

Manual for MRCNN

Version: 13.11.2023

Editor: Stepan Sibirtsev

This manual works for **non-cluster users**. If you want to run MRCNN on your **personal or office computer**, this manual guides you through the installation and application of MRCNN. If you're going to run MRCNN on a cluster, please check the manual for cluster users. We recommend training the MRCNN models on a GPU instead of a CPU to reduce the time required. If you have the opportunity, you should train your MRCNN models on a GPU cluster, especially if a sensitivity analysis, e.g., hyperparameter tuning (see our paper), is performed. Image evaluation with the **MRCNN models trained on a GPU can also be performed on a CPU**. The evaluation is less computationally time-consuming than training, so it works well on a CPU.

These instructions are adapted to **Windows** as an operating system and **Visual Studio Code** (VSC) as a source code editor. However, MRCNN can also be run on Linux, and any source code editor can be used. The instructions must then be adapted accordingly by the user.

Table of Contents

1	Installation	3
1.1	MRCNN repository	3
1.2	Visual Studio Code	3
1.3	Python	3
1.4	Anaconda	3
1.5	Anaconda environment	3
1.6	Assigning Anaconda environment	4
1.7	VGG Image Annotator	5
1.8	Weights & Biases (optional)	5
2	Train MRCNN models	6
2.1	Preparing images (important information!)	Error! Bookmark not defined.
2.2	Preparing training/validation and testing sets	6
2.3	Preparing Weights & Biases project (optional)	8
2.4	MRCNN training	8
3	Image evaluation	9
3.1	Preparing data	9
3.2	Processing images with MRCNN model	9
3.3	Processing images from the manual evaluation with VIA	10
3.4	Evaluating MRCNN detection performance	10
4	Appendix	11
4.1	Description of MRCNN repository folders and files	11
4.2	MRCNN training and processing parameters	13

1 Installation

1.1 MRCNN repository

Download **all folders and files** from the MRCNN repository to a folder of your choice on your personal or office computer. You can find some description of the repository folders and files in **Table 1**.

1.2 Visual Studio Code

Download and install **Visual Studio Code**: <https://code.visualstudio.com/>

1.3 Python

- 1) Start the Visual Studio Code.
- 2) Switch to the "**Extensions**" tab in the bar on the left of VSC.
- 3) Search for **Python** and install it.
- 4) Sometimes, it is helpful to **restart** the source code editor or the computer after installation.

1.4 Anaconda

- 1) Visit the **Anaconda** download page: <https://www.anaconda.com/download>
- 2) Download the **Windows version** from the download page and install it. When installing, be sure to select "**Install for: Just Me**" (so the installation also works on the office computer/server without admin rights) and select "**Add Anaconda to my PATH environment variable**".
- 3) For further assistance, visit Installing Anaconda on Windows **Tutorial**:
<https://www.datacamp.com/tutorial>

1.5 Anaconda environment

We install the anaconda environment from a **YML** file, which ensures that the appropriate and compatible packages are installed directly and automatically into the environment, avoiding errors and saving time. We provide suitable **YML** files depending on the operating system (Windows and Linux) and calculation device (GPU and CPU) located in the **environment folder of the MRCNN repository**.

- 1) Decide whether you want to run MRCNN on **CPU or GPU**. If you want to run MRCNN on GPU, find out which GPU you have, **AMD or Nvidia**.
- 2) Choose the suitable environment located in the environment folder of the MRCNN repository:
 - a) Windows & AMD GPU: env_mrcnn_windows_gpu_amd.yml
 - b) Windows and Nvidia GPU: env_mrcnn_windows_gpu_nvidia.yml
 - c) Windows and CPU: env_mrcnn_windows_cpu.yml
- 3) Start the Visual Studio Code and install the suitable environment as follows via the corresponding **<FileName>.yml** file:
- 4) In the source code editor, navigate to the **<FolderName>** folder where the **<FileName>.yml** file is stored (the **'...'** are very important here to get into the folder):

```
cd '...\<FolderName>\'
```
- 5) Create the environment for MRCNN with Anaconda based on the **<FileName>.yml** file:

```
conda env create -f <FileName>.yml
```
- 6) Check the list of environments for the installed environment to **verify** whether the installation worked correctly. The environment name is the **<FileName>** of the used **YML** file.

```
conda env list
```
- 7) Sometimes it is helpful to restart the source code editor or the computer after installation.
- 8) For further assistance, visit the Managing Environments **Tutorial**:
<https://docs.conda.io/projects/conda/en/4.6.1/user-guide/tasks/manage-environments.html#create-env-file-manually>

1.6 Assigning Anaconda environment

- 1) Open the **Mask_RCNN.code-workspace** file located in the downloaded MRCNN repository with the Visual Studio Code.
- 2) Depending on your VSC version, there are several ways to assign the installed Anaconda environment to the MRCNN Python scripts:
 - a) Specify the Python default interpreter path for your workspace:
Press **"Ctrl + Shift +P"** → enter **"Preferences: Open User Settings"** → navigate to **"Workspace → Extensions → Python"** → enter the **Default**

Interpreter Path of the Anaconda environment (e.g.:

"C:\Anaconda3\envs\env_mrcnn_windows_gpu_nvidia\python.exe").

b) Assign the Anaconda environment to the MRCNN Python scripts individually:

i) Navigate to the **MRCNN (WORKSPACE)** in the explorer on the left of VSC, navigate to **classes/droplet/**, and open the ***train_droplet.py*** script.

ii) Assign the Anaconda environment to the ***train_droplet.py*** script:

Press "**Ctrl + Shift + P**" → enter "**Python: Select Interpreter**" → select the installed Anaconda environment (e.g.: **Python 3.6.10**

(env_mrcnn_windows_gpu_nvidia))

iii) Repeat steps 2-3 if necessary for the scripts **proces_manual_droplet.py** and **process_automated_droplet.py**

1.7 VGG Image Annotator

VGG Image Annotator (VIA) is used to manually mark the droplets in images to create a **JSON** file that contains the marked droplets as coordinates. This **JSON** file and the corresponding images are used to train and test the MRCNN. In our work, we use VIA version 1.0.6. You can open this VIA version under the following link and use it in your internet browser:

<https://www.robots.ox.ac.uk/~vgg/software/via/via-1.0.6.html>

For further assistance and information, visit:

<https://www.robots.ox.ac.uk/~vgg/software/via/>

1.8 Weights & Biases (optional)

Weights & Biases is used to track the sensitivity analysis (see our paper) and to get an overview of the performance of the trained models. Weights & Biases is not necessary but useful, e.g., to easily compare the model performance to each other. You can create a Weights & Biases account under the following link:

<https://wandb.ai/site>

For further assistance and information, visit:

<https://docs.wandb.ai/>

2 Train MRCNN models

2.1 Preparing training/validation and testing sets

1) Important information before procedure:

- If you need to crop the images, do it **before** manually evaluating them with the VIA (section 2.1). You can use the **contrast_normalization.py** script to crop images if required. The script is located in the following folder of the MRCNN repository:
.../pre_processing/contrast_normalization/
- The pre-processing method **CLAHE** can also be applied **after** evaluating the images manually with the VIA (section 2.1).
- **Training/validation images should not exceed a size of approx. 1MB.** Otherwise, problems may occur during training depending on the GPU used due to excessive data volume (allocation of data for several epochs on GPU memory). This point does not apply to the images you want to evaluate with the MRCNN model.
- The image resolution does not have to be the same within an image quality
- If you want to train MRCNN on **several image qualities**, e.g., for MRCNN generalization, individual data sets must be created (section 2.1), **one training/validation and one testing data set for each image quality.** Moreover, each training/validation data set must contain the same number of images.
- The number of images within a **training/validation data set** multiplied with the MRCNN parameter dataset_quantity (s. Table 2) and divided by the MRCNN parameter k_fold (s. Table 2) must result in a whole number because the number of images in each fold must correspond to a whole number and must be the same size.

- 2) Split images into a **training/validation** and a **testing** data set, e.g., 90% of images to training/validation data set and 10% to testing data set, and store the data sets in respective folders, e.g., a training and a testing folder. If you want to train MRCNN on **several image qualities**, individual data sets must be created, **one training/validation and one testing data set for each image quality.**

- 3) Manually evaluate the images of each training/validation and testing data set using the VIA program. Proceed as follows for each individual data set:
 - a) Open the VIA program in the internet browser and load the images of the data set via the "**Image**" → "**Load or Add Images**" tab.
 - b) Use the appropriate "**Region Shape**" to mark the object to be trained (e.g., drops) in each image.
 - c) Save the manual evaluation under the "**Annotation**" → "**Save as JSON**" tab respectively as **train.json** or **test.json** file in the corresponding folder.
- 4) If you want to use the **pre-processing method CLAHE** on your training/validation dataset, you have to process the images at this point as follows. Otherwise, skip this step.
 - a) Copy the training/validation images to the downloaded MRCNN repository under the following path:
`.../pre_processing/contrast_normalization/input/`
 - b) Open the **contrast_normalization.py** script and specify the script parameter. Run the script. The processed output images are saved in the following folder:
`.../pre_processing/contrast_normalization/output/`
- 5) Store the training files (training/validation images and train.json files, see chapter 2.1) in the downloaded MRCNN repository under the following paths, depending on the number of image qualities i:
 - `.../datasets/input/<FolderName>/original/Quality_1`
 - `.../datasets/input/<FolderName>/original/Quality_2`
 - ...
 - `.../datasets/input/<TrainValidationFolderName>/original/Quality_i`

Each "Quality_i" folder must contain the same number of images and one corresponding **JSON** file.

If you have only one image quality, you have only one training/validation image set and thus only the folder "Quality_1".

If you use the CLAHE, the paths change as follows:

`.../datasets/input/<FolderName>/clahe/Quality_i`

- 6) The manually evaluated testing files (testing images and **test.json** files) are used in section **Error! Reference source not found..**

2.2 Preparing Weights & Biases project (optional)

Visit your Weights & Biases account and create a new project. The **YML** environment files in our MRCNN repository include all necessary Weights & Biases packages.

2.3 MRCNN training

- 1) Open the **Mask_RCNN.code-workspace** file located in the downloaded MRCNN repository with the Visual Studio Code.
- 2) Navigate to the **MRCNN (WORKSPACE)** in the explorer on the left of VSC, navigate to **classes/droplet/**, and open the ***train_droplet.py*** script.
- 3) Specify input training parameters under the "**input training parameters**" section in the ***train_droplet.py*** script. For a description and default settings of the training parameters, see Table 2.
- 4) Click on the "**Terminal**" tab at the top bar of the VSC and open a new terminal via "**New Terminal**".
- 5) Click on the "**Run**" tab at the top bar of the VSC and execute the script via "**Run Without Debugging**".
- 6) As soon as the script is finished, the output files of the MRCNN training are saved in the following folder:

.../models/<WeightsFolderName>/

The output files include an Excel file and the weights of each epoch in **H5** file format. The Excel file lists the distribution of the input images to the folds. Each epoch corresponds to one MRCNN model.

3 Image evaluation

3.1 Preparing data

- 1) For image processing with an MRCNN model, store the images you want to evaluate, e.g., images of the testing data set, in the following folder:
...\\datasets\\input\\<EvaluationFolderName>
- 2) For processing images from the manual evaluation with VIA, store the images of the testing data set with the corresponding **JSON** file in the following folder:
...\\datasets\\input\\<EvaluationFolderName>

3.2 Processing images with MRCNN model

- 1) Open the **Mask_RCNN.code-workspace** file located in the downloaded MRCNN repository with the Visual Studio Code.
- 2) Navigate to the **MRCNN (WORKSPACE)** in the explorer on the left of VSC, navigate to **classes/droplet/**, and open the ***process_automated_droplet.py*** script.
- 3) Specify input processing parameters under the "**input processing parameters**" section in the ***process_automated_droplet.py*** script. For a description and default settings of the processing parameters, see Table 3.
- 4) Click on the "**Terminal**" tab at the top bar of the VSC and open a new terminal via "**New Terminal**".
- 5) Click on the "**Run**" tab at the top bar of the VSC and execute the script via "**Run Without Debugging**".
- 6) As soon as the script is finished, the output files of the MRCNN training are saved in the following folder:

...\\datasets\\output\\<OutputFolderName>/

The output files include an Excel file and the evaluated images. The Excel file lists the sizes of the detected droplets. The detected droplets are marked in the evaluated images.

3.3 Processing images from the manual evaluation with VIA

- 7) Open the **Mask_RCNN.code-workspace** file located in the downloaded MRCNN repository with the Visual Studio Code.
- 8) Navigate to the **MRCNN (WORKSPACE)** in the explorer on the left of VSC, navigate to **classes/droplet/**, and open the ***process_manual_droplet.py*** script.
- 9) Specify input processing parameters under the "**input processing parameters**" section in the ***process_automated_droplet.py*** script. For a description and default settings of the processing parameters, see Table 3.
- 10) Click on the "**Terminal**" tab at the top bar of the VSC and open a new terminal via "**New Terminal**".
- 11) Click on the "**Run**" tab at the top bar of the VSC and execute the script via "**Run Without Debugging**".
- 12) As soon as the script is finished, the output files of the MRCNN training are saved in the following folder:

.../datasets/output/<OutputFolderName>/

The output files include an Excel file and the evaluated images. The Excel file lists the sizes of the droplets from the manual evaluation with VIA. The droplets from the manual evaluation with VIA are marked in the evaluated images.

3.4 Evaluating MRCNN detection performance

To evaluate the detection performance of an MRCNN model, you compare the results of the image processing with the MRCNN model (excel output file and evaluated images from section 3.2) with the results of image processing from the manual evaluation with VIA (excel output file and evaluated images from section 3.2). For a qualitative evaluation, you compare the images. For a quantitative evaluation, the Excel output files. This procedure is performed for all epochs/models of the training with a specific training and processing parameter set to identify the epoch/model with the best detection performance. Analogously, the entire process (training, processing, and evaluation) is carried out with modified training and processing parameters to identify the most suitable parameters. For further information, please read our papers.

4 Appendix

4.1 Description of MRCNN repository folders and files

Table 1: Description of MRCNN repository folders and files

Folder/file name	Description
classes	Folders for classes, e.g., droplet, are located here
.../droplet	Script files for the class droplet are located here
.../.../train_droplet.py	Script file for training the MRCNN on the class droplet
.../.../process_automated_droplet.py	Script file for automated processing of images with the trained MRCNN to detect the class droplet
.../.../process_manual_droplet.py	Script file for manual processing of images with marked droplets used the VGG Image Annotator
database	This folder is an archive folder only. The MRCNN has no access to this folder during the training. Databases for MRCNN training are located in a different folder. Here, each database folder contains a ZIP file with images of droplets and the corresponding JSON file with the manually marked droplets. In this folder, you can find the database used in our work/publications. Moreover, you can upload your database here to share it with other users.
datasets	Input files for the MRCNN training as well as input and output files for the MRCNN processing are located here.
.../input	<p>Input files (images and corresponding JSON file) for the MRCNN training and processing are located here. The input files must be stored in a folder. The folders must have the following folder structure.</p> <p><u>For training</u>, depending on the number of image qualities <i>i</i>:</p> <ul style="list-style-type: none">- datasets/input/<FolderName>/original/Quality_1/- datasets/input/<FolderName>/original/Quality_2/- <p>Each “Quality_” folder must contain the same number of images and one, corresponding JSON file. The number of images multiplied with the MRCNN parameter dataset_quantity (s. Table 2) and divided by the MRCNN parameter k_fold (s. Table 2) must result in an integer (whole number). If the CLAHE is used the folder structure changes to:</p>

	<p>- datasets/input/<FolderName>/clahe/Quality_1/</p> <p>- ...</p> <p><u>For processing:</u></p> <p>datasets/input/<FolderName>/</p> <p>For manual processing, folder must contain images and one, corresponding JSON file. For automated processing, folder must contain images.</p>
.../input	Contains folders with processing results. The folder names are defined in the corresponding .py script by the user.
environment	Contains the environment .yml files suitable for the operating system Windows and Linux and calculation devices CPU and GPU (AMD and Nvidia).
.../env_mrcnn_linux_cpu.yml	Environment for Linux and CPU
.../env_mrcnn_linux_gpu.yml	Environment for Linux and GPU
.../env_mrcnn_windows_cpu.yml	Environment for Windows and CPU
.../env_mrcnn_windows_gpu_amd.yml	Environment for Windows and AMD GPU
.../env_mrcnn_windows_gpu_nvidia.yml	Environment for Windows and Nvidia GPU
manual	Manuals are located here. Moreover, example JOB file for clusterjob on RWTH HPC cluster is located here.
models	The trained MRCNN models are saved and stored here.
.../benchmark	MRCNN benchmark models trained on one specific image quality are located here.
.../coco	MRCNN model trained on the COCO database is located here.
.../generalization	MRCNN generalized models trained on several image qualities are located here.
mrcnn	MRCNN architecture is located in this folder.
.../__init__.py	-
.../config.py	Base configuration class of MRCNN. All MRCNN training parameter are located here.
.../model.py	The main MRCNN model implementation.
.../parallel_model.py	Multi-GPU Support for Keras.
.../utils.py	Common utility functions and classes.
.../visualize.py	Display and visualization functions.
pre_processing	Consists separate scripts for pre-preprocessing, e.g., the CLAHE
LICENSE	License.
MRCNN.code-workspace	Workspace.

4.2 MRCNN training and processing parameters

Table 2: MRCNN training parameters

Parameter	Description	Default value (best practice)
Cluster = <Boolean>	Is the script executed on the cluster? E.g., RWTH High Performance Computing cluster? True = yes, False = no	False
file_format = <FileFormat>	File format of images	"jpg"
dataset_path = <InputFolderName>	Input dataset folder located in path: "...\\datasets\\input\\..."	"test"
new_weights_path = <WeightsFolderName>	output weights folder located in path: "...\\models\\... Weights of the individual epochs = MRCNN models	"test"
name_result_file = <ExcelFileName>	Name of the excel output file located in path: "...\\models\\<WeightsFolderName>\\". This excel file shows the distribution of the input images to the folds.	"test"
masks = <Boolean>	Generate detection masks? True = yes, False = no	False
device = <Boolean>	Use GPU or CPU? True = GPU, False = CPU	True
epochs = <NumberEpochs>	Epochs to train	50
early_stopping = <Integer>	Should early stopping be used? 0 = no, otherwise value is number of epochs without improvement	0
early_loss = <LossName>	Loss monitored by early stopping	"val_loss"
base_weights = <WeightsName>	Base weights the training starts from	"coco"
dataset_quantity = <QuantityDataset>	Percentage of the training dataset to be used for training [%], e.g., to determine minimum required number of images in training/validation set for accurate detection performance	100
use_wandb = <Boolean>	Use Weights & Biases to track training data? True = yes, False = no	False
wandb_entity = "test"	Enter W&B entity name (check projects in Weights & Biases, entity	"test"

	name is next to the project name)	
wandb_project = "test"	Enter W&B project name	"test"
wandb_group = "test"	Enter group name within the W&B project	"test"
wandb_name = "test"	Enter run name within the group of W&B project	"test"
cross_validation = <Boolean>	Perform a k-fold cross-validation? If you want to train the final models, choose False. True = yes, False = no	True
k_fold = <NumberFolds>	Number of folds for k-fold cross validation	5
k_fold_val = <FoldNumber>	Validation fold. Starting with 0. The remaining folds are training folds	0
backbone_type = <Integer>	backbone (see BACKBONE parameter in config.py). 0 = "resnet50", 1 = "resnet101"	0
train_all_layers = <Boolean>	Which layers should be trained? True = train all layers, False = train only heads	True
images_gpu = <NumberImages>	Number of images used to train the model on each GPU. If only one GPU is used, this parameter is equivalent to batch size (see BATCH_SIZE parameter in config.py). A 12GB GPU can typically handle 2 images of 1024x1024px resolution. Adjust this parameter based on your GPU memory and image resolution.	1
learning = <Integer>	Learning rate (see LEARNING_RATE parameter in config.py). 0 = 0.01, 1 = 0.001, 2 = 0.0001	1
image_max = <Integer>	Image resolution (see IMAGE_MAX_DIM parameter in config.py). Select the closest value corresponding to the largest side of the image. 0 = 512, 1 = 1024, 2 = 2048	1
momentum = <Integer>	Learning momentum (see LEARNING_MOMENTUM parameter in config.py).	1

	0 = 0.8, 1 = 0.9, 2 = 0.99	
w_decay = <Integer>	Weight decay (see WEIGHT_DECAY parameter in config.py). 0 = 0.0001, 1 = 0.001, 2 = 0.01	0
augmentation = <Boolean>	Use augmentation methods? True = yes, False = no	False
flip = <Integer>	Use augmentation method flip? 0 = no, 1 = (0.5, 0.5)	0
cropandpad = <Integer>	Use augmentation method crop? 0 = no, 1 = (-0.25, 0), 2 = (-0.1, 0)	0
rotate = <Integer>	Use augmentation method rotate? 0 = no, 1 = (-45, 45), 2 = (-90, 90)	0
noise = <Integer>	Use augmentation method additive Gaussian noise? 0 = no, 1 = 0.01, 2 = 0.02	0
gamma = <Integer>	Use augmentation method gamma contrast? 0 = no, 1 = yes	0
contrast = <Integer>	Use contrast adjustment? 0 = no, 1 = contrast limited adaptive histogram equalization, 2 = contrast stretching	0

Table 3: MRCNN processing parameters

Parameter	Description	Default value (best practice)
Cluster = <Boolean>	Is the script executed on the cluster? E.g., RWTH High Performance Computing cluster? True = yes, False = no	False
file_format = <FileFormat>	File format of images	"jpg"
dataset_path = <InputFolderName>	Input dataset folder located in path: "...\\datasets\\input\\..."	"test"
save_path = <OutputFolderName>	Output images folder located in path: "...\\datasets\\output\\..."	"test"
name_result_file = <ExcelFileName>	Name of the excel output file located in path: "...\\models\\<WeightsFolderName>\\". This excel file shows the distribution of the input images to the folds.	"test"
weights_path = <WeightsFolderName>	Folder of the MRCNN model located	"test"

	in: "...models\..." Weights of the individual epochs = MRCNN models	
weights_name = <ModelName>	MRCNN model name located in path: "models\"	"test"
masks = <Boolean>	Generate detection masks? True = yes, False = no	False
device = <Boolean>	Use GPU or CPU? True = GPU, False = CPU	True
save_nth_image = <Integer>	Save n-th result image	1
pixelsize = <Double>	Pixel size in [µm/px]. To read from Sopat log file enter pixelsize = 0 (Sopat generates a JSON file with including information)	1
image_crop = <Coordinates>	Do you want the image to be center cropped before detection? no = None, yes = (x, y) coordinates (e.g.: image_crop = (1000, 1500))	None
images_gpu = <NumberImages>	Number of images used to evaluate with the MRCNN model on each GPU. If only one GPU is used, this parameter is equivalent to batch size (see BATCH_SIZE parameter in config.py). A 12GB GPU can typically handle 2 images of 1024x1024px resolution. Adjust this parameter based on your GPU memory and image resolution.	1
image_max = <Integer>	Image resolution (see IMAGE_MAX_DIM parameter in config.py). Select the closest value corresponding to the largest side of the image. 0 = 512, 1 = 1024, 2 = 2048	1
confidence = <Double>	Skip detections with confidence < specified confidence parameter value	0.7
detect_reflections = <Boolean>	Detect and mark reflections in droplets? The detected reflections are excluded from the evaluation and do not appear in the excel output file. Marking color is blue.	False

	True = yes, False = no	
detect_oval_droplets = <Boolean>	<p>Detect and mark oval droplets?</p> <p>The detected oval droplets are excluded from the evaluation and do not appear in the excel output file.</p> <p>Marking color is red.</p> <p>True = yes, False = no</p>	True
min_aspect_ratio = <Double>	Minimum aspect ratio: filter for elliptical shapes	0.9
detect_adhesive_droplets = <Boolean>	<p>Detect and mark adhesive droplets?</p> <p>The detected adhesive droplets are excluded from the evaluation and do not appear in the excel output file.</p> <p>Marking color is orange.</p> <p>True = yes, False = no</p>	False
save_coordinates = <Boolean>	Save coordinates of detected adhesive droplets in excel output file? True = yes, False = no	False
min_velocity = <Double>	Minimum velocity: threshold to filter adhesive droplets minimum distance [% of droplet mean diameter] that a droplet has to travel between 2 frames	0.3
min_size_diff = <Double>	Minimum size difference: threshold to filter adhesive droplets [%] to be considered a different droplet	0.3
n_images_compared = <Integer>	Number of images that are being compared. This is necessary because adhesive droplets may not get detected every frame.	3
n_adhesive_high = <Integer>	Number of times a droplet has to be detected at a similar position to be defined as adhesive.	3
n_adhesive_low = <Integer>		2
low_distance_threshold = <Double>		0.05
edge_tolerance = <Double>	Edge threshold: filter for image border intersecting droplets. Image border intersecting droplets are marked in color red.	
contrast = <Integer>	Use contrast adjustment? 0 = no, 1 = contrast limited adaptive histogram equalization, 2 = contrast stretching	0

