

Manual for MRCNN (cluster users)

Version: 13.11.2023

Editor: Stepan Sibirtsev

This manual works for **cluster users**. If you want to run MRCNN on a cluster, this manual guides you through the **installation and application of MRCNN**. If you're going to run MRCNN on your personal or office computer, please check the manual for non-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 **Linux** as an operating system and **RWTH HPC** as cluster. However, if the cluster you use applies the SLURM as workload manager/job scheduler, these instructions are easy to adapt to the cluster you use.

Table of Contents

1	Installation	3
1.1	Connection to cluster	3
1.1.1	Win SCP	3
1.1.2	FastX	3
1.2	MRCNN repository	4
1.3	Anaconda	4
1.4	Anaconda environment	5
1.5	VGG Image Annotator	5
1.6	Weights & Biases (optional)	6
2	Train MRCNN models	6
2.1	Preparing training/validation and testing data sets	6
2.2	Preparing Weights & Biases project (optional)	8
2.3	Testing MRCNN execution	8
2.4	Creating a cluster job	9
2.5	Submitting a cluster job	10
3	Image evaluation	10
3.1	Preparing data	10
3.2	Creating a cluster job	11
3.3	Submitting a cluster job	11
3.4	Evaluating MRCNN detection performance	12
4	Appendix	13
4.1	Description of MRCNN repository folders and files	13
4.2	Terminal commands	15
4.3	SLURM and job parameters	15
4.4	MRCNN training and processing parameters	16

1 Installation

1.1 Connection to cluster

Get an HPC cluster account: <https://help.itc.rwth-aachen.de/service/rhr4fjjuttf/article/14573fc745ee478ba855539c240108b6/>

There are several ways to connect and communicate with the cluster. I prefer **WinSCP** for remote file transfer and **FastX** as desktop client for job submission etc.

1.1.1 Win SCP

- 1) Visit the **WinSCP** download page, download and install WinSCP:
<https://winscp.net/eng/download.php>
- 2) WinSCP setup/preparation:
 - a) Host name: e.g., login18-x-1.hpc.itc.rwth-aachen.de
 - b) Port number: 22
 - c) User name: e.g., ab123456 (TIM identifier)
 - d) Password: your RWTHonline password
- 3) For further assistance, visit WinSCP **Tutorial**:
<https://help.itc.rwth-aachen.de/service/sl0p0u7tdi5s/article/9229ea232b5f4a1c99ddd57c07a929cb/>

1.1.2 FastX

- 1) Visit the **FastX** download page, download and install FastX:
<https://www.starnet.com/download/fastx-client>
- 2) FastX setup/preparation:
 - a) Host: e.g., login18-x-1.hpc.itc.rwth-aachen.de
 - b) User: e.g., ab123456 (TIM identifier)
 - c) Port number: 22
 - d) Name: any name
 - e) Password: your RWTHonline password
- 3) For further assistance, visit WinSCP **Tutorial**:
<https://help.itc.rwth-aachen.de/service/rhr4fjjuttf/article/25f576374f984c888bb2a01487fef193/>

1.2 MRCNN repository

- 1) Download **all folders and files** from the MRCNN repository. Connect to the cluster via **WinSCP** and transfer the downloaded MRCNN repository to a folder of your choice on the cluster. You can find some description of the repository folders and files in Table 1.
- 2) Navigate to the path "...\\samples\\droplet" and open ***the train_droplet.py*** script. Set the variable cluster to True (**cluster=True**), save and close the script.
- 3) Repeat step 2 for the scripts ***process_automated_droplet.py*** and ***proces_manual_droplet.py***

1.3 Anaconda

- 1) Connect to the cluster via **FastX** (Launch Session → XFCE)
- 2) Visit the **Anaconda** download page on the cluster:
<https://www.anaconda.com/download>
- 3) Select the **Linux** version on the download page and copy the download link.
- 4) Launch **terminal**, navigate to the home directory, create a temporary (tmp) folder and navigate to it:

```
$ cd ~
```

```
$ mkdir tmp
```

```
$ cd tmp
```

- 5) Download Anaconda bash script into the tmp folder using the copied Anaconda download link:

```
$ wget https://repo.continuum.io/archive/Anaconda3<Release>.sh
```

- 6) Navigate to the folder where the bash script was saved and install Anaconda using the downloaded bash script:

```
$ ls Anaconda3<Release>.sh (it's a small L, not an I)
```

```
$ bash Anaconda3<Release>.sh
```

- 7) Restart terminal and verify the installation:

```
$ cd ~
```

```
$ source .bashrc
```

```
$ python
```

- 8) If the Anaconda installation was successful, the following is displayed:

```
Python 3.6.5 |Anaconda, Inc.|...
```

9) Exit Python:

```
$ exit()
```

10) For further assistance, visit Installing Anaconda on Linux **Tutorial**:

<https://problemsolvingwithpython.com/01-Orientation/01.05-Installing-Anaconda-on-Linux/>

1.4 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 calculation device (GPU and CPU) located in the **environment folder of the MRCNN repository**.

- 1) Connect to the cluster via **FastX** (Launch Session → XFCE)
- 2) Decide whether you want to run MRCNN on **CPU or GPU**
- 3) Choose the **suitable environment** located in the environment folder of the MRCNN repository:

a) Linux & GPU: env_mrcnn_linux_gpu.yml

b) Linux and CPU: env_mrcnn_linux_cpu.yml

- 4) Navigate to the folder where the **<FileName>.yml** file is stored and open a terminal there (right-click → open terminal here)

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

```
$ conda env create -f <FileName>.yml
```

- 6) Check whether Anaconda environment has been installed:

```
$ conda env list
```

- 7) 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>

1.5 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/>

1.6 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 data sets

The preparation of data sets is performed on your personal or office computer.

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 5**) and divided by the MRCNN parameter `k_fold` (s. **Table 5**) 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) Connect to the cluster via **WinSCP**

6) Store the training files (training/validation images and train.json files) in the downloaded MRCNN repository on the cluster 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

7) The manually evaluated testing files (testing images and **test.json** files) are used in section 3.

2.2 Preparing Weights & Biases project (optional)

This step is performed on your personal or office computer. The **YML** environment files in our MRCNN repository include all necessary Weights & Biases packages.

- 1) Visit your Weights & Biases account and create a new project.
- 2) If you want to perform a **grid search**, create a new “**sweep**”. Otherwise, skip this step. You can find an example for the sweep configuration in the **sweep_example.txt** file located in the templates folder of the downloaded MRCNN repository. For a description and default settings of the training parameters, see Table 5. For further assistance, visit the Seeps **Tutorial**: <https://docs.wandb.ai/guides/sweeps>

2.3 Testing MRCNN execution

We recommend to perform this step before finally submitting the training jobs to the cluster in order to test whether MRCNN is executed on cluster correctly.

- 1) Connect to the cluster via **FastX** (Launch Session → XFCE) and launch a terminal

- 2) Connect to a **GPU node** (if job will be performed on CPU, skip this step):

```
$ ssh -l <UserId> <NodeName>.hpc.itc.rwth-aachen.de
```

(HPC GPU nodes: <https://help.itc.rwth-aachen.de/service/rhr4fjjuttff/article/3fb4cb953142422dbbb656c1c3253cff/>)
- 3) Load the **Cuda** modul (if job will be performed on CPU, skip this step):

```
$ module load cuda/10.0
```
- 4) Export the path in which **Anaconda** is located:

```
$ export PATH=$PATH:/home/<UserID>/anaconda3/bin
```
- 5) Activate **Anaconda environment**:

```
$ source activate env_mrcnn_gpu
```

 (for GPU job) or

```
$ source activate env_mrcnn_cpu
```

 (for CPU job)
- 6) Navigate to the path where the ***train_droplet.py*** script is located:

```
$ cd /home/<UserID>/.../samples/droplet/
```
- 7) Execute the ***train_droplet.py*** script with memory tracker:

```
$ r_memusage python train_droplet.py --  
dataset_path=<TrainValidationFolderName> --file_format=<FileFormat> --  
image_max=<Integer> --images_gpu=<Integer> --device=<Boolean>
```

Enter command in one piece. These are the required training parameters to be specified. Description/default settings of all training parameters see **Table 5**.
- 8) Cancel after **10 minutes** with “**Ctrl + Shift + C**”
- 9) Round up the entry for “**peak usage: physical memory**” to gigabytes and the calculation time per epoch to minutes, and remember/note the values.

2.4 Creating a cluster job

- 1) Create a **<FileName>.job** file on your personal or office computer and open it with Notepad, Visual Studio Code or an alternative text editor. You can find **examples and templates** of the **JOB** files in the templates folder of the downloaded MRCNN repository.
- 2) Change the **<FileName>.job** file to Unix conversion. Otherwise, the cluster has troubles to handle the file.
In Notepad click on Edit → EOL Conversion → Unix (LF).
- 3) Specify the **#SBATCH, job and training parameters** of the job (for description/default settings of the parameters, see **Table 3, Table 4** and **Table 5**). Note the **required training parameters to be specified**.

- 4) Save the **<FileName>.job** file.
- 5) Connect to the cluster via **WinSCP** and transfer the **<FileName>.job** file to a folder of your choice on the cluster.

2.5 Submitting a cluster job

- 1) Connect to the cluster via **FastX** (Launch Session → XFCE)
- 2) Navigate to the folder where the **<FileName>.job** file is stored and open a terminal there (right-click → open terminal here)
- 3) Submit the job:

```
$ sbatch <FileName>.job
```
- 4) Check whether the job has arrived in the queue:

```
$ squeue -u <UserID>
```
- 5) As soon as the job starts, you will receive a notification by email.
- 6) As soon as the job is finished or is terminated due to an error, you will receive a notification by email. The **<FileName>.txt** job output file with the documented job history (terminal output of the MRCNN) is saved in the job output file folder specified in the **<FileName>.job** file:

/home/<UserID>/.../<JobOutputFolderName>/

In case of a job termination, the **<FileName>.txt** job output file can give you information about the occurring errors.

- 7) As soon as the job is finished, the output files of the MRCNN training are saved in the following folder:

.../droplet_logs/<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) Connect to the cluster via **WinSCP**
- 2) 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 on the cluster:
...\\datasets\\input\\<ProcessingFolderName>

- 3) 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 on the cluster:

...\datasets\input\<ProcessingFolderName>

3.2 Creating a cluster job

- 6) Create a **<FileName>.job** file on your personal or office computer and open it with Notepad, Visual Studio Code or an alternative text editor. You can find **examples and templates** of the **JOB** files in the templates folder of the downloaded MRCNN repository.
- 7) Change the **<FileName>.job** file to Unix conversion. Otherwise, the cluster has troubles to handle the file.
In Notepad click on Edit → EOL Conversion → Unix (LF).
- 8) Specify the **#SBATCH, job and processing parameters** of the job (for description/default settings of the parameters, see **Table 3, Table 4** and **Table 6**). Note the **required training parameters to be specified**. For processing images with MRCNN model choose the ***process_automated_droplet.py*** script. For processing images from the manual evaluation with VIA choose the ***process_manual_droplet.py*** script.
- 9) Save the **<FileName>.job** file.
- 10) Connect to the cluster via **WinSCP** and transfer the **<FileName>.job** file to a folder of your choice on the cluster.

3.3 Submitting a cluster job

- 8) Connect to the cluster via **FastX** (Launch Session → XFCE)
- 9) Navigate to the folder where the **<FileName>.job** file is stored and open a terminal there (right-click → open terminal here)
- 10) Submit the job:

```
$ sbatch <FileName>.job
```
- 11) Check whether the job has arrived in the queue:

```
$ squeue -u <UserID>
```
- 12) As soon as the job starts, you will receive a notification by email.

13) As soon as the job is finished or is terminated due to an error, you will receive a notification by email. The **<FileName>.txt** job output file with the documented job history (terminal output of the MRCNN) is saved in the job output file folder specified in the **<FileName>.job** file:

/home/<UserID>/.../<JobOutputFolderName>/

In case of a job termination, the **<FileName>.txt** job output file can give you information about the occurring errors.

14) As soon as the job 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.

3.4 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 are compared. 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. You can use the information about the MRCNN training at Weights & Biases to get an overview of the training performance. 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:</p> <ul style="list-style-type: none">- datasets/input/<FolderName>/original/Quality_1/- datasets/input/<FolderName>/original/Quality_2/- <p>Each “Quality_i” 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 5) and divided by the MRCNN parameter k_fold (s. Table 5) 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.
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
templates	Templates and examples are located here, e.g., JOB file for jobs on the RWTH HPC cluster.
LICENSE	License.
MRCNN.code-workspace	Workspace.

4.2 Terminal commands

Table 2: Terminal commands

Command	Description
\$ r_memusage <PythonScript>.py To abort: Shift +Ctrl + C	Starts the Python script and returns the amount of memory used after it was aborted
\$ sbatch <JobScript>.job	Submit job
\$ squeue -u <UserID>	Displays running jobs and jobs in the user's queue
\$ scancel <JobID>	Cancel job
\$ sinfo	Displays cluster status with the list of running jobs and jobs in the queue
\$ r_wlm_usage -q	Display core hours consumption

4.3 SLURM and job parameters

Table 3: SLURM parameters (write #SBATCH before each parameter)

Parameter	Description	Default, example or best practice value
#!/usr/bin/zsh	Must appear at the beginning of every job script.	
--account=<ProjectName>	Submitting the job via a project	-
--job-name=<JobName>	Job name	test
--output=/home/<UserID>/.../%x_%J_output.txt	Storage location of the job execution output file. "%x" takes the name of the file. "%J" adds the job ID	-
--time=d-hh:mm:ss	Duration of the job in days-hours:minutes:seconds	0-03:00:00
--mail-user=<EmailAdress>	Email address to receive information about job progress	-
--mail-type=BEGIN --mail-type=END --mail-type=ALL	Information about job progress that you would like to receive by email	ALL
--ntasks=<NumberTasks>	For Processes/MPI. Number of tasks	1
--mem-per-cpu=<AmountMemory>	Memory requirement per CPU (not per task!), e.g. 10 G. K(ilobyte) G(igabyte) T(erabyte) Specify only for CP jobs.	5

--mem-per-gpu=<AmountMemory>	Memory requirement per GPU (not per task!), e.g., 10 G. K(ilobyte) G(igabyte) T(erabyte) Specify only for GPU jobs.	5
--gres=gpu:1 --gres=gpu:2	Number of GPUs required per node. The partition is automatically selected by the SLURM system. Specify only for GPU jobs.	1
--array=x-y	Definition of the array job, starting at 0	0-4

Table 4: Job parameters

Parameter	Description
export PATH=\$PATH:/home/<UserID>/anaconda3/bin	Export path in which Anaconda is located
source activate <EnvironmentName>	Activate environment
cd /home/<UserID>/.../samples/droplet/	Navigate to the path where the droplet.py script is located
python <ScriptName>.py --<ScriptParameterName1>=<ScriptParameterValue1> --<ScriptParameterName2>=<ScriptParameterValue2> --<ScriptParameterName3>=<ScriptParameterValue3> ...	Run a Python script with specified script parameter. The Python script is the train_droplet.py , proces_manual_droplet.py or process_automated_droplet.py script, depending on the job. The script parameters are training or processing parameters depending on the script, see Table 5 and Table 6. Write -- before each script parameter.

4.4 MRCNN training and processing parameters

Table 5: MRCNN training parameters (write -- before each parameter)

Parameter	Description	Default, example or best practice value
required training parameters to be specified		
cluster=<Boolean>	Is the script executed on the cluster? E.g., RWTH High Performance Computing cluster? True=yes, False=no Must be specified directly in the Python	True

	script.	
file_format=<FileFormat>	File format of images	"jpg"
dataset_path=<InputFolderName>	Input dataset folder located in path: "...\\datasets\\input\\..."	"test_input"
new_weights_path=<WeightsFolderName>	output weights folder located in path: "...\\models\\... Weights of the individual epochs=MRCNN models	"weights"
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.	"folds"
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 name is next to the project name)	"test_entity "
wandb_project="test"	Enter W&B project name	"test_project"
wandb_group="test"	Enter group name within the W&B project	"test_group"
wandb_run="test"	Enter run name within the group of W&B project	"test_run"
cross_validation=<Boolean>	Perform a k-fold cross-validation? If you want to train the final models, choose	False

	False. True=yes, False=no	
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). 0=0.8, 1=0.9, 2=0.99	1
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

	0=no, 1=(-0.25, 0), 2=(-0.1, 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 6: MRCNN processing parameters (write -- before each parameter)

Parameter	Description	Default, example or best practice value
required training parameters to be specified		
cluster=<Boolean>	Is the script executed on the cluster? E.g., RWTH High Performance Computing cluster? True=yes, False=no Must be specified directly in the Python script.	True
file_format=<FileFormat>	File format of images	"jpg"
dataset_path=<InputFolderName>	Input dataset folder located in path: "...\\datasets\\input\\..."	"test_input"
save_path=<OutputFolderName>	Output images folder located in path: "...\\datasets\\output\\..."	"test_output"
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.	"DSD"
weights_path=<WeightsFolderName>	Folder of the MRCNN model located in: "...\\models\\..." Weights of the individual epochs=MRCNN models	"weights" See training output
weights_name=<ModelName>	MRCNN model name located in path: "models\\<WeightsFolderName>\\"	"weights_name" See training output

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 [$\mu\text{m}/\text{px}$]. To read the pixel size value from Sopats specifications log file enter pixelsize=0 (Sopat generates a JSON file with including information)	Check camera specifications
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	Should be the same value as for training
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. True=yes, False=no	False
detect_oval_droplets=<Boolean>	Detect and mark oval droplets? The detected oval droplets are excluded from the evaluation and do	False

	not appear in the excel output file. Marking color is red. True=yes, False=no	
min_aspect_ratio=<Double>	Minimum aspect ratio: filter for elliptical shapes	0.9
detect_adhesive_droplets=<Boolean>	Detect and mark adhesive droplets? The detected adhesive droplets are excluded from the evaluation and do not appear in the excel output file. Marking color is orange. True=yes, False=no	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> n_adhesive_low=<Integer> low_distance_threshold=<Double>	Number of times a droplet has to be detected at a similar position to be defined as adhesive.	3 2 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