User manual for running MicroICS analysis pipeline

The MicroICS pipeline provides advanced tools for 3D biofilm image analysis, automatically extracting features and enabling comparison of machine learning classifiers to link structural features to bacterial traits through predictions and feature analysis (such as strain differentiation). It supports detailed analysis of key discriminative features and allows prediction of selected biological traits in unknown images using a trained model, with tools for visual interpretation of results.

Before using this pipeline, users must ensure that the specified image formats, naming conventions and computer system requirements are met. Comprehensive guidance is provided to support users, including those working in the biology or related fields.

The chapters are arranged in sequence to guide users through the tool from initial setup to advanced use:

1. Image requirements

As input, we use 3D images (z-stacks) of biofilms consisting of several image layers. They must be **saved in tif format**, other formats are currently not compatible with the pipeline.

In case of Zeiss LSM10 microscope high-throughput imaging, the images can be acquired at different positions of the microtiter plate in such a way that the images are saved in one file. The images for further analysis must be separated into separate tiffs (so that one image represents only one position, which includes all layers of a 3D stack). This can be done using open source software ImageJ (https://imagej.net/software/fiji/) with a provided ImageJ script for splitting and converting images from lsm files (Zeiss microscopes) (Figure 1).

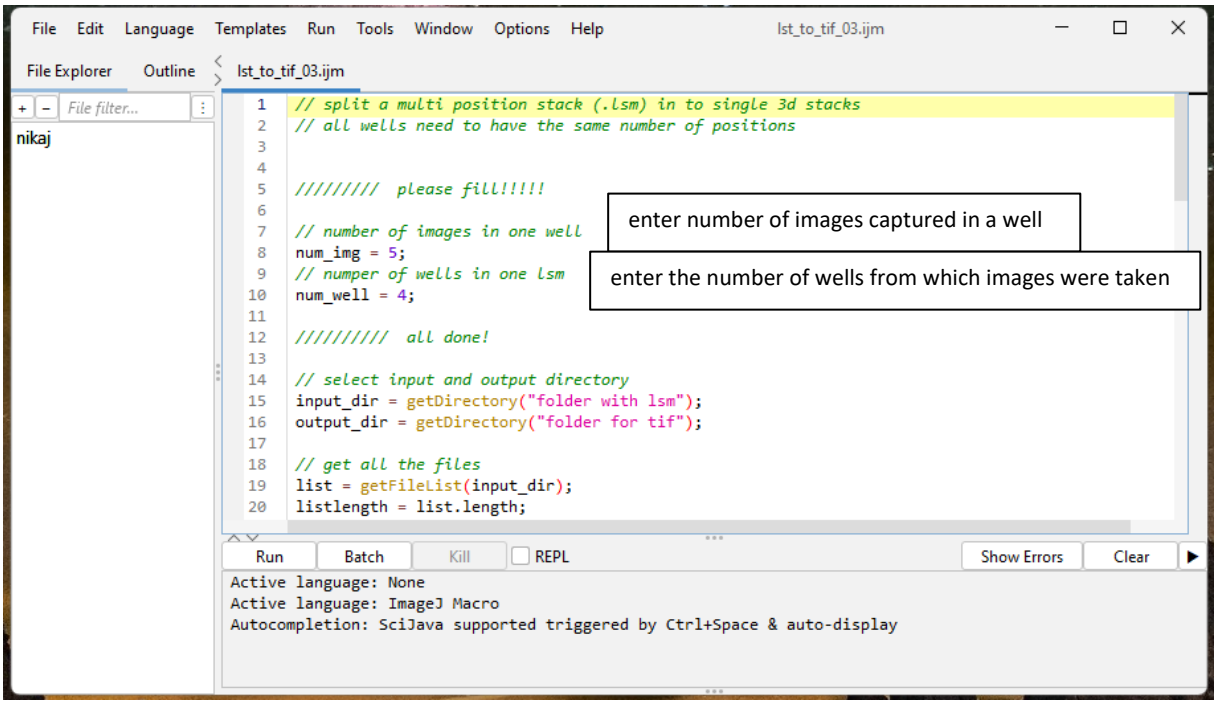


Figure 1. A screenshot of the script “lsm_to_tif,” available at https://imagine.ijs.si, is shown. The script enables the conversion of a lsm file with several 3D stacks taken at different positions in microtiter plate into a single 3D stack per position (one-position--one-set of images – one 3D stack) in tif format. The script is loaded by dragging it onto the ImageJ; once it opens, the number of images per well and the number of wells in which images are captured must be adjusted. This script also relies on the image naming shown below. If you do not follow these, the script will not work.

The execution of the MicroICS pipeline **depends heavily on the information contained in the image names**, which should follow the example below.

Scheme for image naming:
<date>_s_<species>_st_<strain>_p_<well_coordinates>_pos<well_position>_tm_<growth_time>_ch_<channel>_z_<layer_distance>

Example of an image name:
07112023_s_Lm_st_L1823_p_F02_pos005_tm_24_ch_Syto9_z_21

Explanation of the name:

- 07112023: Date currently included in DayMonthYear to make it easier to track results
- s_Lm: s_ indicates the species, Lm means *Listeria monocytogenes*
- st_L1823: st_ denotes the strain, L1823 is the name of the strain (strain identity)
- p_F02: p_ denotes the well coordinates on the 96-well plate, F02 is the actual position on the 96-well plate
- pos005: pos denotes the measured position within the individual well, e.g. position 5 in well F02
- tm_24: tm_ stands for the measurement time, 24 means that the images were taken after 24 hours of biofilm growth
- ch_Syto9: ch_ denotes the channel, Syto9 is the name of the dye and filter used
- z_21: z_ denotes the number of layers in the 3D image, which also reflects the distance between two consecutive layers.

The most important parts of the naming structure are: Date, _s_, _st_, _p_, _pos. The names are parsed on the back-end according to these patterns. Any deviation from the exact patterns listed will result in failed or incorrect processing of the data.

The renaming process can be carried out semi-automatically by the software, e.g. https://www.advancedrenamer.com/, or by user-defined scripts. Examples and the user manual are available here https://www.advancedrenamer.com/user_guide/v3/complete_guide together with graphical representations.

2. PC requirements

The pipeline is designed to run on both Windows and Unix-based systems, or more generally on any system that supports Docker. Hardware requirements include at least two computing cores (CPUs) and 4 gigabytes of ram. The files required to run the pipeline are approximately 500 MB in size, and 1 GB of memory may be required to run the analysis smoothly.
Level required:

User level of windows environment navigation. Acquaintance with command line in Windows, basics available here https://www.geeksforgeeks.org/most-useful-cmd-commands-in-windows/ .

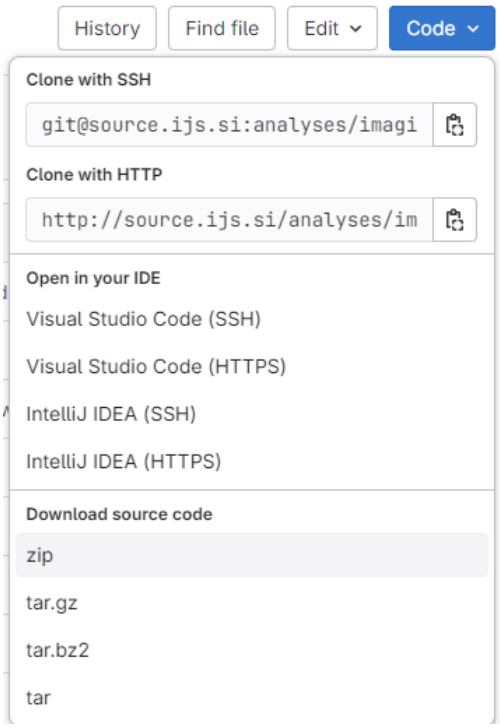
Acquaintance with Docker environment, basics available here https://www.docker.com/blog/getting-started-with-docker-desktop/.

2.1. Setup procedure for Windows systems

2.1.1. Install Docker

1. Download and install the Docker Desktop application from the website. This might take a minute, depending on your system and will require you to restart your computer: https://docs.docker.com/desktop/install/windows-install/
2. Start Docker Desktop by double-clicking on the Docker icon in the Start menu. It might run in the background and you might need to bring it to the front from the tray (bottom right corner).
3. Download "biofilm-classification-main.zip" from the website https://imagine.ijs.si or github:

a. Go to "code"
b. Click on “zip” option in the "download source code" section of the dropdown.
c. Save the zip file to Desktop.



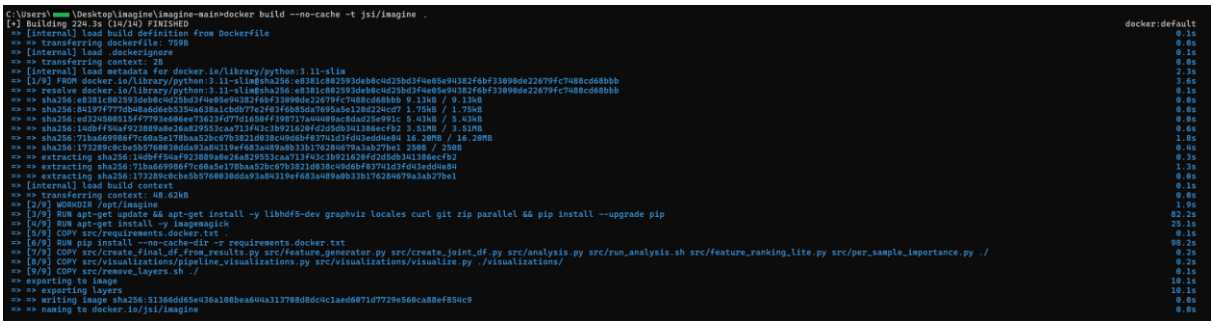
4. Extract “biofilm-classification-main.zip" into the biofilm-classification-main (you can rename it if you want) folder on your desktop.

2.2.2. Building the Docker image

The next step is to build the Docker image.

1. Navigate to the folder where you extracted the zip file, right-click, and select "Open in Terminal" to start the terminal in the "biofilm-classification-main" folder. Link to “open in terminal” can be hidden also in “show more options”. If these options are not available on your system, we suggest extracting the downloaded file onto C:/, open cmd and navigate to the folder using command cd e.g. cd C:/path/to/my/folder. Cmd can be found by typing “cmd” in Windows search and is run by clicking on “Command Prompt”.
2. Connect to the internet prior to running the build command.
3. Write or copy the following command into the terminal to build a Docker image specifically created for running these analyses (the -no-cache parameters ensures a clean build from scratch):

   docker compose build --no-cache



Once the Docker image is built (it takes a while), you can run analyses as described below.

The Docker image will not need to be rebuilt unless changes are made to the scripts prior to the next use of Docker.

You can close the Docker desktop application but leave it running in the background (you will see an icon in the taskbar indicating that the Docker daemon is still running and ready to run containers). You can close the terminal window if you do not intend to continue or leave it open to run the MicroICS pipeline.

3. Preparation of data for running the MicroICS pipeline

Before executing the command to calculate features you need to:

1. Prepare a folder with your input images, e.g. create a folder example_images on the desktop, and remember the path or address of the folder, e.g. /c/Users/my_name/Desktop/example_images).

2. Create a folder in which you want to save the calculation results. For example, create the subfolder results_example on the desktop and make a note of the path or address, e.g. /c/Users/my_name/Desktop/results_example.

(optional) Change the voxel dimensions if necessary: To do this, go to biofilm-classification-main/src, find the feature generator script and change the following part as desired:
constants voxel_size = 0.13 * 0.13 * 0.5 pixel_area = 0.13 ** 2

The voxel size depends on the pixel size in the x, y and z directions, which is defined on the microscope before the images are captured. This information can be retrieved in the original file format or by loading the image into ImageJ, going to “Image” and selecting “Properties”.

If you make changes to the feature generator code, you must recreate the Docker image as described above in point 2.2.2.

3. Open the folder "biofilm-classification-main" (or the name you chose when unzipping the zip file). Search for the file with the name “.env”.

   You will need to replace the path to the images in the .env file:
   "./training_images" with the path or address where your images are located. You can check this in the address bar of the window when the folder with the images is open. This address

needs to be changed slightly to be compatible with Docker by following the rules below: change "\" to "/" and remove ":" as in the following example:

c:\my_folder\test_images, you need to write it as /c/my_folder/test_images.

You must also replace "./results" with the corresponding path if you have created the subfolder in a different location than described above. The address must be changed as described above.

Open the file and change the paths in the .env file as follows:

# change these paths so that they point to your data and desired results folder
IMAGINE_IMAGES=./training_images
IMAGINE_RESULTS=./results

To this one:

IMAGINE_IMAGES=/c/Users/my_name/Desktop/example_images
IMAGINE_RESULTS=/c/Users/my_name/Desktop/results_example

Alternatively, you can also create your own file with the information about the paths. Save this file in the “biofilm-classification-main” folder. If you run generate_features or learning_benchmark, you must specify the name of this file in the command (see below).

4. Generating features using the MicroICS pipeline

To perform the tasks, you must navigate to the "biofilm-classification-main" folder (or the name you chose when unpacking the zip file) and open the terminal in this folder as described above (right-click, then "open in terminal"). The Docker daemon must also be running (you should see an icon in the taskbar). If this is not the case, open the Docker Desktop application as described above.

To generate features run the following commands in the terminal:

docker compose run --rm imagine 1 datafile.tsv 10 generate_features

Alternatively, if you use your own file with paths:

docker compose --env-file test_paths.txt run --rm imagine 1 datafile.tsv 10 generate_features

The parameter “1” determines the parallelization rate — a higher parallelization (should not exceed the number of CPU cores in your computer) works faster, but consumes more resources of the computer, while a lower parallelization (1) works slower, but should also be completed on machines with fewer resources. This task may take quite long for large amounts of images.
The parameter “10” determines how many of the top features are used in the visualisation later in the process, but it is not currently used.

The main output of this step is **datafile.tsv**, where all calculated features are available for review.
Once the process is complete, check the results folder and in particular the datafile.
In the first column of the data file, all analyzed images should be listed by name. If some images are missing, check the format of the missing images.

An example of what kind of results you can expect can be found in the biofilm-classification-main/results_example folder.

The file datafile.tsv can be opened with Excel for checking. If the file becomes too large for Excel, selected rows of the table can be exported to csv and opened with Excel to check the calculations more intuitively:
Open the command prompt as described above.
Navigate to the folder with the results using the cd command as described above. Then use the command:
type datafile.tsv | Select -First 15 > first_15_rows_of_datafile.csv

How features are calculated is described in the last chapter of the manual and in the attached table. Further details can be found in the code itself. Many examples are also given in the manuscript on description of MicroICS based on the use case of *Listeria* biofilms.

The features from the datafile.tsv are now prepared for running the comparison of classification models.

5. Performance comparison of machine learning models

The command to put in the terminal for comparison of the classifiers (benchmark) is as follows:
docker compose run --rm imagine 1 datafile.tsv 10 learning_benchmark

The parameter “1” determines the parallelization rate — a higher parallelization (should not exceed the number of CPU cores in your computer) works faster, but consumes more resources of the computer, while a lower parallelization (1) works slower, but should also be completed on machines with fewer resources.The parameter “10” determines how many of the top features are used in the visualisation later in the process, but it is not currently used.

Under some circumstances (most notably low available memory - commodity laptops) some of the models (AutoGluon) can unfortunately crash the container, however, the software is designed in such a way to store intermediary results so most of the computation will not be wasted. If the software exits prematurely, you can try to rerun it and it should pick up where it left off. In case it stops again, try closing other applications that might be using up the available memory. In the case where a model will not complete even after several tries you can disable it using the instructions provided below. We also advise avoiding activation of TPOT and AutoGluon, as they may cause the container to crash and require rerunning from the beginning.

Disabling individual models:

The file that determines which models will be used in the benchmark is src/feature_ranking_lite.py. If you wish to disable some of the models locate the line 268;

starting from there the following lines should include a definition of which models are used, e.g., as such:

```
    if all_learners:
        models = {
            'dummy': DummyClassifier(),
            'decisiontree': tree.DecisionTreeClassifier(),
            'logistic': LogisticRegression(),
            'rf': tuned_rf,
            'xgb': XGBClassifier(n_estimators=100, max_depth=3, learning_rate=1, objective='binary:logistic'),
            'gridsearch': GridSearchCV(KNeighborsClassifier(), parameters, n_jobs=PARALLELISM),
            #'tpot': TPOTClassifier(generations=5, population_size=20, cv=5, random_state=42, verbosity=2, n_jobs=PARALLELISM, memory='auto'),
        }

        # Add TPOT only if available
        if TPOT_AVAILABLE:
            #models['tpot'] = TPOTClassifier(generations=5, population_size=20, cv=5, random_state=42, verbosity=2, n_jobs=PARALLELISM, memory='auto')
    else:
        # Default behavior: only RandomForest (fast)
        models = {
            'rf': tuned_rf,
        }

    # Add autogluon model only if available
    if AUTOGLUON_AVAILABLE:
        #models['autogluon'] = TabularPredictor(label="label")
```

Comment (put a # sign) in front of two parts of the code:

```
        # Add TPOT only if available
    #if TPOT_AVAILABLE:
        #models['tpot'] = TPOTClassifier(generations=5, population_size=20, cv=5,
random_state=42, verbosity=2, n_jobs=PARALLELISM, memory='auto')
```

and

```
        # Add autogluon model only if available
    #if AUTOGLUON_AVAILABLE:
        #models['autogluon'] = TabularPredictor(label="label")
```

After such modification in the selected models, the change should be rebuilt in the image with (this command builds the change into the existing image and not the entire image once again and takes less time, thus the -no-cache parameter **isn't used**):

docker compose build imagine

Enabling all active (uncommented) models:

By default, the pipeline only runs the random forest method as it presents a good tradeoff between learning time and model quality (predictive accuracy). If you want to run all enabled models add the **--all_learners** parameter at the end of the run command, for example:

docker compose run --rm imagine 1 datafile.tsv 10 learning_benchmark --all_learners

This parameter modifies the function of the learning_benchmark and learning_benchmark_save_models commands.

In some cases, especially when available memory is low (such as on commodity laptops), some models (not necessarily only autogluon) may cause the container to crash. This appears as a terminated task in the terminal, with no models saved in the output folder and only partial results available. However, the software is designed to store intermediate results, so most of the computation will not be lost.

For the classifier comparison, the set of given images is randomly divided by the software into a training set and a prediction set. The training set is used to train the model and the prediction

set is used to test how accurately the names of the strains can be predicted based on the trained model. These results show us how successfully the strains could be predicted.

The comparison results of the classifiers are distributed across multiple files and are presented either as a list or through visualisations in various plots:

- Classification_all
- Classification_no_counts_features
- Ablation_ranking_all
- Rankings_date
- Rankings_label

The label here corresponds to the string that follows _st_ in the name of the image, in this case strain name (strain identity). The classification target is the property predicted by the classifier, and in this case the target corresponds to the label, i.e. the name of the strain. The date refers to the exact date on which the experiment took place and is linked to the biological replicates. The date is also indicated in the name of the image (see naming convention).

Explanations of individual learning benchmark output files:

Classification_all and classification_no_counts_features
These two files summarize the results of classification by selected machine learning algorithms. The columns from left to right are:

- tag – has value RESULT and it has been used for parsing, but is currently not active
- model - type of classifier, random forest, etc. and dummy classifier as sanity test
- upsampling – has value 1
- n_components - number of input features

The classifiers can be fed with all features, currently there are 2700, or the number of features can be reduced beforehand by the SVD decomposition model, which selects the features it considers important and outputs the number of features the user selects - here we reduce the number of features by a factor of two; when comparing the accuracy for the reduced data, you should bear in mind that the features selected by SVD are not the same for all dimensions, which means that the model can be fed with different features and therefore produce different results.

- fold – number of iterations (repetitions)
- accuracy - number of correct predictions (key result of the comparison)
- test_set - name of the strains (labels) used for the classification test,
- thr_features – TRUE if features with threshold are included and FALSE if features with threshold are excluded; for each number of n_components results for TRUE and FALSE are given; for n_components 580 we have only FALSE (all features except the one with threshold) and for 2700 only TRUE (because all features include the one with threshold)

The following files contain information derived exclusively from classifications based on **random forest**.

Ablation_ranking
The report shows how model accuracy changes as features are added incrementally. Determining the highest accuracy for a dataset shows how many features are required to

classify the strains. For interpretation, we recommend selecting the number of features that gives the highest accuracy and then using the same number of features with the highest score to explain which ones contribute the most to the classification.
For example, in ablation_ranking we find that the number of features with the highest accuracy is 81; then we go to the ranking file and select 81 features with the highest score for further interpretation.

Rankings_label and rankings_date *(Here, label refers to the string that follows _st_ in the image name, which in this case is the strain name. In classification, we refer to the property being predicted as the target, so the target here is either the label (strain) or the date.)*
For each feature calculated, the strength of its relationship to the target is indicated, with larger values indicating that the feature was more important in predicting strain and vice versa (rankings_label). We also provide rankings for date prediction (rankings_date) to account for features that may vary as a consequence of repeating the experiment on different days (batch effect is likely due to the methodology or variability of the biological system under study). This is crucial when we provide multiple images from different days (biological replicates) for a single strain to the model, as the variation within a single day is often smaller than between different days. If the model relies on features describing biofilm characteristics that are tied to specific dates rather than strains across different dates, it may lead to bias and poorer results. We suggest that the features included in rankings_label and rankings_date are compared and that the high-ranking, overlapping features are not considered when explaining the complexity of the system.

Visualisations of the comparisons of the performance of classifiers and the ablation ranking

The results of the performance comparison between the selected classifiers (based on calculations from the classification_all and classification_no_counts_features) and the ablation ranking (ablation_ranking) are visualised to facilitate the interpretation of the results. These are found in the folder “Visualizations”.

6. Classification of 3D biofilm images using random forest to identify the strain present in the image

To predict the target, such as the identity of the strains from the new images (inference), we need to specify the path to the folder containing the images for which we want to obtain predictions, as well as the path to the folder where the software provides the prediction results. So open the folder "biofilm-classification-main" (or the name you chose when unzipping the zip file) and look for the file named “.env”.

You will need to replace the path to the images for inference in the .env file:
"./new_images_to_classify" with the path or address where your images are located. You can check this in the address bar of the window when the folder with the images is open. This address needs to be changed slightly to be compatible with Docker by following the rules below: change "\" to "/", remove ":" and add a "/" at the start of the address as in the following example:

c:\my_folder\inference_images, you need to write it as /c/my_folder/inference_images. Make sure that you make no other changes to the text after the ":".

You must also replace "./inference_results" with the corresponding path to the folder for the results of inference e.g. /c/Users/my_name/Desktop/inference_results. The address must be changed as described above.

Open the file and change the paths in the .env file as follows:

# change these paths so that they point to your data and desired results folder
IMAGINE_INFERENCE_INPUTS=./new_images_to_classify
IMAGINE_INFERENCE_OUTPUTS=./inference_results

To this one:

IMAGINE_INFERENCE_INPUTS=/c/my_folder/inference_images
IMAGINE_INFERENCE_OUTPUTS=/c/Users/my_name/Desktop/inference_results

Alternatively, you can also create your own file with the information about the paths. Save this file in the “biofilm-classification-main” folder.

For optimal inference performance with the developed random forest model, we recommend splitting control images from treatment experiments into two equal subsets: one to be included in the training set as data augmentation, and the other reserved exclusively for inference as a sanity check. This approach helps ensure robust model generalisation and accurate evaluation.

Important: Before running training tasks such as learning_benchmark_save_models, ensure that all images in the training set follow the precise naming convention described at the beginning of this manual. Any deviations or extra characters in filenames may cause parsing errors and mislead/interrupt the model training process.

Before you run the inference, you must run the command with which the learning_benchmark task saves the created models for later use. To do this, run:

docker compose run --rm imagine 1 datafile.tsv 10 learning_benchmark_save_models

To activate all active algorithms as described in 5., run:

docker compose run --rm imagine 1 datafile.tsv 10 learning_benchmark_save_models --all_learners

Alternatively, if you use your own file with paths:
docker compose --env-file test_paths.txt run --rm imagine 1 datafile.tsv 10 learning_benchmark_save_model

After this step, check the file “classification_all”, which contains information about the accuracy of the models based on your training set. This will help you estimate the best performing model

for your particular case, but feature ranking is provided only for random forest in the task learning_benchmark. Currently, the random forest algorithm is configured to use the combination of parameters that yields the best accuracy results for inference.

Once this task has been successfully completed, you can carry out the inference task for the new images:

docker compose run --rm imagine 1 datafile.tsv 10 inference

To activate all active algorithms as described in 5., run:

docker compose run --rm imagine 1 datafile.tsv 10 inference --all_learners

Alternatively, if you use your own file with paths:
docker compose --env-file test_paths.txt run --rm imagine 1 datafile.tsv 10 inference

The most important results of the inference are the prediction, the probabilities, and the visualisation of the most significant feature values to support interpretation.

The results of the inference by random forest (rf) are contained in different files:

- inference_summary
- rf_predictions
- rf_probabilities
- rf_features
- explanations

Results of inference for other models, if activated by –all_learners, have 'rf' in the above filenames replaced by the name of the other tested models.

inference_summary

- Model: The name of the model used for inference; "rf" denotes random forest.
- Num_predictions: The total number of images for which inference was performed.
- Unique_predictions: The number of distinct predicted classes in the inference dataset.

rf_features

The feature values calculated for each image in the dataset used for prediction.

rf_predictions

- Sample_name: The identifier or name of the image on which inference was performed.
- Prediction: The inferred class or strain name predicted by the model based on the features extracted from the image.

rf_probabilities

For each class present in the training set, a probability is assigned representing the likelihood that the unknown sample belongs to that class. These probabilities sum to 1 and are calculated based on the proportion of random forest trees voting for each class. This probability provides a measure of confidence in the prediction. When multiple classes have similar probabilities, the prediction is ambiguous, and the user should examine feature-level details via the provided visualisations to better understand the model's decision.

Explanations
For every image in the inference dataset, plots derived from SHAP (SHapley Additive exPlanations) values are provided. SHAP values are a method used to explain the output of machine learning models. They quantify the contribution of each feature to the model's prediction. A higher SHAP value for a feature indicates that this feature has a stronger positive influence on predicting a specific class, while a negative value means the feature decreases the likelihood of that prediction. SHAP values help clarify which characteristics of an input (in this case, image features) drive the model's decisions, making the "black box" of machine learning more interpretable.

For each image a separate plot is generated for each possible class present in the training data, enabling comparison across all relevant categories.

These plots are particularly useful in two scenarios:

- When two classes have similar prediction probabilities, allowing identification of specific features that may have led to confusion or misclassification.
- When misclassification occurs, where the inferred class changes and differs from the true class (e.g., from strain L628 to strain L1764), the plots help identify which features contributed most to this switch compared to the original class.

rf_summary_plot
This plot shows how multiple features affect the model's predictions across samples. The y-axis lists features ranked by overall importance. Each dot represents the SHAP value of a feature for a single image, positioned along the x-axis to indicate the strength of that feature's influence on the predicted class. Dot colors indicate the actual feature value (e.g., red for high values, blue for low), illustrating how feature magnitude affects inference outcomes. Clusters of similarly colored dots demonstrate that predictions depend on the values of these key features, not just their presence or absence, highlighting the combined contribution of multiple features to model decisions.

rf_feature_importance
For each feature calculated, the strength of its relationship to the target is indicated, with larger values showing that the feature was more important in predicting strain, and vice versa. This is similar to the file rankings_label provided for the training set, and here it is provided for the inference set.

rf_shap_values
For each image in the inference set, the table lists all extracted features with their corresponding SHAP values, indicating the contribution and importance of each feature to the model's prediction.

Recommended workflow to interpret machine learning inference results

To avoid confusion, we use the following terms consistently throughout the following description:

- Class: The category or label that the machine learning model assigns to an image (e.g., a predicted identity of a strain).

- Strain: The actual biological category or type of the sample being analyzed, such as a specific bacterial strain. This represents the true class in experimental terms.
- Treatment: The experimental condition or manipulation applied to a biological sample (e.g., a drug or environmental change) before imaging. Multiple images from replicated treatments are used to assess model predictions.

Experimental design

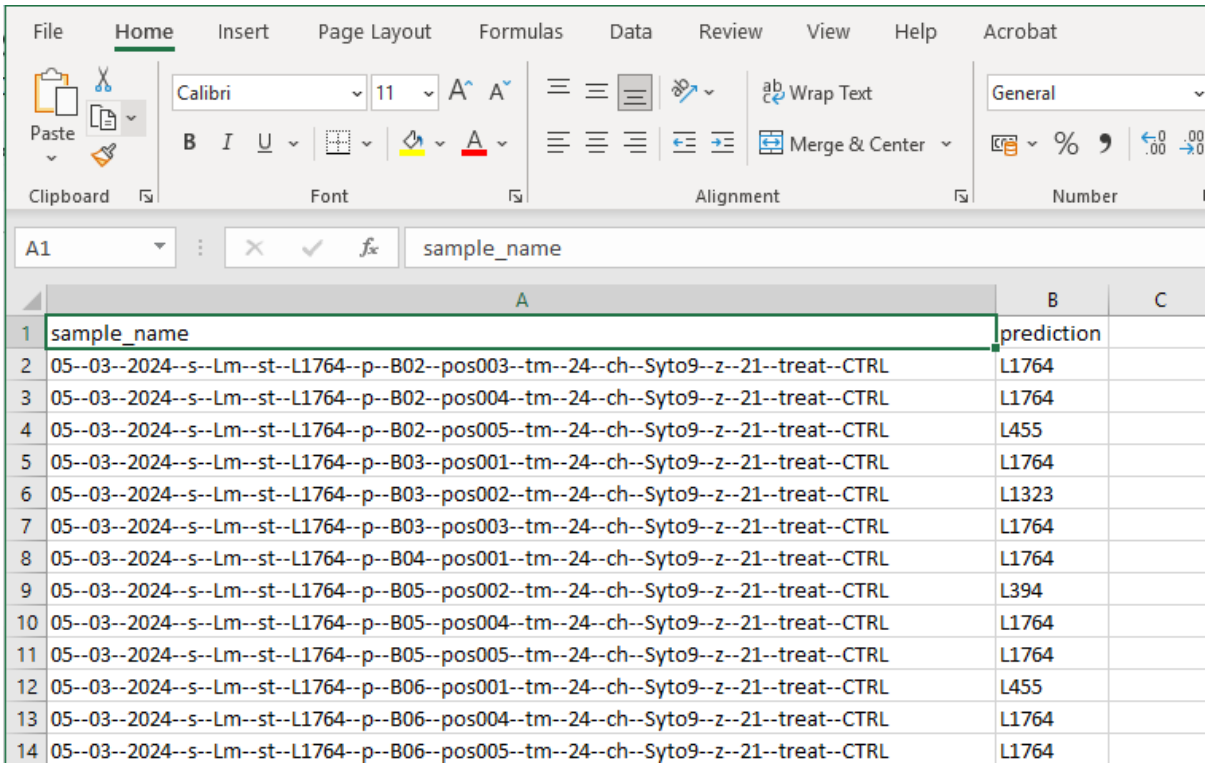
We recommend performing multiple technical replicates and independent experiments on different dates for each treatment and control condition. This approach increases robustness and enables more reliable interpretation of results.

Control experiments integration

Control images should be divided and included in both the training and inference sets as sanity controls. This helps validate the model's performance and supports interpretation.

Aggregation of inference results

After inference, collect all technical and independent repetitions of the same treatment condition. Construct a matrix where inferred classes are counted per treatment, as shown in Table 1. This table records the number of images predicted for each class (strain) from the training set, helping to identify the predominant inferred class per treatment condition. The data for this table are from the file rf_predictions.



	A	B	C
1	sample_name	prediction	
2	05--03--2024--s--Lm--st--L1764--p--B02--pos003--tm--24--ch--Syto9--z--21--treat--CTRL	L1764	
3	05--03--2024--s--Lm--st--L1764--p--B02--pos004--tm--24--ch--Syto9--z--21--treat--CTRL	L1764	
4	05--03--2024--s--Lm--st--L1764--p--B02--pos005--tm--24--ch--Syto9--z--21--treat--CTRL	L455	
5	05--03--2024--s--Lm--st--L1764--p--B03--pos001--tm--24--ch--Syto9--z--21--treat--CTRL	L1764	
6	05--03--2024--s--Lm--st--L1764--p--B03--pos002--tm--24--ch--Syto9--z--21--treat--CTRL	L1323	
7	05--03--2024--s--Lm--st--L1764--p--B03--pos003--tm--24--ch--Syto9--z--21--treat--CTRL	L1764	
8	05--03--2024--s--Lm--st--L1764--p--B04--pos001--tm--24--ch--Syto9--z--21--treat--CTRL	L1764	
9	05--03--2024--s--Lm--st--L1764--p--B05--pos002--tm--24--ch--Syto9--z--21--treat--CTRL	L394	
10	05--03--2024--s--Lm--st--L1764--p--B05--pos004--tm--24--ch--Syto9--z--21--treat--CTRL	L1764	
11	05--03--2024--s--Lm--st--L1764--p--B05--pos005--tm--24--ch--Syto9--z--21--treat--CTRL	L1764	
12	05--03--2024--s--Lm--st--L1764--p--B06--pos001--tm--24--ch--Syto9--z--21--treat--CTRL	L455	
13	05--03--2024--s--Lm--st--L1764--p--B06--pos004--tm--24--ch--Syto9--z--21--treat--CTRL	L1764	
14	05--03--2024--s--Lm--st--L1764--p--B06--pos005--tm--24--ch--Syto9--z--21--treat--CTRL	L1764	

Figure 1. Screenshot of the rf_predictions output file in TSV format, which can be opened using Excel.

Table 1. Summary of image counts per class for each treatment condition.

This table shows the number of images classified as belonging to each class (strain) for repeated experiments under the same condition (combination of strain and treatment). Each

row represents one treatment condition with multiple replicates. The numbers indicate how many images the model assigned to each class.

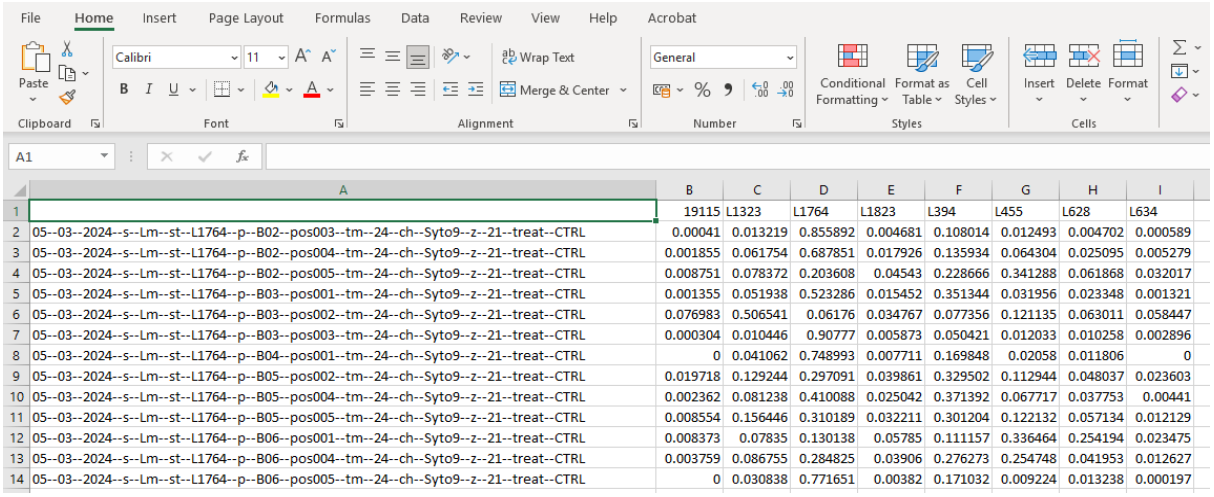
		voted class							
strain	treatment	**L1764**	**L455**	**L1323**	**L394**	**L628**	**ATCC 19115**	**L1823**	**L634**
L1764	control	**19**	2	1	3	0	0	0	0
L1764	melon extract coat	**46**	0	0	4	0	0	0	0
L1764	milk coat	**46**	0	0	2	0	0	2	0
L1764	tatar steak extract coat	**50**	0	0	0	0	0	0	0
L628	control	0	0	0	0	**24**	0	1	0
L628	melon extract coat	**21**	14	0	7	5	0	2	0
L628	milk coat	**42**	0	0	4	0	0	4	0
L628	tatar steak extract coat	**47**	0	0	1	0	0	2	0

The class with the highest count in a given row is considered the predominant inferred class for that treatment condition. This aggregation identifies the overall prediction across multiple images and replicates, increasing confidence in the inference.

Using prediction probabilities

Since individual images may vary, each inference is associated with a probability score indicating the model's confidence in the assigned class. These probabilities are provided in rf_probabilities.

- High probabilities indicate strong confidence in the prediction.
- Probabilities around or below 0.6 require careful examination, especially if the model's probabilities for control classes are similarly close; this may indicate ambiguous or difficult cases. Low probabilities often mean the model assigned the closest available class in the training set, but with low confidence due to dissimilarities.
- We recommend compiling probability values for all repetitions of each treatment in an Excel table and calculating summary statistics such as mean, median, standard deviation, minimum, maximum, and percentiles. This provides insight into the central tendency and variability of the model's confidence per treatment, highlighting any outliers that could skew averages. The average probability then serves as a guide for interpretation.



	A	B	C	D	E	F	G	H	I
1		19115	L1323	L1764	L1823	L394	L455	L628	L634
2	05--03--2024--s--Lm--st--L1764--p--B02--pos003--tm--24--ch--Syto9--z--21--treat--CTRL	0.00041	0.013219	0.855892	0.004681	0.108014	0.012493	0.004702	0.000589
3	05--03--2024--s--Lm--st--L1764--p--B02--pos004--tm--24--ch--Syto9--z--21--treat--CTRL	0.001855	0.061754	0.687851	0.017926	0.135934	0.064304	0.025095	0.005279
4	05--03--2024--s--Lm--st--L1764--p--B02--pos005--tm--24--ch--Syto9--z--21--treat--CTRL	0.008751	0.078372	0.203608	0.04543	0.228666	0.341288	0.061868	0.032017
5	05--03--2024--s--Lm--st--L1764--p--B03--pos001--tm--24--ch--Syto9--z--21--treat--CTRL	0.001355	0.051938	0.523286	0.015452	0.351344	0.031956	0.023348	0.001321
6	05--03--2024--s--Lm--st--L1764--p--B03--pos002--tm--24--ch--Syto9--z--21--treat--CTRL	0.076983	0.506541	0.06176	0.034767	0.077356	0.121135	0.063011	0.058447
7	05--03--2024--s--Lm--st--L1764--p--B03--pos003--tm--24--ch--Syto9--z--21--treat--CTRL	0.000304	0.010446	0.90777	0.005873	0.050421	0.012033	0.010258	0.002896
8	05--03--2024--s--Lm--st--L1764--p--B04--pos001--tm--24--ch--Syto9--z--21--treat--CTRL	0	0.041062	0.748993	0.007711	0.169848	0.02058	0.011806	0
9	05--03--2024--s--Lm--st--L1764--p--B05--pos002--tm--24--ch--Syto9--z--21--treat--CTRL	0.019718	0.129244	0.297091	0.039861	0.329502	0.112944	0.048037	0.023603
10	05--03--2024--s--Lm--st--L1764--p--B05--pos004--tm--24--ch--Syto9--z--21--treat--CTRL	0.002362	0.081238	0.410088	0.025042	0.371392	0.067717	0.037753	0.00441
11	05--03--2024--s--Lm--st--L1764--p--B05--pos005--tm--24--ch--Syto9--z--21--treat--CTRL	0.008554	0.156446	0.310189	0.032211	0.301204	0.122132	0.057134	0.012129
12	05--03--2024--s--Lm--st--L1764--p--B06--pos001--tm--24--ch--Syto9--z--21--treat--CTRL	0.008373	0.07835	0.130138	0.05785	0.111157	0.336464	0.254194	0.023475
13	05--03--2024--s--Lm--st--L1764--p--B06--pos004--tm--24--ch--Syto9--z--21--treat--CTRL	0.003759	0.086755	0.284825	0.03906	0.276273	0.254748	0.041953	0.012627
14	05--03--2024--s--Lm--st--L1764--p--B06--pos005--tm--24--ch--Syto9--z--21--treat--CTRL	0	0.030838	0.771651	0.00382	0.171032	0.009224	0.013238	0.000197

Figure 2. Screenshot of the rf_probabilities output file in TSV format, viewable in Excel. Column headers include the inferred image names and the names of each class from the training set.

Feature importance analysis
For deeper contextual interpretation, analyze the key features driving the model's decisions:

- Extract the SHAP values for each feature from the rf_shap_values file, which lists SHAP values for all features calculated during feature extraction for all images in the inference set.
- Positive SHAP values indicate features pushing the model toward the predicted class; negative values indicate features working against it.
- If true class labels are known for some examples they can be used to identify correctly and incorrectly classified examples.
- Examine rf_feature_importance for information on the entire dataset for inference.

Evaluating "correct" vs. "incorrect" classifications
If true class labels are known, you can combine the data on the top 15 most important features with feature values from the rf_features file to compare characteristics between correctly and incorrectly classified images.

- Correct classification means the predicted class matches the true strain (e.g., strain L628 correctly predicted as L628).
- Incorrect classification means predicted and true strains differ (e.g., L628 predicted as L1764). Although termed "incorrect" in ML terms, such predictions may reflect biological similarity or phenotype convergence between strains, offering valuable insights.

By comparing the feature types and values in these two groups, you can identify which features contribute to class switching and better understand model behavior.

7. MicroICS implementation examples

7.1. How do you use the pipeline if you want to link biofilm features to properties other than the strain name?

The easiest way is to replace the string after _st_ in the image name with the desired property. For example, to associate biofilm features with clinical relevance, replace the string after _st_ (strain name) with CI (clinically relevant) or CNI (clinically not relevant) based on the epidemiology of the strain. Then run the pipeline as described. For the latter example, the renaming of the image would look like this:

The original name is: 13042023_s_Lm_st_L628_p_C06_pos001_tm_24_ch_Syto9_z_21 and then we replace the _st_ with CI, which means that strain L628 is clinically relevant to get the image name: 13042023_s_Lm_st_CI_p_C06_pos001_tm_24_ch_Syto9_z_21.

The same applies to strain L1764, which is not clinically relevant:

The original name 13062023_s_Lm_st_L1764_p_G05_pos002_tm_24_ch_Syto9_z_21 becomes 13062023_s_Lm_st_CNI_p_G05_pos002_tm_24_ch_Syto9_z_2

7.2. Using the pipeline to create images with reduced number of layers and determining whether the number of layers in a 3D image affects classificator performance

The number of layers in the 3D images can be reduced with the command from the pipeline:

docker compose run --rm imagine datafile.tsv reduce_layers <initial number of layers>

To reduce the layers from 21 to 11, you can use the following example:

docker compose run --rm imagine 1 datafile.tsv 10 reduce_layers 21

This command removes every other layer from bottom to top to create a 3D image with half of the original layers. By successively performing the reduce-layers task (21, 11, 6, 3), we can reduce the number of layers down to the bottom layer. Then use images with a reduced number of layers to run the pipeline as described above.

8. Feature descriptions to help with interpretation

A set of quantitative features is extracted from each 3D image (Table 2). The 3D images here consist of several layers, e.g. we have a 3D image consisting of 21 layers (Figure 3). Understanding that 3D images consist of several sub-images is crucial for understanding how calculations are performed.

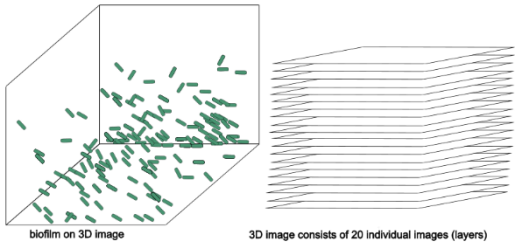


Figure 3. Schematic representation of input images.
Green rods on the left image represent bacteria within the biofilm.

Image analysis typically involves some pre-processing steps to distinguish the background from the foreground and to aid in object recognition within the image. As these procedures require human input, we aim to avoid them. Generally, the input for feature calculation is a raw image; however, for feature counts (number of bacteria), the pixel intensity values in the raw image are also normalised. “Normalisation” for feature count is performed by subtracting each pixel intensity value from the average pixel intensity of that layer and then dividing by the difference between maximum and minimum pixel intensity (x - average) / (max - min). This adjusts the range of pixel values in an image to a standard range, here between 0 and 1. If the feature name in the data file contains “Norminten”, the image has been normalised; if the raw image was used as input, it has not.

All features provide a value for each image but are calculated in two different ways: a) the feature is computed immediately for the whole image or b) the layers are first considered individually and then a value per image is obtained by aggregating the layer values by computing the mean (mean), median (med), minimum (min), maximum (max), standard deviation (std), variance (var) and quantiles (q10, q25, q75 and q90). If the feature is aggregated in the data file, this is noted as _mean, _med, _min, _max, _std, _var, _q10, _q25, _q75, _q90 at the end of the feature's name. In addition, we also introduce normalised values of features that are calculated for each layer as described later.

Below you will find descriptions of the features. Table 1 lists the abbreviations from the data file, the full names of the features, the processing steps and brief explanations.

Morphological features:

These features capture geometric aspects of the biofilm components.

“Biovolume” is the most commonly used feature in the biofilm community, where the number of pixels belonging to the biofilm is counted and then multiplied by the voxel size (here 0.13x0.13x0.5 µm). A threshold is set before counting the pixels and only pixels above this threshold are counted. We generate multiple intensity thresholds, resulting in many biovolume features that differ only by the threshold, so the algorithm can capture the values relevant for classification. This feature is calculated for the image as a whole and not per layer.

The “count" features give the number of bacteria on each layer. For this, a threshold is applied and then the biofilm components (here bacteria) are segmented using the connected components approach from Scipy Multidimensional Image Processing (scipy.ndimage). The objects are then counted for each layer and the counts per layer are aggregated to obtain one value per image. We generate multiple intensity thresholds, resulting in many count features that differ only by the threshold value, so that the algorithm can capture the relevant values for classification. The normalized values of the counts are calculated as described earlier. Although counts intuitively reflect the amount of bacteria, they are based on thresholding and only represent bacteria detected above that threshold, which may not capture all bacteria optimally.

“Substratum Relative Coverage”: Percentage of surface covered by biofilm in the 1st layer (surface of the well) of the image.

The statistics-based intensity features:

The feature “globalMean” is extracted from the entire 3D image by calculating the average value of all pixel intensity values in the 3D image and is not derived layer by layer (Figure 2).

A set of intensity-based features labeled “maxdiffs”, “mindiffs”, “mdiffs” is calculated from the differences of the average pixel intensity values between two consecutive layers, i.e. the average value of the first layer is subtracted from the average value of the second layer and so on until 19 differences are obtained (Figure 2). Then the maximum, minimum and median differences are selected and labeled as “maxdiffs”, “mindiffs”, “mdiffs” and given as individual values for each image.

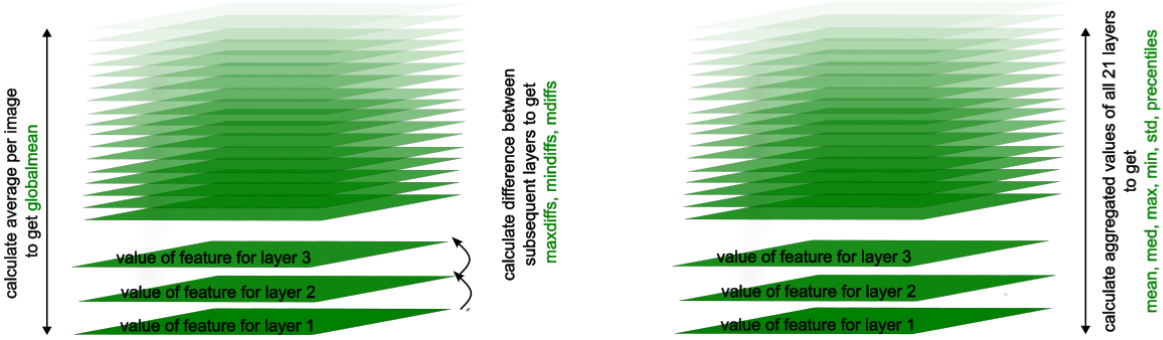


Figure 2. Schematic representation of calculation of intensity-based features.

The features “diff” and “diffNormalized” are calculated by subtracting the maximum pixel intensity value from the minimum pixel value for each layer separately. Consequently, we

obtain several values for a feature (one per layer), which we combine into a single value by aggregation. In addition, we also introduce normalized features that are calculated for each layer, where the values are normalized by dividing the feature value for the individual layer by the sum of the values calculated for that layer. For example, the maximum difference in pixel intensity on layer 1 is divided by the sum of the maximum differences calculated for layers 1, 2, 3, 4 up to 21. Normalized features are indicated by the presence of “Normalized” in the feature name, e.g. “diffNormalized”.

Another set of intensity-based features labeled “mean”, “med”, “max”, “min” and “std” are calculated by computing the mean, median, maximum, minimum or standard deviation of the pixel intensity values for each layer, and these values are then aggregated. Normalized values are also calculated for these features by calculating the proportion for each layer.

Texture features:

Texture features provide a description of an image that contains information about the spatial distribution of pixel intensity values.

Void fraction “minprop” is calculated by counting the number of pixels in a layer whose pixel intensity value is below the average pixel intensity value in that layer. This feature is calculated separately for each layer and then aggregated for the entire image as described above. Normalized values are available as “MinPropNormalized”. This feature describes the empty space of the image that is not occupied by bacteria.

The “Homogeneity” is calculated based on the co-occurrence matrix, which represents the distribution of changes in the pixel intensity values of neighboring pixels in the X, Y and Z directions. The formula for calculating homogeneity has already been described[1]. It measures the similarity of image structures that are spatially close to each other: a higher homogeneity indicates a more homogeneous image structure. This feature is calculated for the image as a whole.

“ThicknessThreshold” is calculated by tracking the pixel intensity value of each pixel from the first layer (substrate) in height (along the normal to the substrate) until the pixel intensity value reaches 0. The number of the layer where the pixel intensity value reaches 0 is marked, and then the average value of these layers (heights) is calculated. One value is calculated per image.

“roughnessThreshold” is calculated by measuring the height differences for consecutive pixels calculated for "ThicknessThreshold" and then calculating the average. It reflects the height differences of the biofilm.

“Spreading” refers to the spread of the biofilm volume in the 3D image and is calculated as described before[2]. Three different values are calculated: horizontal spreading (in xy-direction), "Sxy", vertical spreading (in z-direction), "Sz", and total spreading (in xyz-space).

Large language model (LLM) based features (GPT features)

“GPT volume” refers to the volume of the image layer occupied by the biofilm, where the number of bacteria is multiplied by the voxel size and then aggregated per image.

“GPT fractal dimension” captures the complexity of the geometric patterns on each image layer and is then aggregated into a single value.

"GPTContrast", “GPTDissimilarity", “GPTHomogeneity", “GPTEnergy",” “GPTCorrelation” and “GPTASM" were calculated using the Graycomatrix method. In this method, a gray- level co-occurrence matrix (GLCM) is created by counting how often a pixel with intensity i occurs in a specific spatial relationship to a pixel with intensity j. The correlation measures, for example, how strongly pairs of grey levels vary together, which indicates predictable spatial patterns in the image.

“GPT correlation”: Similar to the Pearson correlation coefficient, the correlation value is calculated based on the correlations between the values in the graycomatrix. If the correlations are high, this indicates that certain grey level pairs occur in a predictable manner.

“GPT Contrast”: The contrast highlights the difference between gray-level values. Larger intensity differences contribute more to the sum, so a higher contrast value indicates greater variability in the intensities of neighbouring pixels.

“GPT dissimilarity”: Similar to contrast, but it is calculated from absolute differences rather than a squared difference. Dissimilarity also captures local variations in grey levels, but its contribution grows more linearly compared to the quadratic term of contrast.

“GPT homogeneity”: It is calculated in a very similar way to non-GPT homogeneity, but here it is first calculated per layer and then summarised per image by aggregation. However, a high homogeneity indicates a more uniform and less varied texture.

“GPT energy”: Energy is another way of expressing the uniformity or regularity of texture. It provides an indication of how consistent the structure of the biofilm is at the microscopic level.

Table 2. List and descriptions of biofilm features calculated by the MicroICS pipeline.

The image can be analysed as a raw image without adjustments or the pixel intensity values can be normalized, which is referred to as norminten. The pixel intensity values in the raw image are normalized per layer by subtracting the average pixel intensity value of the individual layer by the sum of average pixel intensity values for all layers to obtain values between 0 and 1.

All features in the data file are aggregated by calculating the mean, median, maximum, minimum and quartiles 10, 25, 75 and 90, standard deviation and variance across the image layers. If the name of the feature contains "SUMMARY", this means that it is an aggregated value of the feature, and the aggregation type is written after the underscore at the end of the feature name _mean, _med, _min, _max, _q10, _q25, _q75, _q90, _std, _var.

Morphological features:

Feature name in the manuscript	Feature name in the datafile provided by the pipeline	Input	Value normalization	Calculation per _	Description of feature	Threshold
Counts (threshold < n)*	counts(inten<threshold-SUMMARYCustomAlgos.tsv	raw image	no	layer	The number of bacteria in each layer of the 3D image is determined by first applying the threshold value, then segmenting the individual bacteria, and finally counting them. The raw image or an image whose intensity values have been normalized is used as input. For the “normalized” feature, the counts per layer are normalized as a proportion of the sum of all values * 100 (to obtain percentages) per value.	Range from 0.01 to 0.2 increments for 0.01
Counts (normalized intensity, threshold < n)*	counts(Norminten<threshold-SUMMARYCustomAlgos.tsv	intensity normalised on raw image	yes	layer		Range from 0.01 to 0.2 increments for 0.01
Counts (normalized per layer, threshold < n)*	counts(inten<thresholdNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer		Range from 0.01 to 0.2 increments for 0.01
Counts (normalized intensity, normalized per layer, threshold < n)*	counts(Norminten<thresholdNormalized-SUMMARYCustomAlgos.tsv	intensity normalised on raw image	yes	layer		Range from 0.01 to 0.2 increments for 0.01
Biovolume	BioVolumeThr-SUMMARYCustomAlgos.tsv	raw image	no	image	The biofilm volume is calculated by counting the number of pixels belonging to the biofilm after thresholding and multiplying this by the voxel size, which represents the physical dimensions of each pixel in height, width, and length (in micrometres). This gives the total volume occupied by the biofilm in the 3D image. Only one volume value is given per image, and the aggregated values remain consistent.	Range from 0.01 to 0.29 increments for 0.01
GPT volume	GPTVolume-SUMMARYCustomAlgos.tsv	raw image	no	layer	The GPT volume is similar to the biovolume described above, but here the biovolume is first determined layer by layer after thresholding, and then the values of these layer counts are aggregated for each image.	Fixed value determined for each image
Normalized GPT volume	GPTVolumeNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer		Fixed value determined for each image
Substratum relative coverage	SubstratumRelativeCoverage-SUMMARYCustomAlgos.tsv	raw image	no	image	Percentage of the surface of the well (1st layer of the image) covered with biofilm.	Fixed value determined for each image

**Feature “counts”: Counts are a unique feature that captures variations in counts of bacteria within biofilm by changing the threshold used to detect individual bacteria. It is an ensemble of features known to be correlated with each other and useful in machine learning to capture all aspects of thresholding and counting.*

Statistics-based intensity features:

Feature name in the manuscript	Feature name in the datafile provided by the pipeline	Input type	normalization	Calculation per _	Description of feature	Threshold
Global mean	globalMean-SUMMARYDiffGlobal.tsv	raw image	no	image	The average pixel intensity of the entire 3D image reflects the overall brightness of the biofilm. A single value is calculated for each image, whereby all aggregated values are identical.	no
Layer-wise global maximum**	maxdiffs-SUMMARYDiffGlobal.tsv	raw image	no	image	The average pixel intensities are calculated for each layer of the 3D image, and the difference between the average of each layer and the next is calculated. Only the maximum difference is given, which represents the largest change in brightness between the layers and extends the understanding of the global mean.	no
Layer-wise global minimum**	mindiffs-SUMMARYDiffGlobal.tsv	raw image	no	image	The difference between the average intensities of consecutive layers is calculated in a similar way, but only the minimum difference is given to highlight areas of little or no change, which may correspond to stable biofilm regions or the background.	no
Layer-wise global mean**	mdiffs-SUMMARYDiffGlobal.tsv	raw image	no	image	The mean difference between the intensities of successive layers is calculated and reported and reflects the overall uniformity or variability of brightness in the vertical dimension of the biofilm.	no
/	eigen-SUMMARYDiffGlobal.tsv	raw image	no	/	A value representing the differences in the overall 3D image is noted but is not currently calculated.	no
Intensity differences**	diff-SUMMARYCustomAlgos.tsv	raw image	no	layer	The difference between maximum and minimum pixel intensity within each layer is calculated and aggregated, where: (a) the mean aggregation indicates the image contrast, b) the median reflects the typical intensity, c) the standard deviation indicates the variability of the contrast between the layers, and d) the quantiles describe the distribution skewness of the contrast between the layers.	no
Normalized intensity differences**	diffNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer	The differences between the maximum and minimum intensities per layer are normalized as a percentage of the total max-min sum across all layers and then aggregated to characterize the relative contrast distribution.	no
Mean intensity	mean-SUMMARYCustomAlgos.tsv	raw image	no	layer	The mean pixel intensity is calculated for each layer and then averaged across the layers, representing the average brightness per layer; higher values indicate denser or brighter regions and vice versa.	no
Normalized mean intensity	meanNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer	The mean intensity on each layer is normalized as a percentage of the total sum of mean intensities per image before averaging across layers.	no
Median intensity	med-SUMMARYCustomAlgos.tsv	raw image	no	layer	The median pixel intensity is calculated for each layer, and then the median values are aggregated across all layers.	no
Normalized median intensity	medNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer	The median intensity per layer is normalized as a percentage of the total sum of the median intensities per image and then aggregated across layers.	no
Maximum intensity	max-SUMMARYCustomAlgos.tsv	raw image	no	layer	The maximum pixel intensity is calculated per layer and aggregated, highlighting localized bright spots such as clumps or active cells.	no

Feature name in the manuscript	Feature name in the datafile provided by the pipeline	Input type	normalization	Calculation per _	Description of feature	Threshold
Normalized maximum intensity	maxNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer	The maximum intensity per layer is normalized as a percentage of the total maximum per image and then aggregated.	no
Minimum intensity	min-SUMMARYCustomAlgos.tsv	raw image	no	layer	The minimum pixel intensity per layer is calculated and aggregated to capture the darkest or least dense biofilm regions; in combination with other statistics, it shows biofilm gaps, heterogeneity and spatial structure.	no
Normalized minimum intensity	minNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer	The minimum intensity per layer is normalized as a percentage of the total minimum sum per image before aggregation.	no
Standard deviation of intensity	std-SUMMARYCustomAlgos.tsv	raw image	no	layer	The standard deviation of pixel intensities per layer is calculated and aggregated; low values indicate uniform intensity (low contrast), while high values indicate high contrast and high variability.	no
Normalized standard deviation of intensity	stdNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer	The standard deviation per layer is normalized as a percentage of the total standard deviation sum per image and then aggregated.	no

***Layer wise features:*
Layer wise features capture how much the intensity fluctuates between the layers and how features differ across the volume.

Texture features:

Feature name in the manuscript	Feature name in the datafile provided by the pipeline	Input type	normalization	Calculation per _	Description of feature	Threshold
Void fraction (min Prop)	minProp-SUMMARYCustomAlgos.tsv	raw image	no	layer	The number of pixels with intensity below the average pixel intensity is counted for each image. Min prop refers to dark areas, such as empty regions not occupied by biofilm.	no
Normalized void fraction	minPropNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer		no
Homogeneity	Homogenity-SUMMARYCustomAlgos.tsv	raw image	no	image	Homogeneity is calculated using the co-occurrence matrix, which captures the distribution of intensity changes between neighbouring pixels in the X, Y and Z directions (after Beyenal et al. 2004, doi: 10.1016/j.mimet.2004.08.003). It measures how similar neighbouring image objects are, with higher values indicating greater uniformity. One value is given per image, and the aggregated values are identical.	no
Thickness	ThicknessThreshold=0.01-SUMMARYCustomAlgos.tsv	raw image	no	image	Pixel intensities are tracked vertically from the first image layer, recording for each pixel the height at which the intensity drops to zero. The average of these heights is reported as a single image value using the same aggregation.	Range from 0.01 to 0.29 increments for 0.01
Roughnes	RoughnessThreshold=threshold-SUMMARYCustomAlgos.tsv	raw image	no	image	The average height difference between neighbouring pixels at a given layer thickness is calculated and reflects the roughness of the biofilm surface. One value per image is given, with uniform aggregation.	Range from 0.01 to 0.29 increments for 0.01
Horizontal spreading	SpreadingHorizontal-SUMMARYCustomAlgos.tsv	raw image	no	image	Three distributions (Dx, Dy, Dz) of the x, y, and z coordinates of the biofilm pixels are created, as described by Mueller et al. (2006, doi: 10.1186/1472-6785-6-1), and the variances for each axis are calculated. Horizontal spreading is calculated by summing up variances in x and y. Vertical spreading (in z direction) equals the variance of z. Total spreading (in z direction) is calculated by summing up variances in x, y and z.	no
Total spreading	SpreadingTotal-SUMMARYCustomAlgos.tsv	raw image	no	image		no
Vertical spreading	SpreadingVertical-SUMMARYCustomAlgos.tsv	raw image	no	image		no
GPT fractal dimension	GPTFractalDim-SUMMARYCustomAlgos.tsv	raw image	no	layer	The fractal dimension reflects the complexity and irregularity of the surface or boundary of the biofilm and quantifies this complexity at different scales; higher values indicate more complex and branched structures rather than smooth shapes.	no
Normalized GPT fractal dimension	GPTFractalDimNormalized-SUMMARYCustomAlgos.tsv	raw image	yes	layer		no
GPT homogeneity	GPTHomogeneity-SUMMARYCustomAlgos.tsv	raw image	no	image	This feature is calculated similarly to non-GPT homogeneity, but is first computed for each layer and then aggregated for each image; higher homogeneity indicates a more uniform and consistent texture.	no

The following features were calculated by Graycomatrix method, which constructs a grayscale co-occurrence matrix (GLCM) by calculating how many times a pixel with intensity value (grayscale) *i* occurs in a given spatial relationship to a pixel with value *j*.

Feature name in the manuscript	Feature name in the datafile provided by the pipeline	Input type	normalization	Calculation per _	Description of feature	Threshold
GPT contrast	GPTContrast-SUMMARYCustomAlgos.tsv	raw image	no	image	Contrast emphasises the difference between grey-level values. Larger differences in intensity contribute more to the sum, so a higher contrast value indicates greater variability in the intensities of neighbouring pixels.	no
GPT correlation	GPTCorrelation-SUMMARYCustomAlgos.tsv	raw image	no	image	The correlation value is calculated similarly to the Pearson correlation coefficient, using the correlations between values in the graycomatrix. High correlations indicate that certain grey-level pairs occur in a predictable manner.	no
GPT dissimilarity	GPTDissimilarity-SUMMARYCustomAlgos.tsv	raw image	no	image	Similar to contrast, but it is calculated from absolute differences instead of a squared difference. Dissimilarity also captures local variations in gray levels, but its contribution grows more linearly compared to contrast's squared term. Dissimilarity is opposite to homogeneity.	no
GPT energy	GPTEnergy-SUMMARYCustomAlgos.tsv	raw image	no	image	Energy is another way to express the uniformity or orderliness in the texture. It is often used because it is on a [0,1] scale if P is normalized.	no

LITERATURE

1. Beyenal, H., Donovan, C., Lewandowski, Z. & Harkin, G. Three-dimensional biofilm structure quantification. *J Microbiol Methods* 59, 395–413 (2004).

2. Mueller, L. N., de Brouwer, J. F., Almeida, J. S., Stal, L. J. & Xavier, J. B. Analysis of a marine phototrophic biofilm by confocal laser scanning microscopy using the new image quantification software PHLIP. *BMC Ecology 2006 6:1* 6, 1–15 (2006).