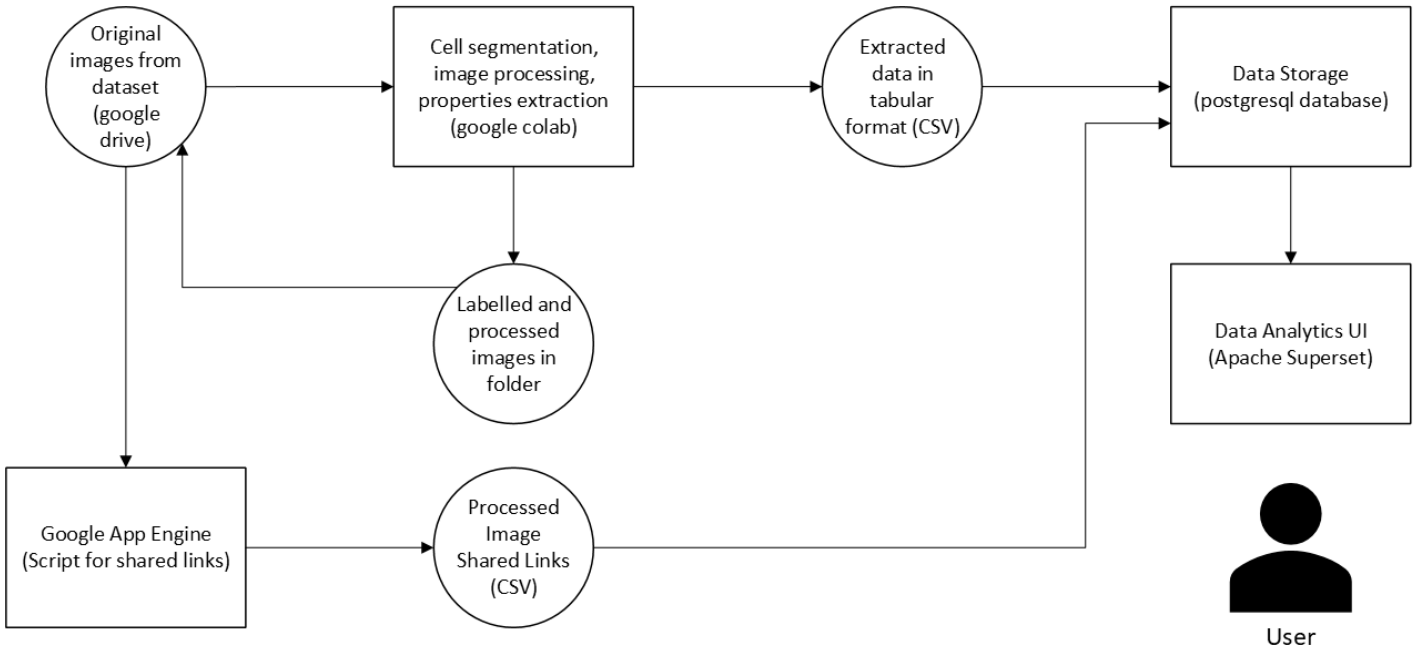
Datascience and bioengineering: Final Project

Image-Based Bioengineering Data Analysis and Query Tool Development

Project description:
In this project, the aim was to automate the segmentation of cells in the dataset provided by images from bioengineering experiments to extract meaningful attributes. Traditional manual analysis is time-consuming, making automated approaches crucial for accuracy and scalability.

Approach and design:
This is our workflow and different components involved in the end-to-end process.



The image files and dataset were provided in the project and were stored in drive while I coded in colab but here have been put in github.

Assumptions:

1. Incomplete cells at the border were not considered as cells and are ignored.

Tools used:

1. Python/Colab - Coding
2. Docker - Deployment of superset and postgres
3. Google drive, app engine - Datasource and sharing
4. Apache Superset - UI and query tool
5. LabelMe - Annotating data for *ongoing* work

Initial Approach: OpenCV and Watershed Segmentation:
Initially, I planned to segment individual cells and extract key attributes using OpenCV-based contour detection. As I explored further, I decided to refine the approach using:

- Watershed segmentation
- Contrast Limited Adaptive Histogram Equalization (CLAHE)

- Thresholding with distance transform
- Gaussian blur

..and more.

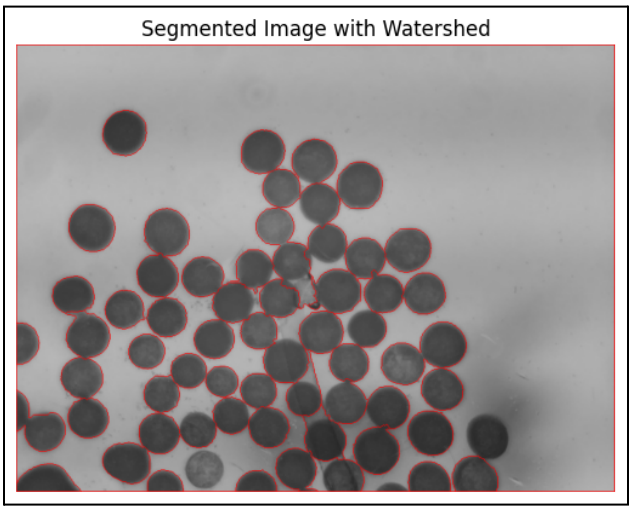
Watershed segmentation usually is useful for places where objects (cells in our context) overlap/connect. The cells get separated by treating the image taken as input like a topographic surface. In this, high-intensity regions represent peaks and low-intensity regions represent valleys. The algorithm "floods" the valleys filling basins until they meet and that forms segmentation boundaries.

Thresholding enhances object separation before applying watershed. It converts an image to binary (black/white) and helps identify foreground (cells) and background.

I initially tested the approach on single images before applying it to the entire dataset. My first attempt involved converting images to grayscale before segmentation. However, I soon noticed that:

- Some overlapping/touching cells were not segmented properly.
- The contrast between the background and cells was similar so insufficient for effective boundary detection.
- While separation improved, extensive preprocessing was required.

An example of the output is shown below, which appears promising at first glance but is ineffective upon closer inspection.



Exploring Other Methods: Mask R-CNN and HSV Processing:
After evaluating the limitations of OpenCV segmentation, I considered machine learning approaches such as:

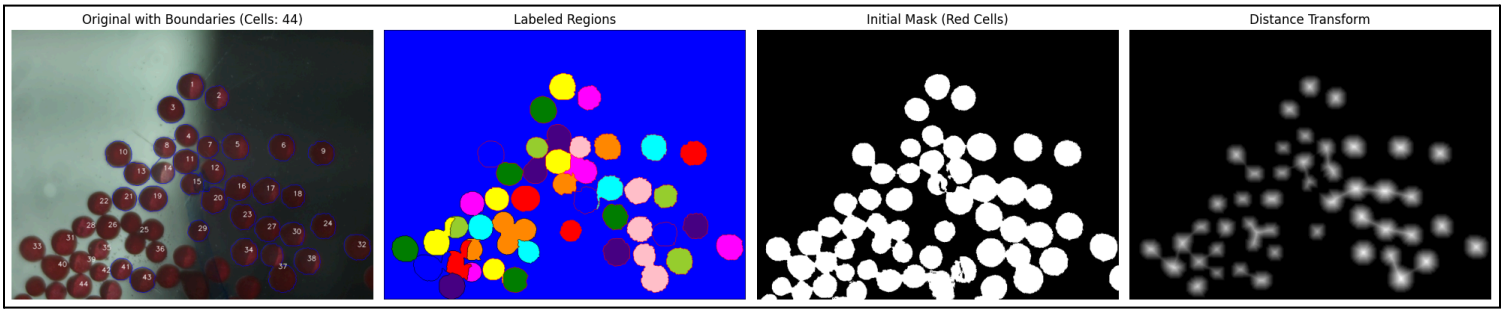
- Mask R-CNN
- Segment Anything Model (SAM) by Meta

However, the lack of annotated training data forced us to explore different segmentation methods. Since I had already spent significant time on watershed segmentation, I decided to refine it further.

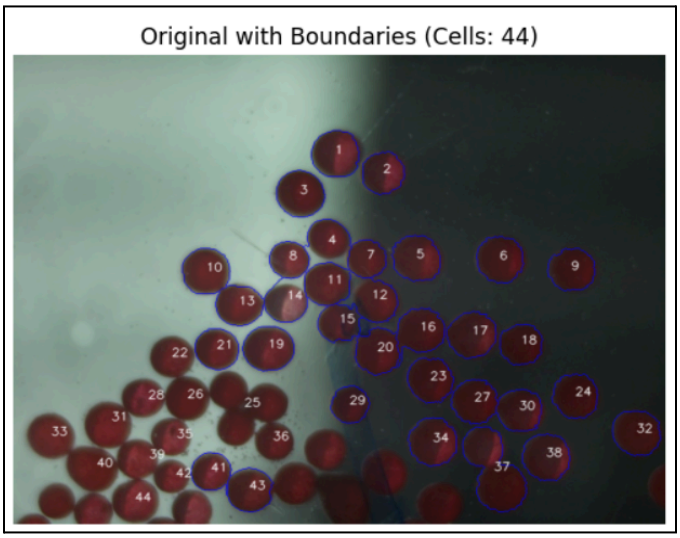
Instead of grayscale, I switched to HSV (Hue, Saturation, Value) image processing, as the dataset contained red cells while the rest of the image had grayscale or blue marks. Isolating the red color helped improve contour detection.
Challenges with HSV Processing:
Initially, I encountered issues like:

- Incorrect labeling
- Inaccurate feature extraction

After several refinements, I was able to achieve ~70% accuracy in processed images, as shown below.



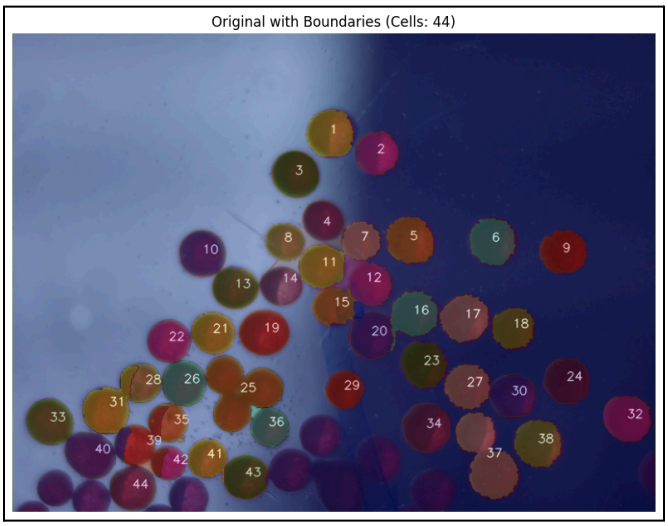
Additional Issues: Displaying Segmentation Boundaries:



A new problem emerged—boundaries were not displayed correctly for many cells. Some:

- Were improperly outlined (e.g., grainy or pixelated edges)
- Had conjoined parts with other cells (e.g., Cell 31 and Cell 28)
- Were grouped together as one contour (e.g., Cell 25 contained three cells)
- Were entirely missed in detection (e.g., Cells 40, 29, 34, and 43)

To improve visualization, I overlaid the “labeled regions” image (from the whole processed list above including original with boundaries, labelled regions, initial mask red cells and distance transform) on top of the segmented output to help users differentiate cells more easily. Below is a look of the final output image.



Obviously it is realised that there are shortcomings (as mentioned before) and a lot of improvements can be made in extracting these cells like (few examples):

1. Cell 31 has a conjoined part with cell 28 and isn't segmented well (same with another bunch of cells).
2. Cell 25 consists of 3 cells grouped together in one contour. Cell 37 the same but with 2 grouped.
3. Few cells like below 40, 29, 34 and beside 43 + more are not being counted.
4. Some boundaries are grainy and not properly circled over the whole cell.

Scaling Up: Processing the Entire Dataset:
I added a loop to iterate over all 136 images in the dataset, segmenting cells and extracting their features.

Due to time constraints, I had to wrap up image processing at this stage and move on to the next step:

- Extracting structured data
- Transforming it into a tabular format
- Developing an interactive query and analysis tool

The extracted attributes included:

- Cell size (since no scale was provided, at pixel level)
- Centroid (position)
- Perimeter
- Circularity (shape)
- Bounding box
- Aspect ratio
- Solidity

At first I used it to print out all the data for each cell in single images as shown below.

```
Extracted Attributes:
Cell 1:
  Size (Area): 2774.5
  Centroid: (495, 214)
  Perimeter: 205.58073377609253
  Circularity: 0.8249540699247633
  Bounding Box: (466, 183, 59, 63)
  Aspect Ratio: 0.9365079365079365
  Solidity: 0.9685809041717577
Cell 2:
  Size (Area): 2278.5
  Centroid: (421, 257)
  Perimeter: 182.2670258283615
  Circularity: 0.8618718237301098
  Bounding Box: (393, 233, 58, 51)
  Aspect Ratio: 1.1372549019607843
  Solidity: 0.9783168741949334
Cell 3:
```

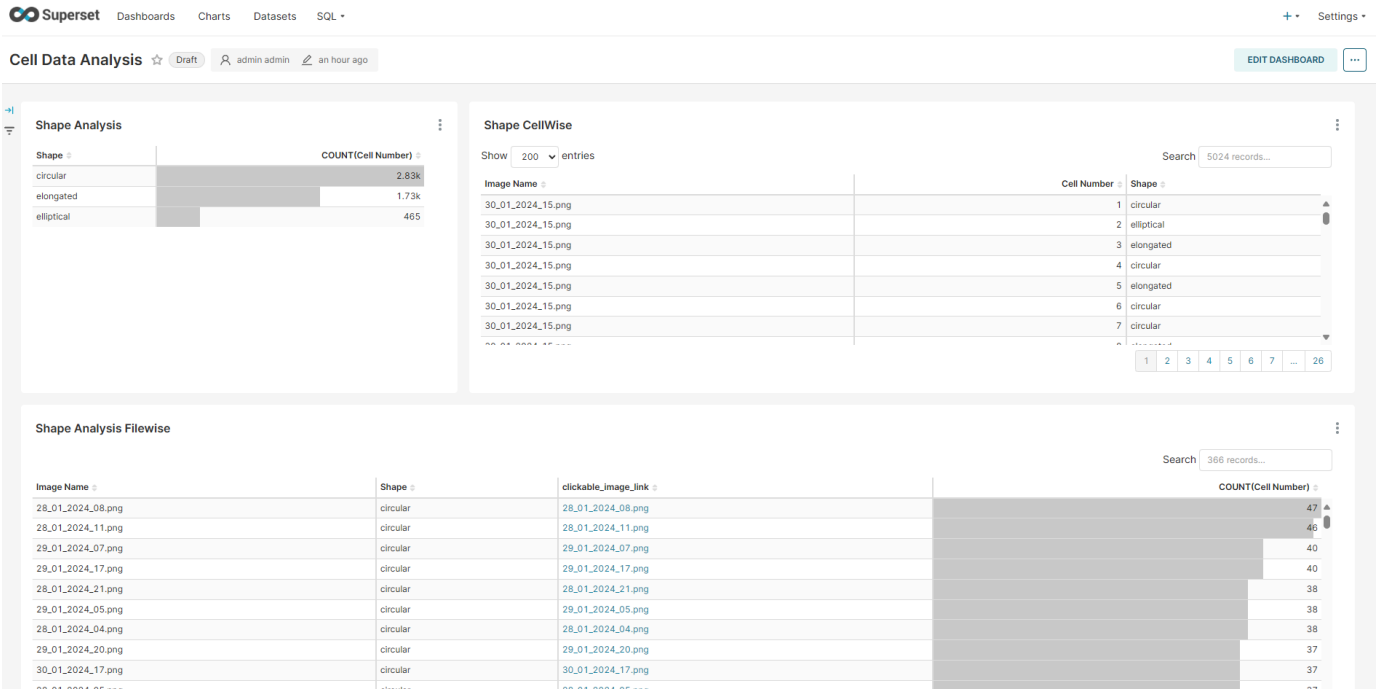
Later on after I confirmed that it worked according to my images, I looped it for the whole dataset like earlier and using pandas- made a DataFrame where I put my data in tabular format to save and export in CSV. I chose CSV for easy access across devices, as my code ran on Google Colab, and the planned UI (using Apache Superset) would be deployed on my local network.

Visualization with Apache Superset:
This provided an intuitive interface to query the dataset and visualise results in different formats and for our data we have chose to visualise it with:

- Tables
    - Shape wise (overall in all the files)

    - Shape analysis cell wise (trackable to each individual cell in each file)
    - Shape analysis file wise (file wise shape distribution)
- Categorising the cell shape as circular, elongated or elliptical.
    - It is based on a custom calculation as written below and can be changed in the superset easily.
        - If Circularity is between 0.8 and 1 and Aspect Ratio is between 0.9 and 1.1, the shape is circular.
        - If Circularity is between 0.7 and 1.0 and Aspect Ratio is greater than 1.1, the shape is elliptical.
        - Otherwise, the shape is elongated.
- Superset as a query tool. You can write your own SQL queries on the data.

Below is a screenshot of the UI's look:



Ongoing & Future Work: Implementing Mask R-CNN:
While the current image processing method is functional, it is not efficient. I did not want to stop here despite needing to submit the project before achieving the desired level of accuracy.

I returned to Mask R-CNN, a deep learning framework for generating segmentation masks. Since no annotated training data was provided, I began manually annotating images using LabelMe.

Current Progress:

- Annotating 5 images from the dataset
- Working on the first image, manually labeling cells

After completing the annotation, I plan to use pre-trained models and fine-tune them. I am currently exploring Detectron2 and similar models. My primary learning resources are YouTube tutorials and ChatGPT. As of now, I am confident that Mask R-CNN will be an effective approach.

References:

1. ChatGPT - as a teacher throughout the project and proofreading
2. Youtube to help with understanding of different algorithms and models
3. LabelMe - Annotation Tool, https://github.com/wkentaro/labelme