Computer Science and Bioengineering, 2025

Image-Based Bioengineering Data Analysis and Query Tool Development Report

Contributor: Qizhi Tian[1], Xuan Gu[2], Jiatong Liu[3]

Abstract. This paper presents the development of an automated image analysis and query tool designed to transform unstructured bioengineering biology cell image data into structured, tabular formats, supporting natural language or SQL-like queries. The tool facilitates efficient data extraction, analysis, and visualization, enabling researchers to derive insights from complex cell-based datasets. By leveraging image processing pipelines, feature extraction techniques, and an interactive query interface, this project aims to streamline bioengineering research workflows and enhance data-driven decision-making.

Key words and phrases: Bioengineering, Automated Image Analysis, Data Transformation, Flexible query engine, Feature Extraction, Data Visualization, Computer vision

Table of Contents:

[1] *Equally contributor: Qizhi Tian, UCI, qizhit@uci.edu*
[2] *Equally contributor: Xuan Gu, UCI, xuang7@uci.edu*
[3] *Equally contributor: Jiatong Liu, UCI, jiatonl9@uci.edu*

1. Introduction

1.1 Background

Bioengineering research often relies on the analysis of complex datasets derived from experimental imaging, such as microscopy images of biological cells. These images contain valuable information about cellular attributes, including size, shape, density, and spatial distribution. However, the unstructured nature of image data poses significant challenges for efficient analysis and interpretation. Traditional manual methods of data extraction are time-consuming, error-prone, and impractical for large datasets. As a result, there is a growing need for automated tools that can process, analyze, and query image-based data effectively.

1.2 Problem Statement

The primary challenge in bioengineering image analysis lies in converting unstructured biology cell image data into structured, analyzable formats. Current methods often require extensive manual intervention, limiting scalability and reproducibility. Additionally, researchers lack intuitive tools to query and visualize complex datasets, hindering their ability to derive meaningful insights. Addressing these challenges requires the development of an automated pipeline for image analysis, feature extraction, and data transformation, coupled with an interactive query tool for seamless data exploration.

1.3 Objectives

This project aims to:

1. Develop an image processing pipeline to analyze bioengineering images, segment individual cells, and extract key cellular attributes.
2. Transform the extracted data into a structured, tabular format suitable for analysis.
3. Build an interactive query tool that supports natural language or SQL-like queries for efficient data exploration.
4. Provide visualization capabilities to represent query results effectively.

1.4 Significance

The proposed tool has the potential to revolutionize bioengineering research by automating data extraction and analysis processes. It will enable researchers to focus on hypothesis testing and discovery rather than manual data processing. By providing a user-friendly interface for querying and visualizing data, the tool will enhance accessibility and usability for researchers with varying levels of technical expertise.

2. Image Analysis and Feature Extraction

The image analysis and feature extraction pipeline is the core component of this project. It involves processing bioengineering images to identify and segment individual cells, followed by the extraction of key cellular attributes such as size, shape, position, and density. Below, we describe the methodology and implementation details of this pipeline, supported by mathematical formulations and code examples.

The codes provided in the implementation section are not fully included; they are presented as references to illustrate key concepts and improvements.

2.1 Image Processing

The first step in the pipeline is preprocessing the input images to enhance their quality and prepare them for segmentation. This includes noise reduction and contrast enhancement.

Noise Reduction:

Noise in the images is reduced using a median filter, which replaces each pixel's value with the median value of its neighborhood. This is particularly effective for removing salt-and-pepper noise while preserving edges. The operation is defined as:

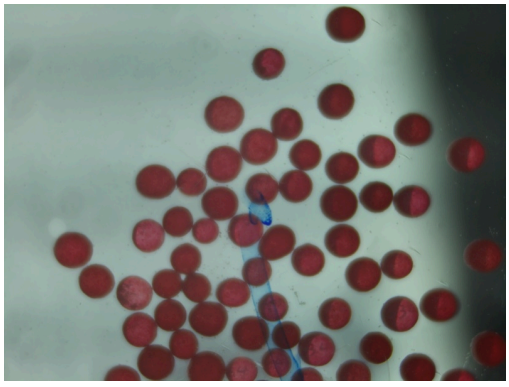
$$I\_\text{filtered}\,(x,y) = \text{median}(I\,(x-k,y-l)), \forall\,(k,l) \in W$$

where I(x,y) is the pixel intensity at the location (x,y), and W is the neighborhood window.

Contrast Enhancement:

Contrast-limited adaptive histogram equalization (CLAHE) is applied to improve the visibility of cell boundaries. CLAHE operates on small regions of the image, enhancing local contrast while limiting the amplification of noise

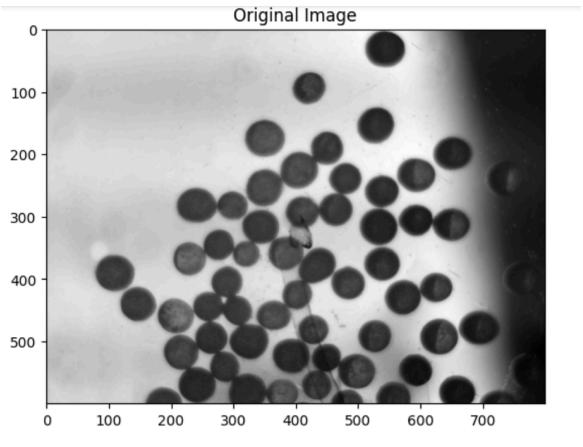
Code Implementation and effect:

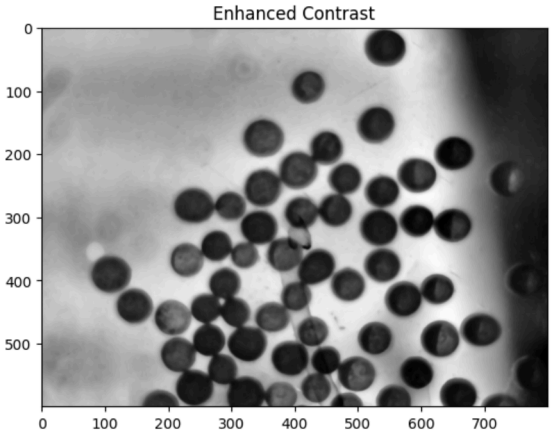


```python
# Load image in grayscale
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Apply median blur for noise reduction
image_filtered = cv2.medianBlur(image, 5)

# Apply CLAHE for contrast enhancement
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
enhanced_image = clahe.apply(image_filtered)
```





2.2 Threshold Segmentation

After preprocessing, the image is segmented into foreground (cells) and background using binary thresholding. The threshold value is chosen empirically, and morphological operations are applied to remove noise and small artifacts.

Binary Thresholding:

$$I_{\text{binary}}(x,y) = \begin{cases} 255 & \text{if } I_{\text{enhanced}}(x,y) > T \\ 0 & \text{otherwise} \end{cases}$$

where T is the threshold value.

Morphological Operations:

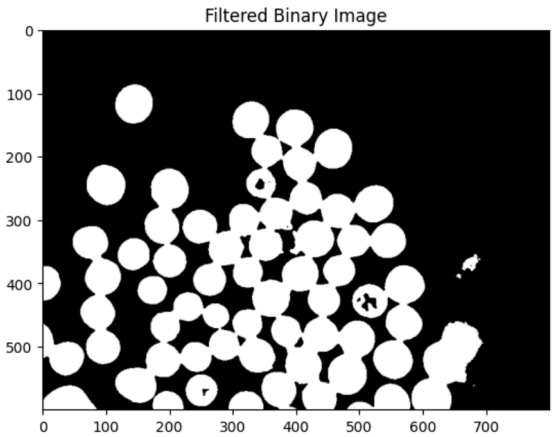
Morphological opening is performed to remove small noise regions. This involves erosion followed by dilation using a structuring element K:

$$I_{\text{opened}} = (I_{\text{binary}} \ominus K) \oplus K$$

Code Implementation and effect:

```
# Apply binary thresholding
_, binary_image = cv2.threshold(enhanced_image, 127, 255, cv2.THRESH_BINARY_INV)

# Perform morphological opening
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
binary_image = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)
```



2.3 Distance Transform and Seed Detection

The distance transform is applied to the binary image to compute the Euclidean distance of each foreground pixel to the nearest background pixel. Local maxima in the distance transform are identified as seed points for cell segmentation.

Distance Transform:

$$D(x, y) = \min_{(i,j) \in \text{background}} \sqrt{(x - i)^2 + (y - j)^2}$$

Local Maxima Detection:

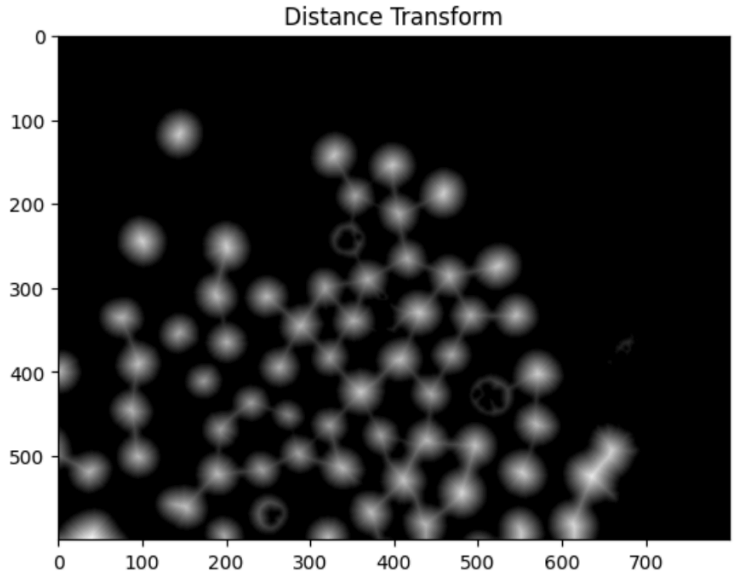
$$M(x, y) = \begin{cases} 1 & \text{if } D(x, y) \text{ is a local maximum} \\ 0 & \text{otherwise} \end{cases}$$

Code Implementation and effect:

```
# Compute distance transform
dist_transform = cv2.distanceTransform(binary_image, cv2.DIST_L2, 5)

# Detect local maxima
local_max = maximum_filter(dist_transform, size=30)
maxima = (dist_transform == local_max) & (dist_transform > 0.3 * dist_transform.max())

# Label seed points
labeled_maxima, num_cells = label(maxima)
```



2.4 Watershed Segmentation

The watershed algorithm is used to segment the cells based on the detected seed points. This algorithm treats the image as a topographic surface, flooding it from the seed points to separate regions.

Watershed Algorithm:

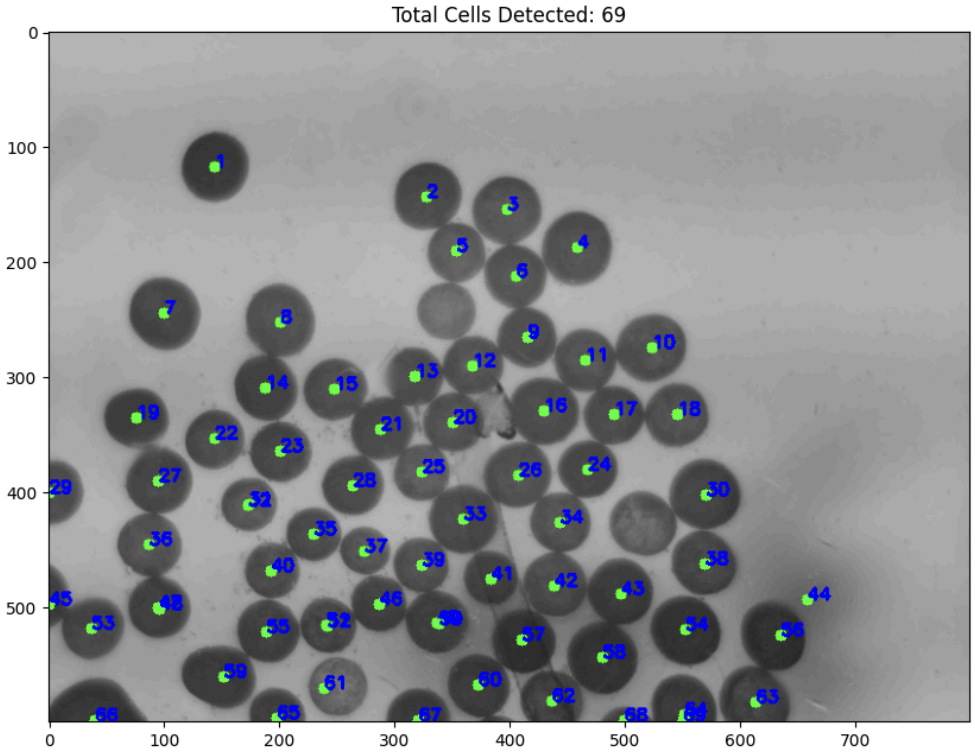
$$L(x, y) = \text{watershed}(I_{\text{enhanced}}, \text{markers})$$

where L(x,y) is the labeled segmentation result.

Code Implementation and effect:

```
# Apply watershed algorithm
markers = np.zeros_like(binary_image, dtype=np.int32)
seed_indices = np.argwhere(labeled_maxima)
for idx, (y, x) in enumerate(seed_indices, start=1):
    markers[y, x] = idx

watershed_labels = cv2.watershed(cv2.cvtColor(binary_image, cv2.COLOR_GRAY2BGR), markers)
```




```
Cell 1: Area = 2883, Perimeter = 169, Position = (144, 117), Local Density = 0
Cell 2: Area = 2662, Perimeter = 164, Position = (328, 142), Local Density = 0
Cell 3: Area = 2671, Perimeter = 166, Position = (397, 155), Local Density = 0
Cell 4: Area = 2943, Perimeter = 173, Position = (459, 187), Local Density = 0
Cell 5: Area = 3232, Perimeter = 258, Position = (352, 205), Local Density = 0
Cell 6: Area = 2291, Perimeter = 168, Position = (406, 212), Local Density = 1
Cell 7: Area = 3047, Perimeter = 173, Position = (99, 245), Local Density = 0
Cell 8: Area = 2976, Perimeter = 176, Position = (200, 251), Local Density = 0
Cell 9: Area = 2102, Perimeter = 155, Position = (415, 266), Local Density = 2
Cell 10: Area = 2761, Perimeter = 167, Position = (523, 274), Local Density = 0
Cell 11: Area = 2334, Perimeter = 164, Position = (466, 286), Local Density = 0
Cell 12: Area = 4312, Perimeter = 331, Position = (368, 268), Local Density = 1
Cell 13: Area = 2012, Perimeter = 148, Position = (317, 301), Local Density = 0
Cell 14: Area = 2453, Perimeter = 161, Position = (187, 309), Local Density = 0
```

2.5 Feature Extraction

For each segmented cell, key features such as area, perimeter, centroid, and local density are computed.

Area:

$$A = \sum_{(x,y)\in R} 1$$

Perimeter:

$$P = \sum_{(x,y)\in \partial R} 1$$

Centroid:

$$(c_x, c_y) = \left(\frac{\sum_{(x,y)\in R} x}{A}, \frac{\sum_{(x,y)\in R} y}{A} \right)$$

Local Density:

$$\rho = \frac{\text{Number of neighbors within radius } r}{\pi r^2}$$

Code Implementation:

```
def compute_perimeter(region_mask):
    eroded_image = binary_erosion(region_mask, structure=np.array([[0, 1, 0], [1, 1, 1], [0, 1,
0]]))
    boundary = logical_xor(region_mask, eroded_image)
    return np.sum(boundary)

def compute_cell_centroids(watershed_labels, num_cells):
    centroids = []
    for cell_label in range(1, num_cells + 1):
        region_mask = (watershed_labels == cell_label).astype(np.uint8)
        moments = cv2.moments(region_mask)
        if moments["m00"] != 0:
            cx = int(moments["m10"] / moments["m00"])
            cy = int(moments["m01"] / moments["m00"])
            centroids.append({"label": cell_label, "x": cx, "y": cy})
    return centroids

def compute_local_density(centroids, radius=50):
    positions = np.array([[cell["x"], cell["y"]] for cell in centroids])
    distances = cdist(positions, positions)
    local_densities = np.sum(distances < radius, axis=1) - 1
    for i, cell in enumerate(centroids):
        cell["local_density"] = local_densities[i]

# Compute features for each cell
cell_features = []
centroids = compute_cell_centroids(watershed_labels, num_cells)
compute_local_density(centroids, radius=50)

for centroid in centroids:
    region_mask = (watershed_labels == centroid["label"]).astype(np.uint8)
    area = np.sum(region_mask)
    perimeter = compute_perimeter(region_mask)
    cell_features.append({
        "Image Name": image_name,
        "Cell ID": centroid["label"],
        "X Position": centroid["x"],
        "Y Position": centroid["y"],
        "Area": area,
        "Perimeter": perimeter,
        "Local Density": centroid["local_density"]
    })
```

2.6 Summary Statistics

For each image, summary statistics such as average cell area, perimeter, and local density are computed and stored.

Code Implementation:

```
# Compute image-level summary statistics
avg_area = np.mean([cell["Area"] for cell in cell_features])
avg_perimeter = np.mean([cell["Perimeter"] for cell in cell_features])
avg_local_density = np.mean([cell["Local Density"] for cell in cell_features])

image_summary = {
    "Image Name": image_name,
    "Total Cells": num_cells,
    "Average Area": avg_area,
    "Average Perimeter": avg_perimeter,
    "Average Local Density": avg_local_density
}
```

3. Transformation and Tabular Format

3.1 Overview

In this study, we developed a structured approach to transform unstructured image-based bioengineering data into a tabular format. The dataset consists of images from bioengineering experiments containing groups of cells. Through image processing and feature extraction, we converted this unstructured visual data into a structured dataset that allows for efficient storage, analysis, and querying.

3.2 Structuring the Data

After extracting individual cell attributes, we organized the data into a well-structured tabular format:

1. Cell-Level Data (cell_data.csv) – Stores information for each detected cell:

Image Name	Cell ID	X Position	Y Position	Area	Perimeter	Local density
30_01_2024_18.png	15	48	491	3145	178	0
30_01_2024_18.png	16	407	293	466202	3247	0
26_01_2024_05.png	1	25	77	751	100	1
26_01_2024_05.png	2	13	97	612	108	1
26_01_2024_05.png	4	253	298	2925	170	0
26_01_2024_05.png	5	108	302	373	67	0
26_01_2024_05.png	6	39	306	3089	292	0

2. Image-Level Summary (image_summary.csv) – Aggregates statistics for each image:

Image Name	Total Cells	Average Area	Average Perimeter	Average Local Density
30_01_2024_18.png	16	234673.5	1712.5	0.0
26_01_2024_05.png	8	68058.28571428571	622.2857142857143	0.2857142857142857
30_01_2024_15.png	12	960.6666666666666	100.91666666666667	2.1666666666666665
28_01_2024_08.png	93	4654.698924731183	238.68817204301075	2.0
27_01_2024_10.png	68	6652.940298507463	249.91044776119404	1.671641791044776

These structured tables enable efficient querying and visualization, making it easier for researchers to analyze trends across different images and individual cells.

4. Interactive Query and Analysis Tool

4.1 Overview

To enable efficient exploration and analysis, we implemented an interactive query tool that supports:

1. SQL-like filtering using Pandas .query() for structured analysis.
2. Data visualization through histograms, scatter plots, and heatmaps.
3. Selection for Analysis of Cell-Level or Summary Data, allowing researchers to choose between analyzing individual cell attributes or image-level statistics.

To facilitate this, we provide an option to load either cell-level data (cell_data.csv) or image summary data (image_summary.csv), based on the user's selection.

This functionality allows researchers to toggle between granular cell analysis and broader image-level insights, making the tool adaptable for different research needs.

4.2 SQL-Like Querying

Researchers can retrieve specific subsets of data using SQL-like queries. The tool allows filtering based on cell attributes, location, and density.

Example Queries:

Research Question	Query Syntax
Find large cells with high density	`run_query("Area > 1000 and Local Density > 2", df_cells)`
Retrieve all cells from a specific image	`run_query("Image Name == 'image_01.png'", df_cells)`
Find images with high average cell density	`run_query("Average Local Density > 3", df_images)`
Identify irregularly shaped cells	`run_query("Perimeter / Area > 0.5", df_cells)`

Query executed: Area > 1000 and `Local density` > 2

	Image Name	Cell ID	X Position	Y Position	Area	Perimeter	Local density
28	28_01_2024_08.png	8	662	107	1356	124	3
31	28_01_2024_08.png	11	636	123	1064	111	4
44	28_01_2024_08.png	24	618	200	1242	120	3
54	28_01_2024_08.png	34	469	289	1841	156	7
61	28_01_2024_08.png	41	508	289	1367	143	8

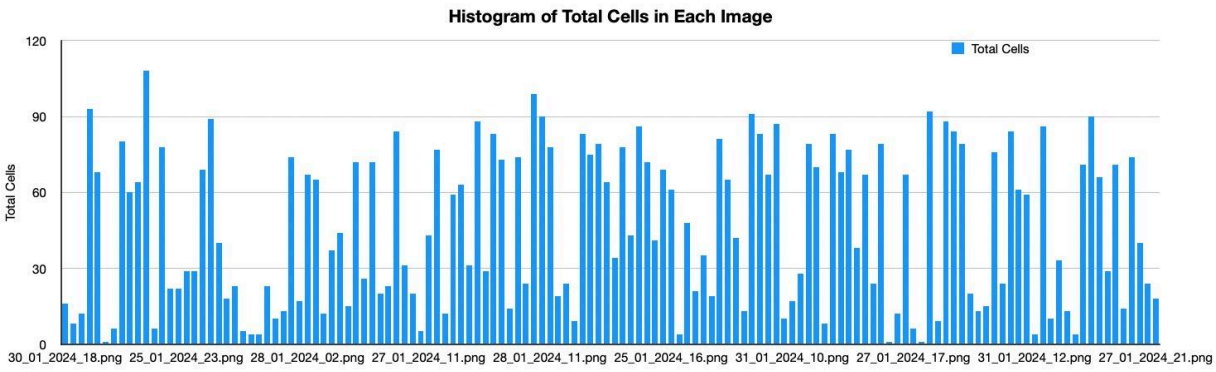
4.3 Data Visualization

To support deeper insights, the tool provides multiple visualization options:

1. Histogram (plot_histogram(column, df, data_type="cell")) – Displays distributions of cell attributes.
2. Scatter Plot (plot_scatter(x_col, y_col, df, data_type="cell")) – Visualizes spatial distribution of cells.
3. Heatmap (plot_heatmap(column, df, data_type="cell")) – Shows density and intensity of cell attributes across images.

Example Visualization:

Visualization Type	Purpose
Histogram	Explore the distribution of cell area, perimeter, or density
Scatter Plot	Visualize the spatial distribution of detected cells
Heatmap	Identify dense regions of high cell activity



5. Summary of Improvement from Version 1 to Version 2

The transition from Version 1 to Version 2 of the image analysis and feature extraction pipeline introduced several key improvements to enhance the system's robustness, accuracy, and usability. These improvements address challenges such as handling low-quality images, improving segmentation accuracy, and optimizing the pipeline for scalability. Below is a summary of the major improvements made in Version 2:

1. Adaptive Area Filtering

Version 1: The area threshold for filtering connected components was fixed, which sometimes led to under-segmentation (too few cells detected) in images with varying cell densities.

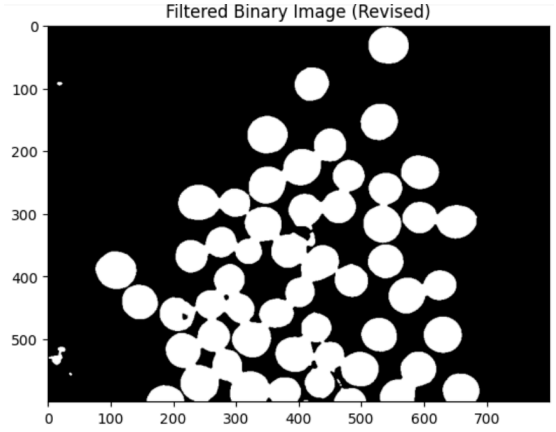
Version 2: Introduced an adaptive area filtering mechanism that dynamically adjusts the area threshold based on the number of detected cells. If the number of detected cells is below a predefined threshold (e.g., 10), the area threshold is iteratively reduced until a sufficient number of cells are detected.

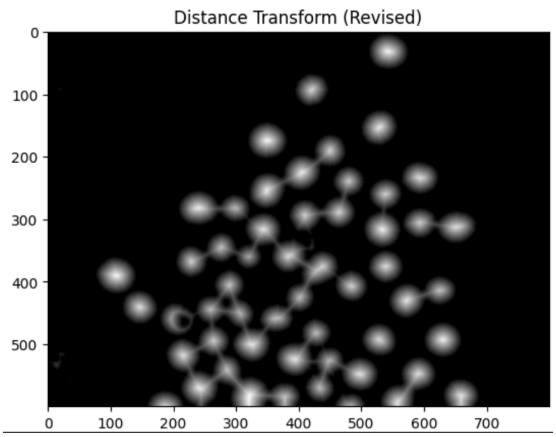
Impact: Improves segmentation accuracy for images with varying cell densities and sizes.

2. Enhanced Seed Point Detection

The distance transform highlights potential seed points, but some regions may still have overlapping or indistinct maxima, leading to under or over-segmentation.

Improvement: Introduce a multi-scale maxima detection approach: Apply the distance transform at multiple scales (e.g., varying window sizes for local maxima detection). Combine results from different scales to identify robust seed points. Then Reapplying area filtering with the stricter threshold.





Improvement	Description	Impact
Adaptive area filtering	Dynamically adjusts the area threshold for connected component filtering.	Improves segmentation accuracy for images with varying cell densities.
Enhanced Seed Point Detection	Introduce a multi-scale maxima detection approach	Reduces errors in seed point detection, especially in regions with overlapping cells or irregular shapes.
Modularization	Encapsulates distance transform and seed detection logic in a reusable function.	Improves code readability, maintainability, and reusability.

6. Conclusion and Future Improvements

This project successfully developed an automated pipeline for analyzing bioengineering images, extracting key cellular features, and transforming unstructured image data into a structured, tabular format. The proposed system addresses the challenges of manual data extraction and analysis, offering a scalable and reproducible solution for bioengineering research. By automating these processes, the tool significantly reduces the time and effort required for data processing, allowing researchers to focus on deriving insights and advancing their studies. The integration of natural language querying capabilities further enhances the usability of the tool, making it accessible to researchers with varying levels of technical expertise.

The results demonstrate the effectiveness of the pipeline in handling diverse bioengineering datasets, providing accurate and reliable feature extraction. The visualization capabilities, including histograms, scatter plots, and heatmaps, enable researchers to explore and interpret their data intuitively. Overall, this project contributes to the growing field of bioinformatics by providing a powerful tool for image-based data analysis and querying.

Future Improvements

Looking ahead, we aim to enhance the system by incorporating several key advancements:

1. Natural Language Querying: Implementing NLP-based queries will allow researchers to interact with the dataset using conversational language, making data analysis even more intuitive.
2. Machine Learning-Based Cell Classification: We plan to integrate deep learning models, such as convolutional neural networks (CNNs), to improve cell

segmentation and feature extraction accuracy, especially for images with overlapping cells or complex backgrounds.

3. Cloud-Based Deployment: Deploying the tool on a cloud platform will enable researchers to access and analyze data remotely, facilitating collaboration and data sharing across institutions.
4. Enhanced Visualization for Different Requirements: To accommodate diverse research needs, we plan to refine and expand visualization options. This includes customizable color schemes, interactive plots, and adaptive graphical outputs tailored to user-defined parameters.

By implementing these improvements, we aim to further enhance usability, accuracy, and accessibility, making this tool a robust platform for bioengineering data analysis.

7. Appendices

Dataset Sources:

The bioengineering cell images used in this project were provided by the Data Science Programs for Bioengineering Applications, a collaborative initiative between the University of California, Irvine (UCI) and the University of California, San Diego (UCSD). These datasets comprise high-resolution microscopy images of biology cell, which are essential for developing and validating the image analysis and query tool described in this paper.

Acknowledgments:

We would like to express our sincere gratitude to the following individuals for their guidance, and support to this project: Professor Faisal Nawab, Professor Chen Li, and Professor Reem Khojah.

Their mentorship has been instrumental in the successful completion of this project, and we are deeply thankful for their time and encouragement.