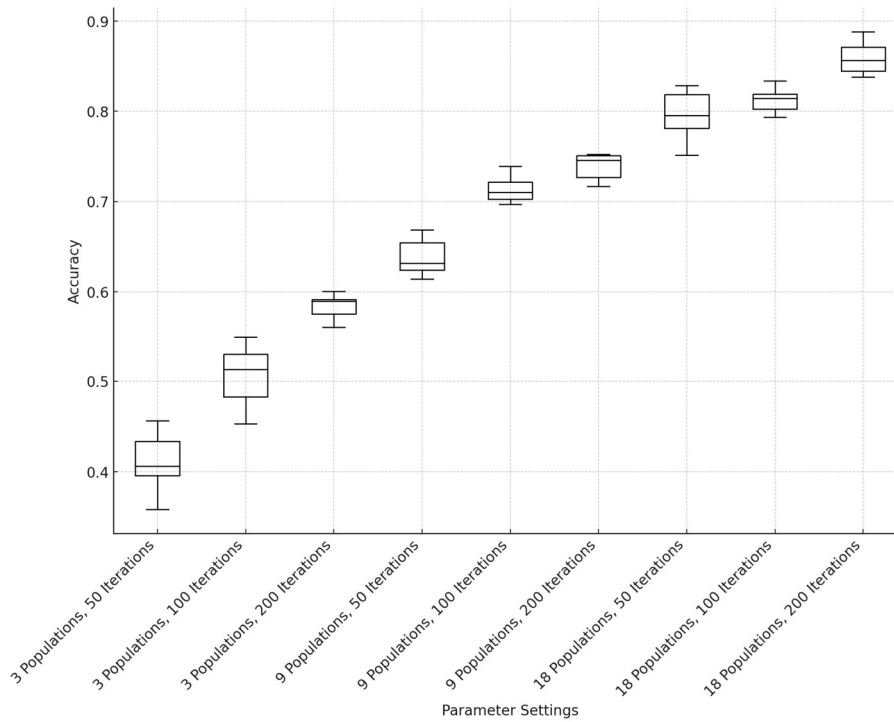


Figure 76: Box Plot of classification Model 5-Fold Cross-Validation Results.

Table 47: Summary Statistics for 5-Fold CV Mean Accuracy (%) for Symbolic Classification.

| Parameter setting             | Mean Accuracy | Standard deviation |
|-------------------------------|---------------|--------------------|
| 3 Populations, 50 Iterations  | 41.80         | 0.0164             |
| 3 Populations, 100 Iterations | 50.10         | 0.0204             |
| 3 Populations, 200 Iterations | 58.10         | 0.0082             |
| 9 Populations, 50 Iterations  | 64.70         | 0.0121             |
| 9 Populations, 100 Iterations | 71.90         | 0.0111             |
| 9 Populations, 200 Iterations | 73.90         | 0.0113             |

| Parameter setting              | Mean Accuracy | Standard deviation |
|--------------------------------|---------------|--------------------|
| 3 Populations, 50 Iterations   | 41.80         | 0.0164             |
| 18 Populations, 50 Iterations  | 79.30         | 0.0143             |
| 18 Populations, 100 Iterations | 81.90         | 0.0112             |
| 18 Populations, 200 Iterations | 87.10         | 0.0123             |

The classification accuracy on the test data is reported as 92.31%, suggesting that the model generalises well to unseen data. The low standard deviations across different configurations, particularly those with higher populations and iterations, imply consistent performance and robustness of the model.

#### 4.9.2.3 Symbolic regression and classification model

In this section, a model for symbolic regression is presented for each well based on their treatment. The  $y$  value represents cell growth,  $x_0$  represents time,  $x_1$  represents migration speed, and  $x_2$  represents angular velocity. Regarding the control treatment, well 0 is represented by equation 18, well 1 corresponds to equation 19, well 2 is described by equation 20, and well 3 is depicted by equation 21. The visualisation of the equations, showing the relationship between variables, is provided only for well 0. For wells 1, 2, and 3, please refer to Appendix Figures A.37 through A.39.

Symbolic regression model for Well 0

$$y = 6.78 \times 10^{-1} \cdot A_1 \cdot x_0 \cdot x_2 \cdot (x_1 - x_2) + 8.42 \times 10^{-2} \cdot A_2 - x_1 \cdot A_4 + 2.25 \times 10^2 \quad (18)$$

Where;

$$A_1 = x_2 - 0.7032$$

$$A_2 = x_0 - 31.865$$

$$A_3 = x_1 - 0.433$$

$$A_4 = A_2 \cdot A_3 \cdot x_1 - 145.1451$$

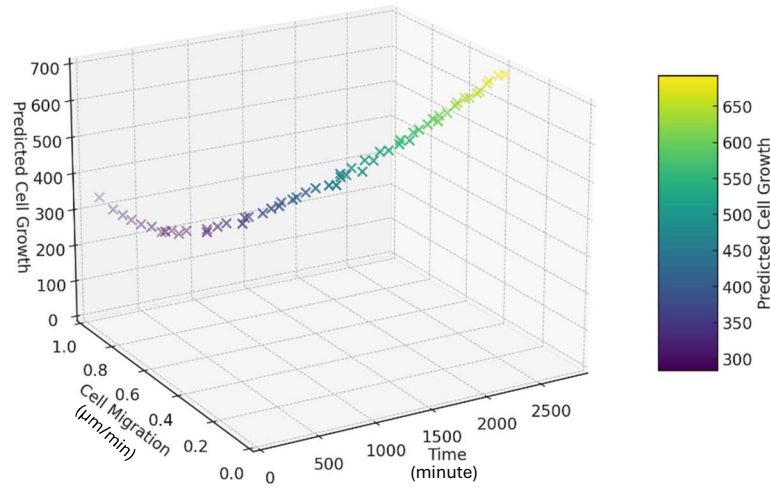


Figure 77: 3D Surface Plot of Predicted Cell Growth vs. Migration Speed and Time. For Well 0

Figure 77 presents a 3D scatter plot that demonstrates the relationship between predicted cell growth, time, and cell migration for Normal Human Urothelial (NHU) cells over a span of 48 hours based on the symbolic regression equation derived for Well 0. The X-axis represents time, indicating the duration of observation. The Y-axis displays cell migration speed, and the Z-axis illustrates predicted cell growth. Each point on the plot is colour-coded to represent different levels of cell growth, ranging from 300 to over 650, as shown on the colour gradient bar. This visualisation highlights the pattern of cell growth over time in relation to migration speed, providing insights into how cell migration influences growth dynamics in a controlled environment.

Symbolic regression model for Well 1

$$y = 6.74 \times 10^{-5} \cdot x_0^2 + 3.72 \times 10^1 \cdot A_1^2 \cdot A_2^4 - 4.6 \times 10^{-1} \quad (19)$$

Where;

$$A_1 = 1 - 1.7 \times 10^{-1} \cdot x_1$$

$$A_2 = 1 + 9.8 \times 10^{-1} \cdot x_1$$

Symbolic regression model for Well 2

$$y = -6.43 \times 10^3 \cdot (x_1 x_2)^2 \cdot (A_1 - 9.4 \times 10^{-1} \cdot x_2)^2 + 1.30 \times 10^3 \cdot A_2^2 + 1.0 \times 10^{-1} \cdot A_3 \cdot x_1^{-1} \quad (20)$$

Where;

$$A_1 = x_1 + 1.0 \times 10^{-2}$$

$$A_2 = x_1 - 3.5 \times 10^{-1}$$

$$A_3 = x_0 + 2.46$$

Symbolic regression model for Well 3

$$y = -4.57 \times 10^{-5} \cdot A_1 + 4.27 \times 10^1 \cdot x_1 + A_2 \cdot (A_3 + 9.25 \times 10^{-1}) + 2.22 \times 10^2 \quad (21)$$

Where;

$$A_1 = x_0 \cdot (x_1 - x_0)$$

$$A_2 = \left(\frac{x_1}{x_2}\right)^2 + \frac{7.81 \times 10^{-1} \cdot x_2}{x_1 - 4.99 \times 10^{-1}}$$

$$A_3 = x_1 \cdot (x_2 - x_1)$$

As for the TGF-beta treatment, well 4 corresponds to equation 22, well 5 is described by equation 23, well 6 is depicted by equation 24, and well 7 matches equation 25. The visualisation of the equations, showing the relationship between variables, is provided only for well 4. For wells 5, 6, and 7, please refer to Appendix Figures A.40 through A.42.

Symbolic regression model for Well 4

$$y = (1.06 \cdot x_1^3 + A_1) \cdot (3.0 \times 10^{-2} \cdot x_0 - 1.15 \cdot A_2 + A_3 + 2.85 \times 10^2) \quad (22)$$

Where;

$$A_1 = \frac{x_0 \cdot (8.4 \times 10^{-1} - x_1) + x_1}{x_0 \cdot (5.0 \times 10^{-2} \cdot x_2 + 3.8 \times 10^{-1})}$$

$$A_2 = \frac{x_2^3}{(x_1 - 4.6 \times 10^{-1}) \cdot (x_2 - 5.6 \times 10^{-1})}$$

$$A_3 = \frac{x_2^2}{6.7 \times 10^{-1} - x_1}$$

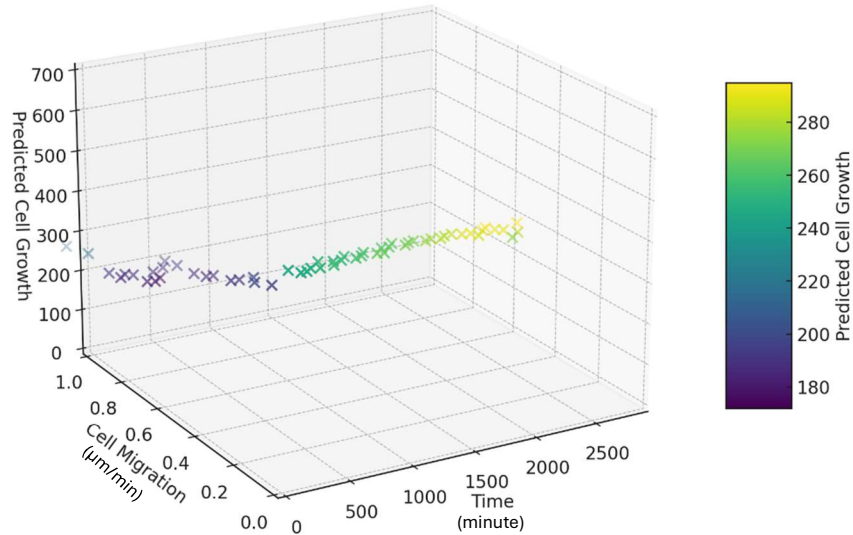


Figure 78: 3D Surface Plot of Predicted Cell Growth vs. Migration Speed and Time for Well 4.

Figure 78 showcases a 3D scatter plot that maps the relationship between predicted cell growth, time, and cell migration for Normal Human Urothelial (NHU) cells over a 48-hour period, utilising a symbolic regression equation derived for Well 4. The X-axis denotes time, representing the duration of observation, while the Y-axis measures cell migration speed. The Z-axis displays predicted cell growth. Each point on the plot is colour-coded to indicate various levels of cell growth, ranging from 180 to 280, as depicted on the colour gradient bar. This visualisation underscores the pattern of cell growth over time in connection to migration speed, offering valuable insights into how migration dynamics impact cell growth under TGF- $\beta$  treatment.

Symbolic regression model for Well 5

$$y = -x_2^2 \cdot (x_2 + 3.0) + A_1 \cdot A_2 - A_3 + 1.20 \times 10^2 \tag{23}$$

Where;

$$A_1 = 2.39 \times 10^{-6} \cdot (1.84 \times 10^{-3} \cdot x_0 - 1)^2 \cdot x_2$$

$$A_2 = 1.25 \times 10^7 \cdot (2.83 \times 10^{-4} \cdot x_0 - 1)^2 + 5.49 \times 10^6 \cdot (4.27 \times 10^{-4} \cdot x_0 - 1)^2$$

$$A_3 = \frac{7.19 \cdot x_1 - 4.92}{x_1^2 - x_2 + 2.53 \times 10^{-1}}$$

Symbolic regression model for Well 6

$$y = -1.73 \times 10^{-3} \cdot x_0 + A_1 \cdot x_1^3 + A_2 \cdot x_1 + 8.21 \cdot x_2^2 + 1.40 \times 10^2 \quad (24)$$

Where;

$$A_1 = 4.29 \times 10^{-2} \cdot (x_0 - 1.10 \times 10^1)$$

$$A_2 = \frac{(1.14 - 1.45 \cdot x_1) \cdot (x_0 - 1.40 \times 10^2)}{x_2 \cdot (3.96 \times 10^{-6} \cdot x_0^2 + 4.28 \cdot x_1^2)}$$

Symbolic regression model for Well 7

$$y = A_1 \cdot (4.65 \times 10^{-3} \cdot x_0 - 2.08) + A_2 + 1.08 \times 10^2 \cdot A_3^2 \quad (25)$$

Where;

$$A_1 = \frac{x_1 - 6.02 \times 10^{-1}}{x_2 - 8.24 \times 10^{-1}}$$

$$A_2 = \frac{x_1 - 5.93 \times 10^{-1}}{x_2^2 - 9.55 \times 10^{-1}}$$

$$A_3 = 1.83 \times 10^{-4} \cdot x_0 + 9.62 \times 10^{-2} \cdot x_2^2 + 1$$

In terms of the SB431542 treatment, well 8 is described by equation 26, well 9 matches equation 27, well 10 is depicted by equation 28, and well 11 corresponds to equation 29. The visualisation of the equations, showing the relationship between variables, is provided only for well 8. For wells 9, 10, and 11, please refer to Appendix Figures A.43 through A.45.

Symbolic regression model for Well 8

$$y = (-1.44 \times 10^{-8}) \cdot A_1 \cdot A_2 \cdot x_0^2 - x_2 + A_3 + A_4 \quad (26)$$

Where;

$$A_1 = (8.83 \times 10^{-3} \cdot x_0 - 1)^2$$

$$A_2 = 9.08 \times 10^{-8} \cdot x_0^2 - 1$$

$$A_3 = 1.34 \times 10^2 + \frac{9.32 \times 10^{-1} - x_2}{9.31 \times 10^{-1} - x_1}$$

$$A_4 = \frac{7.11 \times 10^{-1} - x_2^2}{(6.80 \times 10^{-1} - x_1) \cdot (x_1 - 3.20 \times 10^{-2})}$$

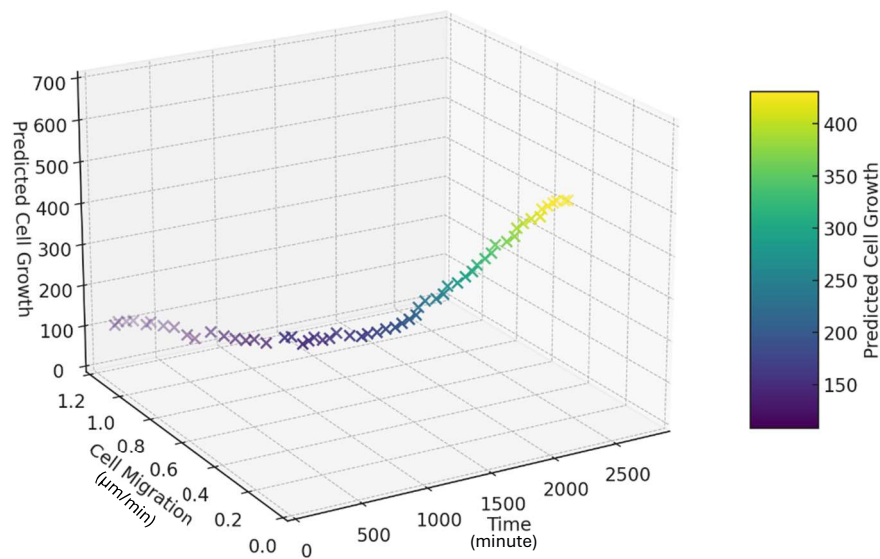


Figure 79: 3D Plot of Predicted Cell Growth vs. Migration Speed and Time for Well 8.

In Figure 79, a 3D scatter plot elucidates the dynamics of cell growth influenced by migration speed and time for Normal Human Urothelial (NHU) cells, modelled over a 48-hour period using a symbolic regression equation for Well 8. Time is plotted on the X-axis, illustrating the duration for which observations were recorded. Cell migration speed is charted on the Y-axis, whilst predicted cell growth is delineated on the Z-axis. The varying levels of cell growth, ranging from 150 to 400, are visually differentiated using a colour gradient bar. This chart

offers a lucid visual representation of how migration speeds correlate with the growth patterns of NHU cells under conditions influenced by SB431542 treatment, thereby highlighting key trends and potential biological implications.

Symbolic regression model for Well 9

$$y = -x_2 + (x_1 - 1.25) \cdot A_1 + x_1 \cdot A_2 + 2.3 \times 10^{-1} \cdot A_3 + 1.26 \times 10^2 \quad (27)$$

Where;

$$A_1 = (-3.66 \times 10^{-5} \cdot x_0 - 1.0 \times 10^{-2}) \cdot (x_0 + 1.95 \times 10^2)$$

$$A_2 = \frac{-2.67 \cdot x_2 \cdot x_0 + 6.0 \times 10^{-2} \cdot x_0 - 1.29 \times 10^2 \cdot x_2}{x_2 \cdot x_0 \cdot (3.90 \cdot x_1 \cdot x_2 + 4.03 \cdot x_1 - 3.99)}$$

$$A_3 = \frac{1}{x_2 \cdot (x_1 - 9.4 \times 10^{-1})}$$

Symbolic regression model for Well 10

$$y = A_1 \cdot x_0^2 + 1.44 \times 10^{-2} \cdot A_2 \cdot x_0 + 1.88 \times 10^2 \cdot x_2 - 3.15 \times 10^1 + A_3 \quad (28)$$

Where;

$$A_1 = \frac{-1}{x_1^2} \cdot (1.08 \times 10^{-5} - 4.94 \times 10^{-9} \cdot x_0)$$

$$A_2 = \frac{1}{x_1 \cdot x_2}$$

$$A_3 = -\frac{1.50 \times 10^2}{x_0 - 1.86 \times 10^2} + \frac{6.38}{x_2^4}$$

Symbolic regression model for Well 11

$$y = 8.83 \times 10^{-2} \cdot x_0 + 1.15 \times 10^2 + A_1 - A_2 \quad (29)$$

Where;



$$A_1 = \frac{6.90 \times 10^{-3}}{x_1 - 4.94 \times 10^{-1}}$$

$$A_2 = \frac{1.35 \times 10^2}{-8.17 \times 10^{-3} \cdot x_0 - 2.36}$$

In this study, the stability of each model was assessed individually, where each symbolic regression produced an equation that was evaluated against unseen data to predict cell growth. Table 48 summarises the coefficient of determination  $R^2$  values for each well, which serve as an indicator of how well the models were able to predict cell growth based on the test data. The  $R^2$  values for all wells ranged between 0.816 and 0.909, indicating that the models consistently performed well in predicting cell growth across different wells. This narrow range of  $R^2$  values suggests a high level of stability and consistent predictive accuracy across the different wells, even though the specific symbolic regression equations generated for each well were not identical. The consistently high  $R^2$  values across all wells suggest that the symbolic regression models are stable and reliable in predicting cell growth. Despite the differences in the equations generated for each well, the models exhibit similar trends in predictive accuracy, as evidenced by the closely clustered  $R^2$  values. This consistency across multiple datasets under similar conditions reinforces the stability of the models and their ability to generalise well to unseen data.

Table 48:  $R^2$  Values for Symbolic Regression Models for All Wells.

| Well | Coefficient of determination, $R^2$ |
|------|-------------------------------------|
| 0    | 0.870                               |
| 1    | 0.854                               |
| 2    | 0.816                               |
| 3    | 0.886                               |
| 4    | 0.867                               |
| 5    | 0.816                               |
| 6    | 0.841                               |
| 7    | 0.909                               |
| 8    | 0.878                               |
| 9    | 0.896                               |
| 10   | 0.818                               |
| 11   | 0.869                               |

For symbolic classification,  $y_0$  represents output 0,  $y_1$  represents output 1, and  $y_2$  represents output 2. This setup represents the multi-binary output for classification, where  $x_0$  corresponds to growth 1,  $x_1$  corresponds to growth 2,  $x_2$  corresponds to growth 3, and  $x_3$  corresponds to growth 4.

Symbolic classification model

$$y_0 = \frac{1}{1+\exp(-20(0.35 \times x_2 + (x_3 \times 1.21 - x_0) - 100))} - \frac{1}{1+\exp(-20(x_1 \times 0.24 - 100))} \quad (30)$$

$$y_1 = 1 - \frac{1}{1+\exp(-20(2.15 \times x_1 - x_0 - 100))} \quad (31)$$

$$y_2 = \frac{1}{1+\exp(-2(0.45 \times x_0 - x_2 \times 0.30 - 1))} - \frac{1}{1+\exp(-20(2.19 \times (x_1 - x_2) - 100))} \times (2.31 - x_3 \times 0.01) \times (-3.36 + 0.01 \times x_3) \quad (32)$$

#### 4.9.3 Comparison of Classification with Different Machine Learning Algorithms

Long Short-Term Memory (LSTM) [290] is a specialised type of recurrent neural network architecture designed to address the vanishing gradient problem in traditional RNNs. Developed by Hochreiter and Schmidhuber in 1997, LSTM networks are particularly adept at learning and processing sequential data, making them invaluable for tasks that require understanding context over extended periods. The key innovation of LSTMs lies in their unique cell structure, which incorporates gates (input, forget, and output gates) to control the flow of information, allowing the network to remember or discard information as needed selectively. This design enables LSTMs to capture long-term dependencies in data, especially for sequential and time series prediction.

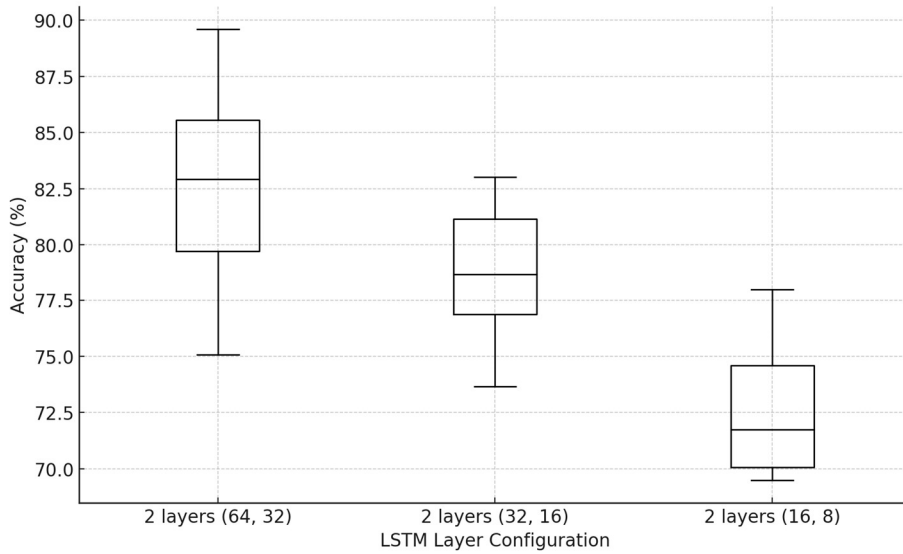


Figure 80: LSTM Box Plot of Classification Model 5-Fold Cross-Validation Results.

Table 49: Summary Statistics for 5-Fold CV Mean Accuracy of LSTM Classification.

| Parameter setting | Mean accuracy (%) | Standard deviation |
|-------------------|-------------------|--------------------|
| 2 layer (64,32)   | 82.61             | 4.76               |
| 2 layer (32,16)   | 78.71             | 3.14               |
| 2 layer (16,8)    | 72.67             | 3.07               |

The classification results of LSTM models with varying layer configurations are depicted in the box plot titled Figure 80, comparing the performances of different LSTM layer setups. The visual representation shows a trend where larger LSTM layers achieve higher accuracy but also exhibit greater variability. The median values, represented by the lines within the boxes, indicate that the accuracy tends to be highest for the configuration with 2 layers (64, 32), followed by 2 layers (32, 16), and lowest for 2 layers (16, 8). The interquartile ranges (IQR) illustrate the spread of accuracy values, with larger boxes indicating higher variability, particularly noticeable in the 2 layers (64, 32) configuration. Detailed statistical data on the classification performance of various LSTM configurations are shown in Table 49. The table includes each configuration's parameter settings, mean accuracy (%), and standard deviation. The configuration with two layers (64, 32) has a mean accuracy of 82.61% and a standard

deviation of 4.76, indicating higher performance but greater variability. The two layers (32, 16) setup records a mean accuracy of 78.71% and a lower standard deviation of 3.14, showing more consistent outcomes. The configuration with two layers (16, 8) has the lowest mean accuracy at 72.67% and the smallest standard deviation of 3.07. The analysis shows that the configuration with two layers (64, 32), where the first layer has 64 units and the second layer has 32 units, achieves the highest mean accuracy but also has significant variability, indicating inconsistent performance. The configuration with two layers (32, 16) demonstrates moderate accuracy with reduced variability, suggesting consistent performance. The setup of the two layers (16, 8) has the lowest mean accuracy but the least variability, indicating high consistency. These findings highlight a trade-off between achieving high accuracy and maintaining consistent performance across different LSTM configurations, emphasising the impact of layer setup on network behaviour.

Random Forest [291] is a machine-learning algorithm developed by Leo Breiman in 2001. It operates under ensemble learning methods, combining multiple decision trees to create a robust and accurate predictive model. During training, it constructs many decision trees and outputs either the mode of the classes (for classification) or the mean prediction (for regression) of the individual trees. Random Forest uses bagging, or bootstrap aggregating, where each tree is built from a sample drawn with replacement from the training set. Additionally, it employs feature bagging, selecting a random subset of features for consideration at each node split. This randomness enhances the model's robustness and reduces overfitting compared to individual decision trees.

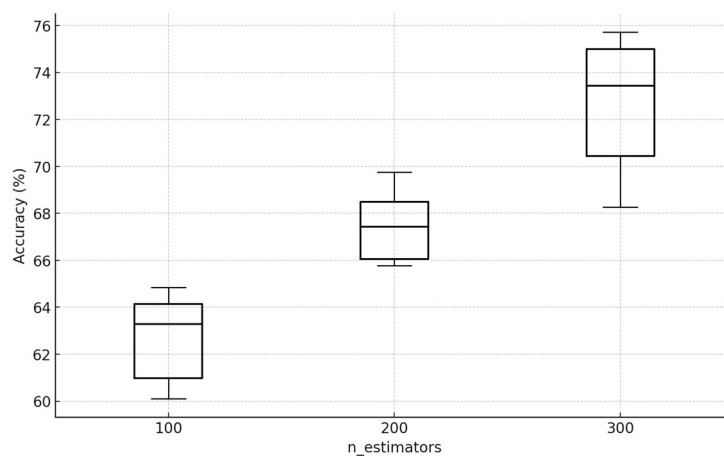


Figure 81: Random Forest Box Plot of Classification Model 5-Fold Cross-Validation Results.

Table 50: Summary Statistics for 5-Fold CV Mean Accuracy of Random Forest Classification.

| Parameter setting  | Mean accuracy (%) | Standard deviation |
|--------------------|-------------------|--------------------|
| n_estimators = 100 | 62.69             | 1.86               |
| n_estimators = 200 | 67.47             | 1.47               |
| n_estimators = 300 | 72.64             | 2.83               |

The box plot titled Figure 81 illustrates the classification performance of a Random Forest model using 5-fold cross-validation across different numbers of estimators. The `n_estimators` parameter varies among three settings: 100, 200, and 300. The plot shows that the median accuracy increases as the number of estimators rises, with `n_estimators = 300` achieving the highest median accuracy. The interquartile ranges (IQR) indicate the spread of the accuracy values, with larger spreads observed at higher `n_estimators` values, particularly at 300, suggesting greater variability. The accompanying table, Table 50, provides summary statistics for the mean accuracy and standard deviation for each setting of the `n_estimators` parameter. The configuration with `n_estimators = 100` shows a mean accuracy of 62.69% and a standard deviation of 1.86, indicating relatively low performance with moderate variability. The configuration with `n_estimators = 200` records a mean accuracy of 67.47% and a lower standard deviation of 1.47, indicating improved performance with reduced variability. Finally, the configuration with `n_estimators = 300` presents the highest mean accuracy at 72.64% and the highest standard deviation at 2.83, indicating both high performance and greater variability. The analysis displays a trend where increasing the number of estimators in the Random Forest model leads to higher mean accuracy. However, the configuration with `n_estimators = 300`, despite achieving the highest mean accuracy, shows the greatest variability, indicating potential inconsistencies in performance. In contrast, `n_estimators = 200` provides a balance with moderate accuracy and reduced variability, suggesting more consistent performance. The configuration with `n_estimators = 100` maintains moderate variability despite having the lowest mean accuracy. The `n_estimators` parameter affects the Random Forest model's behaviour and outputs, highlighting a trade-off between accuracy and consistency.

Support Vector Machine (SVM) [292] is a supervised machine learning algorithm for classification and regression tasks. Developed by Vladimir Vapnik and his colleagues in the

1990s, SVM focuses on finding the optimal hyperplane that separates different classes in a high-dimensional feature space. By mapping input data to a higher-dimensional space, SVM can establish a linear decision boundary even for non-linearly separable data. The algorithm maximises the margin between classes, defined as the distance between the decision boundary and the nearest data points from each class, known as support vectors. Margin maximisation improves the model's generalisation capability, enhancing performance on unseen data. SVMs can handle both linear and non-linear classification tasks through various kernel functions, which perform the mapping to higher-dimensional spaces implicitly, making the algorithm computationally efficient.

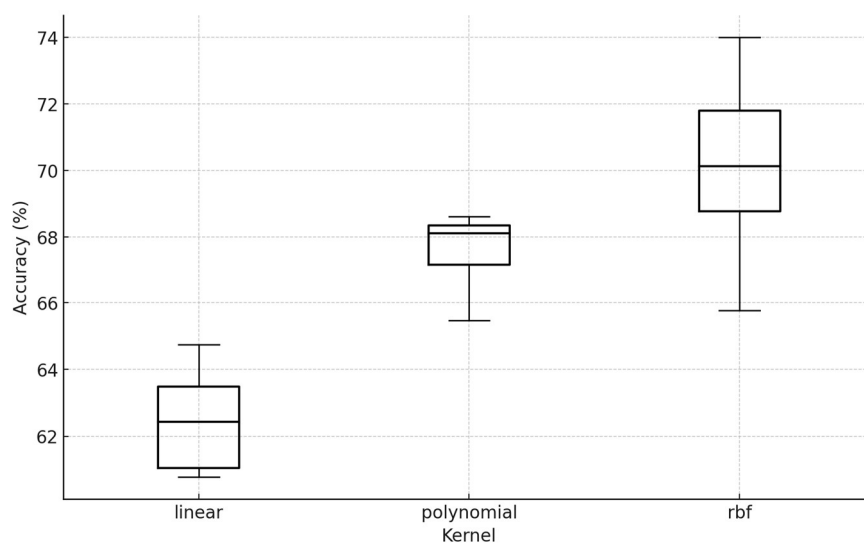


Figure 82: Support Vector Machine Box Plot of Classification Model 5-Fold Cross-Validation Results.

Table 51: Summary Statistics for 5-Fold CV Mean Accuracy of SVM Classification.

| Parameter setting   | Mean accuracy (%) | Standard deviation |
|---------------------|-------------------|--------------------|
| Kernel = linear     | 62.47             | 1.47               |
| Kernel = polynomial | 67.60             | 1.10               |
| Kernel = rbf        | 70.11             | 2.64               |

Figure 82 depicts the classification performance of a Support Vector Machine (SVM) model using 5-fold cross-validation across different kernel functions: linear, polynomial, and radial

basis function (RBF). The box plot exhibits that the median accuracy varies with different kernel functions, with the RBF kernel achieving the highest median accuracy. The interquartile ranges (IQR) reflect the spread of accuracy values, with the RBF kernel showing a wider spread, suggesting greater variability. The whiskers and outliers provide further insights into the range and extremities of the accuracy data. Table 51 shows summary statistics for each kernel function's mean accuracy and standard deviation. The linear kernel configuration shows a mean accuracy of 62.47% and a standard deviation of 1.47, indicating lower performance with moderate variability. The polynomial kernel records a mean accuracy of 67.60% and a lower standard deviation of 1.10, suggesting improved performance with reduced variability. The RBF kernel presents the highest mean accuracy at 70.11% and the highest standard deviation at 2.64, indicating high performance and more significant variability. The analysis indicates that the choice of kernel function significantly affects the performance of the SVM model. The RBF kernel achieves the highest mean accuracy but exhibits the most significant variability, suggesting potential inconsistencies in performance. In contrast, the polynomial kernel strikes a balance with moderate accuracy and reduced variability, indicating more consistent performance. The linear kernel maintains moderate variability while having the lowest mean accuracy. The results underscore the balance required between attaining high accuracy and ensuring consistent performance, emphasising the critical role of selecting an appropriate kernel function for the SVM model.

#### **4.9.3.1 Test Result Classification with Different Machine Learning Algorithms**

The table 72 compares test accuracy percentages for different classification methods: PySR, RCGP, LSTM, Random Forest, and Support Vector Machine (SVM). The table summarises the test accuracy of five classification methods. PySR achieved a test accuracy of 92.31%, indicating that it is a highly effective model for the dataset used. RCGP, on the other hand, recorded a test accuracy of 88.59%, which is slightly lower but still represents a substantial level of accuracy. The LSTM model achieves the test accuracy at 80.96% and Random Forest model follows with a test accuracy of 71.43%, showing respectable performance but trailing behind the LSTM. The SVM model records the lowest test accuracy at 68.63%, suggesting it is less effective than the other two methods in this scenario. The comparison reveals that PySR achieved the highest accuracy, followed by RCGP, the LSTM model, the Random Forest, and the Support Vector Machine (SVM) regarding test accuracy.

Table 52: Comparison of Test Accuracy for Different Classification Methods (PySR, RCGP, LSTM, Random Forest, and SVM).

| Classification method  | Test accuracy (%) |
|------------------------|-------------------|
| PySR                   | 92.31             |
| RCGP                   | 88.59             |
| LSTM                   | 80.96             |
| Random forest          | 71.43             |
| Support Vector Machine | 68.63             |

This order highlights the relative effectiveness of these models in handling the specific classification task. PySR, being the most accurate, demonstrates its robustness for this dataset, while RCGP also performs strongly. LSTM models, though less effective than PySR and RCGP, still handle sequential and temporal data well due to their recurrent structure, which captures dependencies over time. While performing reasonably well, the Random Forest model does not match the accuracy of the LSTM. Random Forest models excel at capturing complex interactions between features through their ensemble approach of multiple decision trees. The SVM model shows the lowest performance among these methods. SVM can be prone to overfitting, mainly when model complexity or the data contains noise, leading to lower generalisation performance on the test set. Additionally, SVM may struggle with datasets with complex structures or non-linear relationships that are not well captured by the chosen kernel.

## 5.0 Conclusion

In conclusion, this chapter has demonstrated the implementation of various advanced computational techniques in analysing cell culture data, underscoring novel approaches in model construction and analytical methods. The illumination correction techniques, such as FFTF, PFFC, and BaSiC, were critically evaluated, with BaSiC achieving the lowest mean coefficient of variation, thereby enhancing image quality significantly. Besides, Quantile-Based Normalisation was identified as the most efficacious method for standardising images, ensuring consistency across analyses. Tools like Ilastik were shown to be highly accurate in



cell segmentation, while the Follow Neighbour method provided the highest MOTA scores in cell tracking. Detailed analyses of cell growth curves and rates under various treatments yielded important biological insights, particularly highlighting the inhibitory effect of TGF-beta on cell proliferation. Moreover, migration speed and angular velocity measurements further elucidated these treatments' impact on cell dynamics, with TGF-beta significantly enhancing cell motility, whereas SB431542 moderated it.

The application of RCGP with a pairwise and ensemble classification approach, alongside PySR, which combines symbolic regression with symbolic classification, introduced novel modelling capabilities within the domain of machine learning classification. These models demonstrated high classification rates and predictive accuracies and facilitated the generation of symbolic equations and network diagrams that reveal underlying biological behaviours. This aspect is highly valuable, providing interpretable "white box" models that contribute profoundly to our understanding of the data. This is very different from 'black box' methods like LSTM, which, although accurate, don't clearly show how they work. A comparative analysis with other machine learning models, such as Random Forest and SVM, underscored the trade-offs between accuracy and interpretability, highlighting the advantages of novel approaches employed in RCGP and PySR for detailed scientific analysis. These findings emphasise the critical role of selecting appropriate analytical tools to enhance our understanding and methodologies in cellular biology research, ensuring that precision in analysis and depth in interpretative clarity are maintained.

## CHAPTER 5

### CONCLUSIONS AND FUTURE WORK

#### 5.1 Introduction

This thesis aimed to develop a computational tool pipeline to quantify cell behaviour from time-lapse microscopy images, facilitating an objective understanding of cell culture across different conditions and environments. The study achieved its objectives and made several significant contributions to computational biology. Below is a summary of the essential findings and contributions of this research.

#### 5.2 Summary of Key Findings

Initially, the first objective was to establish a pipeline of computer vision tools to reliably segment and track unlabelled cells grown in culture as adherent monolayers using normal human urothelial cells as a representative example. This objective was achieved by meticulously designing and implementing a robust methodology detailed in Chapter 3. The study commenced with cell culture preparation across twelve plates, each containing four replicates, under three different treatments: a control, TGF- $\beta$ , and SB431542 as a TGF- $\beta$  inhibitor. Time-lapse microscopy captured detailed images, which were processed using the BaSiC plugin to correct uneven illumination and normalised using a quantile-based ImageJ plugin. Cell segmentation was performed using Ilastik for pixel classification and ImageJ to convert probabilities into binary images for precise object detection. Cell tracking was conducted using CellProfiler's Follow Neighbour method. This comprehensive pipeline established a reliable cell segmentation and tracking framework, validating its effectiveness in analysing unlabelled cells grown as adherent monolayers. While this pipeline was developed specifically for normal human urothelial cells, its underlying principles are applicable to other cell types, particularly those that exhibit similar growth patterns, such as epithelial cells in other tissues. Future work could explore adjustments to pixel classification and segmentation parameters to tailor the pipeline for different cell morphologies, thus broadening its applicability. Furthermore, the versatility of the pipeline extends to different imaging configurations. Although it was primarily tested with time-lapse microscopy, the segmentation

and tracking techniques employed are compatible with various imaging methods. For example, with additional pre-processing, the pipeline could be adapted for use with fluorescence microscopy or high-resolution imaging, enhancing its utility across a range of imaging setups in computational cell biology. This led to Contribution 1: Development of a Computational Tools Pipeline. The research successfully developed and validated a computational tools pipeline for accurate cell segmentation and tracking, tailored explicitly for analysing adherent monolayers of unlabelled cells. This achievement fulfilled the aim of creating a robust computational tools pipeline.

The second objective was to use the pipeline to describe the variations in growth rate, migration speed, and angular velocity. This objective was achieved as demonstrated in Chapter 4. Advanced computational techniques were implemented to analyse cell culture data, including illumination correction techniques (FFTF, PFFC, and BaSiC), with BaSiC achieving the lowest mean coefficient of variation and significantly enhancing image quality. Quantile-Based Normalisation ensured consistency across analyses. Ilastik proved highly accurate in cell segmentation, and the Follow Neighbour method achieved the highest MOTA scores in cell tracking. Detailed analyses of cell growth curves and rates under various treatments revealed significant biological insights, particularly the inhibitory effect of TGF-beta on cell proliferation. Migration speed and angular velocity measurements further elucidated the treatments' impact on cell dynamics, with TGF-beta significantly enhancing cell motility and SB431542 moderating. These findings underscore the pipeline's utility in capturing critical aspects of cell behaviour, offering biologists a robust tool for quantifying cellular responses to different treatments. By providing precise measurements of growth rate and motility, the pipeline enables a deeper understanding of the molecular mechanisms through which TGF-beta and its inhibitors modulate cell behaviour, thus offering valuable insights for further biological research. These analyses objectively described variations in cell growth rate, migration speed, and angular velocity. This led to Contribution 3: Characterisation of Cell Behaviour Variability. Chapter 4 focused on identifying and analysing features that describe variations in cell growth, migration speed, and angular velocity of NHU cells. These findings provided an objective framework for understanding cell behaviour variability, achieving a level of analysis not previously attained.

The third objective was to characterise the behaviour leading to changes and the differences in cell populations. This objective was addressed in Chapters 3 and 4, where the methodologies and analyses provided detailed characterisations of cell behaviour under various

treatments. The comparative study of cell behaviours under different conditions (control, TGF- $\beta$ , and SB431542) revealed significant insights into the inhibitory effect of TGF-beta on cell proliferation and its enhancement of cell motility. These findings highlighted the differences in cell populations and their responses to different environmental conditions, comprehensively depicting the behaviours leading to changes in cell populations. These findings align closely with the original aims of the study, demonstrating how specific treatments impact cellular behaviour. For biologists, this research provides predictive insights into how similar treatments might influence other cell types or complex tissue environments, potentially guiding future research into the effects of TGF-beta and its inhibitors in contexts such as cancer or tissue regeneration. This accomplishment supports Contribution 4: Insights into Cellular Responses to Treatments. The comparative study of cell behaviours under different treatment conditions, including control, TGF- $\beta$ , and SB431542, offered valuable insights into cellular responses to compounds. These insights are critical for future research on cellular responses to treatments, providing a deeper understanding of how different compounds influence cell behaviour.

The fourth objective was to investigate using the features within an interpretable "white box" machine learning context to further characterise NHU cell behaviours. This objective was achieved by applying RCGP and PySR, as discussed in Chapter 4. These tools introduced novel modelling capabilities within machine learning classification, demonstrating high classification rates and predictive accuracies. The models facilitated the generation of symbolic equations and network diagrams that revealed underlying biological behaviours, providing interpretable "white box" models. This approach contrasted sharply with more opaque "black box" methods like LSTM, which, despite yielding good accuracy, offered less insight into their operational mechanics. These machine-learning models allowed for a deeper characterisation of NHU cell behaviours, uncovering patterns and relationships in the data that traditional analysis methods might overlook. The potential to adapt these interpretable machine learning models for other cell types presents an exciting avenue for future research. By applying these models to diverse biological datasets, researchers could further our understanding of cellular behaviour across a range of contexts, enhancing the broader applicability of the tools developed in this study. This aligns with Contribution 5: Application of Interpretable "White Box" Machine Learning. The research implemented a novel approach using RCGP and PySR to characterise NHU cell behaviours. These tools provided high classification rates and predictive accuracies, generating symbolic equations and network diagrams that offer significant insights into cell behaviour data.

Additionally, introducing practical methods and significant improvements to existing segmentation techniques were critical achievements of this research. As demonstrated in Chapter 3, the methodology enhanced the precision and reliability of cell segmentation and tracking for unlabelled cells. Tools like Ilastik were shown to be highly accurate in cell segmentation, while the Follow Neighbour method provided the highest MOTA scores in cell tracking. These improvements ensured that the segmentation and tracking processes were precise and reliable, addressing the need for advanced computational techniques in cell analysis. This contribution directly aligns with Contribution 2: Segmentation and Tracking Methodology by providing enhanced tools and methodologies that improve the segmentation and tracking of unlabelled cells.

### **5.3 Contributions to the Field**

This research has made several significant contributions to the field of computational biology. Firstly, the development and validation of a computational tools pipeline for cell segmentation and tracking addressed the need for reliable analytical tools. Secondly, the segmentation and tracking methodology introduced practical improvements, enhancing the precision and reliability of these processes. Thirdly, the characterisation of cell behaviour variability provided an objective framework for understanding variations in cell growth, migration speed, and angular velocity. Fourthly, insights into cellular responses to treatments offered valuable implications for future research on cellular responses to compounds. Lastly, the application of interpretable “white box” machine learning revealed patterns and relationships in cell behaviour data that traditional methods might overlook. These contributions suggest exciting possibilities for future research, particularly in adapting these tools and methodologies to different cell types and imaging conditions, thus enhancing their general applicability across diverse biological contexts. The impact of these contributions extends beyond computational biology, potentially influencing future research in cell biology, drug development, and personalized medicine by providing tools that can more accurately model and predict cellular behaviour under various conditions.

## 5.4 Recommendations for Future Research

Despite the robust findings, the study acknowledged limitations such as the specificity of NHU cells and the conditions tested. Potential biases in image-based analysis and the need for broader data sets for model validation suggest areas for future methodological improvement. Future research should generalise the pipeline to other cell types and experimental conditions to validate its robustness and versatility. Specifically, future studies could explore the pipeline's effectiveness on a variety of cell types and imaging configurations, ensuring that the methodologies developed are broadly applicable and useful across different biological contexts. Integrating multi-modal imaging and enhanced computational techniques could deepen understanding and expand applicability in biological research. Additionally, refining the machine learning models to enhance their interpretability and accuracy would be beneficial.

In conclusion, this thesis advances theoretical and practical knowledge in computational biology, providing a robust foundation for future research. The development and validation of a computational tool pipeline for cell segmentation and tracking mark significant contributions to the field. The novelty of this research lies in the application of interpretable machine learning models, which reveal underlying patterns and relationships in cell behaviour data. These insights not only advance our understanding of NHU cells but also offer potential applications to other cell types and experimental setups, thereby broadening the impact and relevance of this research across various domains of cellular analysis. Looking forward, the methodologies and findings presented in this thesis could serve as a cornerstone for future advancements in cellular analysis, potentially leading to more accurate and predictive models of cellular behaviour, which are crucial for fields such as drug development, cancer research, and regenerative medicine. This innovative approach enhances our understanding and sets the stage for ongoing advancements in cellular analysis.

## REFERENCES

- [1] T. Sri Ranjani and C. Ramadevi, "Evaluation of Cell Biology and Genetics using VIKOR Method," *Agricultural, Biologicals and Food Science*, vol. 2, no. 1, pp. 01–05, May 2023, doi: 10.46632/abfs/2/1/1.
- [2] Q. Chen, X. Wang, and M. He, "Cell-in-Cell: From Cell Biology to Translational Medicine," *BioMed Research International*, vol. 2022, pp. 1–7, Sep. 2022, doi: 10.1155/2022/7608521.
- [3] "Cell Biology," *Biochips and Medical Imaging*, pp. 1–25, Aug. 2022, doi: 10.1002/9781118910573.ch1.
- [4] "Cells," *Understanding Metaphors in the Life Sciences*, pp. 67–87, Apr. 2022, doi: 10.1017/9781108938778.006.
- [5] R. M. Perera, "Zooming in on the cell biology of disease," *Molecular Biology of the Cell*, vol. 32, no. 22, Dec. 2021, doi: 10.1091/mbc.e21-09-0459.
- [6] R. Gupta and N. Gupta, "Prokaryotic Cell Structure and Function," *Fundamentals of Bacterial Physiology and Metabolism*, pp. 43–79, 2021, doi: 10.1007/978-981-16-0723-3\_2.
- [7] C. T. Mierke, "Focus on Eukaryotic Cells," *Cellular Mechanics and Biophysics*, pp. 35–56, 2020, doi: 10.1007/978-3-030-58532-7\_2.
- [8] M. Pavlovic, "Cell Physiology: Liaison Between Structure and Function," *Bioengineering*, pp. 23–35, Sep. 2014, doi: 10.1007/978-3-319-10798-1\_3.
- [9] M. Carter, R. Essner, N. Goldstein, and M. Iyer, "Microscopy," *Guide to Research Techniques in Neuroscience*, pp. 115–143, 2022, doi: 10.1016/b978-0-12-818646-6.00002-6.
- [10] Crystal, "Prokaryotes vs. Eukaryotes: What Are the Main Differences?," *A-Z Animals*, Apr. 17, 2023. <https://a-z-animals.com/blog/prokaryotes-vs-eukaryotes-what-are-the-main-differences/>
- [11] A.-M. Tanasescu, "Cell Culture Techniques and Practices to Avoid Contamination by Fungi and Bacteria in the Research Cell Culture Laboratory," *Journal of Visualized Experiments*, no. 197, Jul. 2023, doi: 10.3791/64769.

- [12] E. Ortega-Soto and M. Chopin-Doroteo, "Cell Cultures: A Laboratory Tool for Studying Viruses," *Frontiers for Young Minds*, vol. 11, Apr. 2023, doi: 10.3389/frym.2023.943570.
- [13] C. Zhao, "Cell culture: in vitro model system and a promising path to in vivo applications," *Journal of Histotechnology*, vol. 46, no. 1, pp. 1–4, Jan. 2023, doi: 10.1080/01478885.2023.2170772.
- [14] S. M. Afify and M. Seno, "Culture of Cells: Basic Principles," *Methods in Cancer Stem Cell Biology*, pp. 23–32, 2023, doi: 10.1007/978-981-99-1331-2\_2.
- [15] S. Weiskirchen, S. K. Schröder, E. M. Buhl, and R. Weiskirchen, "A Beginner's Guide to Cell Culture: Practical Advice for Preventing Needless Problems," *Cells*, Feb. 21, 2023. <https://doi.org/10.3390/cells12050682>
- [16] X. Zhan, Ed., "Cell Culture - Advanced Technology and Applications in Medical and Life Sciences," *Biochemistry*, Jun. 2022, Published, doi: 10.5772/intechopen.94823.
- [17] C. Kenneth, *Microscope image processing*. Academic press. [Online]. Available: [https://books.google.com/books?hl=en&lr=&id=IGFIEAAAQBAJ&oi=fnd&pg=PP1&dq=microscope&ots=HK1\\_s1nbKz&sig=rZCZYtI4--FbFnkk1f7UKkPYxLk](https://books.google.com/books?hl=en&lr=&id=IGFIEAAAQBAJ&oi=fnd&pg=PP1&dq=microscope&ots=HK1_s1nbKz&sig=rZCZYtI4--FbFnkk1f7UKkPYxLk)
- [18] J. N. Joshua, "Microscope basics," 2013, [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780124077614000014>
- [19] C. Michele, A. R. I. A. C. M., A. Giuseppe, and Meschini. "Structural and functional alterations of cellular components as revealed by electron microscopy Stefania, "Structural and functional alterations of cellular components as revealed by electron microscopy," 2013, [Online]. Available: <https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/abs/10.1002/jemt.22266>
- [20] N. Smitha, A. Neha, S. Mridula, K. Yury, and Mazumder. "Exploring the future of regenerative medicine: U. the potential of optical microscopy for structural and functional imaging of stem cells Nirmal, "Exploring the future of regenerative medicine: Unveiling the potential of optical microscopy for structural and functional imaging of stem cells," 2024, [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/jbio.202300360>



- [21] K. Thorn, “A quick guide to light microscopy in cell biology,” *Molecular Biology of the Cell*, vol. 27, no. 2, pp. 219–222, Jan. 2016, doi: 10.1091/mbc.e15-02-0088.
- [22] “Fluorescence Imaging Techniques and Analysis Methods,” *Sensors and Probes for Bioimaging*, pp. 313–327, Jun. 2023, doi: 10.1002/9783527834365.ch10.
- [23] C. Bhan, P. Dipankar, S. P. Dash, P. Chakraborty, N. Dalpati, and P. P. Sarangi, “Recent Advances in Imaging and Analysis of Cellular Dynamics in Real Time,” *Biotechnological Advances for Microbiology, Molecular Biology, and Nanotechnology*, pp. 311–350, Feb. 2022, doi: 10.1201/9781003161158-15.
- [24] R. S. Randall *et al.*, “Image analysis workflows to reveal the spatial organization of cell nuclei and chromosomes,” *Nucleus*, vol. 13, no. 1, pp. 279–301, Nov. 2022, doi: 10.1080/19491034.2022.2144013.
- [25] D. Merenich, K. E. Van Manen-Brush, C. Janetopoulos, and K. A. Myers, “Advanced microscopy techniques for the visualization and analysis of cell behaviors,” *Cell Movement in Health and Disease*, pp. 303–321, 2022, doi: 10.1016/b978-0-323-90195-6.00010-3.
- [26] T. Wen *et al.*, “Review of research on the instance segmentation of cell images,” *Computer Methods and Programs in Biomedicine*, vol. 227, p. 107211, Dec. 2022, doi: 10.1016/j.cmpb.2022.107211.
- [27] R. Yazdi and H. Khotanlou, “A survey on automated cell tracking: challenges and solutions,” *Multimedia Tools and Applications*, Mar. 2024, **Published**, doi: 10.1007/s11042-024-18697-9.
- [28] T. Stiehl and A. Marciniak-Czochra, “How to Characterize Stem Cells? Contributions from Mathematical Modeling,” *Current Stem Cell Reports*, vol. 5, no. 2, pp. 57–65, May 2019, doi: 10.1007/s40778-019-00155-0.
- [29] M. T. Figge, “Quantitative bioimage analysis of cell characteristics,” *Cytometry Part A*, vol. 93, no. 3, pp. 278–280, Mar. 2018, doi: 10.1002/cyto.a.23349.
- [30] M. Sonka, V. Hlavac, and R. Boyle, “Image Processing, Analysis and Machine Vision,” 1993, **Published**, doi: 10.1007/978-1-4899-3216-7.

- [31] C. J. Soelistyo, K. Ulicna, and A. R. Lowe, “Machine learning enhanced cell tracking,” *Frontiers in Bioinformatics*, vol. 3, Jul. 2023, doi: 10.3389/fbinf.2023.1228989.
- [32] W.-C. Tang, “Cell movement during development,” *Cell Movement in Health and Disease*, pp. 151–157, 2022, doi: 10.1016/b978-0-323-90195-6.00004-8.
- [33] F. H. Stephenson, “Cell Growth,” *Calculations for Molecular Biology and Biotechnology*, pp. 42–76, 2003, doi: 10.1016/b978-012665751-7/50044-5.
- [34] H.-J. You and S.-K. Han, “Cell Therapy for Wound Healing,” *Journal of Korean Medical Science*, vol. 29, no. 3, p. 311, 2014, doi: 10.3346/jkms.2014.29.3.311.
- [35] Y. Zhao, L. Gu, H. Sun, X. Sha, and W. J. Li, “Physical Cytometry: Detecting Mass-Related Properties of Single Cells,” *ACS Sensors*, vol. 7, no. 1, pp. 21–36, Jan. 2022, doi: 10.1021/acssensors.1c01787.
- [36] “Cell Structure and Function,” *Microbial Physiology*, pp. 277–349, Jun. 2002, doi: 10.1002/0471223867.ch7.
- [37] J. A. I. Johnson *et al.*, “Digitize your Biology! Modeling multicellular systems through interpretable cell behavior,” Sep. 2023, Published, doi: 10.1101/2023.09.17.557982.
- [38] D. Xia *et al.*, “Advances in Targeting Drug Biological Carriers for Enhancing Tumor Therapy Efficacy,” *Macromolecular Bioscience*, vol. 23, no. 12, Aug. 2023, doi: 10.1002/mabi.202300178.
- [39] *The Cell: A Molecular Approach. 2nd Edition.* 2000. [Online]. Available: [http://books.google.ie/books?id=c75DzQEACAAJ&dq=The+Cell:+A+Molecular+Approach.+2nd+edition.&hl=&cd=2&source=gbs\\_api](http://books.google.ie/books?id=c75DzQEACAAJ&dq=The+Cell:+A+Molecular+Approach.+2nd+edition.&hl=&cd=2&source=gbs_api)
- [40] A. Kintonova, A. Sabitov, I. Povkhan, D. Khaimulina, and G. Gabdreshov, “Organization of online learning using the intelligent metasystem of open semantic technology for intelligent systems,” *Eastern-European Journal of Enterprise Technologies*, vol. 1, no. 2 (121), pp. 29–40, Feb. 2023, doi: 10.15587/1729-4061.2023.272952.
- [41] H. M. Taïeb, L. Bertinetti, T. Robinson, and A. Cipitria, “FUCCItrack: An all-in-one software for single cell tracking and cell cycle analysis,” *PLOS ONE*, vol. 17, no. 7, p. e0268297, Jul. 2022, doi: 10.1371/journal.pone.0268297.

- [42] Y. Han *et al.*, “Label-Free Mammalian Cell Tracking Enhanced by Precomputed Velocity Fields,” Jan. 2023, **Published**, doi: 10.1101/2023.01.25.525598.
- [43] S. Amarteifio, T. Fallesen, G. Pruessner, and G. Sena, “A random-sampling approach to track cell divisions in time-lapse fluorescence microscopy,” *Plant Methods*, vol. 17, no. 1, Mar. 2021, doi: 10.1186/s13007-021-00723-8.
- [44] K. R. Begnini, L. C. Pereira, J. L. Faccioni, G. Lenz, and E. C. Filippi-Chiela, “Bioimaging approaches for quantification of individual cell behavior during cell fate decisions,” *Biochemical Society Transactions*, vol. 50, no. 1, pp. 513–527, Feb. 2022, doi: 10.1042/bst20210534.
- [45] C. Bhan, P. Dipankar, S. P. Dash, P. Chakraborty, N. Dalpati, and P. P. Sarangi, “Recent Advances in Imaging and Analysis of Cellular Dynamics in Real Time,” *Biotechnological Advances for Microbiology, Molecular Biology, and Nanotechnology*, pp. 311–350, Feb. 2022, doi: 10.1201/9781003161158-15.
- [46] R. Liu, S. Xia, and H. Li, “Native top-down mass spectrometry for higher-order structural characterization of proteins and complexes,” *Mass Spectrometry Reviews*, vol. 42, no. 5, pp. 1876–1926, Jun. 2022, doi: 10.1002/mas.21793.
- [47] A. S.-Y. Leong, C. Suthipintawong, and S. Vinyuvat, “Immunostaining of Cytologic Preparations: A Review of Technical Problems,” *Applied Immunohistochemistry & Molecular Morphology*, vol. 7, no. 3, p. 214, Sep. 1999, doi: 10.1097/00129039-199909000-00007.
- [48] C. Roffay *et al.*, “Technical insights into fluorescence lifetime microscopy of mechanosensitive Flipper probes,” Sep. 2022, **Published**, doi: 10.1101/2022.09.28.509885.
- [49] S. Badillo *et al.*, “An Introduction to Machine Learning,” *Clinical Pharmacology & Therapeutics*, vol. 107, no. 4, pp. 871–885, Mar. 2020, doi: 10.1002/cpt.1796.
- [50] H.-G. Lee and S.-C. Lee, “Morphological Multi-Cell Discrimination for Robust Cell Segmentation,” *IEEE Access*, vol. 8, pp. 49837–49847, 2020, doi: 10.1109/access.2020.2980561.

- [51] Z. Wang, “Cell Segmentation for Image Cytometry: Advances, Insufficiencies, and Challenges,” *Cytometry Part A*, vol. 95, no. 7, pp. 708–711, Dec. 2018, doi: 10.1002/cyto.a.23686.
- [52] K. Ruzaeva, J.-C. Cohrs, K. Kasahara, D. Kohlheyer, K. Nöh, and B. Berkels, “Cell tracking for live-cell microscopy using an activity-prioritized assignment strategy,” *2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)*, Dec. 2022, **Published**, doi: 10.1109/ipas55744.2022.10053036.
- [53] D. Jin, Y. Sung, N. Lue, Y. Kim, P. T. C. So, and Z. Yaqoob, “Large population cell characterization using quantitative phase cytometer,” *Cytometry Part A*, vol. 91, no. 5, pp. 450–459, Apr. 2017, doi: 10.1002/cyto.a.23106.
- [54] E. Zerhouni, B. Prisacari, Q. Zhong, P. Wild, and M. Gabrani, “Heterogeneity characterization of immunohistochemistry stained tissue using convolutional autoencoder,” *SPIE Proceedings*, Mar. 2017, **Published**, doi: 10.1117/12.2256238.
- [55] J. Demongeot, J. Bezy-Wendling, J. Mattes, P. Haignon, N. Glade, and J. L. Coatrieux, “Multiscale modeling and imaging: the challenges of biocomplexity,” *Proceedings of the IEEE*, vol. 91, no. 10, pp. 1723–1737, Oct. 2003, doi: 10.1109/jproc.2003.817878.
- [56] P. Wehrli *et al.*, “Spatial Chemometrics and Comprehensive Chemical Imaging based Molecular Histopathology Delineates Anatomical Heterogeneity at Single Pixel Resolution,” Feb. 2020, **Published**, doi: 10.26434/chemrxiv.11800209.v1.
- [57] F. Shi, “High Dimensional Statistical and Computational Methods for Knowledge Discovery and Data Mining in Biomedical Data,” 2018. <https://escholarship.org/uc/item/2gn5j62f>
- [58] A. H. K. Roeder, A. Cunha, M. C. Burl, and E. M. Meyerowitz, “A computational image analysis glossary for biologists,” *Development*, vol. 139, no. 17, pp. 3071–3080, Sep. 2012, doi: 10.1242/dev.076414.
- [59] K. D. Ahlquist, L. A. Sugden, and S. Ramachandran, “Enabling interpretable machine learning for biological data with reliability scores,” *PLOS Computational Biology*, vol. 19, no. 5, p. e1011175, May 2023, doi: 10.1371/journal.pcbi.1011175.

- [60] A. P. Browning, “Model complexity in biology and bioengineering,” **Published**, doi: 10.5204/thesis.eprints.227787.
- [61] X. Tang *et al.*, “Explainable multi-task learning for multi-modality biological data analysis,” *Nature Communications*, vol. 14, no. 1, May 2023, doi: 10.1038/s41467-023-37477-x.
- [62] L. Zhang, M. Karimzadeh, M. Welch, C. McIntosh, and B. Wang, “Analytics methods and tools for integration of biomedical data in medicine,” *Artificial Intelligence in Medicine*, pp. 113–129, 2021, doi: 10.1016/b978-0-12-821259-2.00007-7.
- [63] E. Meijering, V. D. Calhoun, G. Menegaz, D. J. Miller, and J. C. Ye, “Deep Learning in Biological Image and Signal Processing [From the Guest Editors],” *IEEE Signal Processing Magazine*, vol. 39, no. 2, pp. 24–26, Mar. 2022, doi: 10.1109/msp.2021.3134525.
- [64] R. C. Sobti *et al.*, “Emerging techniques in biological sciences,” *Advances in Animal Experimentation and Modeling*, pp. 3–18, 2022, doi: 10.1016/b978-0-323-90583-1.00013-1.
- [65] Y. Kanai, “Image segmentation using intensity and color information,” *SPIE Proceedings*, Jan. 1998, Published, doi: 10.1117/12.298383.
- [66] S. Chakraborty, “An Advanced Approach to Detect Edges of Digital Images for Image Segmentation,” *Advances in Computational Intelligence and Robotics*, pp. 90–118, 2020, doi: 10.4018/978-1-7998-2736-8.ch004.
- [67] Noor khalid, “Review on Region-Based Segmentation Using Watershed and Region Growing Techniques and their Applications in Different Fields,” *Journal La Multiapp*, vol. 3, no. 5, pp. 241–249, Nov. 2022, doi: 10.37899/journallamultiapp.v3i5.714.
- [68] G. Kaur and R. Kumar, “Comparative Analysis of Image Segmentation using Thresholding,” *International Journal of Engineering and Applied Sciences (IJEAS)*, vol. 7, no. 6, Jun. 2020, doi: 10.31873/ijeas.7.06.12.
- [69] Z. Suhail and R. Zwiggelaar, “Histogram-based approach for mass segmentation in mammograms,” *15th International Workshop on Breast Imaging (IWBI2020)*, May 2020, Published, doi: 10.1117/12.2563621.

- [70] D. Zahorodnia, Y. Pigovsky, and P. Bykovyy, "CANNY-BASED METHOD OF IMAGE CONTOUR SEGMENTATION," *International Journal of Computing*, pp. 200–205, Sep. 2016, doi: 10.47839/ijc.15.3.853.
- [71] S. GL and B. E, "Sobel edge detection technique implementation for image steganography analysis," *Biomedical Research*, 2018, Published, doi: 10.4066/biomedicalresearch.29-17-1212.
- [72] H. Kong, H. C. Akakin, and S. E. Sarma, "A Generalized Laplacian of Gaussian Filter for Blob Detection and Its Applications," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1719–1733, Dec. 2013, doi: 10.1109/tsmcb.2012.2228639.
- [73] C. Li, J. Huang, X. Tian, and L. Tang, "Application of the Watershed Algorithm in the Image Segmentation," *Advances in Intelligent Systems and Computing*, pp. 1676–1680, 2020, doi: 10.1007/978-981-15-2568-1\_235.
- [74] A. Chaturvedi, R. Khanna, and V. Kumar, "An Analysis of Region Growing Image Segmentation Schemes," *International Journal of Computer Trends and Technology*, vol. 34, no. 1, pp. 46–51, Apr. 2016, doi: 10.14445/22312803/ijctt-v34p108.
- [75] D. Chudasama, T. Patel, S. Joshi, and G. I. Prajapati, "Image Segmentation using Morphological Operations," *International Journal of Computer Applications*, vol. 117, no. 18, pp. 16–19, May 2015, doi: 10.5120/20654-3197.
- [76] C.-S. Chang, J.-J. Ding, Y.-F. Wu, and S.-J. Lin, "Cell Segmentation Algorithm Using Double Thresholding with Morphology-Based Techniques," 2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), May 2018, Published, doi: 10.1109/icce-china.2018.8448467.
- [77] F. Mostajer Kheirkhah, H. R. Sadegh Mohammadi, and A. Shahverdi, "Modified histogram-based segmentation and adaptive distance tracking of sperm cells image sequences," *Computer Methods and Programs in Biomedicine*, vol. 154, pp. 173–182, Feb. 2018, doi: 10.1016/j.cmpb.2017.11.005.
- [78] G. L. B. Ramalho, D. S. Ferreira, A. G. C. Bianchi, and C. M. Carneiro, "CELL RECONSTRUCTION UNDER VORONOI AND ENCLOSING ELLIPSES FROM 3D MICROSCOPY," in *Proc. IEEE Int. Symp. Biomed. Imaging*, 2015, pp. 1–2.

- [79] J. CHALFOUN, M. MAJURSKI, A. PESKIN, C. BREEN, P. BAJCSY, and M. BRADY, “Empirical gradient threshold technique for automated segmentation across image modalities and cell lines,” *Journal of Microscopy*, vol. 260, no. 1, pp. 86–99, Jun. 2015, doi: 10.1111/jmi.12269.
- [80] H. Xu, C. Lu, R. Berendt, N. Jha, and M. Mandal, “Automatic Nuclei Detection Based on Generalized Laplacian of Gaussian Filters,” *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 3, pp. 826–837, May 2017, doi: 10.1109/jbhi.2016.2544245.
- [81] R. Rahali, Y. B. Salem, N. Dridi, and H. Dahman, “New foreground markers for Drosophila cell segmentation using marker-controlled watershed,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 12, no. 5, p. 5055, Oct. 2022, doi: 10.11591/ijece.v12i5.pp5055-5062.
- [82] R. Stoklasa, L. Balek, P. Krejci, and P. Matula, “Automated cell segmentation in phase-contrast images based on classification and region growing,” *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*, Apr. 2015, Published, doi: 10.1109/isbi.2015.7164149.
- [83] F. Zhang, Y. Wu, M. Xu, S. Liu, C. Peng, and Z. Gao, “A Morphological Image Segmentation Algorithm for Circular Overlapping Cells,” *Intelligent Automation & Soft Computing*, vol. 32, no. 1, pp. 301–321, 2022, doi: 10.32604/iasc.2022.021929.
- [84] S. Dimopoulos, C. E. Mayer, F. Rudolf, and J. Stelling, “Accurate cell segmentation in microscopy images using membrane patterns,” *Bioinformatics*, vol. 30, no. 18, pp. 2644–2651, May 2014, doi: 10.1093/bioinformatics/btu302.
- [85] J. Chalfoun, M. Majurski, A. Dima, C. Stuelten, A. Peskin, and M. Brady, “FogBank: a single cell segmentation across multiple cell lines and image modalities,” *BMC Bioinformatics*, vol. 15, no. 1, Dec. 2014, doi: 10.1186/s12859-014-0431-x.
- [86] K. E. G. Magnusson, J. Jalden, P. M. Gilbert, and H. M. Blau, “Global Linking of Cell Tracks Using the Viterbi Algorithm,” *IEEE Transactions on Medical Imaging*, vol. 34, no. 4, pp. 911–929, Apr. 2015, doi: 10.1109/tmi.2014.2370951.

- [87] C. Versari et al., “Long-term tracking of budding yeast cells in brightfield microscopy: CellStar and the Evaluation Platform,” *Journal of The Royal Society Interface*, vol. 14, no. 127, p. 20160705, Feb. 2017, doi: 10.1098/rsif.2016.0705.
- [88] D. Vasilescu and M. Filzmoser, "Machine invention systems: a (r)evolution of the invention process?", *AI & SOCIETY*, vol. 36, no. 3, p. 829-837, 2020. <https://doi.org/10.1007/s00146-020-01080-1>
- [89] O. Hilsenbeck et al., “fastER: a user-friendly tool for ultrafast and robust cell segmentation in large-scale microscopy,” *Bioinformatics*, vol. 33, no. 13, pp. 2020–2028, Feb. 2017, doi: 10.1093/bioinformatics/btx107.
- [90] T. J. Cleophas and A. H. Zwinderman, “Support Vector Machines,” *Machine Learning in Medicine*, pp. 155–161, 2013, doi: 10.1007/978-94-007-6886-4\_15.
- [91] I. Arganda-Carreras et al., “Trainable Weka Segmentation: a machine learning tool for microscopy pixel classification,” *Bioinformatics*, vol. 33, no. 15, pp. 2424–2426, Mar. 2017, doi: 10.1093/bioinformatics/btx180.
- [92] D. E. Hernandez, S. W. Chen, E. E. Hunter, E. B. Steager, and V. Kumar, “Cell Tracking with Deep Learning and the Viterbi Algorithm,” 2018 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS), Jul. 2018, Published, doi: 10.1109/marss.2018.8481231.
- [93] S. Berg et al., “ilastik: interactive machine learning for (bio)image analysis,” *Nature Methods*, vol. 16, no. 12, pp. 1226–1232, Sep. 2019, doi: 10.1038/s41592-019-0582-9.
- [94] H.-F. Tsai, J. Gajda, T. F. W. Sloan, A. Rares, and A. Q. Shen, “Usiigaci: Instance-aware cell tracking in stain-free phase contrast microscopy enabled by machine learning,” *SoftwareX*, vol. 9, pp. 230–237, Jan. 2019, doi: 10.1016/j.softx.2019.02.007.
- [95] W. Wang et al., “Learn to segment single cells with deep distance estimator and deep cell detector,” *Computers in Biology and Medicine*, vol. 108, pp. 133–141, May 2019, doi: 10.1016/j.combiomed.2019.04.006.



- [96] Z. Zhou, F. Wang, W. Xi, H. Chen, P. Gao, and C. He, “Joint Multi-frame Detection and Segmentation for Multi-cell Tracking,” *Lecture Notes in Computer Science*, pp. 435–446, 2019, doi: 10.1007/978-3-030-34110-7\_36.
- [97] A. Arbelle and T. R. Raviv, “Microscopy Cell Segmentation Via Convolutional LSTM Networks,” 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019), Apr. 2019, Published, doi: 10.1109/isbi.2019.8759447.
- [98] A. Ayanzadeh, O. Y. Ozuysal, D. P. Okvur, S. Onal, B. U. Tgreyin, and D. Unay, “Deep Learning based Segmentation Pipeline for Label-Free Phase-Contrast Microscopy Images,” 2020 28th Signal Processing and Communications Applications Conference (SIU), Oct. 2020, Published, doi: 10.1109/siu49456.2020.9302304.
- [99] E. Fazeli et al., “Automated cell tracking using StarDist and TrackMate,” *F1000Research*, vol. 9, p. 1279, Dec. 2020, doi: 10.12688/f1000research.27019.2.
- [100] J.-B. Lugagne, H. Lin, and M. J. Dunlop, “DeLTA: Automated cell segmentation, tracking, and lineage reconstruction using deep learning,” *PLOS Computational Biology*, vol. 16, no. 4, p. e1007673, Apr. 2020, doi: 10.1371/journal.pcbi.1007673.
- [101] T. Scherr, K. Löffler, M. Böhlend, and R. Mikut, “Cell segmentation and tracking using CNN-based distance predictions and a graph-based matching strategy,” *PLOS ONE*, vol. 15, no. 12, p. e0243219, Dec. 2020, doi: 10.1371/journal.pone.0243219.
- [102] Y. Chen et al., “Celltrack R-CNN: A Novel End-To-End Deep Neural Network For Cell Segmentation And Tracking In Microscopy Images,” 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI), Apr. 2021, Published, doi: 10.1109/isbi48211.2021.9434057.
- [103] L. von Chamier et al., “Democratising deep learning for microscopy with ZeroCostDL4Mic,” *Nature Communications*, vol. 12, no. 1, Apr. 2021, doi: 10.1038/s41467-021-22518-0.
- [104] A. Zargari et al., “DeepSea is an efficient deep-learning model for single-cell segmentation and tracking in time-lapse microscopy,” *Cell Reports Methods*, vol. 3, no. 6, p. 100500, Jun. 2023, doi: 10.1016/j.crmeth.2023.100500.

- [105] T. Vicar et al., “Self-supervised pretraining for transferable quantitative phase image cell segmentation,” *Biomedical Optics Express*, vol. 12, no. 10, p. 6514, Sep. 2021, doi: 10.1364/boe.433212.
- [106] C. Wen et al., “3DeeCellTracker, a deep learning-based pipeline for segmenting and tracking cells in 3D time lapse images,” *eLife*, vol. 10, Mar. 2021, doi: 10.7554/elife.59187.
- [107] S. M. Mota et al., “Automated mesenchymal stem cell segmentation and machine learning-based phenotype classification using morphometric and textural analysis,” *Journal of Medical Imaging*, vol. 8, no. 01, Feb. 2021, doi: 10.1117/1.jmi.8.1.014503.
- [108] J. Schindelin et al., “Fiji: an open-source platform for biological-image analysis,” *Nature Methods*, vol. 9, no. 7, pp. 676–682, Jun. 2012, doi: 10.1038/nmeth.2019.
- [109] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *Lecture Notes in Computer Science*, pp. 234–241, 2015, doi: 10.1007/978-3-319-24574-4\_28.
- [110] K. Minton, “A safety net for successful mitosis,” *Nature Reviews Molecular Cell Biology*, vol. 15, no. 4, pp. 220–221, Mar. 2014, doi: 10.1038/nrm3777.
- [111] Y. Wang, H. Mao, and Z. Yi, “Stem cell motion-tracking by using deep neural networks with multi-output,” *Neural Computing and Applications*, vol. 31, no. 8, pp. 3455–3467, Nov. 2017, doi: 10.1007/s00521-017-3291-2.
- [112] Q. Jiang et al., “An Advanced Framework for Time-lapse Microscopy Image Analysis,” Sep. 2020, Published, doi: 10.1101/2020.09.21.303800.
- [113] F. Arroyo, V. Mitrana, A. Păun, and M. Păun, “A Multi-agent Model for Cell Population,” *Agents and Multi-agent Systems: Technologies and Applications 2019*, pp. 95–104, Jun. 2019, doi: 10.1007/978-981-13-8679-4\_8.
- [114] H. Yoshida, Y. Miwa, and M. Kaneko, “Elliptic curves and Fibonacci numbers arising from Lindenmayer system with symbolic computation,” *Applicable Algebra in Engineering, Communication and Computing*, vol. 22, no. 2, pp. 147–164, Mar. 2011, doi: 10.1007/s00200-011-0143-7.

- [115] J. S. Yu and N. Bagheri, “Agent-Based Models Predict Emergent Behavior of Heterogeneous Cell Populations in Dynamic Microenvironments,” *Frontiers in Bioengineering and Biotechnology*, vol. 8, Jun. 2020, doi: 10.3389/fbioe.2020.00249.
- [116] T. Kihara, K. Kashitani, and J. Miyake, “In silico characterization of cell–cell interactions using a cellular automata model of cell culture,” *BMC Research Notes*, vol. 10, no. 1, Jul. 2017, doi: 10.1186/s13104-017-2613-x.
- [117] J. Li, S. K. Schnyder, M. S. Turner, and R. Yamamoto, “Role of the Cell Cycle in Collective Cell Dynamics,” *Physical Review X*, vol. 11, no. 3, Jul. 2021, doi: 10.1103/physrevx.11.031025.
- [118] R. Wiecek, “Hydrodynamic limit of a stochastic model of proliferating cells with chemotaxis,” *Kinetic and Related Models*, vol. 16, no. 3, pp. 373–393, 2023, doi: 10.3934/krm.2022032.
- [119] Z. Shen et al., “Deciphering controversial results of cell proliferation on TiO<sub>2</sub> nanotubes using machine learning,” *Regenerative Biomaterials*, vol. 8, no. 4, Jun. 2021, doi: 10.1093/rb/rbab025.
- [120] J. Ma et al., “Using deep learning to model the hierarchical structure and function of a cell,” *Nature Methods*, vol. 15, no. 4, pp. 290–298, Mar. 2018, doi: 10.1038/nmeth.4627.
- [121] G. H. Balde, Md. A. Ala Walid, S. P. V. R. Yella, M. Soumya, and R. Rastogi, “A Novel Cell Density Prediction Design using Optimal Deep Learning with Salp Swarm Algorithm,” 2023 7th International Conference on Trends in Electronics and Informatics (ICOEI), Apr. 2023, Published, doi: 10.1109/icoei56765.2023.10125711.
- [122] L. Wiggins et al., “The CellPhe toolkit for cell phenotyping using time-lapse imaging and pattern recognition,” *Nature Communications*, vol. 14, no. 1, Apr. 2023, doi: 10.1038/s41467-023-37447-3.
- [123] R. Li, Q. Gao, and K. Rohr, “Multi-Object Dynamic Memory Network For Cell Tracking in Time-Lapse Microscopy Images,” 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI), Apr. 2021, Published, doi: 10.1109/isbi48211.2021.9433828.

- [124] M. Gamarra, E. Zurek, H. J. Escalante, L. Hurtado, and H. San-Juan-Vergara, “Split and merge watershed: A two-step method for cell segmentation in fluorescence microscopy images,” *Biomedical Signal Processing and Control*, vol. 53, p. 101575, Aug. 2019, doi: 10.1016/j.bspc.2019.101575.
- [125] S. P. Shen et al., “Automatic Cell Segmentation by Adaptive Thresholding (ACSAT) for Large-Scale Calcium Imaging Datasets,” *eneuro*, vol. 5, no. 5, p. ENEURO.0056-18.2018, Sep. 2018, doi: 10.1523/eneuro.0056-18.2018.
- [126] P. K. S. Tam, “Modular expert network approach to histogram thresholding,” *Journal of Electronic Imaging*, vol. 6, no. 3, p. 286, Jul. 1997, doi: 10.1117/12.269904.
- [127] Y. Fang and B. Zhong, “Cell segmentation in fluorescence microscopy images based on multi-scale histogram thresholding,” *Mathematical Biosciences and Engineering*, vol. 20, no. 9, pp. 16259–16278, 2023, doi: 10.3934/mbe.2023726.
- [128] Z. Wang and H. Li, “Generalizing cell segmentation and quantification,” *BMC Bioinformatics*, vol. 18, no. 1, Mar. 2017, doi: 10.1186/s12859-017-1604-1.
- [129] S. Kaliman, C. Jayachandran, F. Rehfeldt, and A.-S. Smith, “Limits of Applicability of the Voronoi Tessellation Determined by Centers of Cell Nuclei to Epithelium Morphology,” *Frontiers in Physiology*, vol. 7, Nov. 2016, doi: 10.3389/fphys.2016.00551.
- [130] Y. Xu, T. Wu, F. Gao, J. R. Charlton, and K. M. Bennett, “Improved small blob detection in 3D images using jointly constrained deep learning and Hessian analysis,” *Scientific Reports*, vol. 10, no. 1, Jan. 2020, doi: 10.1038/s41598-019-57223-y.
- [131] R. D and Dr. M. Rafi, “A REVIEW ON MARKER-CONTROLLED WATER SHED,” *International Journal of Engineering Applied Sciences and Technology*, vol. 7, no. 4, pp. 138–141, Aug. 2022, doi: 10.33564/ijeast.2022.v07i04.019.
- [132] W. Han, A. M. Cheung, M. J. Yaffe, and A. L. Martel, “Cell segmentation for immunofluorescence multiplexed images using two-stage domain adaptation and weakly labelled data for pre-training,” *Scientific Reports*, vol. 12, no. 1, Mar. 2022, doi: 10.1038/s41598-022-08355-1.

- [133] T. Vicar et al., “Cell segmentation methods for label-free contrast microscopy: review and comprehensive comparison,” *BMC Bioinformatics*, vol. 20, no. 1, Jun. 2019, doi: 10.1186/s12859-019-2880-8.
- [134] K. O. Oyeboode and J. R. Tapamo, “ADAPTIVE PARAMETER SELECTION FOR GRAPH CUT-BASED SEGMENTATION ON CELL IMAGES,” *Image Analysis & Stereology*, vol. 35, no. 1, p. 29, Mar. 2016, doi: 10.5566/ias.1333.
- [135] J. QI, “Dense nuclei segmentation based on graph cut and convexity–concavity analysis,” *Journal of Microscopy*, vol. 253, no. 1, pp. 42–53, Nov. 2013, doi: 10.1111/jmi.12096.
- [136] Y. Al-Kofahi, W. Lassoued, W. Lee, and B. Roysam, “Improved Automatic Detection and Segmentation of Cell Nuclei in Histopathology Images,” *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 4, pp. 841–852, Apr. 2010, doi: 10.1109/tbme.2009.2035102.
- [137] A. Boroomand, A. Wong, and K. Bizheva, “A Conditional Random Field Weakly Supervised Segmentation Approach for Segmenting keratocytes Cells in Corneal Optical Coherence Tomography Images,” *Vision Letters*, vol. 1, no. 1, Oct. 2015, doi: 10.15353/vsnl.v1i1.48.
- [138] P. Dave et al., “MIMO U-Net: efficient cell segmentation and counting in microscopy image sequences,” *Medical Imaging 2023: Digital and Computational Pathology*, Apr. 2023, Published, doi: 10.1117/12.2655627.
- [139] Z. Wang and H. Li, “Generalizing cell segmentation and quantification,” *BMC Bioinformatics*, vol. 18, no. 1, Mar. 2017, doi: 10.1186/s12859-017-1604-1.
- [140] Z. Wang, “Cell Segmentation for Image Cytometry: Advances, Insufficiencies, and Challenges,” *Cytometry Part A*, vol. 95, no. 7, pp. 708–711, Dec. 2018, doi: 10.1002/cyto.a.23686.
- [141] F. Li, X. Zhou, H. Zhao, and S. T. C. Wong, “Cell Segmentation Using Front Vector Flow Guided Active Contours,” *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2009*, pp. 609–616, 2009, doi: 10.1007/978-3-642-04271-3\_74.

- [142] C. Collewet, “Polar snakes: A fast and robust parametric active contour model,” 2009 16th IEEE International Conference on Image Processing (ICIP), Nov. 2009, Published, doi: 10.1109/icip.2009.5414444.
- [143] E. Karami, M. S. Shehata, and A. Smith, “Adaptive Polar Active Contour for Segmentation and Tracking in Ultrasound Videos,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 4, pp. 1209–1222, Apr. 2019, doi: 10.1109/tcsvt.2018.2818072.
- [144] G. Paul, T. Varghese, K. V. Purushothaman, and A. Singh, “Automated Segmentation of MR Images by Implementing Multi SVM Technique,” *Lecture Notes in Electrical Engineering*, pp. 1509–1516, Nov. 2014, doi: 10.1007/978-81-322-2119-7\_147.
- [145] W. Wu, H. Pan, Y. An, and G. Cheng, “Image Segmentation Based on Ball Vector Machine,” 2012 International Conference on Computer Science and Service System, Aug. 2012, Published, doi: 10.1109/csss.2012.567.
- [146] S. Winfree, “User-Accessible Machine Learning Approaches for Cell Segmentation and Analysis in Tissue,” *Frontiers in Physiology*, vol. 13, Mar. 2022, doi: 10.3389/fphys.2022.833333.
- [147] D. Lu et al., “Deep-learning based multiclass retinal fluid segmentation and detection in optical coherence tomography images using a fully convolutional neural network,” *Medical Image Analysis*, vol. 54, pp. 100–110, May 2019, doi: 10.1016/j.media.2019.02.011.
- [148] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *Lecture Notes in Computer Science*, pp. 234–241, 2015, doi: 10.1007/978-3-319-24574-4\_28.
- [149] S. Wang, K. He, D. Nie, S. Zhou, Y. Gao, and D. Shen, “CT male pelvic organ segmentation using fully convolutional networks with boundary sensitive representation,” *Medical Image Analysis*, vol. 54, pp. 168–178, May 2019, doi: 10.1016/j.media.2019.03.003.
- [150] F. Milletari, N. Navab, and S.-A. Ahmadi, “V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation,” 2016 Fourth International Conference on 3D Vision (3DV), Oct. 2016, Published, doi: 10.1109/3dv.2016.79.

- [151] “Classification of Acute Lymphoblastic Leukemia on White Blood Cell Microscopy Images Based on Instance Segmentation Using Mask R-CNN,” *International Journal of Intelligent Engineering and Systems*, vol. 15, no. 5, pp. 625–637, Oct. 2022, doi: 10.22266/ijies2022.1031.54.
- [152] P. Shrestha, N. Kuang, and J. Yu, “Efficient end-to-end learning for cell segmentation with machine generated weak annotations,” *Communications Biology*, vol. 6, no. 1, Mar. 2023, doi: 10.1038/s42003-023-04608-5.
- [153] R. Vasan, M. P. Rowan, C. T. Lee, G. R. Johnson, P. Rangamani, and M. Holst, “Applications and Challenges of Machine Learning to Enable Realistic Cellular Simulations,” *Frontiers in Physics*, vol. 7, Jan. 2020, doi: 10.3389/fphy.2019.00247.
- [154] G. Zhan, W. Wang, H. Sun, Y. Hou, and L. Feng, “Auto-CSC: A Transfer Learning Based Automatic Cell Segmentation and Count Framework,” *Cyborg and Bionic Systems*, vol. 2022, Jan. 2022, doi: 10.34133/2022/9842349.
- [155] C. Karabağ, M. A. Ortega-Ruiz, and C. C. Reyes-Aldasoro, “Impact of Training Data, Ground Truth and Shape Variability in the Deep Learning-Based Semantic Segmentation of HeLa Cells Observed with Electron Microscopy,” *Journal of Imaging*, vol. 9, no. 3, p. 59, Mar. 2023, doi: 10.3390/jimaging9030059.
- [156] M. Stevens, A. Nanou, L. W. M. M. Terstappen, C. Driemel, N. H. Stoecklein, and F. A. W. Coumans, “StarDist Image Segmentation Improves Circulating Tumor Cell Detection,” *Cancers*, vol. 14, no. 12, p. 2916, Jun. 2022, doi: 10.3390/cancers14122916.
- [157] R. Sarma and Y. K. Gupta, “A comparative study of new and existing segmentation techniques,” *IOP Conference Series: Materials Science and Engineering*, vol. 1022, no. 1, p. 012027, Jan. 2021, doi: 10.1088/1757-899x/1022/1/012027.
- [158] Z. Zhang, Q. Li, W. Song, P. Wei, and J. Guo, “A novel concavity based method for automatic segmentation of touching cells in microfluidic chips,” *Expert Systems with Applications*, vol. 202, p. 117432, Sep. 2022, doi: 10.1016/j.eswa.2022.117432.
- [159] Z. Chen, “Medical Image Segmentation Based on U-Net,” *Journal of Physics: Conference Series*, vol. 2547, no. 1, p. 012010, Jul. 2023, doi: 10.1088/1742-6596/2547/1/012010.

- [160] “TRACKING OF EXTENDED CROSSING OBJECTS USING THE VITERBI ALGORITHM,” Proceedings of the First International Conference on Informatics in Control, Automation and Robotics, 2004, Published, doi: 10.5220/0001129601420149.
- [161] C. Haubold, M. Schiegg, A. Kreshuk, S. Berg, U. Koethe, and F. A. Hamprecht, “Segmenting and Tracking Multiple Dividing Targets Using ilastik,” Focus on Bio-Image Informatics, pp. 199–229, 2016, doi: 10.1007/978-3-319-28549-8\_8.
- [162] M. Schiegg, P. Hanslovsky, B. X. Kausler, L. Hufnagel, and F. A. Hamprecht, “Conservation Tracking,” 2013 IEEE International Conference on Computer Vision, Dec. 2013, Published, doi: 10.1109/iccv.2013.364.
- [163] V. Kordic, Ed., “Kalman Filter,” May 2010, Published, doi: 10.5772/233.
- [164] K. Löffler, T. Scherr, and R. Mikut, “A graph-based cell tracking algorithm with few manually tunable parameters and automated segmentation error correction,” PLOS ONE, vol. 16, no. 9, p. e0249257, Sep. 2021, doi: 10.1371/journal.pone.0249257.
- [165] N. Chenouard et al., “Objective comparison of particle tracking methods,” Nature Methods, vol. 11, no. 3, pp. 281–289, Jan. 2014, doi: 10.1038/nmeth.2808.
- [166] “Introduction to Trackpy — trackpy 0.6.2 documentation.” <http://soft-matter.github.io/trackpy/dev/introduction.html>
- [167] F. Wang, H. Li, W. Yang, S. Jin, and P. Gao, “Cell tracking with multifeature fusion,” The Journal of Supercomputing, vol. 79, no. 17, pp. 20001–20018, Jun. 2023, doi: 10.1007/s11227-023-05384-z.
- [168] C. J. Soelistyo, K. Ulicna, and A. R. Lowe, “Machine learning enhanced cell tracking,” Frontiers in Bioinformatics, vol. 3, Jul. 2023, doi: 10.3389/fbinf.2023.1228989.
- [169] D. Lin, Y. Cheng, Y. Li, S. Prasad, and A. Guo, “MLSA-UNet: End-to-End Multi-Level Spatial Attention Guided UNet for Industrial Defect Segmentation,” 2022 IEEE International Conference on Image Processing (ICIP), Oct. 2022, Published, doi: 10.1109/icip46576.2022.9897416.



- [170] E. Arkin, N. Yadikar, X. Xu, A. Aysa, and K. Ubul, “A survey: object detection methods from CNN to transformer,” *Multimedia Tools and Applications*, vol. 82, no. 14, pp. 21353–21383, Oct. 2022, doi: 10.1007/s11042-022-13801-3.
- [171] T. He, H. Mao, J. Guo, and Z. Yi, “Cell tracking using deep neural networks with multi-task learning,” *Image and Vision Computing*, vol. 60, pp. 142–153, Apr. 2017, doi: 10.1016/j.imavis.2016.11.010.
- [172] H. Alanazi, A. J. Canul, A. Garman, J. Quimby, and A. E. Vasdekis, “Robust microbial cell segmentation by optical-phase thresholding with minimal processing requirements,” *Cytometry Part A*, vol. 91, no. 5, pp. 443–449, Mar. 2017, doi: 10.1002/cyto.a.23099.
- [173] A. X. Lu, T. Zarin, I. S. Hsu, and A. M. Moses, “YeastSpotter: accurate and parameter-free web segmentation for microscopy images of yeast cells,” *Bioinformatics*, vol. 35, no. 21, pp. 4525–4527, May 2019, doi: 10.1093/bioinformatics/btz402.
- [174] L. Chen, K. Painter, C. Surulescu, and A. Zhigun, “Mathematical models for cell migration: a non-local perspective,” *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 375, no. 1807, p. 20190379, Jul. 2020, doi: 10.1098/rstb.2019.0379.
- [175] A. H. Fathy Navid and A. Bagheri, “Cellular Learning Automata and Its Applications,” *Emerging Applications of Cellular Automata*, May 2013, Published, doi: 10.5772/52953.
- [176] L. Boyer, V. Poupet, and G. Theyssier, “On the Complexity of Limit Sets of Cellular Automata Associated with Probability Measures,” *Lecture Notes in Computer Science*, pp. 190–201, 2006, doi: 10.1007/11821069\_17.
- [177] J. T. Wootton, “Local interactions predict large-scale pattern in empirically derived cellular automata,” *Nature*, vol. 413, no. 6858, pp. 841–844, Oct. 2001, doi: 10.1038/35101595.
- [178] M. Feig and Y. Sugita, “Whole-Cell Models and Simulations in Molecular Detail,” *Annual Review of Cell and Developmental Biology*, vol. 35, no. 1, pp. 191–211, Oct. 2019, doi: 10.1146/annurev-cellbio-100617-062542.

- [179] M. Ahmadian, J. J. Tyson, J. Peccoud, and Y. Cao, “A hybrid stochastic model of the budding yeast cell cycle,” *npj Systems Biology and Applications*, vol. 6, no. 1, Mar. 2020, doi: 10.1038/s41540-020-0126-z.
- [180] J. M. Osborne, A. G. Fletcher, J. M. Pitt-Francis, P. K. Maini, and D. J. Gavaghan, “Comparing individual-based approaches to modelling the self-organization of multicellular tissues,” *PLOS Computational Biology*, vol. 13, no. 2, p. e1005387, Feb. 2017, doi: 10.1371/journal.pcbi.1005387.
- [181] C. Zhang, C. Liu, X. Zhang, and G. Almpanidis, “An up-to-date comparison of state-of-the-art classification algorithms,” *Expert Systems with Applications*, vol. 82, pp. 128–150, Oct. 2017, doi: 10.1016/j.eswa.2017.04.003.
- [182] J. Li, Y. Li, X. Xiang, S.-T. Xia, S. Dong, and Y. Cai, “TNT: An Interpretable Tree-Network-Tree Learning Framework using Knowledge Distillation,” *Entropy*, vol. 22, no. 11, p. 1203, Oct. 2020, doi: 10.3390/e22111203.
- [183] A. Natekin and A. Knoll, “Gradient boosting machines, a tutorial,” *Frontiers in Neuroinformatics*, vol. 7, 2013, doi: 10.3389/fnbot.2013.00021.
- [184] X. Huang et al., “ParsVNN: parsimony visible neural networks for uncovering cancer-specific and drug-sensitive genes and pathways,” *NAR Genomics and Bioinformatics*, vol. 3, no. 4, Oct. 2021, doi: 10.1093/nargab/lqab097.
- [185] J. Brownlee, “A Gentle Introduction to LSTM Autoencoders,” *MachineLearningMastery.com*, Aug. 27, 2020. <https://machinelearningmastery.com/lstm-autoencoders/>
- [186] H. Chen and R. F. Murphy, “Evaluation of cell segmentation methods without reference segmentations,” *Molecular Biology of the Cell*, vol. 34, no. 6, May 2023, doi: 10.1091/mbc.e22-08-0364.
- [187] M. C. Robitaille, J. M. Byers, J. A. Christodoulides, and M. P. Raphael, “Self-supervised machine learning for live cell imagery segmentation,” *Communications Biology*, vol. 5, no. 1, Nov. 2022, doi: 10.1038/s42003-022-04117-x.

- [188] Y. Wang, W. Wang, D. Liu, W. Hou, T. Zhou, and Z. Ji, “GeneSegNet: a deep learning framework for cell segmentation by integrating gene expression and imaging,” *Genome Biology*, vol. 24, no. 1, Oct. 2023, doi: 10.1186/s13059-023-03054-0.
- [189] M. C. Robitaille, J. M. Byers, J. A. Christodoulides, and M. P. Raphael, “Self-supervised machine learning for live cell imagery segmentation,” *Communications Biology*, vol. 5, no. 1, Nov. 2022, doi: 10.1038/s42003-022-04117-x.
- [190] H. S. Deter, M. Dies, C. C. Cameron, N. C. Butzin, and J. Buceta, “A Cell Segmentation/Tracking Tool Based on Machine Learning,” *Computer Optimized Microscopy*, pp. 399–422, 2019, doi: 10.1007/978-1-4939-9686-5\_19.
- [191] 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, Jun. 2012, doi: 10.1038/nmeth.2089.
- [192] S. Berg et al., “ilastik: interactive machine learning for (bio)image analysis,” *Nature Methods*, vol. 16, no. 12, pp. 1226–1232, Sep. 2019, doi: 10.1038/s41592-019-0582-9.
- [193] D. R. Stirling, M. J. Swain-Bowden, A. M. Lucas, A. E. Carpenter, B. A. Cimini, and A. Goodman, “CellProfiler 4: improvements in speed, utility and usability,” *BMC Bioinformatics*, vol. 22, no. 1, Sep. 2021, doi: 10.1186/s12859-021-04344-9.
- [194] M. I. A. Barbosa, J. Belinha, R. M. N. Jorge, and A. X. Carvalho, “Computationally modelling cell proliferation: A review,” *International Journal for Numerical Methods in Biomedical Engineering*, vol. 39, no. 7, May 2023, doi: 10.1002/cnm.3715.
- [195] R. Kruse, S. Mostaghim, C. Borgelt, C. Braune, and M. Steinbrecher, “Introduction to Evolutionary Algorithms,” *Texts in Computer Science*, pp. 225–254, 2022, doi: 10.1007/978-3-030-42227-1\_11.
- [196] A. Slowik and H. Kwasnicka, “Evolutionary algorithms and their applications to engineering problems,” *Neural Computing and Applications*, vol. 32, no. 16, pp. 12363–12379, Mar. 2020, doi: 10.1007/s00521-020-04832-8.
- [197] Southgate, J., Masters, J. R., & Trejdosiewicz, L. K. (2002, April 12). Culture of Human Urothelium. Culture of Epithelial Cells, 381–399. <https://doi.org/10.1002/0471221201.ch12>

- [198] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), Dec. 2016, Published, doi: 10.1109/icgtspicc.2016.7955308.
- [199] T. Yu, R. Riolo, and B. Worzel, "Genetic Programming: Theory and Practice," Genetic Programming Theory and Practice III, pp. 1–14, doi: 10.1007/0-387-28111-8\_1.
- [200] "Product - Sarstedt." <https://www.sarstedt.com/en/products/laboratory/cell-tissue-culture/cultivation/product/83.3921/>
- [201] K. Tzavlaki and A. Moustakas, "TGF- $\beta$  Signaling," *Biomolecules*, vol. 10, no. 3, p. 487, Mar. 2020, doi: 10.3390/biom10030487.
- [202] S. K. Halder, R. D. Beauchamp, and P. K. Datta, "A Specific Inhibitor of TGF- $\beta$  Receptor Kinase, SB-431542, as a Potent Antitumor Agent for Human Cancers," *Neoplasia*, vol. 7, no. 5, pp. 509–521, May 2005, doi: 10.1593/neo.04640.
- [203] "Time-lapse microscopy," Wikipedia, Nov. 03, 2023. [https://en.wikipedia.org/wiki/Time-lapse\\_microscopy](https://en.wikipedia.org/wiki/Time-lapse_microscopy)
- [204] J.-A. Conchello and J. W. Lichtman, "Optical sectioning microscopy," *Nature Methods*, vol. 2, no. 12, pp. 920–931, Nov. 2005, doi: 10.1038/nmeth815.
- [205] Chalfie, M., & Kain, S. R. (2005). *Green fluorescent protein: Properties, applications, and protocols*. John Wiley & Sons.
- [206] Zernike, F. (1942). Phase contrast, a new method for the microscopic observation of transparent objects. *Physica*, 9(7), 686-698. [https://doi.org/10.1016/S0031-8914\(42\)80035-X](https://doi.org/10.1016/S0031-8914(42)80035-X)
- [207] Planchon, T. A., Gao, L., Milkie, D. E., Davidson, M. W., Galbraith, J. A., Galbraith, C. G., & Betzig, E. (2011). Rapid three-dimensional isotropic imaging of living cells using Bessel beam plane illumination. *Nature Methods*, 8(5), 417-423. <https://doi.org/10.1038/nmeth.1586>
- [208] Inoué, S. (2006). Foundations of confocal scanned imaging in light microscopy. In *Handbook of Biological Confocal Microscopy* (pp. 1-16). Springer, Boston, MA. [https://doi.org/10.1007/978-0-387-45524-2\\_1](https://doi.org/10.1007/978-0-387-45524-2_1)

- [209] “Differential Interference Contrast - Comparison of Phase Contrast and DIC Microscopy | Olympus LS.” <https://www.olympus-lifescience.com/en/microscope-resource/primer/techniques/dic/dicphasecomparison/>
- [210] Libretexts, “3.3B: Phase-Contrast Microscopy,” Biology LibreTexts, Dec. 24, 2022. [https://bio.libretexts.org/Bookshelves/Microbiology/Microbiology\\_%28Boundless%29/03:\\_Microscopy/3.03:\\_Other\\_Types\\_of\\_Microscopy/3.3B:\\_Phase-Contrast\\_Microscopy](https://bio.libretexts.org/Bookshelves/Microbiology/Microbiology_%28Boundless%29/03:_Microscopy/3.03:_Other_Types_of_Microscopy/3.3B:_Phase-Contrast_Microscopy)
- [211] “Phase-Contrast Microscopy,” Teledyne Photometrics, Mar. 23, 2020. <https://www.photometrics.com/learn/microscopy-basics/phase-contrast-microscopy>
- [212] F. J. W.-M. Leong, “Correction of uneven illumination (vignetting) in digital microscopy images,” *Journal of Clinical Pathology*, vol. 56, no. 8, pp. 619–621, Aug. 2003, doi: 10.1136/jcp.56.8.619.
- [213] E. K. Mahlandt and J. Goedhart, “Visualizing and Quantifying Data from Time-Lapse Imaging Experiments,” *Methods in Molecular Biology*, pp. 329–348, 2022, doi: 10.1007/978-1-0716-2051-9\_19.
- [214] T. J. Collins, “ImageJ for microscopy,” *BioTechniques*, vol. 43, no. 1S, pp. S25–S30, Jul. 2007, doi: 10.2144/000112517.
- [215] J. Schindelin et al., “Fiji: an open-source platform for biological-image analysis,” *Nature Methods*, vol. 9, no. 7, pp. 676–682, Jun. 2012, doi: 10.1038/nmeth.2019.
- [216] T. Peng et al., “A BaSiC tool for background and shading correction of optical microscopy images,” *Nature Communications*, vol. 8, no. 1, Jun. 2017, doi: 10.1038/ncomms14836.
- [217] “‘Quantile Based Normalization’ ImageJ PlugIn.” <https://www.longair.net/edinburgh/ImageJ/quantile-normalization/#:~:text=Essentially%20the%20idea%20is%20that,quantile%20across%20all%20the%20images.>
- [218] B. S. Deshmukh and V. H. Mankar, “Segmentation of Microscopic Images: A Survey,” 2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies, Jan. 2014, Published, doi: 10.1109/icesc.2014.68.

- [219] C. Sommer, C. Straehle, U. Kothe, and F. A. Hamprecht, "Ilastik: Interactive learning and segmentation toolkit," 2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro, Mar. 2011, Published, doi: 10.1109/isbi.2011.5872394.
- [220] "Anna Kreshuk," Wikipedia, Mar. 15, 2024. [https://en.wikipedia.org/wiki/Anna\\_Kreshuk](https://en.wikipedia.org/wiki/Anna_Kreshuk)
- [221] L. R. Dice, "Measures of the Amount of Ecologic Association Between Species," *Ecology*, vol. 26, no. 3, pp. 297–302, Jul. 1945, doi: 10.2307/1932409.
- [222] Sørensen, T, "A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons." *Kongelige Danske Videnskabernes Selskab*, 5(4), 1-34.
- [223] A. Sacan, H. Ferhatosmanoglu, and H. Coskun, "CellTrack: an open-source software for cell tracking and motility analysis," *Bioinformatics*, vol. 24, no. 14, pp. 1647–1649, May 2008, doi: 10.1093/bioinformatics/btn247.
- [224] A. E. Carpenter, T. R. Jones, M. R. Lamprecht, C. Clarke, I. H. Kang, O. Friman, D. A. Guertin, J. H. Chang, R. A. Lindquist, J. Moffat, P. Golland, and D. M. Sabatini, "CellProfiler: image analysis software for identifying and quantifying cell phenotypes," *\*Genome Biology\**, vol. 7, no. 10, pp. R100, 2006. doi: 10.1186/GB-2006-7-10-R100.
- [225] C. McQuin et al., "CellProfiler 3.0: Next-generation image processing for biology," *PLOS Biology*, vol. 16, no. 7, p. e2005970, Jul. 2018, doi: 10.1371/journal.pbio.2005970.
- [226] R. Delgado-Gonzalo, N. Denervaud, S. Maerkl, and M. Unser, "Multi-target tracking of packed yeast cells," 2010 IEEE International Symposium on Biomedical Imaging: From Nano to Macro, Apr. 2010, Published, doi: 10.1109/isbi.2010.5490288.
- [227] F. Padovani, B. Mairhörmann, P. Falter-Braun, J. Lengfeld, and K. M. Schmoller, "Segmentation, tracking and cell cycle analysis of live-cell imaging data with Cell-ACDC," *BMC Biology*, vol. 20, no. 1, Aug. 2022, doi: 10.1186/s12915-022-01372-6.
- [228] R. M. Maier, "Chapter 3 – Bacterial Growth," in *Environmental Microbiology*, 2nd ed., Academic Press, 2009.

- [229] T. Straube, C. Müller, and A. R. Perestrelo, “How to do a Proper Cell Culture Quick Check,” Science Lab | Leica Microsystems, Feb. 06, 2023. <https://www.leica-microsystems.com/science-lab/life-science/how-to-do-a-proper-cell-culture-quick-check/>
- [230] J. I. López and I. M. De la Fuente, “An Approach to Cell Motility as a Key Mechanism in Oncology,” *Cancers*, vol. 13, no. 14, p. 3576, Jul. 2021, doi: 10.3390/cancers13143576.
- [231] H. Chelly and P. Recho, “Cell motility as an energy minimization process,” *Physical Review E*, vol. 105, no. 6, Jun. 2022, doi: 10.1103/physreve.105.064401.
- [232] I. M. De la Fuente and J. I. López, “Cell Motility and Cancer,” *Cancers*, vol. 12, no. 8, p. 2177, Aug. 2020, doi: 10.3390/cancers12082177.
- [233] J. R. Koza and R. Poli, “Genetic Programming,” *Search Methodologies*, pp. 127–164, doi: 10.1007/0-387-28356-0\_5.
- [234] K. Staats, “About GP,” Genetic Programming. <https://geneticprogramming.com/>
- [235] A. J. Turner and J. F. Miller, “Recurrent Cartesian Genetic Programming,” *Parallel Problem Solving from Nature – PPSN XIII*, pp. 476–486, 2014, doi: 10.1007/978-3-319-10762-2\_47.
- [236] S.-H. Park and J. Fürnkranz, “Efficient Pairwise Classification,” *Machine Learning: ECML 2007*, pp. 658–665, 2007, doi: 10.1007/978-3-540-74958-5\_65.
- [237] K. Nordhausen, “Ensemble Methods: Foundations and Algorithms by Zhi-Hua Zhou,” *International Statistical Review*, vol. 81, no. 3, pp. 470–470, Nov. 2013, doi: 10.1111/insr.12042\_10.
- [238] M. F. Korns, “Genetic Programming Symbolic Classification: A Study,” *Genetic Programming Theory and Practice XV*, pp. 39–54, 2018, doi: 10.1007/978-3-319-90512-9\_3.
- [239] Miles Cranmer. 2020. PySR: Fast & Parallelized Symbolic Regression in Python/Julia. <https://doi.org/10.5281/zenodo.4041459>
- [240] D. Angelis, F. Sofos, and T. E. Karakasidis, “Artificial Intelligence in Physical Sciences: Symbolic Regression Trends and Perspectives,” *Archives of Computational Methods in Engineering*, vol. 30, no. 6, pp. 3845–3865, Apr. 2023, doi: 10.1007/s11831-023-09922-z.

- [241] Miller, J.F., Thomson, P., Fogarty, T.C.: Designing Electronic Circuits Using Evolutionary Algorithms: Arithmetic Circuits: A Case Study. In: D. Quagliarella, J. Periaux, C. Poloni, G. Winter (eds.) Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications, pp. 105–131. Wiley (1998).
- [242] Miller, J.F.: An Empirical Study of the Efficiency of Learning Boolean Functions using a Cartesian Genetic Programming Approach. In: Proc. Genetic and Evolutionary Computation Conference, pp. 1135–1142. Morgan Kaufmann (1999)
- [243] Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Proc. European Conference on Genetic Programming, LNCS, vol. 1802, pp. 121–132. Springer (2000)
- [244] Miller, J.F. (ed.): Cartesian Genetic Programming. Springer (2011)
- [245] P. J. Angeline, “Genetic programming: On the programming of computers by means of natural selection,” *Biosystems*, vol. 33, no. 1, pp. 69–73, Jan. 1994, doi: 10.1016/0303-2647(94)90062-0.
- [246] Turner, A.J., Miller, J.F.: Cartesian Genetic Programming: Why No Bloat? In: Genetic Programming: 17th European Conference (to appear, 2014)
- [247] V. K. Vassilev and J. F. Miller, “The Advantages of Landscape Neutrality in Digital Circuit Evolution,” *Evolvable Systems: From Biology to Hardware*, pp. 252–263, 2000, doi: 10.1007/3-540-46406-9\_25.
- [248] J. F. Miller and S. L. Smith, “Redundancy and computational efficiency in Cartesian genetic programming,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 2, pp. 167–174, Apr. 2006, doi: 10.1109/tevc.2006.871253.
- [249] B. W. Goldman and W. F. Punch, “Length bias and search limitations in cartesian genetic programming,” *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, Jul. 2013, Published, doi: 10.1145/2463372.2463482.
- [250] K. J. Simpson et al., “Identification of genes that regulate epithelial cell migration using an siRNA screening approach,” *Nature Cell Biology*, vol. 10, no. 9, pp. 1027–1038, Aug. 2008, doi: 10.1038/ncb1762.



- [251] V. Hakim and P. Silberzan, "Collective cell migration: a physics perspective," *Reports on Progress in Physics*, vol. 80, no. 7, p. 076601, Apr. 2017, doi: 10.1088/1361-6633/aa65ef.
- [252] Y. Hu, M. L. Becker, and R. K. Willits, "Quantification of cell migration: metrics selection to model application," *Frontiers in Cell and Developmental Biology*, vol. 11, May 2023, doi: 10.3389/fcell.2023.1155882.
- [253] E. R. Jerison and S. R. Quake, "Heterogeneous T cell motility behaviors emerge from a coupling between speed and turning in vivo," *eLife*, vol. 9, May 2020, doi: 10.7554/elife.53933.
- [254] X. H. Cao, I. Stojkovic, and Z. Obradovic, "A robust data scaling algorithm to improve classification accuracies in biomedical data," *BMC Bioinformatics*, vol. 17, no. 1, Sep. 2016, doi: 10.1186/s12859-016-1236-x.
- [255] A. Rácz, D. Bajusz, & K. Héberger, "Effect of dataset size and train/test split ratios in qsar/qspr multiclass classification", *Molecules*, vol. 26, no. 4, p. 1111, 2021. <https://doi.org/10.3390/molecules26041111>
- [256] O. Oyedele, "Determining the optimal number of folds to use in a K-fold cross-validation: A neural network classification experiment," *Research in Mathematics*, vol. 10, no. 1, May 2023, doi: 10.1080/27684830.2023.2201015.
- [257] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *\*Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)\**, vol. 14, no. 2, pp. 1137-1145, Aug. 1995.
- [258] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics Surveys*, vol. 4, pp. 40-79, 2010.
- [259] K. Nordhausen, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition by Trevor Hastie, Robert Tibshirani, Jerome Friedman," *International Statistical Review*, vol. 77, no. 3, pp. 482–482, Oct. 2009, doi: 10.1111/j.1751-5823.2009.00095\_18.x.

- [260] K. Nordhausen, “An Introduction to Statistical Learning—with Applications in R by Gareth James, Daniela Witten, Trevor Hastie & Robert Tibshirani,” *International Statistical Review*, vol. 82, no. 1, pp. 156–157, Apr. 2014, doi: 10.1111/insr.12051\_19.
- [261] G. Varoquaux, P. R. Raamana, D. A. Engemann, A. Hoyos-Idrobo, Y. Schwartz, and B. Thirion, “Assessing and tuning brain decoders: Cross-validation, caveats, and guidelines,” *NeuroImage*, vol. 145, pp. 166–179, Jan. 2017, doi: 10.1016/j.neuroimage.2016.10.038.
- [262] S.-H. Park and J. Fürnkranz, “Efficient Pairwise Classification,” *Machine Learning: ECML 2007*, pp. 658–665, 2007, doi: 10.1007/978-3-540-74958-5\_65.
- [263] J. Fürnkranz and E. Hüllermeier, "pairwise preference learning and ranking", p. 145-156, 2003. [https://doi.org/10.1007/978-3-540-39857-8\\_15](https://doi.org/10.1007/978-3-540-39857-8_15)
- [264] O. Lézoray and H. Cardot, "Comparing combination rules of pairwise neural networks classifiers", *Neural Processing Letters*, vol. 27, no. 1, p. 43-56, 2007. <https://doi.org/10.1007/s11063-007-9058-5>.
- [265] M. Hanifi, F. Sèdes, D. Aboutajdine, & A. Lasfar, "Multi-class support vector machine: a new approach to characterize a texture", 2007. <https://doi.org/10.1109/isie.2007.4374861>
- [266] L. Vanneschi and S. Silva, “Ensemble Methods,” *Natural Computing Series*, pp. 283–288, 2023, doi: 10.1007/978-3-031-17922-8\_11.
- [267] M. Cranmer, "Interpretable machine learning for science with PySR and SymbolicRegression," arXiv preprint arXiv:2305.01582, 2023.
- [268] Anne Brindle. *Genetic Algorithms for Function Optimization*. PhD thesis, 1980
- [269] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized Evolution for Image Classifier Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4780–4789, July 2019.
- [270] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, May 1983
- [271] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. General considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, March 1970

- [272] “Julia Documentation · The Julia Language.” <https://docs.julialang.org/en/v1/>
- [273] J. B. S. K. Al Viral Shah, Alan Edelman, Et, “Julia Micro-Benchmarks.” <https://julialang.org/benchmarks/>
- [274] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, et al. Scikitlearn: Machine Learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, November 2011.
- [275] Fast Fourier Transformation (FFT) - MIPAV," MIPAV. [Online]. Available: [https://mipav.cit.nih.gov/pubwiki/index.php/Fast\\_Fourier\\_Transformation\\_%28FFT%29](https://mipav.cit.nih.gov/pubwiki/index.php/Fast_Fourier_Transformation_%28FFT%29). [Accessed: 25-Jun-2024]
- [276] “BioVoxxel Toolbox,” ImageJ Wiki. <https://ImageJ.net/plugins/biovoxxel-toolbox#flat-field-and-pseudo-flat-field-correction>
- [277] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?,” Sep. 2009, doi: 10.1109/iccv.2009.5459469.
- [278] K. Zuiderveld, “Contrast Limited Adaptive Histogram Equalization,” in Elsevier eBooks, 1994, pp. 474–485. doi: 10.1016/b978-0-12-336156-1.50061-6.
- [279] I. Arganda-Carreras et al., “Trainable Weka Segmentation: a machine learning tool for microscopy pixel classification,” *Bioinformatics*, vol. 33, no. 15, pp. 2424–2426, Mar. 2017, doi: 10.1093/bioinformatics/btx180.
- [280] M. Arzt et al., “LABKIT: Labeling and Segmentation Toolkit for Big Image Data,” *Frontiers in Computer Science*, vol. 4, Feb. 2022, doi: 10.3389/fcomp.2022.777728.
- [281] J. Chalfoun, A. Cardone, A. A. Dima, D. P. Allen, and M. W. Halter, “Overlap-based cell tracker,” *Journal of Research of the National Institute of Standards and Technology*, vol. 115, no. 6, p. 477, Nov. 2010, doi: 10.6028/jres.115.034.
- [282] Y. T. Fukai and K. Kawaguchi, “LapTrack: linear assignment particle tracking with tunable metrics,” *Bioinformatics*, vol. 39, no. 1, Dec. 2022, doi: 10.1093/bioinformatics/btac799.

- [283] J. J. Franco, Y. Atieh, C. D. Bryan, K. M. Kwan, and G. T. Eisenhoffer, “Cellular crowding influences extrusion and proliferation to facilitate epithelial tissue repair,” *Molecular Biology of the Cell*, vol. 30, no. 16, pp. 1890–1899, Jul. 2019, doi: 10.1091/mbc.e18-05-0295.
- [284] A. Bajpai, J. Tong, W. Qian, Y. Peng, and W. Chen, “The Interplay Between Cell-Cell and Cell-Matrix Forces Regulates Cell Migration Dynamics,” *Biophysical Journal*, vol. 117, no. 10, pp. 1795–1804, Nov. 2019, doi: 10.1016/j.bpj.2019.10.015.
- [285] L. Kubiczkova, L. Sedlarikova, R. Hajek, and S. Sevcikova, “TGF- $\beta$  – an excellent servant but a bad master,” *Journal of Translational Medicine*, vol. 10, no. 1, Sep. 2012, doi: 10.1186/1479-5876-10-183.
- [286] P.-Y. Chen, L. Qin, and M. Simons, “TGF $\beta$  signaling pathways in human health and disease,” *Frontiers in Molecular Biosciences*, vol. 10, Jun. 2023, doi: 10.3389/fmolb.2023.1113061.
- [287] J. M. Fleming et al., “Differentiation-Associated Reprogramming of the Transforming Growth Factor  $\beta$  Receptor Pathway Establishes the Circuitry for Epithelial Autocrine/Paracrine Repair,” *PloS One*, vol. 7, no. 12, p. e51404, Dec. 2012, doi: 10.1371/journal.pone.0051404.
- [288] A. J. Lee et al., “Sustained Delivery of SB-431542, a Type I Transforming Growth Factor Beta-1 Receptor Inhibitor, to Prevent Arthrofibrosis,” *Tissue Engineering. Part A*, vol. 27, no. 21–22, pp. 1411–1421, Nov. 2021, doi: 10.1089/ten.tea.2021.0029.
- [289] S. K. Halder, R. D. Beauchamp, and P. K. Datta, “A Specific Inhibitor of TGF- $\beta$  Receptor Kinase, SB-431542, as a Potent Antitumor Agent for Human Cancers,” *Neoplasia*, vol. 7, no. 5, pp. 509–521, May 2005, doi: 10.1593/neo.04640.
- [290] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
- [291] L. Breiman, “Random Forest,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, Jan. 2001, doi: 10.1023/a:1010933404324.
- [292] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1007/bf00994018.

## APPENDIX A

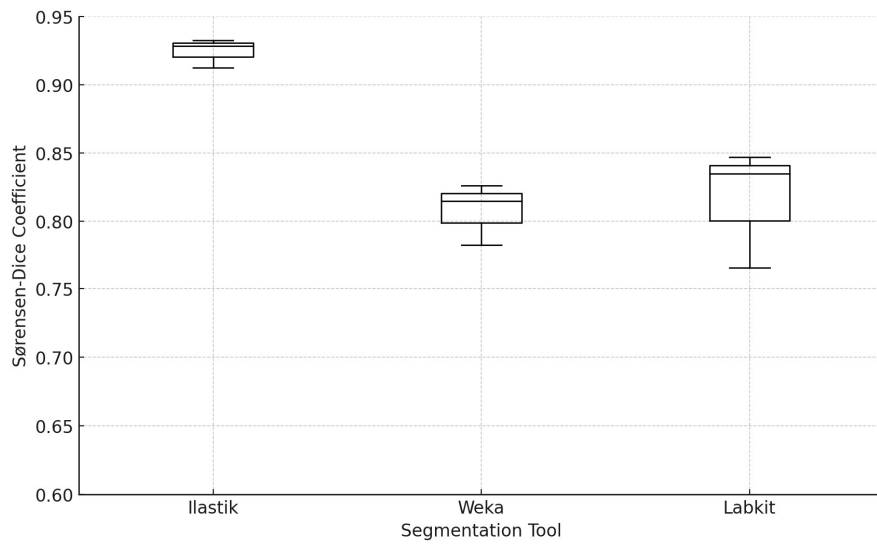


Figure A.1: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 1

Table A.1: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 1

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.9245 | 0.0107             |
| Weka              | 0.8080 | 0.0227             |
| Labkit            | 0.8160 | 0.0438             |

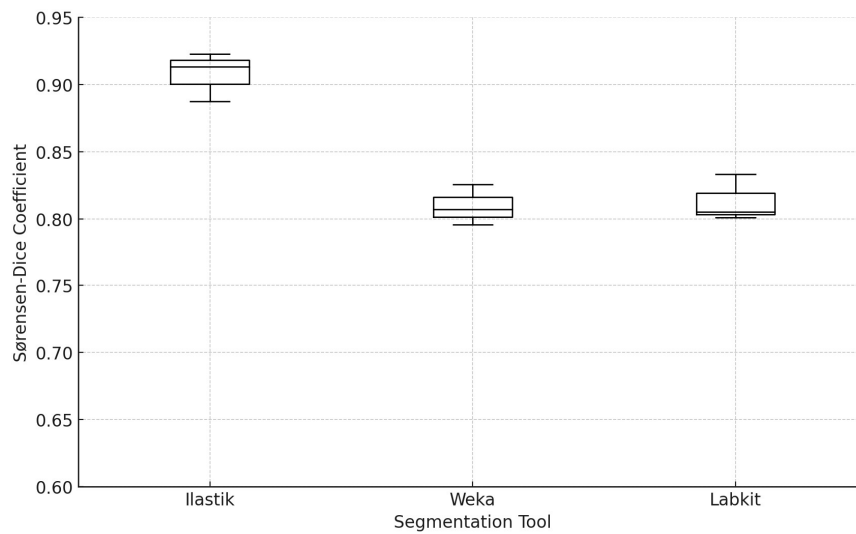


Figure A.2: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 2

Table A.2: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 2

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.9079 | 0.0182             |
| Weka              | 0.8097 | 0.0151             |
| Labkit            | 0.8134 | 0.0174             |

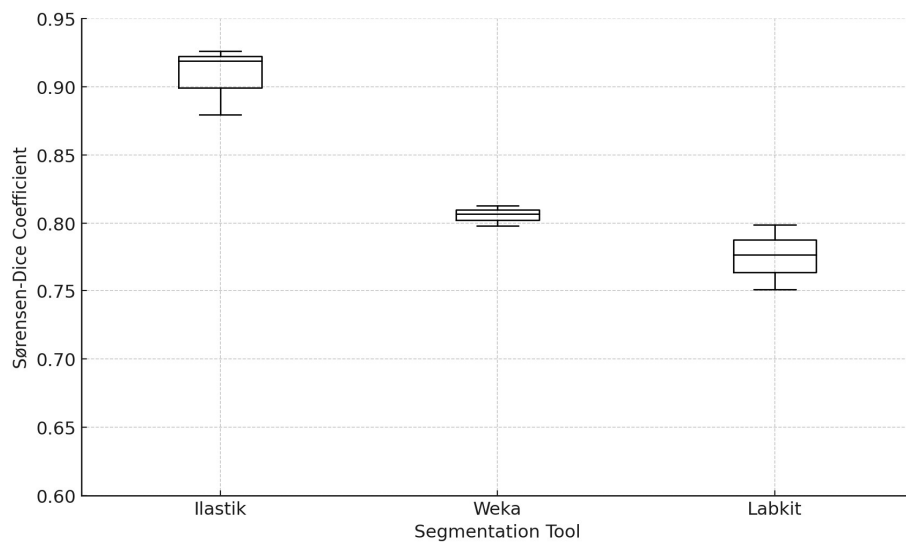


Figure A.3: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 3

Table A.3: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 3

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.9082 | 0.0251             |
| Weka              | 0.8060 | 0.0076             |
| Labkit            | 0.7757 | 0.0241             |

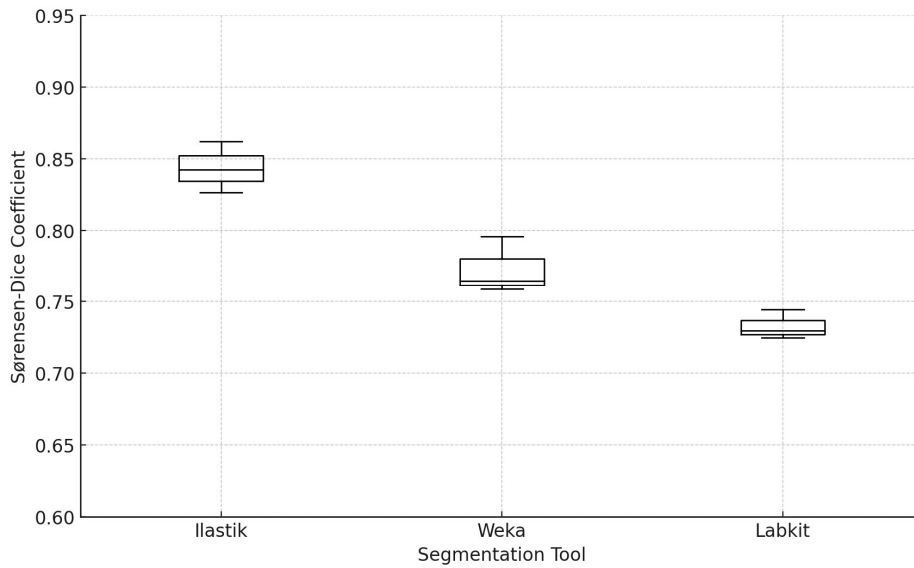


Figure A.4: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 5

Table A.4: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 5

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.8436 | 0.0178             |
| Weka              | 0.7733 | 0.0198             |
| Labkit            | 0.7328 | 0.0102             |

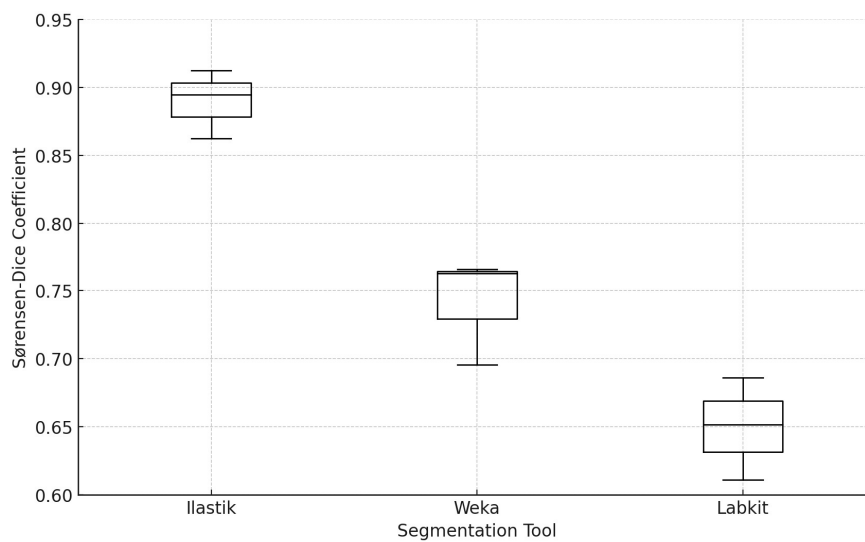


Figure A.5: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 6

Table A.5: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 6

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.8899 | 0.0253             |
| Weka              | 0.7418 | 0.0401             |
| Labkit            | 0.6496 | 0.0375             |

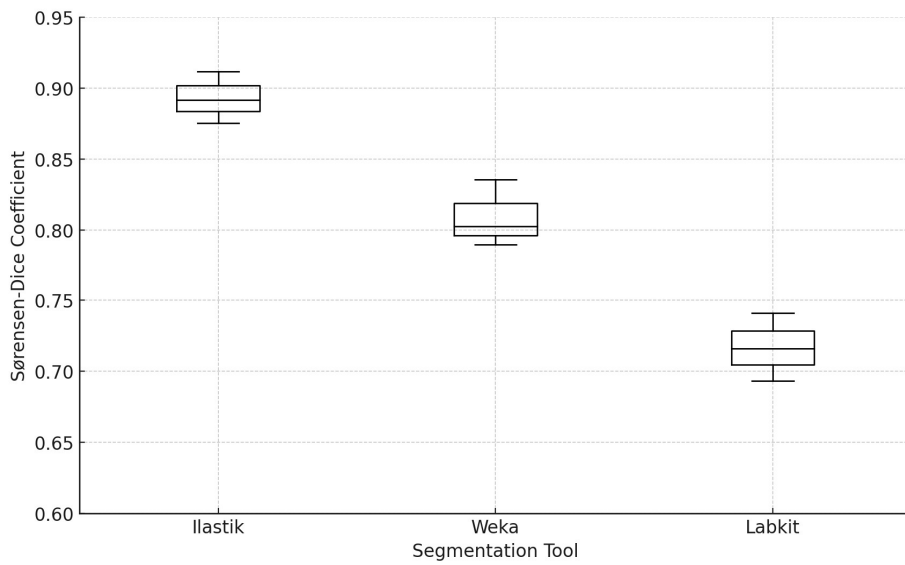


Figure A.6: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 7

Table A.6: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 7

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.8931 | 0.0182             |
| Weka              | 0.8093 | 0.0236             |
| Labkit            | 0.7167 | 0.0237             |



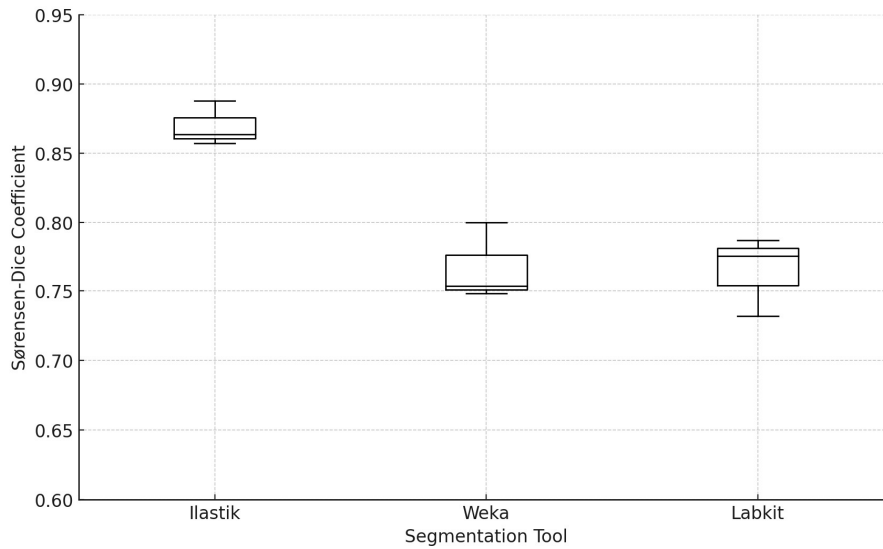


Figure A.7: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 9

Table A.7: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 9

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.8696 | 0.0161             |
| Weka              | 0.7671 | 0.0286             |
| Labkit            | 0.7650 | 0.0291             |

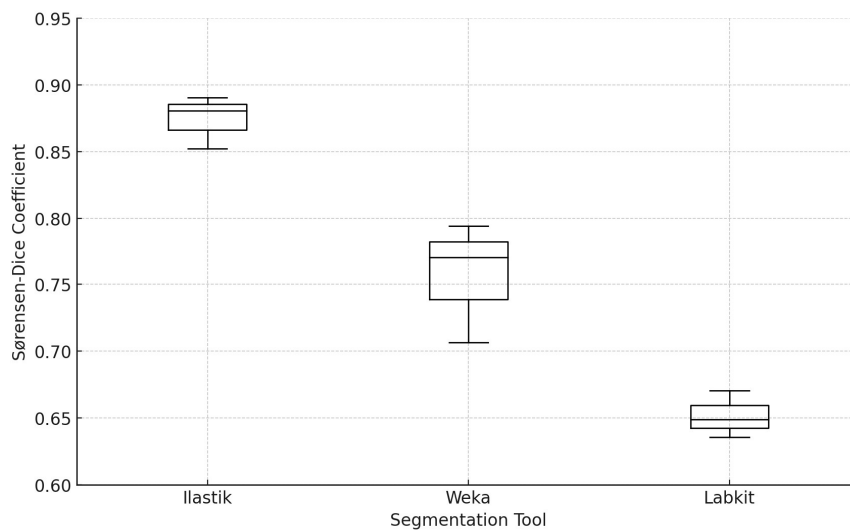


Figure A.8: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 10

Table A.8: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 10

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.8745 | 0.0200             |
| Weka              | 0.7572 | 0.0456             |
| Labkit            | 0.6517 | 0.0176             |

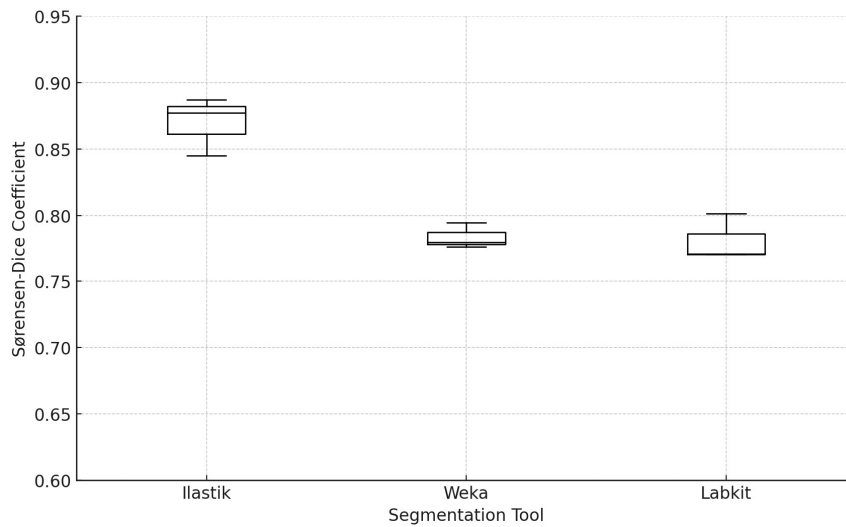


Figure A.9: Sørensen-Dice Coefficient Comparison Across Different Segmentation Tools for Well 11

Table A.9: Mean and SD of Sørensen-Dice Coefficient for Different Segmentation Tools for Well 11

| Segmentation tool | Mean   | Standard deviation |
|-------------------|--------|--------------------|
| Ilastik           | 0.8699 | 0.0220             |
| Weka              | 0.7838 | 0.0095             |
| Labkit            | 0.7813 | 0.0176             |

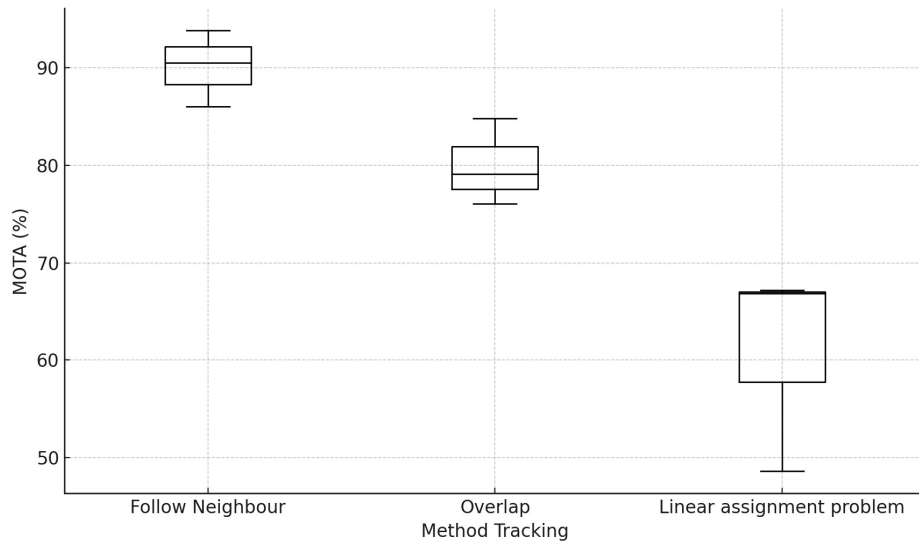


Figure A.10: MOTA (%) Comparison Across Different Tracking Methods for Well 1

Table A.10: Mean and SD of MOTA (%) for Different Tracking Method for Well 1

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 90.10 | 3.89               |
| Overlap                   | 79.97 | 4.44               |
| Linear assignment problem | 60.83 | 10.61              |

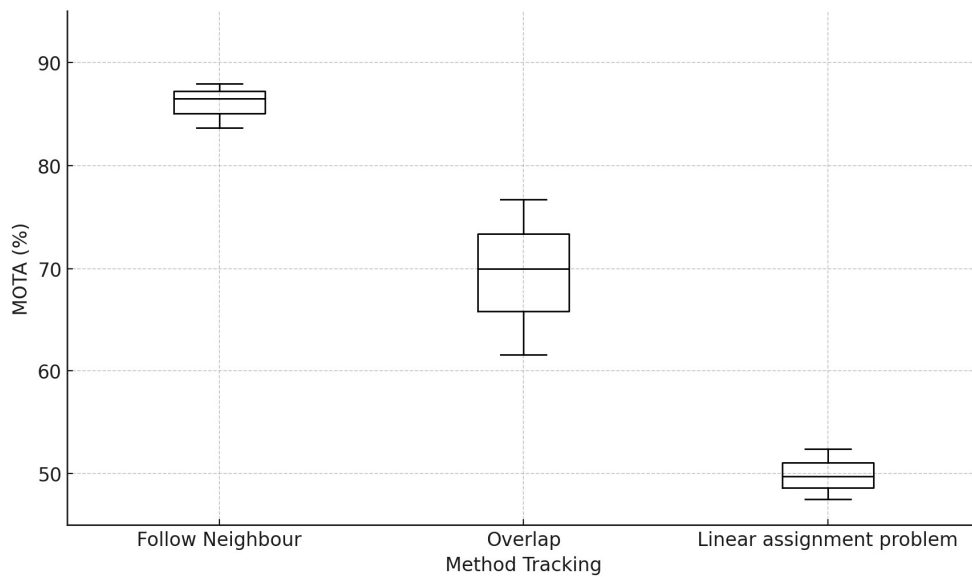


Figure A.11: MOTA (%) Comparison Across Different Tracking Methods for Well 2

Table A.11: Mean and SD of MOTA (%) for Different Tracking Method for Well 2

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 86.03 | 2.17               |
| Overlap                   | 69.45 | 7.60               |
| Linear assignment problem | 49.88 | 2.46               |

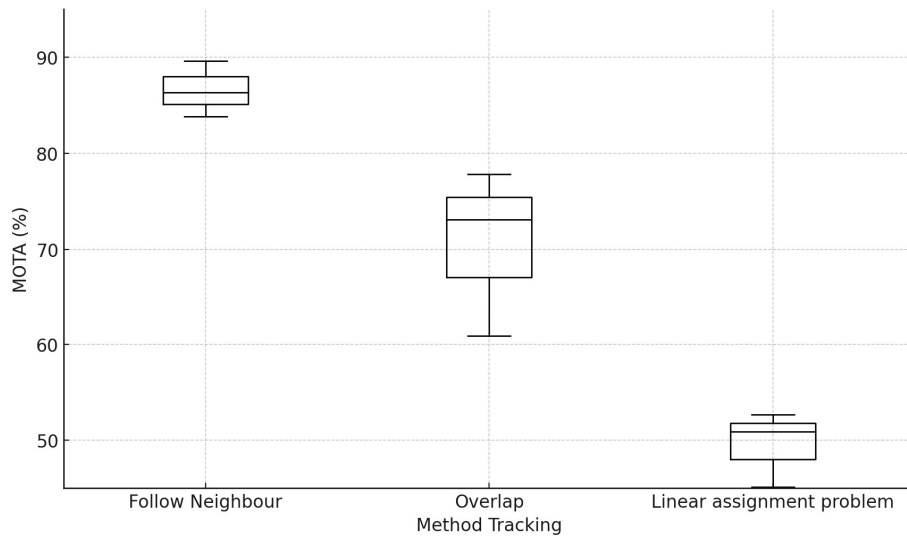


Figure A.12: MOTA (%) Comparison Across Different Tracking Methods for Well 3

Table A.12: Mean and SD of MOTA (%) for Different Tracking Method for Well 3

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 86.60 | 2.90               |
| Overlap                   | 70.61 | 8.75               |
| Linear assignment problem | 49.57 | 3.94               |

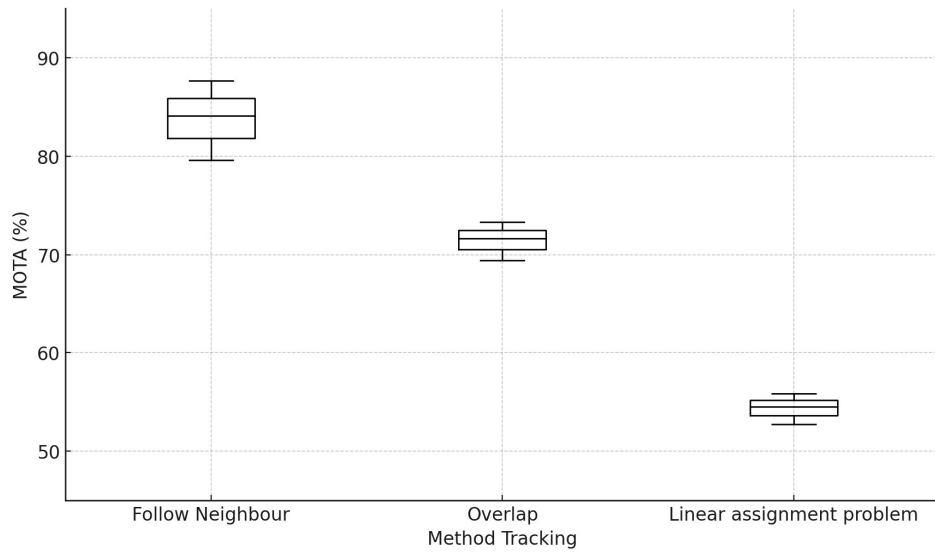


Figure A.13: MOTA (%) Comparison Across Different Tracking Methods for Well 5

Table A.13: Mean and SD of MOTA (%) for Different Tracking Method for Well 5

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 83.80 | 4.02               |
| Overlap                   | 71.50 | 1.96               |
| Linear assignment problem | 54.37 | 1.58               |

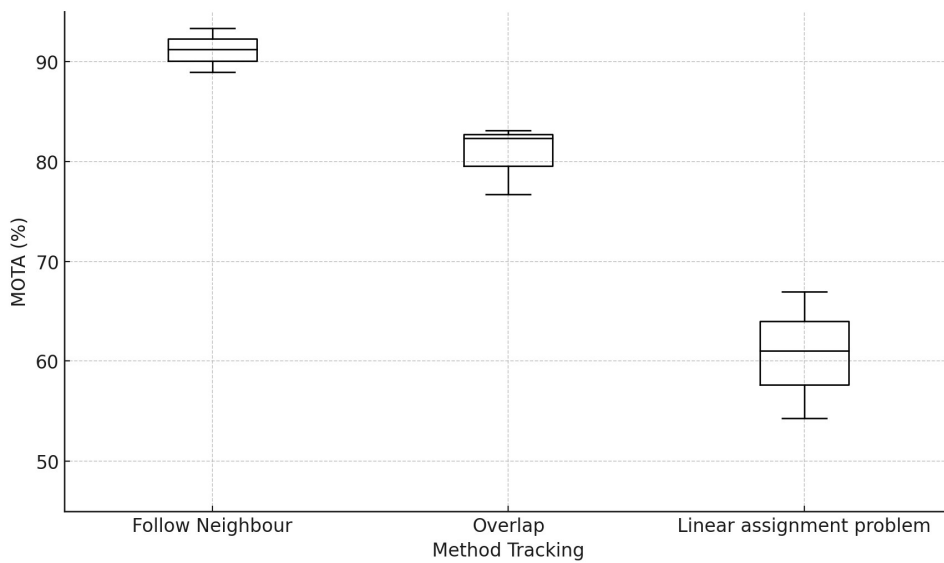


Figure A.14: MOTA (%) Comparison Across Different Tracking Methods for Well 6

Table A.14: Mean and SD of MOTA (%) for Different Tracking Method for Well 6

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 91.17 | 2.20               |
| Overlap                   | 80.74 | 3.47               |
| Linear assignment problem | 60.73 | 6.31               |

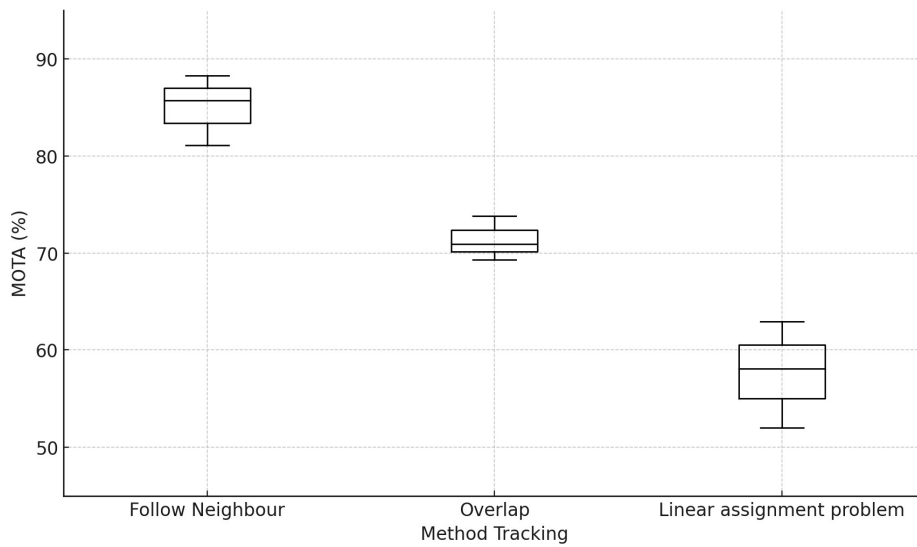


Figure A.15: MOTA (%) Comparison Across Different Tracking Methods for Well 7

Table A.15: Mean and SD of MOTA (%) for Different Tracking Method for Well 7

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 85.04 | 3.64               |
| Overlap                   | 71.38 | 2.29               |
| Linear assignment problem | 57.66 | 5.47               |

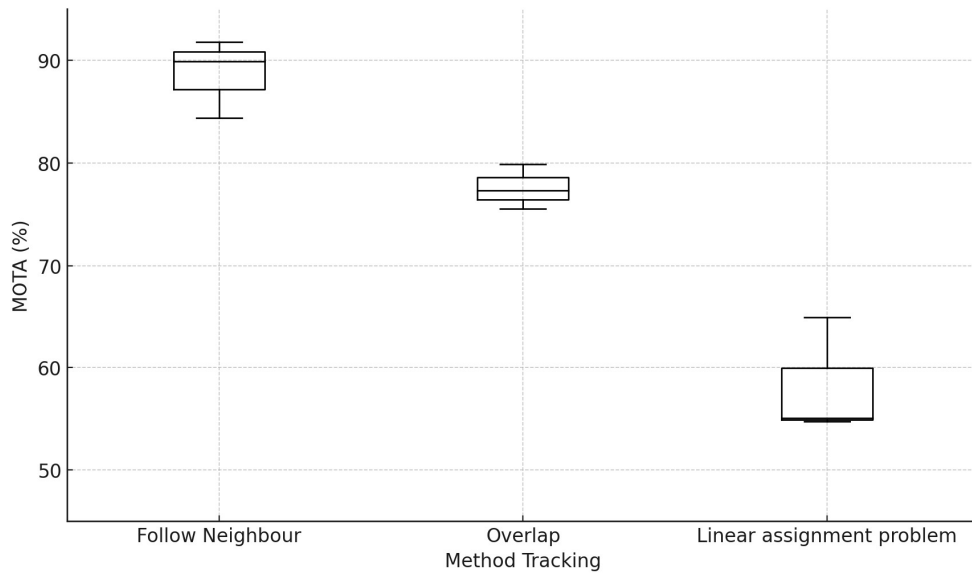


Figure A.16: MOTA (%) Comparison Across Different Tracking Methods for Well 9

Table A.16: Mean and SD of MOTA (%) for Different Tracking Method for Well 9

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 88.71 | 3.85               |
| Overlap                   | 77.62 | 2.17               |
| Linear assignment problem | 58.23 | 5.76               |

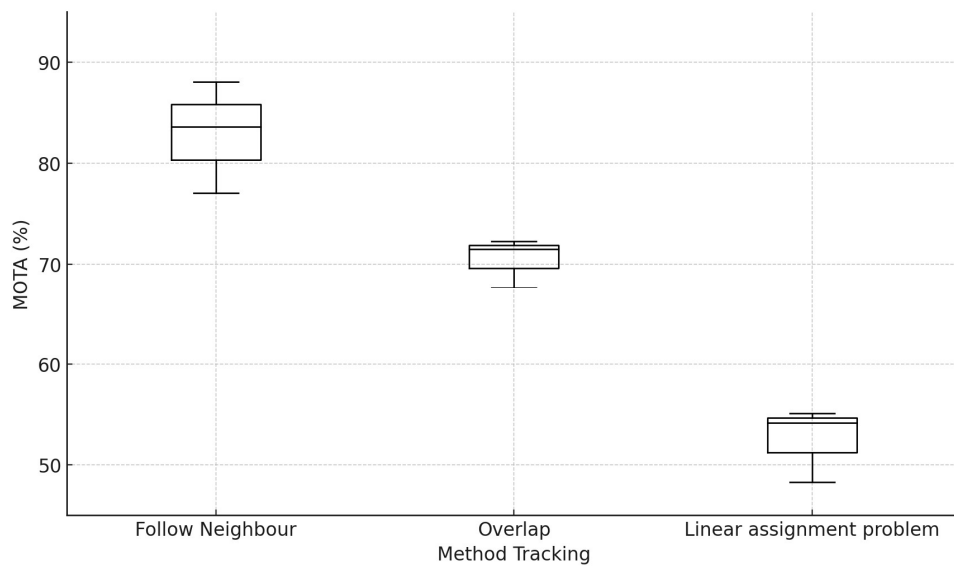


Figure A.17: MOTA (%) Comparison Across Different Tracking Methods for Well 10

Table A.17: Mean and SD of MOTA (%) for Different Tracking Method for Well 10

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 82.92 | 5.52               |
| Overlap                   | 70.50 | 2.47               |
| Linear assignment problem | 52.53 | 3.70               |

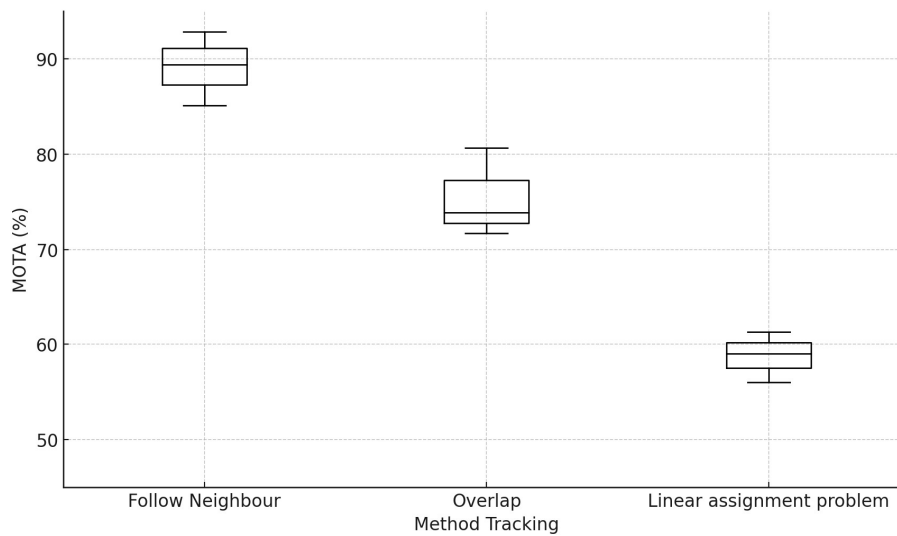


Figure A.18: MOTA (%) Comparison Across Different Tracking Methods for Well 11

Table A.18: Mean and SD of MOTA (%) for Different Tracking Method for Well 11

| Method tracking           | Mean  | Standard deviation |
|---------------------------|-------|--------------------|
| Follow neighbour          | 89.12 | 3.85               |
| Overlap                   | 75.43 | 4.68               |
| Linear assignment problem | 58.78 | 2.64               |



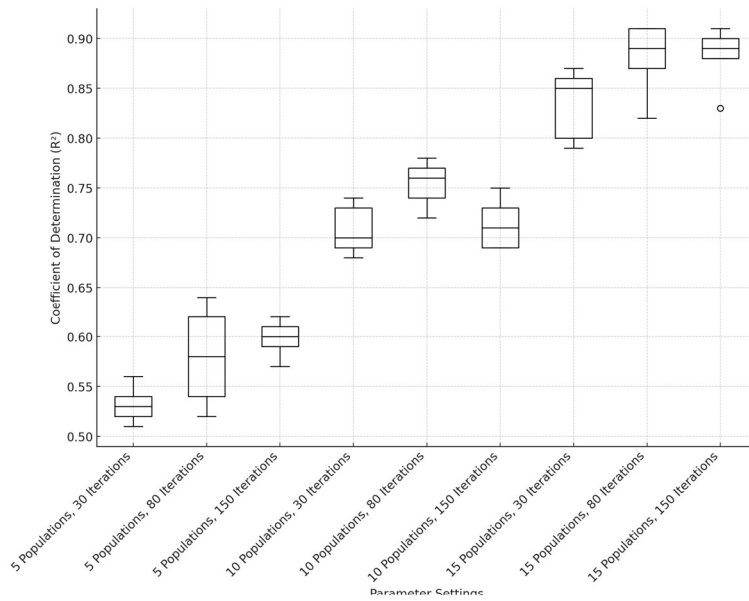


Figure A.19: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 1

Table A.19: Summary Statistics for 5-Fold CV  $R^2$  Mean for Well 1 Symbolic Regression.

| Parameter setting              | $R^2$ Mean | Standard deviation |
|--------------------------------|------------|--------------------|
| 5 Populations, 30 Iterations   | 0.542      | 0.0102             |
| 5 Populations, 80 Iterations   | 0.583      | 0.0255             |
| 5 Populations, 150 Iterations  | 0.595      | 0.0102             |
| 10 Populations, 30 Iterations  | 0.733      | 0.0324             |
| 10 Populations, 80 Iterations  | 0.757      | 0.0245             |
| 10 Populations, 150 Iterations | 0.718      | 0.0164             |
| 15 Populations, 30 Iterations  | 0.826      | 0.0296             |
| 15 Populations, 80 Iterations  | 0.883      | 0.0376             |
| 15 Populations, 150 Iterations | 0.885      | 0.0080             |

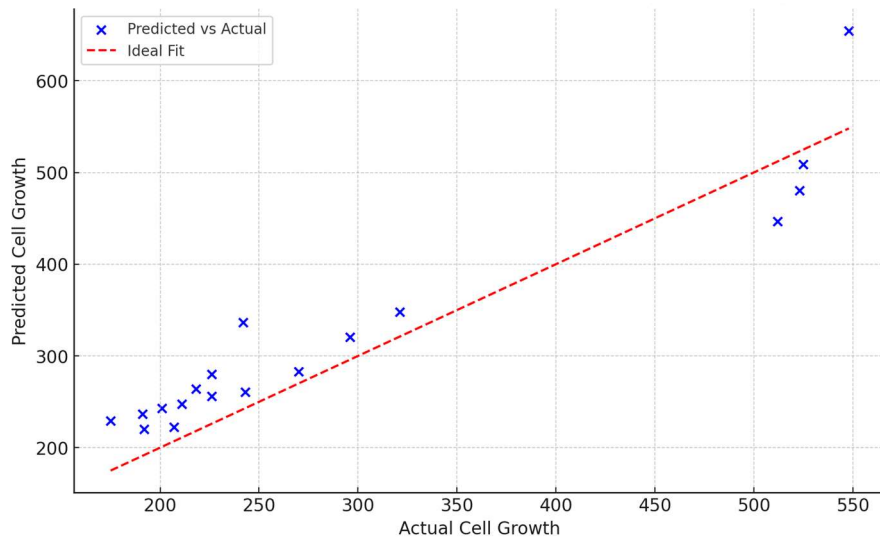


Figure A.20: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.854$ )

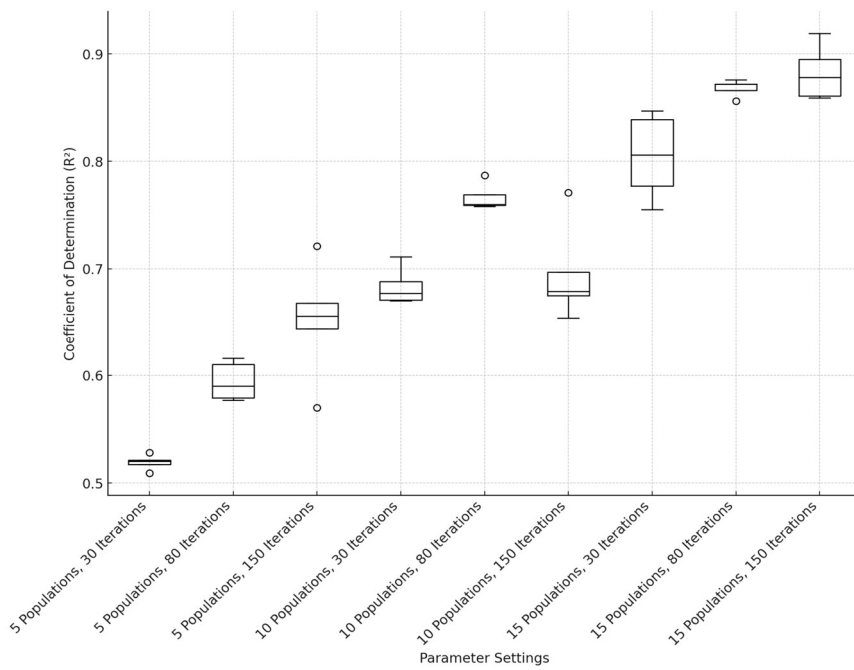


Figure A.21: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 2

Table A.20: Summary Statistics for 5-Fold CV  $R^2$  Mean for Well 2 Symbolic Regression.

| Parameter setting              | $R^2$ Mean | Standard deviation |
|--------------------------------|------------|--------------------|
| 5 Populations, 30 Iterations   | 0.518      | 0.0031             |
| 5 Populations, 80 Iterations   | 0.599      | 0.0098             |
| 5 Populations, 150 Iterations  | 0.661      | 0.0041             |
| 10 Populations, 30 Iterations  | 0.685      | 0.0085             |
| 10 Populations, 80 Iterations  | 0.773      | 0.0050             |
| 10 Populations, 150 Iterations | 0.685      | 0.0084             |
| 15 Populations, 30 Iterations  | 0.801      | 0.0182             |
| 15 Populations, 80 Iterations  | 0.861      | 0.0043             |
| 15 Populations, 150 Iterations | 0.876      | 0.0153             |

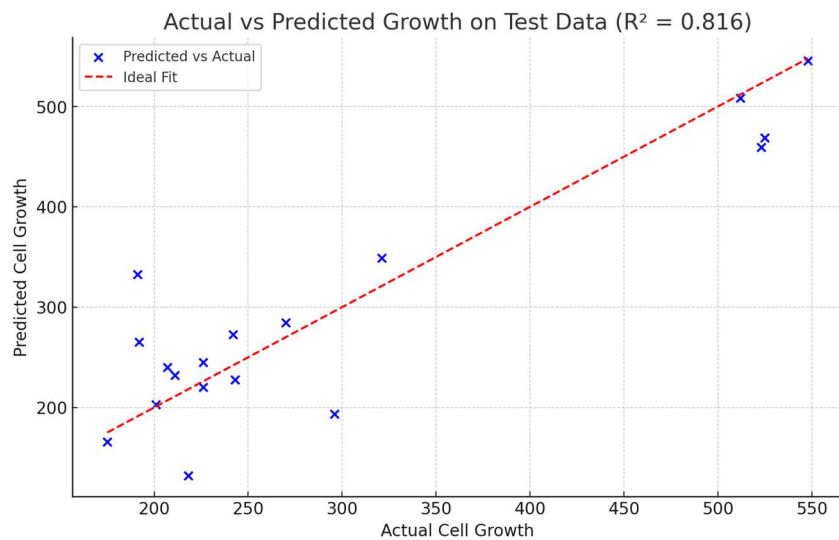


Figure A.22: Actual Vs Predicted Growth on Test Data ( $R^2 = 0.816$ ).

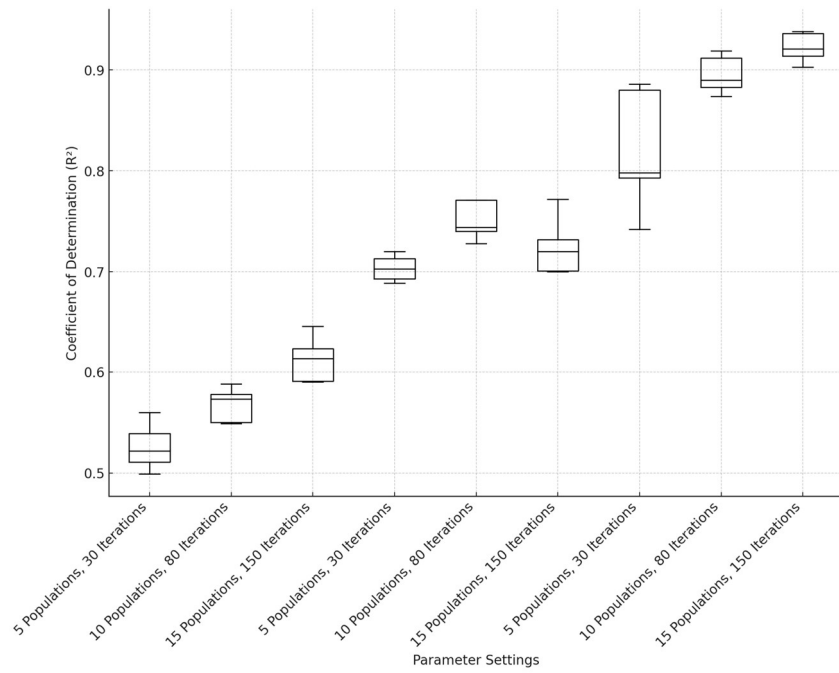


Figure A.23: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 3.

Table A.21: Summary Statistics for 5-Fold CV  $R^2$  Mean for Well 3 Symbolic Regression.

| Parameter setting              | $R^2$ Mean | Standard deviation |
|--------------------------------|------------|--------------------|
| 5 Populations, 30 Iterations   | 0.532      | 0.0047             |
| 5 Populations, 80 Iterations   | 0.566      | 0.0095             |
| 5 Populations, 150 Iterations  | 0.619      | 0.0115             |
| 10 Populations, 30 Iterations  | 0.704      | 0.0087             |
| 10 Populations, 80 Iterations  | 0.748      | 0.0092             |
| 10 Populations, 150 Iterations | 0.737      | 0.0147             |
| 15 Populations, 30 Iterations  | 0.810      | 0.0308             |
| 15 Populations, 80 Iterations  | 0.893      | 0.0114             |
| 15 Populations, 150 Iterations | 0.914      | 0.0056             |

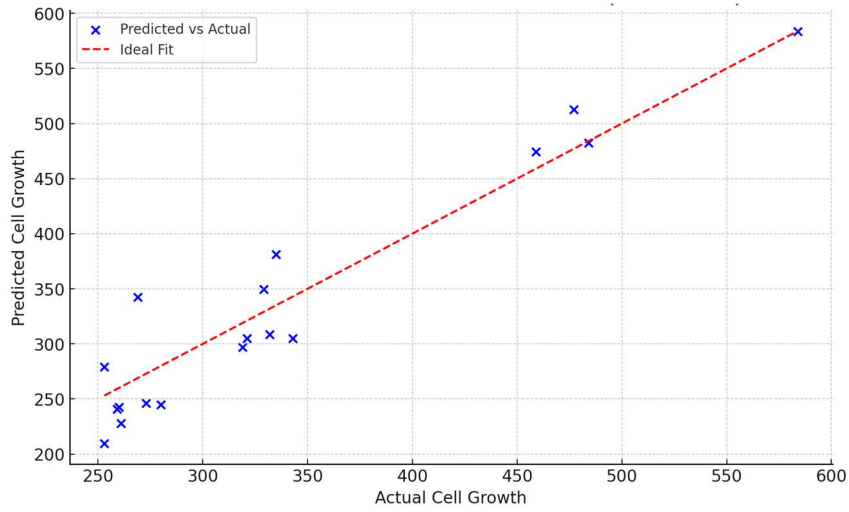


Figure A.24: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.886$ ).

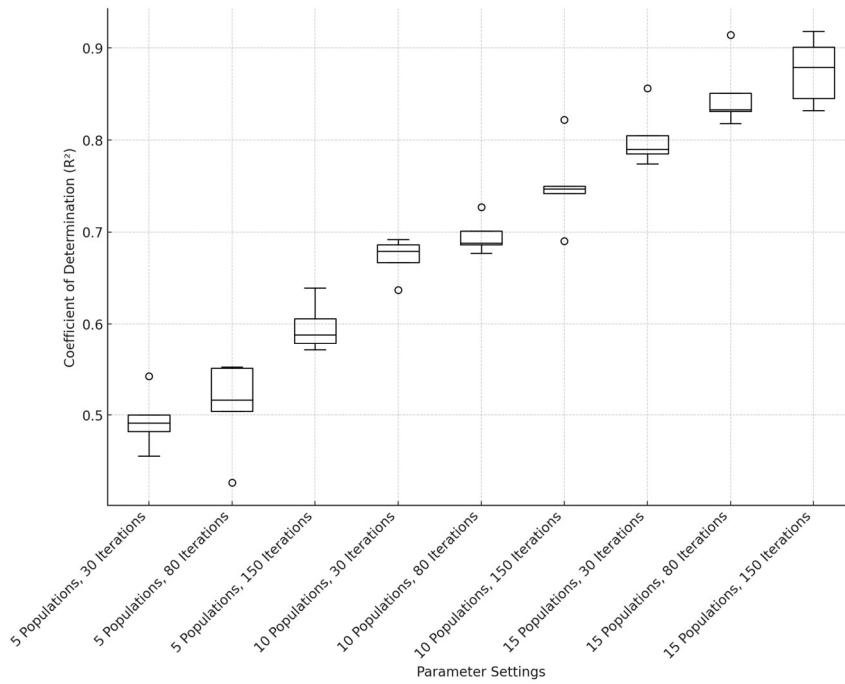


Figure A.25: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 5.

Table A.22: Summary Statistics for 5-Fold CV R<sup>2</sup> Mean for Well 5 Symbolic Regression.

| Parameter setting              | R <sup>2</sup> Mean | Standard deviation |
|--------------------------------|---------------------|--------------------|
| 5 Populations, 30 Iterations   | 0.476               | 0.0054             |
| 5 Populations, 80 Iterations   | 0.521               | 0.0195             |
| 5 Populations, 150 Iterations  | 0.613               | 0.0163             |
| 10 Populations, 30 Iterations  | 0.660               | 0.0147             |
| 10 Populations, 80 Iterations  | 0.707               | 0.0096             |
| 10 Populations, 150 Iterations | 0.751               | 0.0041             |
| 15 Populations, 30 Iterations  | 0.795               | 0.0086             |
| 15 Populations, 80 Iterations  | 0.833               | 0.0048             |
| 15 Populations, 150 Iterations | 0.878               | 0.0212             |

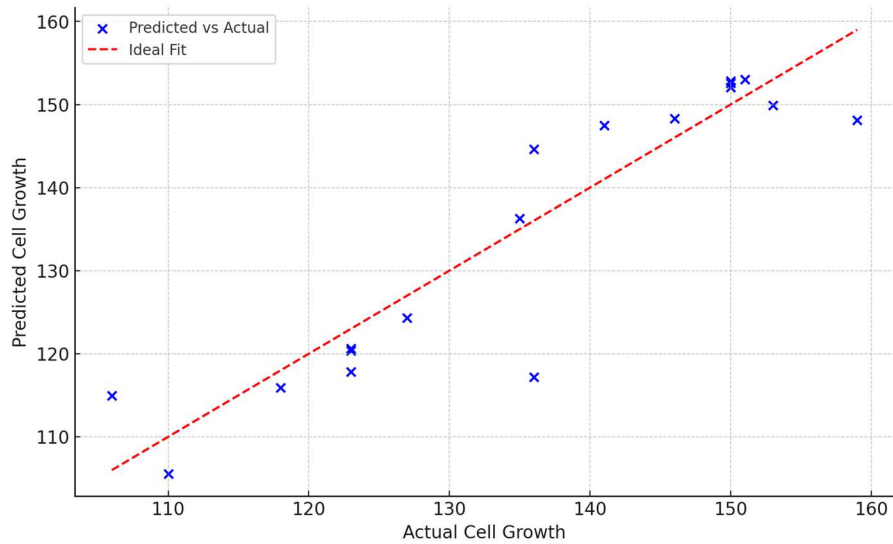


Figure A.26: Actual Vs Predicted Growth On Test Data (R<sup>2</sup> = 0.816).

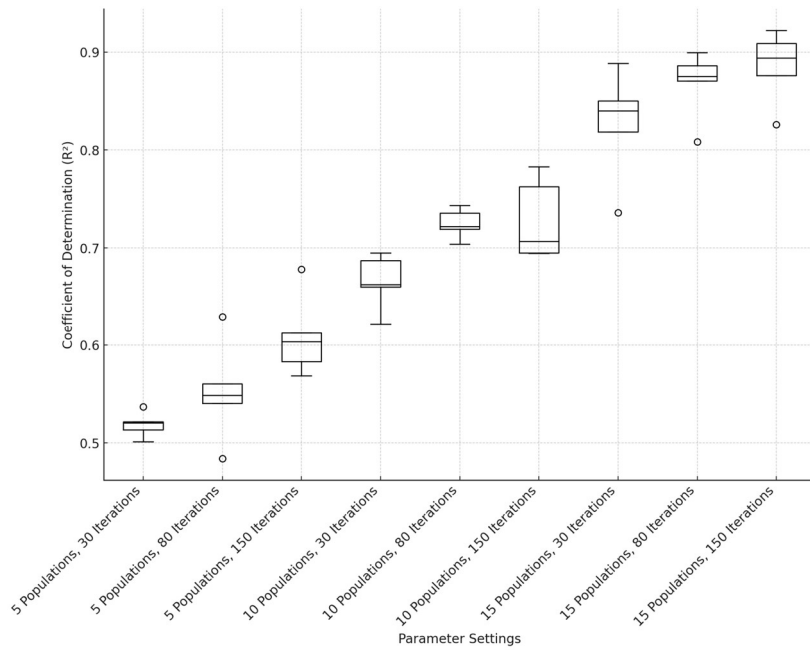


Figure A.27: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 6.

Table A.23: Summary Statistics for 5-Fold CV R<sup>2</sup> Mean for Well 6 Symbolic Regression.

| Parameter setting              | R <sup>2</sup> Mean | Standard deviation |
|--------------------------------|---------------------|--------------------|
| 5 Populations, 30 Iterations   | 0.517               | 0.0071             |
| 5 Populations, 80 Iterations   | 0.560               | 0.0037             |
| 5 Populations, 150 Iterations  | 0.595               | 0.0090             |
| 10 Populations, 30 Iterations  | 0.657               | 0.0149             |
| 10 Populations, 80 Iterations  | 0.730               | 0.0129             |
| 10 Populations, 150 Iterations | 0.736               | 0.0192             |
| 15 Populations, 30 Iterations  | 0.851               | 0.0133             |
| 15 Populations, 80 Iterations  | 0.880               | 0.0078             |
| 15 Populations, 150 Iterations | 0.898               | 0.0110             |

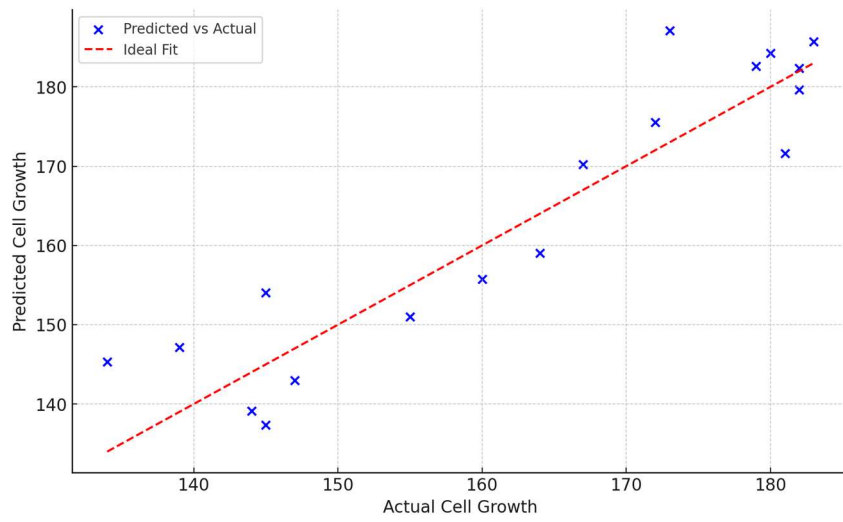


Figure A.28: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.841$ ).

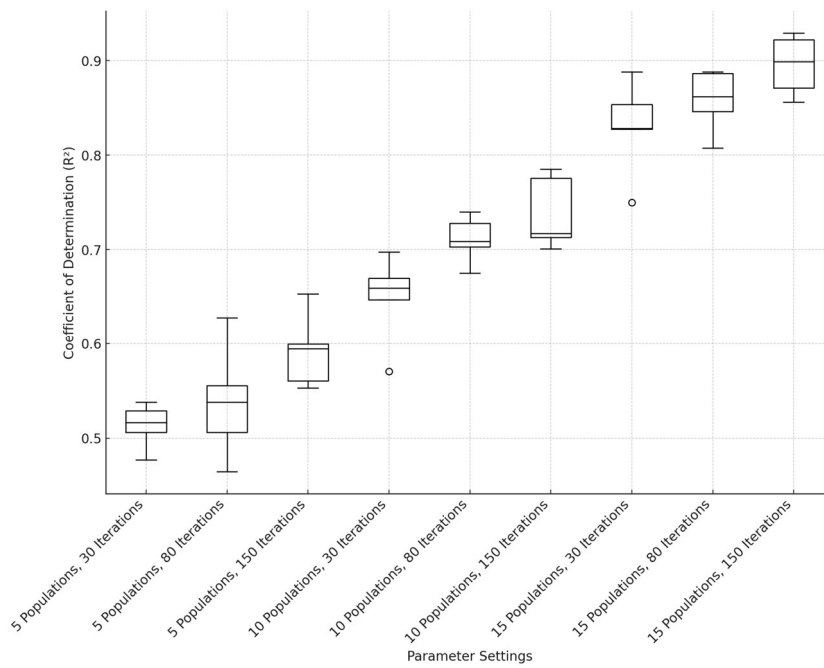


Figure A.29: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 7.



Table A.24: Summary Statistics for 5-Fold CV  $R^2$  Mean for Well 7 Symbolic Regression.

| Parameter setting              | $R^2$ Mean | Standard deviation |
|--------------------------------|------------|--------------------|
| 5 Populations, 30 Iterations   | 0.506      | 0.0149             |
| 5 Populations, 80 Iterations   | 0.544      | 0.0344             |
| 5 Populations, 150 Iterations  | 0.590      | 0.0212             |
| 10 Populations, 30 Iterations  | 0.665      | 0.0124             |
| 10 Populations, 80 Iterations  | 0.706      | 0.0156             |
| 10 Populations, 150 Iterations | 0.744      | 0.0185             |
| 15 Populations, 30 Iterations  | 0.825      | 0.0153             |
| 15 Populations, 80 Iterations  | 0.847      | 0.0156             |
| 15 Populations, 150 Iterations | 0.896      | 0.0163             |

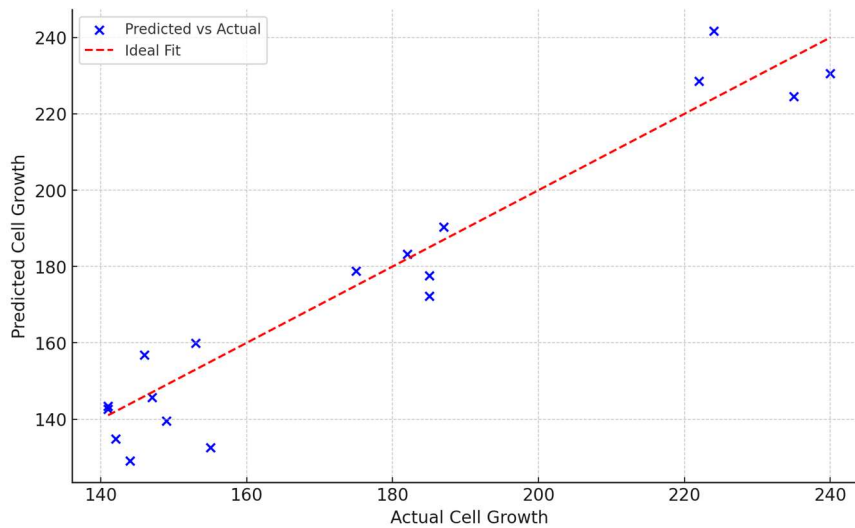


Figure A.30: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.909$ ).

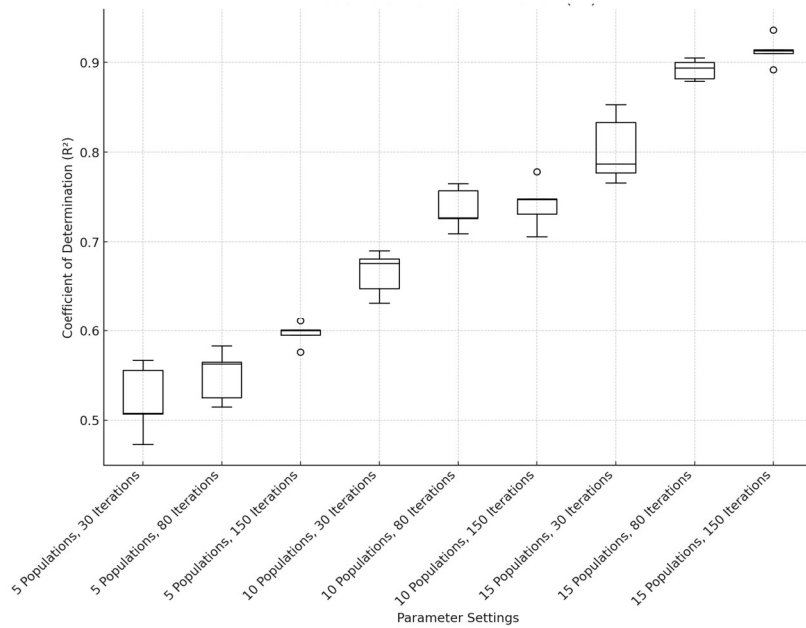


Figure A.31: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 9.

Table A.25: Summary Statistics for 5-Fold CV  $R^2$  Mean for Well 9 Symbolic Regression.

| Parameter setting              | $R^2$ Mean | Standard deviation |
|--------------------------------|------------|--------------------|
| 5 Populations, 30 Iterations   | 0.523      | 0.0227             |
| 5 Populations, 80 Iterations   | 0.554      | 0.0110             |
| 5 Populations, 150 Iterations  | 0.597      | 0.0024             |
| 10 Populations, 30 Iterations  | 0.660      | 0.0118             |
| 10 Populations, 80 Iterations  | 0.744      | 0.0121             |
| 10 Populations, 150 Iterations | 0.732      | 0.0076             |
| 15 Populations, 30 Iterations  | 0.805      | 0.0213             |
| 15 Populations, 80 Iterations  | 0.892      | 0.0071             |
| 15 Populations, 150 Iterations | 0.919      | 0.0046             |

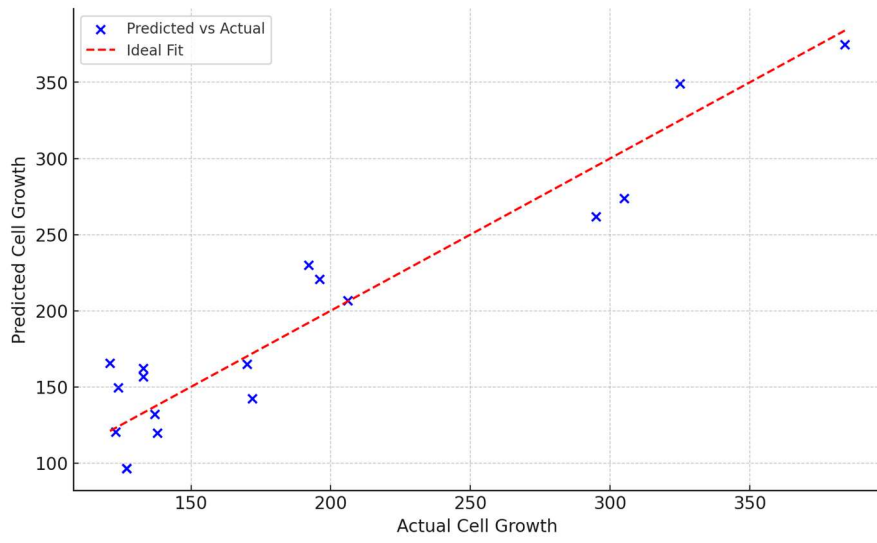


Figure A.32: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.896$ ).

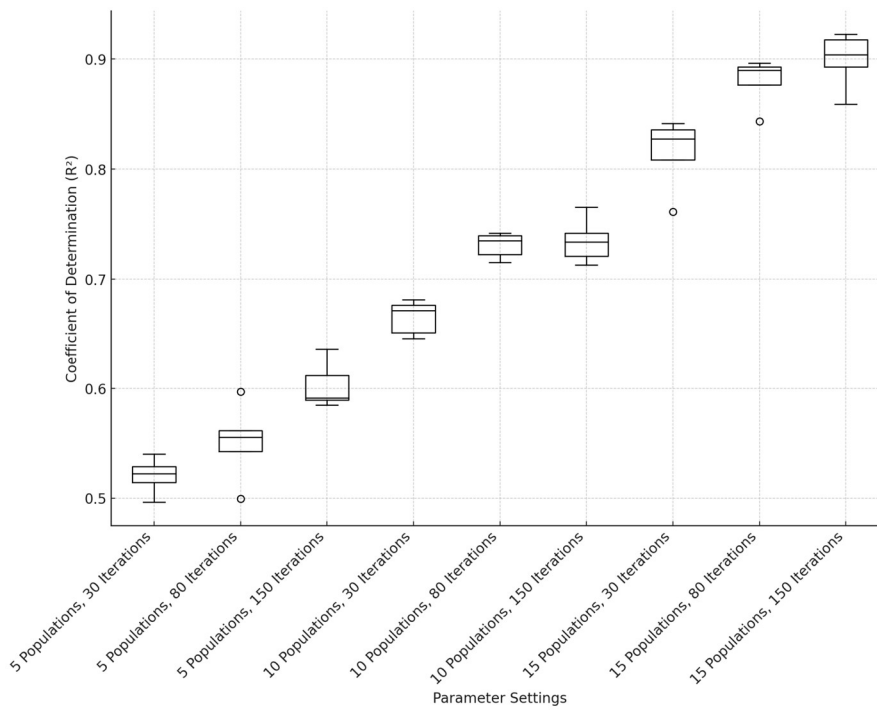


Figure A.33: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 10.

Table A.26: Summary Statistics for 5-Fold CV  $R^2$  Mean for Well 10 Symbolic Regression.

| Parameter setting              | $R^2$ Mean | Standard deviation |
|--------------------------------|------------|--------------------|
| 5 Populations, 30 Iterations   | 0.523      | 0.0121             |
| 5 Populations, 80 Iterations   | 0.565      | 0.0037             |
| 5 Populations, 150 Iterations  | 0.622      | 0.0129             |
| 10 Populations, 30 Iterations  | 0.673      | 0.0084             |
| 10 Populations, 80 Iterations  | 0.739      | 0.0095             |
| 10 Populations, 150 Iterations | 0.745      | 0.0106             |
| 15 Populations, 30 Iterations  | 0.828      | 0.0097             |
| 15 Populations, 80 Iterations  | 0.885      | 0.0147             |
| 15 Populations, 150 Iterations | 0.914      | 0.0232             |

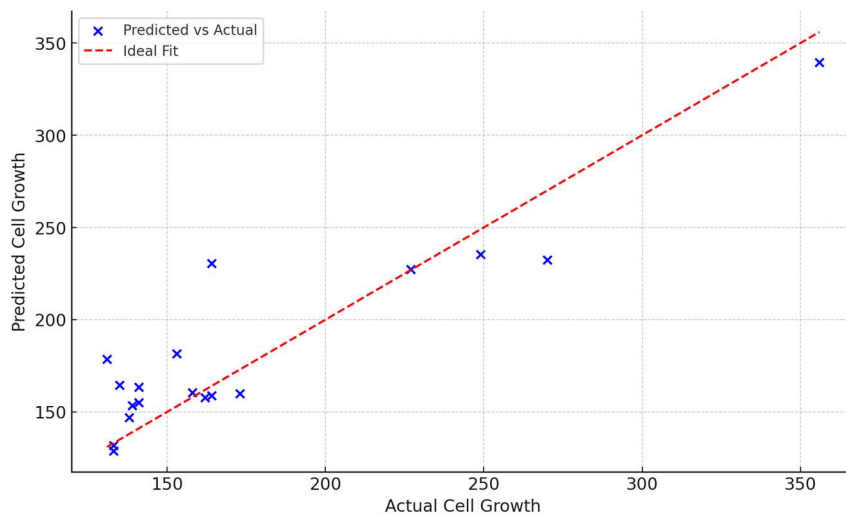


Figure A.34: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.818$ ).

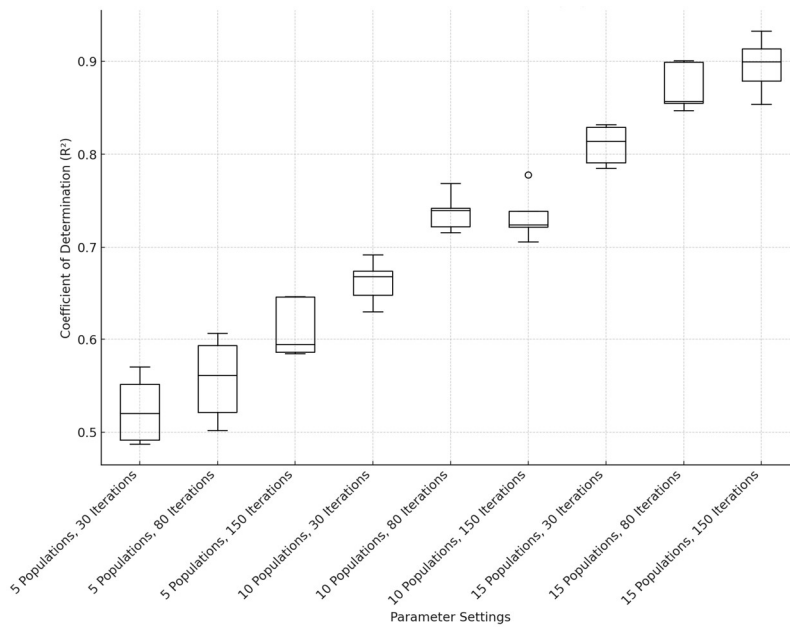


Figure A.35: Box Plot of Regression Model 5-Fold Cross-Validation Results for Well 11

Table A.27: Summary Statistics for 5-Fold CV  $R^2$  Mean for Well 11 Symbolic Regression.

| Parameter setting              | $R^2$ Mean | Standard deviation |
|--------------------------------|------------|--------------------|
| 5 Populations, 30 Iterations   | 0.5285     | 0.0182             |
| 5 Populations, 80 Iterations   | 0.5610     | 0.0233             |
| 5 Populations, 150 Iterations  | 0.6180     | 0.0151             |
| 10 Populations, 30 Iterations  | 0.6600     | 0.0114             |
| 10 Populations, 80 Iterations  | 0.7445     | 0.0132             |
| 10 Populations, 150 Iterations | 0.7340     | 0.0102             |
| 15 Populations, 30 Iterations  | 0.8130     | 0.0118             |
| 15 Populations, 80 Iterations  | 0.8785     | 0.0104             |
| 15 Populations, 150 Iterations | 0.9010     | 0.0190             |

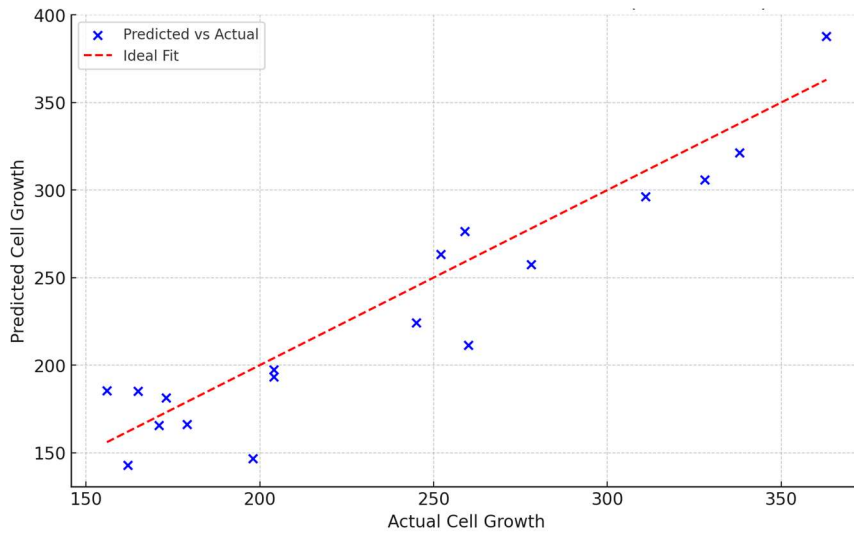


Figure A.36: Actual Vs Predicted Growth On Test Data ( $R^2 = 0.869$ ).

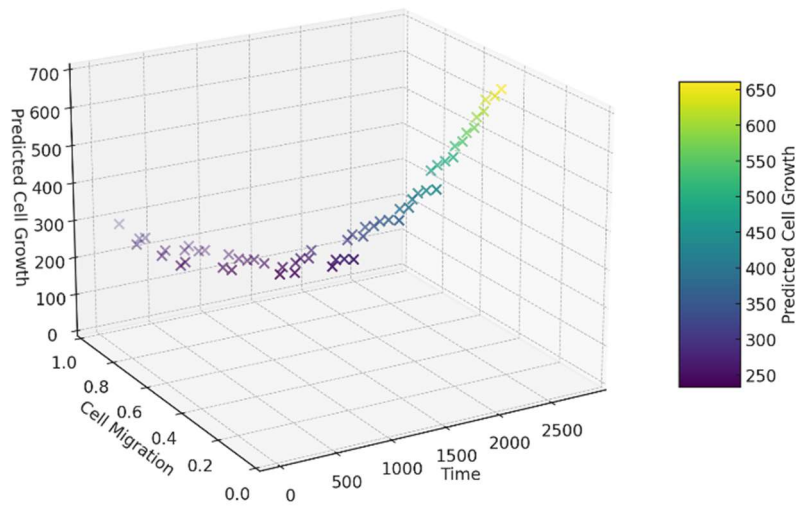


Figure A.37: 3D Scatter plot of Predicted Cell Growth vs. Time and Cell Migration for Well 1

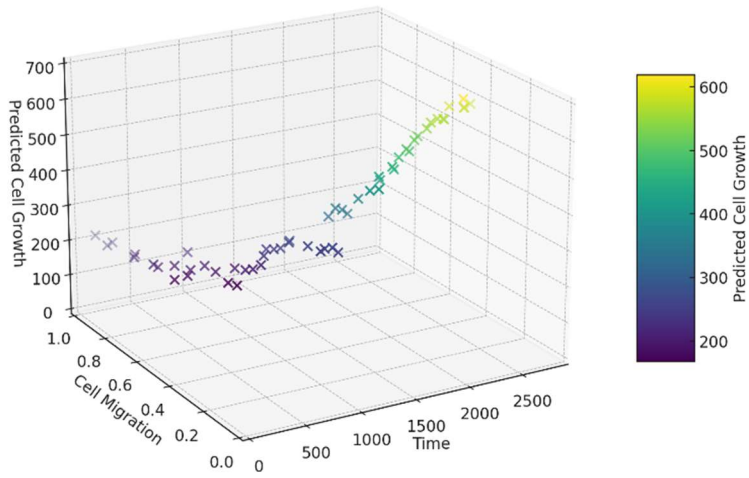


Figure A.38: 3D Scatter plot of Predicted Cell Growth vs. Time and Cell Migration for Well 2

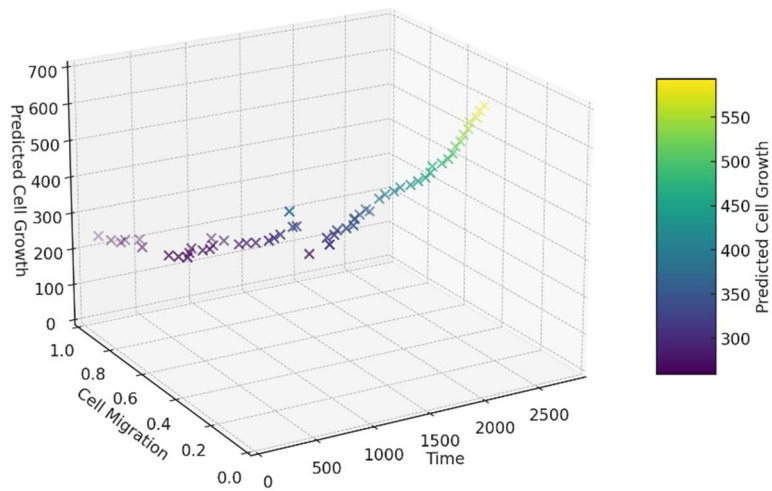


Figure A.39: 3D Scatter plot of Predicted Cell Growth vs. Time and Cell Migration for Well 3

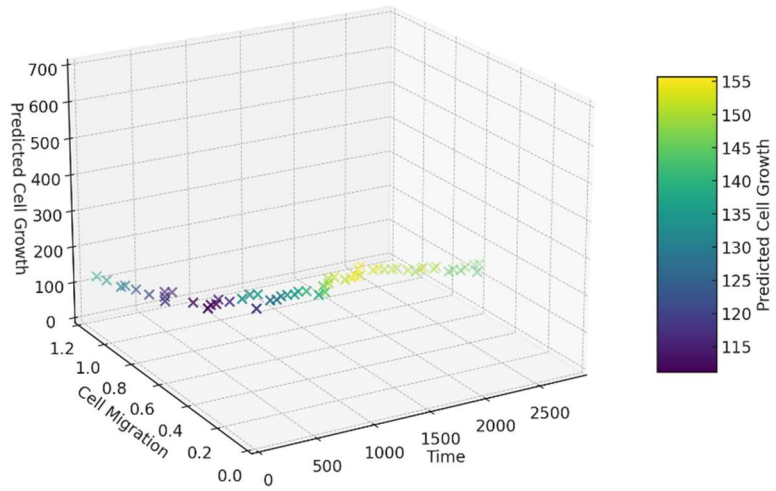


Figure A.40: 3D Scatter plot of Predicted Cell Growth vs. Time and Cell Migration for Well 5

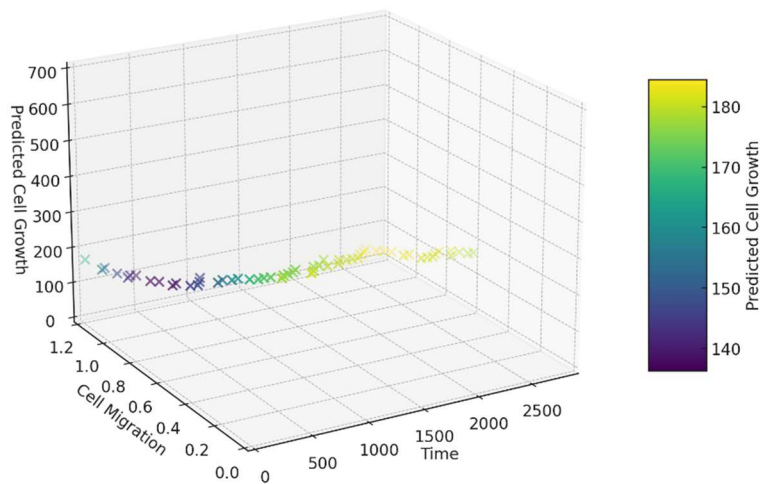


Figure A.41: 3D Scatter plot of Predicted Cell Growth vs. Time and Cell Migration for Well 6



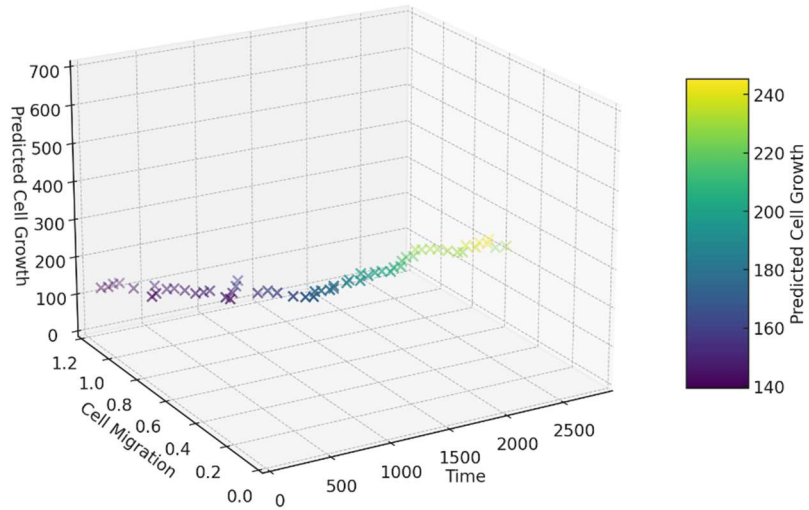


Figure A.42: 3D Scatter plot of Predicted Cell Growth vs. Time and Cell Migration for Well 7

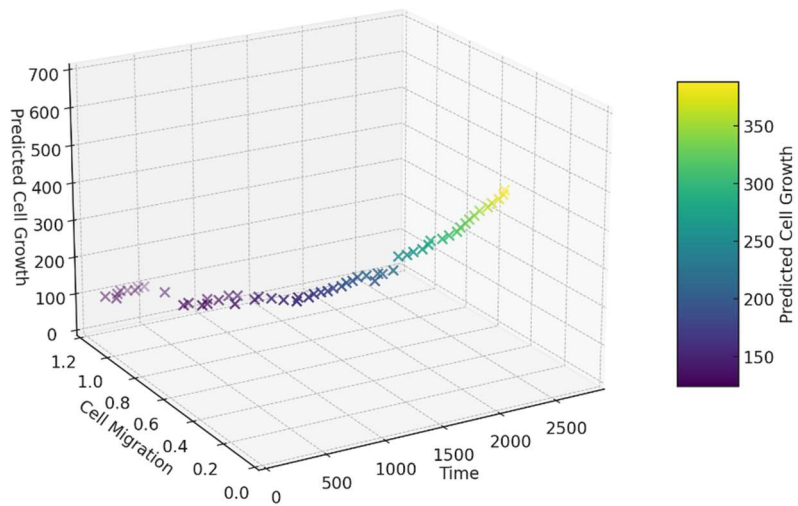


Figure A.43: 3D Scatter plot of Predicted Cell Growth vs. Time and Cell Migration for Well 9

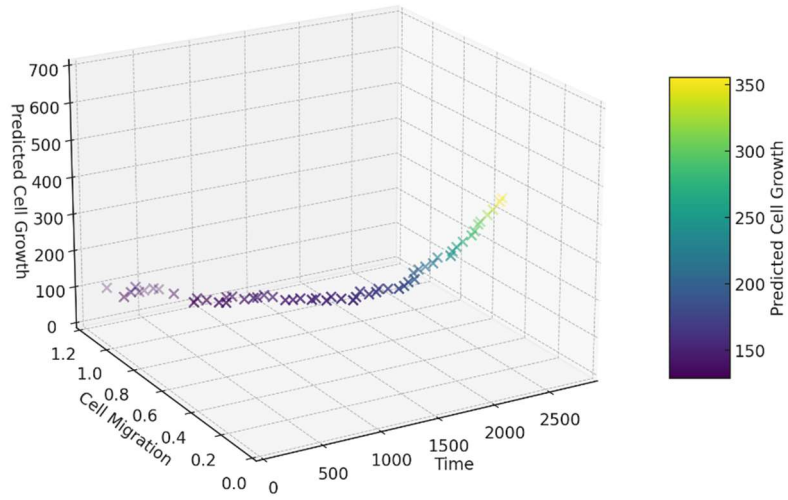


Figure A.44: 3D Scatter plot of Predicted Cell Growth vs. Time and Cell Migration for Well 10

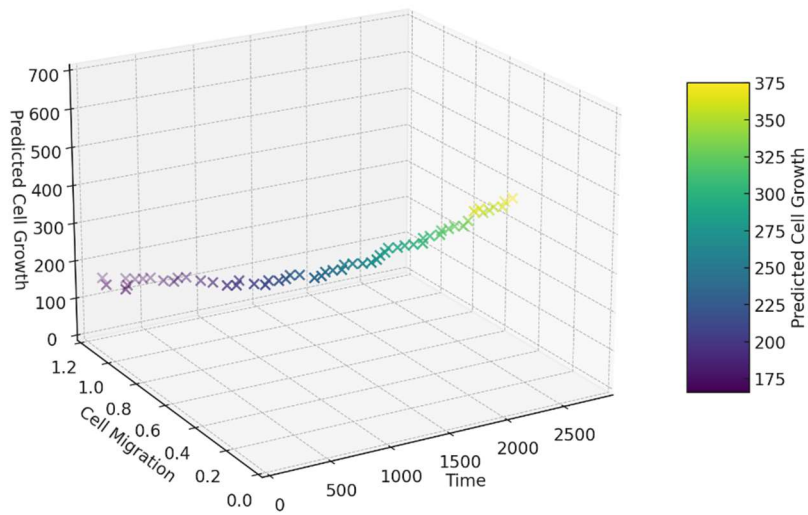


Figure A.45: 3D Scatter plot of Predicted Cell Growth vs. Time and Cell Migration for Well 11