CellEstimator

Project report Group 06

Raimundo Becerra Parra Leonardo Fernandes Oliveira Konstantinos Larintzakis

Wafa Laroussi Michael Lemanov Ivan Nikolovski

July 25, 2023

Motivation

The field of blood cell analysis is central in modern healthcare, as it provides valuable insights into various health aspects. One of the most common blood tests is the complete blood count. This test serves not only to check a patient's overall health but also to diagnose and track different conditions such as anemia and leukemia [1]. In order to perform this test, peripheral blood is taken from the patient and analyzed manually or by automated analyzers. In the automated procedure, if the result is reported as abnormal, a manual peripheral blood smears review is usually required. The abnormality rate could range between 10% to 20% in certain cases [2]. These conventional methods usually suffer from cost and time inefficiencies. The cost of a hematology analyzer is high, and sometimes separate devices for different diagnoses are required. Regarding sample preparation, the time needed with traditional methods, like the widely used Giemsa stained blood smear, could be up to 45 minutes for a single slide [3]. For that, medical professionals must be paid, which can also be costly for healthcare institutions. Moreover, the analysis itself can also prove to be inefficient in time, as it depends on the task's complexity and the laboratory's workload. The aforementioned drawbacks highlight the need for alternative approaches that enable more timely interventions.

1 Project Description And Goals

In response to these challenges, the CellFace project was initiated [4]. The leading advanced methods used in that work are flow cytometry, in combination with digital holographic microscopy. This approach offers the capability to capture 3D images of individual blood cells while processing many samples quickly and efficiently. Further integration of state-of-the-art machine learning and computer vision models bears the potential of reducing costs and increasing the efficiency of blood cell analysis by limiting manual intervention and streamlining the entire process.

Our contribution to this work focuses on developing a pipeline that interpreters phase images extracted by the generated 3D holograms. The pipeline should support and display various steps, including

- **Segmentation:** The pipeline should employ advanced image processing techniques that achieve reliable cell segmentation. On top of that, the web interface should allow for manual segmentation in case of misclassified or missed cells.
- **Feature Extraction:** Feature engineering techniques should be applied to extract the morphological characteristics from the cells and, thus, communicate the results better among the medical target group.
- **Classification:** State-of-the-art machine learning models should be used to distinguish different cell types based on extracted features. High classification accuracy is desirable but not the main focus of this work.
- **Active Learning:** The pipeline should incorporate active learning techniques to improve the classification models' performance. Retraining of the segmentation methods used is beyond the scope of this work.
- **User Interface:** A user interface should be designed to present the results clearly, enabling medical experts to make informed decisions based on the analysis.

Two main challenges should guide the creation of the described software application. First, the interpretability of the results should be prioritized over accuracy. Various black-box deep learning models that learn directly on the provided images promise high accuracy rates but often lack the ability to explain the criteria involved in prediction outcomes. As these criteria may also have no medical meaning, accepting this work from the medical community may be compromised. Furthermore, the user interface should be ergonomic and intuitive for a non-machine learning expert, ensuring a seamless adoption and productive usability of the designed software.

2 Data basis

Our project employs Quantitative Phase Imaging (QPI), a subset of holographic microscopy offering a unique cell imaging perspective. Unlike traditional light microscopy that focuses on light intensity and absorption, QPI investigates phase shift, i.e., how much light is delayed when it traverses an optically denser material. This technique is particularly well-suited to cellular imaging, as cells, while mostly transparent, are constructed from materials of varying optical densities. With QPI, we can observe cells in the phase domain without the necessity of labor-intensive staining processes.

Interpreting the resulting images requires an understanding that we are not dealing with ordinary images. The captured holograms undergo a reconstruction process to yield phase and amplitude images with unique properties. These images are stored as tensors, each containing just one channel with values in float16 format. For amplitude images, pixel values denote light absorption, while for phase images, they represent the delay of light depending on the cell size and its optical density.

The data we are working with are derived from real blood samples from healthy donors. These samples comprise various cell types, including Red Blood Cells (RBCs), White Blood Cells (WBCs), and Platelets (PLT). Additionally, there may be instances of cell aggregation (AGG), where cells stick together, and out-of-focus (OOF) cells, which occur when a cell floats by above or below the microscope's focal plane.

The data will come in several formats: real-world samples, simplified samples, and an example prediction file. The real-world samples reflect whole blood samples with a significant presence of red blood cells. Simplified samples contain a more balanced distribution of different cell types. Finally, the example prediction file offers a selection of segmented and labeled cells from each cell type, providing a snapshot of the different cells you can expect to encounter.

3 Segmentation

In order to perform cell segmentation, our app supports three alternatives:

- **Threshold:** a threshold-based segmentator, which applies "classical" image processing techniques found in the OpenCV library in order to come up with cell masks. It applies a global threshold to find general regions of interest (ROIs), and then applies a regional threshold to each ROI in order to segment finer details. Optimized for *amplitude* images.
- **Cellpose:** open source deep learning-based segmentation method [5, 6] trained on images of multiple cell types, with API available as a Python library. We use their pre-trained *"cyto"* model, with parameters optimized for *phase* images.
- **FastSAM:** a patched implementation[1] of the FastSAM [7] segmentation method, with pre-trained weights. FastSAM is an optimized version of the "Segment Anything" model [8], a generalist segmentation model proposed by Meta researchers. It works well in both *amplitude* and *phase* images. Overlapping masks are common when using this method, meaning that the user will have to delete some predicted masks for an accurate cell count estimation.

In the user web interface, the user is able to choose between one of these methods, which enables him to analyze what suits him best in terms of segmentation speed and precision.

4 Feature Extraction

Given the small number of labeled examples, a traditional machine learning approach was necessary. This means that the feature extraction had to be done manually before any classification effort. In works [9, 10, 11], the authors demonstrated the effectiveness of multiple features extracted from QPI images, in the classification of different cell types. Note that all of these features require an

[1] `https://github.com/leofernandeso/FastSAM`

accurate pixel-level mask of the cell. In this work, we selected a subset of these features. For some features, we use the local standard deviation values of the image. These values are calculated by applying a square kernel around each pixel and calculating the standard deviation of all the values inside said kernel. In the user web interface, the user is able to visualize a 2D feature space, choosing any two features from the following list:

- **Volume:** Cell volume in μm^3.
- **Roundness:** How similar the cell's shape is to a perfect circle. The greater the value, the closest it is to a circle. Unitless.
- **Opacity:** How much light the cell lets pass through, in $1/\mu$m^2.
- **AmplitudeVariance:** Variance of the amplitude values found in the cell area. Unit-less.
- **AmplitudeSkewness:** Skewness of the amplitude values found in the cell area. Unit-less.
- **MaxAmplitude:** Maximum amplitude value found in the cell area. Unit-less.
- **MinAmplitude:** Minimum amplitude value found in the cell area. Unit-less.
- **DryMassDensity:** Dry mass density (i.e. its dry mass divided by the cell volume) of the cell in pg$/\mu$m^3.
- **MaxPhase:** Maximum phase value found in the cell area, in rad.
- **MinPhase:** Minimum phase value found in the cell area, in rad.
- **PhaseVariance:** Variance of the phase values found in the cell area, in rad^2.
- **PhaseSkewness:** Skewness of the phase values found in the cell area. Unit-less.
- **PhaseSTDLocalMean:** Mean of the local standard deviation phase values found in the cell area, in rad.
- **PhaseSTDLocalVariance:** Variance of the local standard deviation phase values found in the cell area, in rad^2.
- **PhaseSTDLocalSkewness:** Skewness of the local standard deviation phase values found in the cell area. Unit-less.
- **PhaseSTDLocalKurtosis:** Kurtosis of the local standard deviation phase values found in the cell area. Unit-less.
- **PhaseSTDLocalMin:** Minimum local standard deviation phase value found in the cell area, in rad.
- **PhaseSTDLocalMax:** Maximum local standard deviation phase value found in the cell area, in rad.
- **AmplitudeSTDLocalMean:** Mean of the local standard deviation amplitude values found in the cell area. Unit-less.
- **AmplitudeSTDLocalVariance:** Variance of the local standard deviation amplitude values found in the cell area. Unit-less.
- **AmplitudeSTDLocalSkewness:** Skewness of the local standard deviation amplitude values found in the cell area. Unit-less.

- **AmplitudeSTDLocalKurtosis:** Kurtosis of the local standard deviation amplitude values found in the cell area. Unit-less.
- **AmplitudeSTDLocalMin:** Minimum local standard deviation amplitude value found in the cell area. Unit-less.
- **AmplitudeSTDLocalMax:** Maximum local standard deviation amplitude value found in the cell area. Unit-less.

For the exact formulas for each of these features, please refer to the supplementary material of [9, 10, 11]. The following parameters were used in the calculation of the aforementioned features:

- **Area to pixel ratio:** 0.08 μm$^2/$pixel. Used to calculate all features that required length or area dimensions.
- **Center wavelength of incident light:** 530 nm. Used to calculate the dry mass density.
- **Specific refractive increment:** 0.2 mL/g. Used to calculate the dry mass density.
- **Kernel size:** 4 pixels, equivalent to approximately 1 μm. Used to calculate the local standard deviation features.

5 Classification

Classifying the cells into predefined classes is one of the most fundamental tasks of the machine learning pipeline. In our project, two different approaches are explored, namely, one-step and three-step classification based on the extracted features from the images. Both methods are trained and evaluated on training and test sets generated from the "prediction.seg" file. This file contains 111 phase and amplitude images with corresponding masks and labels.

5.1 One-step Classifier

For the one-step classification, our methodology involves creating balanced training and test sets using the phase images and the masks. To ensure that the training and test sets represent each class, the class proportions are maintained after the data split, guaranteeing balanced datasets.

In order to improve robustness and model generalization, data augmentation techniques are applied to the images. Some of them include rotating, flipping, and zooming in on the image. Specific images have missing parts, represented by white pixels. To enhance the data quality, those white pixel values are replaced with the minimum value of the respective image, making the images more consistent with the darker background and avoiding numerical instabilities during feature extraction.

All the features extracted are represented by numerical values. In order to bring the numerical features to a consistent range, a min-max scaler is applied. This practice ensures that no single feature dominates the classification process due to its magnitude.

Four different classifiers are evaluated: k-Nearest Neighbors (KNN), Support Vector Classifier (SVC), Naive Bayes, and Random Forest Classifier (RFC). These classifiers are chosen due to their widespread use and effectiveness in various classification tasks. In order to determine the best-performing classifiers for each of the four methods, a grid search is employed. To assess the performance of the one-step classifier, confusion matrices for each classifier are generated. Figure

1 shows the confusion matrices for the SVC and KNN classifiers. For testing, 20% of the available labeled data are used. The results are satisfactory. Since SVC, KNN and RFC display similar performance, we introduce only two of them in the final version of the software, namely the SVC and the KNN, to avoid overloading the user interface.

Applications of Naive Bayes indicate that the method may not generalize well when the features are not independent [12], which might be the case with our extracted features. Hence, the Naive Bayes approach is excluded from the classification options.

The described procedure for training the one-step classifiers and generating the confusion matrices can be found in the `main_extended` branch, in folder `notebooks` under `one_step_classification.ipynb`.

5.2 Three-step Classifier

This classification procedure was born out of the realization that classifying a cell image as in-focus vs out-of-focus, as a single-cell vs an aggregation of cells, and as a RBC, WBC or PLT, were three very different problems. Additionally, both OOF and AGG classes had a small amount of training examples, meaning that class imbalance had to be dealt with somehow. As a solution to both of these concerns, three-step classification was proposed: two binary classifiers, one for OOF vs non-OOF and the other for AGG vs non-AGG, and a third multi-class classifier, for classifying between RBC, WBC and PLT (the cell type classes). The advantage of having binary classifiers for the imbalanced classes, is that we can easily control the precision and recall that we want for these, independently of the cell type classification. For example, we can find the probability threshold for our binary classifiers that maximizes either the f1-score, the f0.5-score or the area under the precision-recall curve (PR-AUC).

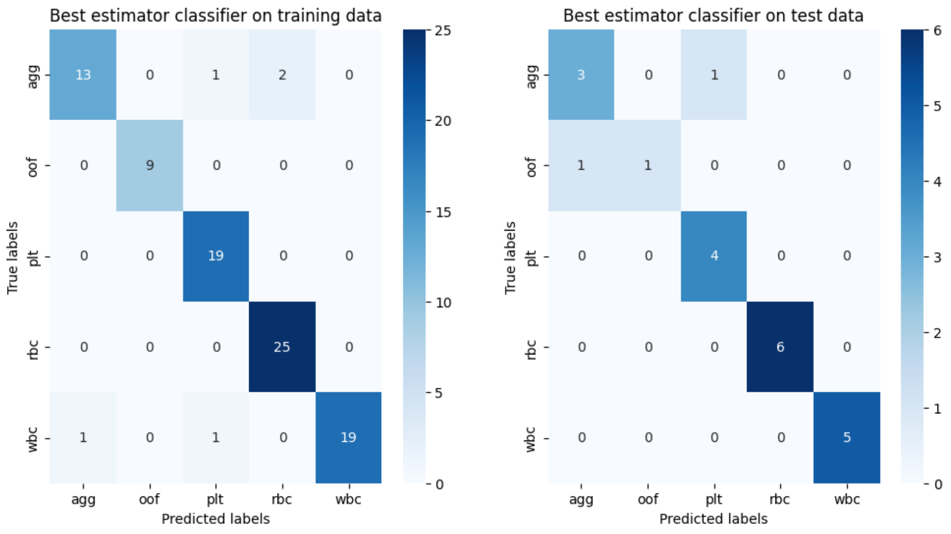
5.2.1 Model Selection

For the model and hyperparameter selection, we performed a grid-search with repeated stratified k-fold cross-validation[2] with 75% of the available labeled data, while the rest was used as test set. We evaluated two classification methods for each classifier: XGBoost [13] and logistic regression. For the three classifiers, we evaluated selecting only the first k features that maximize the ANOVA F-value[3]. For the binary classifiers, we evaluated the addition of SMOTE oversampling and random undersampling, both provided by the imblearn [14] Python library.

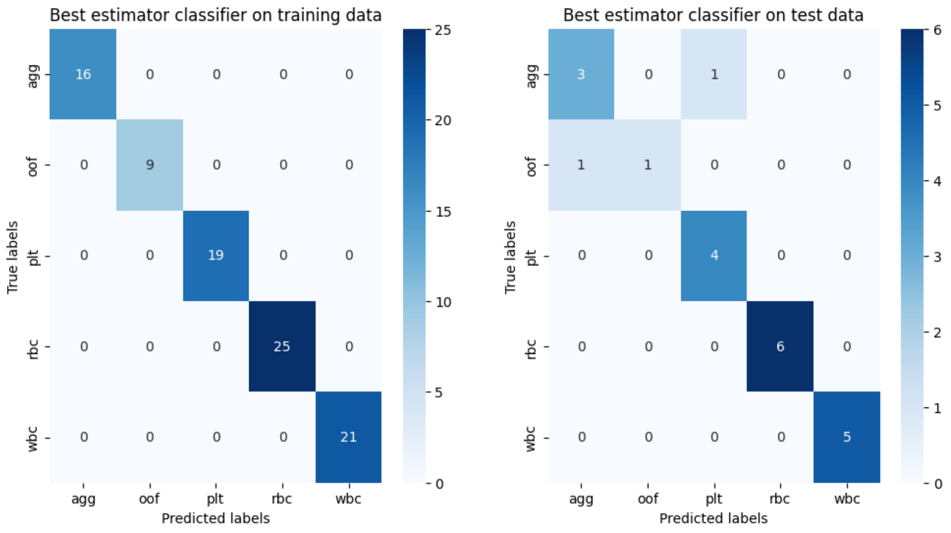
- **OOF Classifier:** For OOF classification, precision is more important than recall, since it is worse to have false positives than false negatives (it is worse that an in-focus cell is labeled as OOF than a OOF cell to be labeled in-focus). For each classification method, two grid-searches were performed, one maximizing the f0.5-score and another maximizing the PR-AUC. The entire training data was divided into OOF and non-OOF. We chose logistic regression over XGBoost, as the former had better precision performance on the test set. Finally, we chose the logistic regression model optimized over the PR-AUC metric, as it could maintain the same precision performance on the test set as the model optimized over the

[2]`https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedStratifiedKFold.html`

[3]`https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html`



(a) SVC



(b) KNN

Figure 1: Confusion matrices of the different classifiers for the training and test sets

f0.5-score metric, but with better PR-AUC score. We chose the probability threshold as the threshold that maximizes the f1-score.

- **AGG Classifier:** Again, for AGG classification, precision is more important than recall, since it is worse to have false positives than false negatives. For each classification method, two grid-searches were performed, one maximizing the f0.5-score and another maximizing the PR-AUC. The entire training data was divided into AGG and non-AGG. We chose XGBoost optimized over the PR-AUC metric, as it had the better precision performance than the other options, on the test set. We chose the probability threshold as the threshold that maximizes the f1-score.
- **Cell Type Classifier:** For cell type classification, the classes are balanced and precision is as important as recall. Because of this, we do only one grid-search for each classification method, optimizing over the f1-score metric in the *macro*[4] setting, meaning that we calculate the f1-score for each class, and find their unweighted mean. Note that in this case, the training data is filtered such that it only contains RBC, WBC and PLT training examples (we exclude AGG and OOF examples). We chose logistic regression as it has slightly better accuracy performance.

The models used for inference in our website are the models taken straight out of the grid-search procedure. The code used in this step can be found in the `main_extended` branch, folder `notebooks`, under `tsp_optimal_threshold.ipynb`, `tsp_gridsearch.ipynb` and `gridsearch_utils.py`.

5.2.2 Training

In the training step, each classifier is trained independently. For the OOF classifier, the training set will be divided into OOF and non-OOF; for the AGG classifier, the training set will be divided into AGG and non-AGG; and for the cell type classifier, the training set will be filtered such that it only contains RBC, WBC, and PLT training examples. At the end of the training step, the optimal probability threshold must be selected for the binary classifiers. This is done by running repeated stratified k-fold cross-validation using the test set and calculating the threshold that maximizes the f1-score on each run. The final optimal threshold is set to be the average of the thresholds found on the cross-validation runs.

5.2.3 Inference

For inference, the models will be applied sequentially. First, the OOF classifier is applied. All the examples labeled as OOF by the OOF classifier are considered as part of the OOF class. The examples labeled as non-OOF by the OOF classifier go to the next step: they are fed to the AGG classifier. After this, all the examples labeled as AGG by the AGG classifier are considered as part of the AGG class, while the examples labeled as non-AGG go to the next step: they are fed to the cell type classifier. Once the cell type classifier has labeled the remaining examples, the inference step is done and all examples are labeled. If instead of labels, we ask for class probabilities, we

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`

use the following formulas:

$$\begin{aligned}
P(\text{OOF}) &= P_{oof}(\text{OOF}) & (1)\\
P(\text{AGG}) &= P_{oof}(\text{in-focus})P_{agg}(\text{AGG}) & (2)\\
P(\text{RBC}) &= P_{oof}(\text{in-focus})P_{agg}(\text{single-cell})P_{cell}(\text{RBC}) & (3)\\
P(\text{WBC}) &= P_{oof}(\text{in-focus})P_{agg}(\text{single-cell})P_{cell}(\text{WBC}) & (4)\\
P(\text{PLT}) &= P_{oof}(\text{in-focus})P_{agg}(\text{single-cell})P_{cell}(\text{PLT}), & (5)
\end{aligned}$$

where $P(\cdot)$ represents the total three-step classifier class probability, while $P_{oof}(\cdot), P_{agg}(\cdot), P_{cell}(\cdot)$ represent the class probabilities given by the OOF classifier, the AGG classifier and the cell type classifier respectively. Note that to get these formulas, we assume that each classifier is independent of each other.

6 Active Learning

After the classification and the correction of the generated labels are completed, the user can save the corresponding features, masks, and labels, facilitating data collection for future improvements. Once a sufficient amount of new data is collected by performing additional classifications, the retraining of the classification model is advised. This retraining is implemented by using the data already saved in the backend, combined with the newly acquired data from the current session. The active learning procedure can be triggered when new data are generated by clicking the *Retrain Classification Model* button. The classification model that is retrained is the one currently selected in the user interface. This allows for continuous enhancement of the training data without having to label an enormous amount of images manually before ever being able to use the application.

For the user's convenience, we highlight in the frontend the cells which the classification model has predicted with low confidence. We do this by rendering the points around the mask in color purple. The confidence score we use is the prediction entropy, given by:

$$\text{Entropy} = -\sum_{\ell \in \mathcal{L}} P(\ell) \log_2(P(\ell)), \tag{6}$$

where $\mathcal{L}$ is our set of classes and $P(\cdot)$ is the class probability of the classifier. A uniformly random prediction would have the maximum entropy possible: $\log_2(5)$ bits ≈ 2.32 bits. The less entropy, the more confident our model is in its prediction. We have set an entropy threshold of 1.3 bits, such that all the cells whose prediction entropy is larger than this threshold, will be highlighted.

7 Docker and deployment

This section will focus on deploying our web-based application on a Kubernetes cluster. The application is composed of a frontend and a backend, both of which are containerized. The frontend is accessible on port 3000 and serves user interfaces, while the backend is accessible on port 8000 and processes user requests and provides the necessary data for the frontend.

The application architecture is as follows:

Frontend : The frontend is a single-page application (SPA) that is built with React.js. It is containerized using Docker and is deployed to a Kubernetes pod.

Backend : The backend is a RESTful API that is built with FastAPI using Python, which is also containerized and deployed together with the backend.

7.1 Containerization with Docker

We used Docker to containerize the application to ensure environment consistency. The Dockerfile defines the environment for running the application, including the base OS, dependencies, and the application itself. The Docker Compose file orchestrates the application services. It details how to build and/or pull the application Docker image, maps network ports for frontend (3000) and backend (8000) services, specifies environment variables, and mounts volumes from a remote file share server, among other configurations.

7.2 Continuous Integration and Deployment (CI/CD)

We use GitLab for CI/CD, which provides an efficient way to automate building and deploying our Docker images. The GitLab pipeline is divided into two stages: build and deploy. In the build stage, Kaniko builds Docker images directly within the Gitlab Runner environment and pushes them to our registry. The deploy stage uses Flux to reconcile the cluster's state with the latest changes in the Git repository. In this way, when a developer pushes to main, the image is directly built, and the application is automatically deployed right after, removing the need for manual deployment via the command line.

7.3 Kubernetes Deployment

We deploy our application on the provided Kubernetes cluster, which allows us to manage, scale, and maintain the application efficiently. The deployment manifests define the desired state for our application in the cluster, such as the Docker image to use, the commands to run, and the ports to expose. For instance, the frontend service listens on port 3000, and the backend service on port 8000. Our application also relies on persistent storage provided by PersistentVolumeClaims. The configuration for these resources, as well as their mounting points in our containers, is also defined in the deployment manifests.

7.4 Security and Best Practices

We adhered to various security practices and Kubernetes best practices throughout the project. ImagePullSecrets were utilized to pull from our private Docker registry securely. Defining resource requests and limits in our deployments ensured the efficient use of cluster resources. Also, we use Traefik[5] as a reverse proxy responsible for routing our API and frontend domains to the specific port. More details about the reverse proxy can be found in Figure 2.

[5] `https://traefik.io/traefik/`

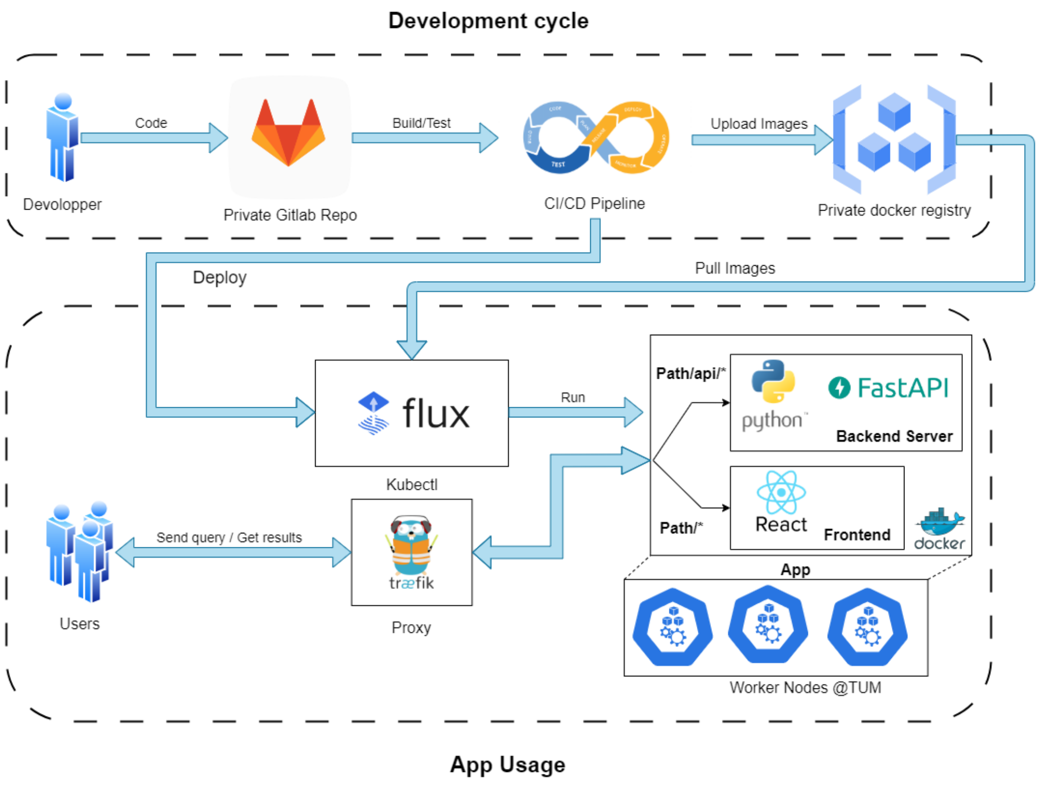


Figure 2: App Deployment Pipeline

7.5 Challenges and Solutions

Managing and understanding the interaction between various components posed a significant challenge. Learning and adapting to Kubernetes, Docker, and GitLab's best practices was another challenge, which improved our deployment process once overcome. We understand that our current deployment pipeline is probably not the best one, but we agree that we are happy with the current results since we were able to achieve automatic deployment when the code is pushed to the main branch. In the future, one could think about having different deployments as environments, such as development and production.

8 User interface

In order to interact with the app, we implemented a frontend using HTML + CSS + Javascript. Namely, we used the React framework in order to build the interface. The main goal was to make a simplistic and minimalist website where the user does not feel overwhelmed while scrolling through many pages. Thus, we decided to go for a single web page interface that is displayed in Figure 3. With this interface, the user can:

- Load a dataset of cell images. For this, it is only necessary that the user adds the dataset file to the specified data share.
- Navigate between different images.
- Segment and classify images.
- Edit the segmentation masks and relabel images.
- Perform active learning for model retraining.

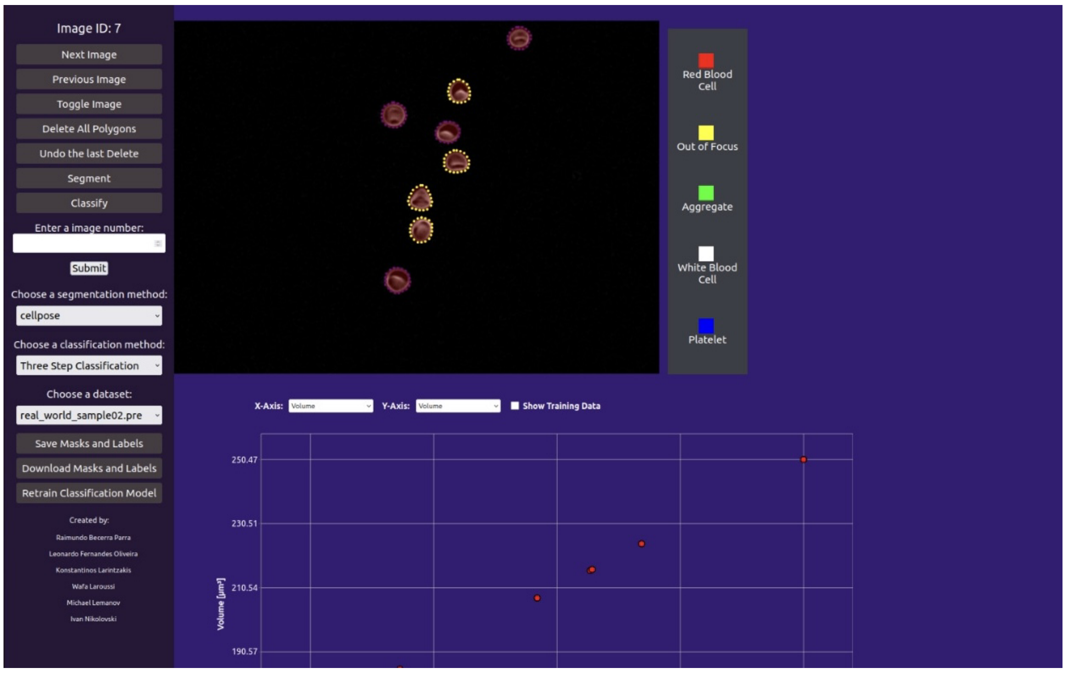


Image ID: 7

Choose a segmentation method: cellpose

Choose a classification method: Three Step Classification

Choose a dataset: real_world_sample02.pre

X-Axis: Volume Y-Axis: Volume

Figure 3: CellEstimator Application

8.1 Image Selection and Navigation

In order to achieve the above-stated goals, we created a menu bar on the left side of the interface. The menu contains selector drop-down elements to select the dataset that should be displayed and the segmentation and classification method that can be selected from the implemented options. The dataset options consist of all the datasets that can be found in a specified data share. Once a dataset is selected, the first image in that dataset will be displayed on the main part of the interface. The user can then navigate between the images of the dataset using the buttons *Next Image* and *Previous Image* or simply specify the image number of an image in the dataset. Furthermore, it is possible to toggle between the amplitude and phase image of the dataset as performing segmentation and classification on both delivers different results. That way, the user can freely choose and experiment which results are more suitable for the given use case.

8.2 Segmentation and Mask Handling

Once an image of interest is selected by the user, the next step is to segment the image with the *Segment* button. After segmentation, the masks of the segmented cells can be found in the image. Here the masks can be evaluated by the user, and if necessary, every mask can be modified or new ones added. The options for modification include moving the entire mask by dragging the mask to the new location, modifying the mask by dragging any of the yellow dots on the edge of the mask to a new location, or simply deleting one or even all masks. To delete a single mask, the user right-clicks on the mask and selects *Delete mask* in the newly opened menu on the right. Deleting all polygons is done by using the *Delete All Polygons* button in the menu on the left. Every delete can be undone in reverse order by the *Undo the last Delete* button. After deleting one or all masks or discovering that the segmentation process missed a cell, the user can draw new masks by setting points of a polygon with a click in the image and pressing escape once the polygon sufficiently covers the cell. The number of cells or masks found can be seen in the top left corner of the app.

8.3 Classification

Once the segmentation step is completed, the user can click on the *Classify* button in the menu on the left, which classifies the cells using the selected classifier. The masks are now color-coded to represent the classification result. A legend for the colors can be found to the right of the image. The yellow dots indicating the edge of the mask can be found on every mask but one, where the dots have a purple color. This is an indication to have a closer look at this cell as the classification yielded the lowest confidence score. The masks can be modified in the same way as during the segmentation step. One modification option that was not yet mentioned as it is more relevant for or after the classification is the option to change the color of the mask and, therefore, change the classification result. It is done by right-clicking a cell and then selecting the new class by clicking on the corresponding button on the right. That way, the user can correct classification results.

8.4 Saving Results and Retraining

If the user is satisfied with the results of the image, he can click on the *Save Masks and Labels* button to save the current masks and corresponding labels. This can be done to any number of classification results, and after saving masks and labels, the saved results can be used for active learning. By clicking on the *Retrain Classification Model* button, the model gets retrained using the saved masks and labels with the purpose to produce better classification results. The *Download Masks and Labels* button gives the user the option to download the saved masks and corresponding labels to save them locally or process them otherwise to their liking. This feature outputs a .pre file containing the dataset-ID, the image-ID, and the masks and labels for the image.

8.5 Visualization of Classification Results

This concludes the overall workflow of the application, but there is still one implemented feature missing, which is a plot to understand the classification results better. After classifying, a plot pops up underneath the image. Above the plot, two features from the above-mentioned list of features can be selected to visualize a 2D feature space. With the checkbox *Show Training Data*, it can be selected if the feature space should be visualized, including the training data or only the data of

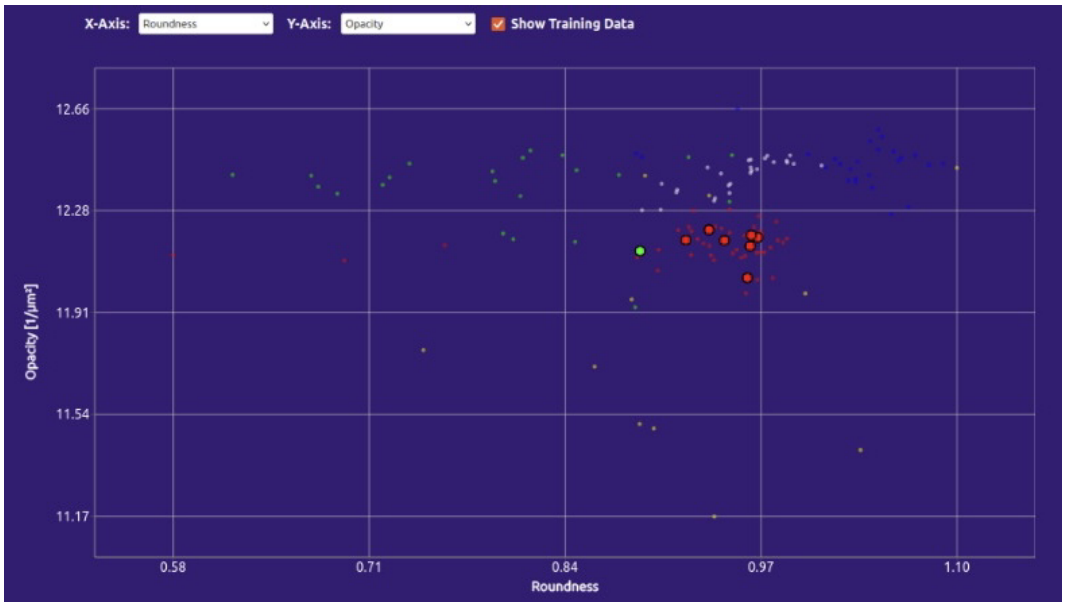


Figure 4: Visualization of the feature space including the training data

the current image classification. By hovering over the points that represent the cells of the current image, the corresponding mask is highlighted in the image to identify what point represents what mask. The aim is to help the user understand how the classification was performed on the training data, what features are dominant in a given class, and how this translates to the labeling of the current image. In Figure 4, it is demonstrated how the bigger dots (representing the features of the current image's cells) are plotted alongside the smaller dots (representing the features of the training examples). By doing so, the user can easily examine where the current image's cells lie in the feature space, and by comparing them to the training examples, they can judge whether the classification is fitting based on the selected features. In this example, we can see how the bigger red dots can be found in the cluster of smaller red dots, which is an indication of why they were classified as red blood cells. The bigger green dot, on the other hand, has no green cluster in proximity to it, which could be an indication that the cell was missclassified.

9 Future Work

Future work on the developed software should be concentrated on improving the user interface and extending the capabilities of the segmentation tools. More specifically, further steps should include:

- **Feedback allocation from the medical professionals** who will use the software. More accurate and meaningful enhancements can be implemented by better understanding their needs and preferences. Evaluation would reveal weaknesses in the user experience, which could then be addressed.
- **Using Cellpose's 3D segmentation capabilities** by feeding it both phase and amplitude

images. This approach is expected to yield better segmentation results, providing more accurate cell identification.

- **Retraining the segmentation methods** to further improve the accuracy of cell detection and minimize the need for manual segmentation. Both Cellpose and FastSAM allow for human-in-the-loop retraining of their models.
- **Implementation of smart annotation techniques** that allow the user to click on "missed" cells and automatically segment them without manually drawing the contour lines.
- **Incorporation of AI tools** that support automatic text generation. The user can make more informative decisions by outputting a detailed explanation of the classification results.

10 Comments to the Group Work Experience

The group work experience throughout the project was highly efficient and cohesive. From the beginning, we conducted regular meetings that helped us communicate our ideas, discuss our progress and issues and establish a productive workflow to remain constantly on track. Vital was also the mutual respect and the strong sense of responsibility displayed by all the members when it came to the individual and group tasks. Furthermore, each member is characterized by a unique and complementary set of skills and the willingness to help others, even outside of their designated assignments. This factor allowed us to tackle challenges together and establish a friendly and collaborative environment. As a result, we functioned as a team, making the work both productive and enjoyable.

References

[1] Mayo Clinic Staff. Complete blood count (CBC). URL https://www.mayoclinic.org/tests-procedures/complete-blood-count/about/pac-20384919. Accessed: 23.07.2023.

[2] Ayalew Tefferi, Curtis A. Hanson, and David J. Inwards. How to interpret and pursue an abnormal complete blood cell count in adults. *Mayo Clinic Proceedings*, 80(7):923–936, 2005. ISSN 0025-6196. doi: https://doi.org/10.4065/80.7.923. URL https://www.sciencedirect.com/science/article/pii/S0025619611615681.

[3] Blood Specimens - Staining. URL https://www.cdc.gov/dpdx/diagnosticprocedures/blood/staining.html. Accessed: 23.07.2023.

[4] CellFace. URL https://wiki.tum.de/display/ldv/CellFace. Accessed: 23.07.2023.

[5] Carsen Stringer, Tim Wang, Michalis Michaelos, and Marius Pachitariu. Cellpose: a generalist algorithm for cellular segmentation. *Nature Methods*, 18(1):100–106, January 2021. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-020-01018-x. URL https://www.nature.com/articles/s41592-020-01018-x.

[6] Marius Pachitariu and Carsen Stringer. Cellpose 2.0: how to train your own model. *Nature Methods*, 19(12):1634–1641, December 2022. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-022-01663-4. URL https://www.nature.com/articles/s41592-022-01663-4.

[7] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast Segment Anything, June 2023. URL http://arxiv.org/abs/2306.12156. arXiv:2306.12156 [cs].

[8] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment Anything, April 2023. URL http://arxiv.org/abs/2304.02643. arXiv:2304.02643 [cs].

[9] Kelvin C.M. Lee, Maolin Wang, Kathryn S.E. Cheah, Godfrey C.F. Chan, Hayden K.H. So, Kenneth K.Y. Wong, and Kevin K. Tsia. Quantitative Phase Imaging Flow Cytometry for Ultra-Large-Scale Single-Cell Biophysical Phenotyping. *Cytometry Part A*, 95(5):510–520, May 2019. ISSN 1552-4922, 1552-4930. doi: 10.1002/cyto.a.23765. URL https://onlinelibrary.wiley.com/doi/10.1002/cyto.a.23765.

[10] Kelvin C. M. Lee, Andy K. S. Lau, Anson H. L. Tang, Maolin Wang, Aaron T. Y. Mok, Bob M. F. Chung, Wenwei Yan, Ho C. Shum, Kathryn S. E. Cheah, Godfrey C. F. Chan, Hayden K. H. So, Kenneth K. Y. Wong, and Kevin K. Tsia. Multi-ATOM: Ultrahigh-throughput single-cell quantitative phase imaging with subcellular resolution. *Journal of Biophotonics*, 12(7), July 2019. ISSN 1864-063X, 1864-0648. doi: 10.1002/jbio.201800479. URL https://onlinelibrary.wiley.com/doi/10.1002/jbio.201800479.

[11] Dickson M. D. Siu, Kelvin C. M. Lee, Michelle C. K. Lo, Shobana V. Stassen, Maolin Wang, Iris Z. Q. Zhang, Hayden K. H. So, Godfrey C. F. Chan, Kathryn S. E. Cheah, Kenneth K. Y. Wong, Michael K. Y. Hsin, James C. M. Ho, and Kevin K. Tsia. Deep-learning-assisted biophysical imaging cytometry at massive throughput delineates cell population heterogeneity. *Lab on a Chip*, 20(20):3696–3708, 2020. ISSN 1473-0197, 1473-0189. doi: 10.1039/D0LC00542H. URL http://xlink.rsc.org/?DOI=D0LC00542H.

[12] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[13] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, San Francisco California USA, August 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL `https://dl.acm.org/doi/10.1145/2939672.2939785`.

[14] Guillaume Lemaitre, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning, September 2016. URL `http://arxiv.org/abs/1609.06570`. arXiv:1609.06570 [cs].