Searching for the Liquid Crystal sedimentation in Polarized Optical Microscope photos

Szymon Baczyński
KD-158

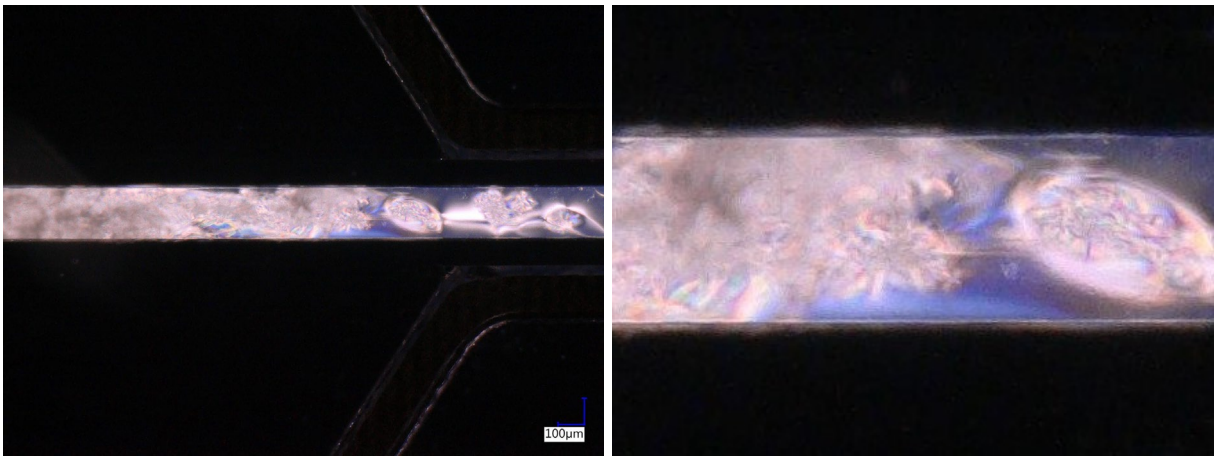
1. Project subject

During my research on my PhD thesis "Optofluidic systems for sensing applications", where I fill channels made in the PDMS (polymer) with various types of liquid crystals. These include the nematics E7, 5CB and 6CHBT. Some kind of sediment was formed during the work with E7. I have carried out many trials that resulted in the sedimentation of the liquid crystal. Due to the long process of sediment formation, one of the ideas for observation and reporting was to create a laboratory stand to collect data over time. In the meantime, many samples were checked, thanks to which I collected a database of several hundred photos. All photos are taken with the Polarized Optical Microscope - a microscope that uses crossed polarizers.

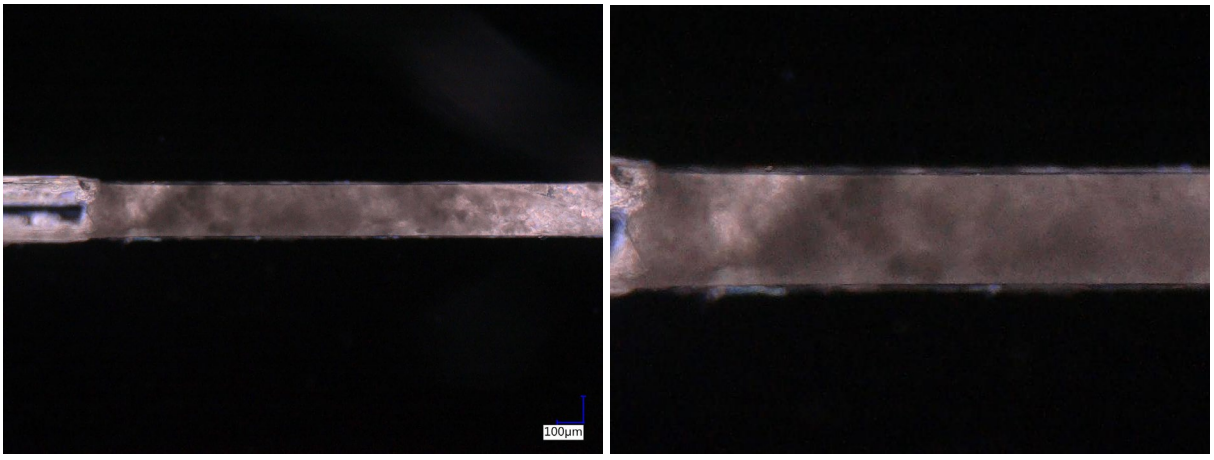
The purpose of this project is an initial review of image recognition models and checking the available tools. Due to little experience in working with image recognition tools and models, preliminary research in this field will be presented. The results and ideas for improving the system will be presented at the end.

2. Input Data

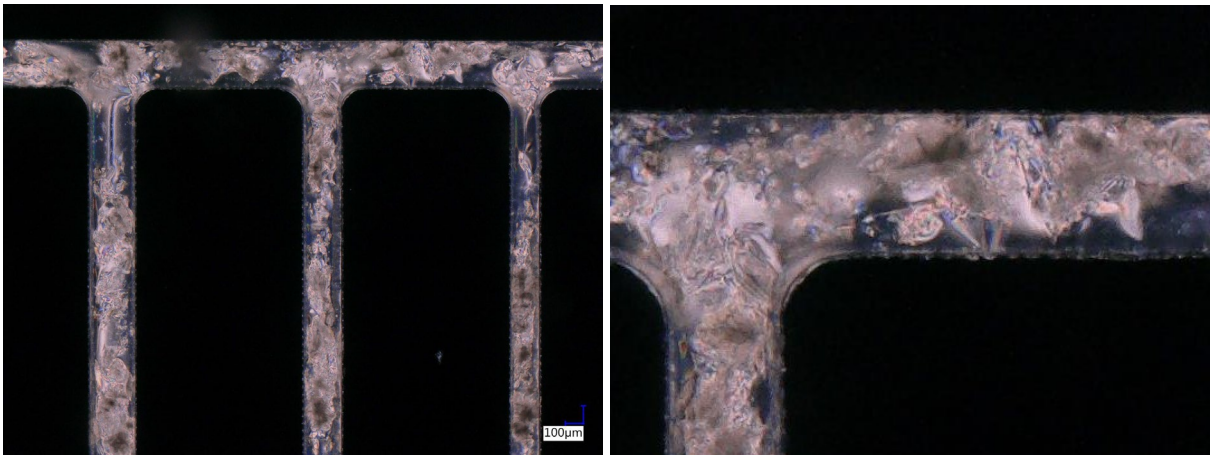
The input data used for training is a collection of 53 photos from different stages of the research. The set includes photos taken with the Keyence VHX5000 microscope. The resolution of a single photo is 1600x1200 pixels, but photos were also taken using the “Stitching function”, thanks to which the images have a higher resolution. Pictures were taken with light passing through crossed polarizers. The areas where there is no liquid crystal (LC) are completely black (isotropic medium). Interesting parts of the photos where liquid crystal sedimentation occurs are gray and have a heterogeneous texture. In addition to photos with channels where we observe a destructive phenomenon, there are also several photos of Liquid Crystal droplets on the PDMS. In this case, the different stages of sedimentation formation can be seen very clearly. Below, there are some sample photos to depict the variety of cases.



Microchannel in PDMS with Liquid Crystal - day 3 (after filling with LC).
On the right, magnification of an area with progressive sedimentation.

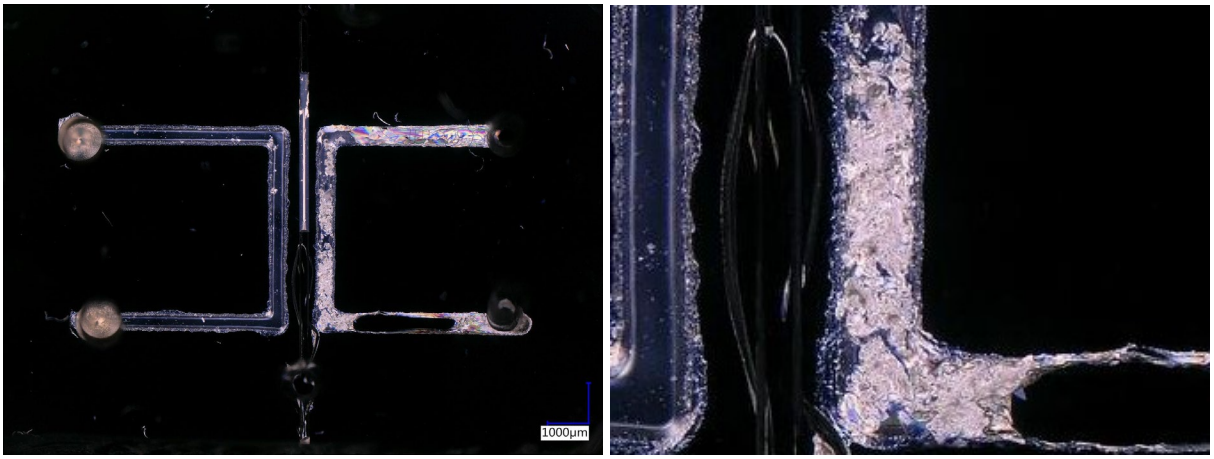


Microchannel in PDMS with Liquid Crystal - day 3 (after filling with LC).
On the right, a magnification of an exemplary area of interest.

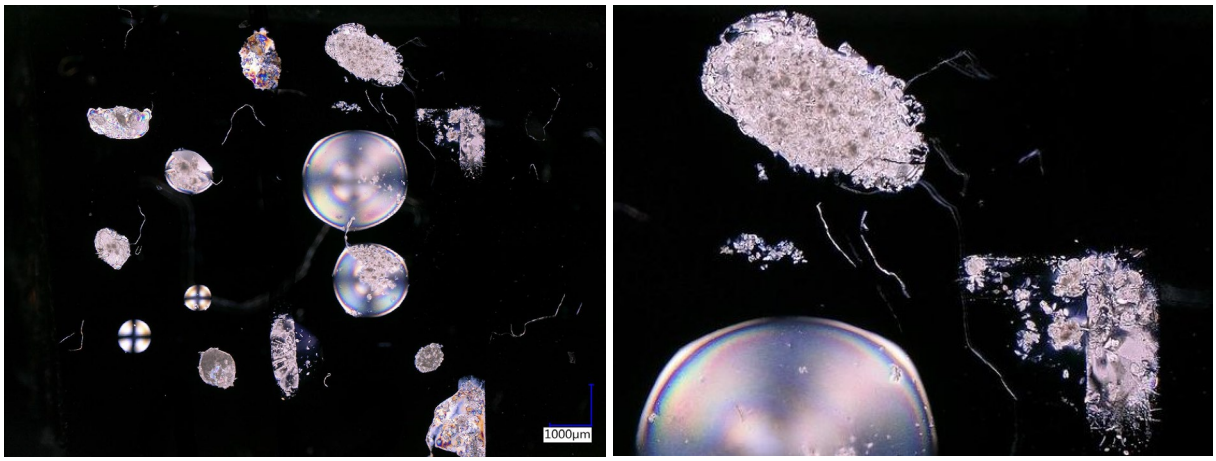


Microchannels in PDMS with Liquid Crystal - day 4 (after filling with LC).
On the right, a magnification of an exemplary area of interest.

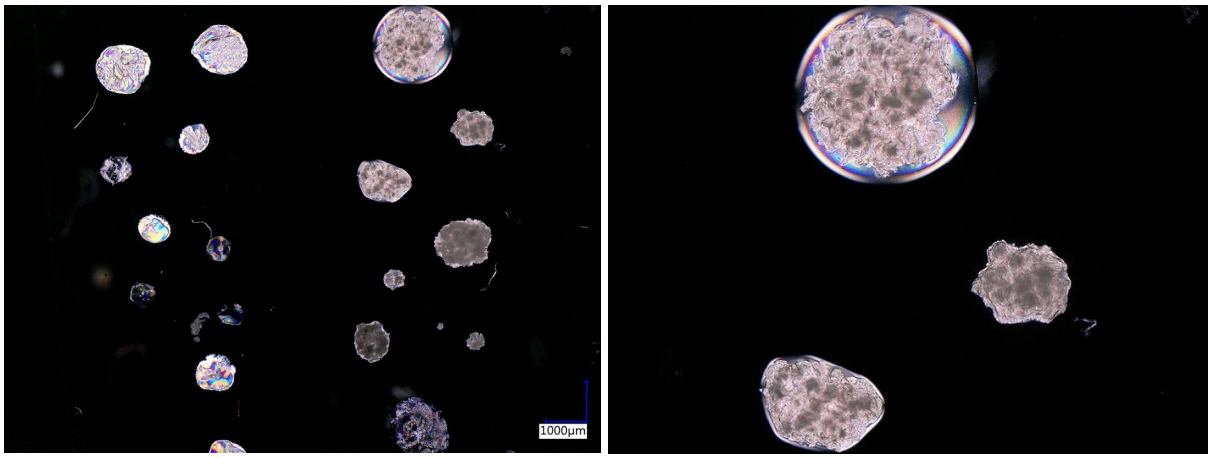
Microchannels in PDMS with Liquid Crystal - day 4 (after filling with LC).
On the right, a magnification of an exemplary area of interest.



Microchannels in PDMS with Liquid Crystal - day 7 (after filling with LC).
On the right, a magnification of an exemplary area of interest.



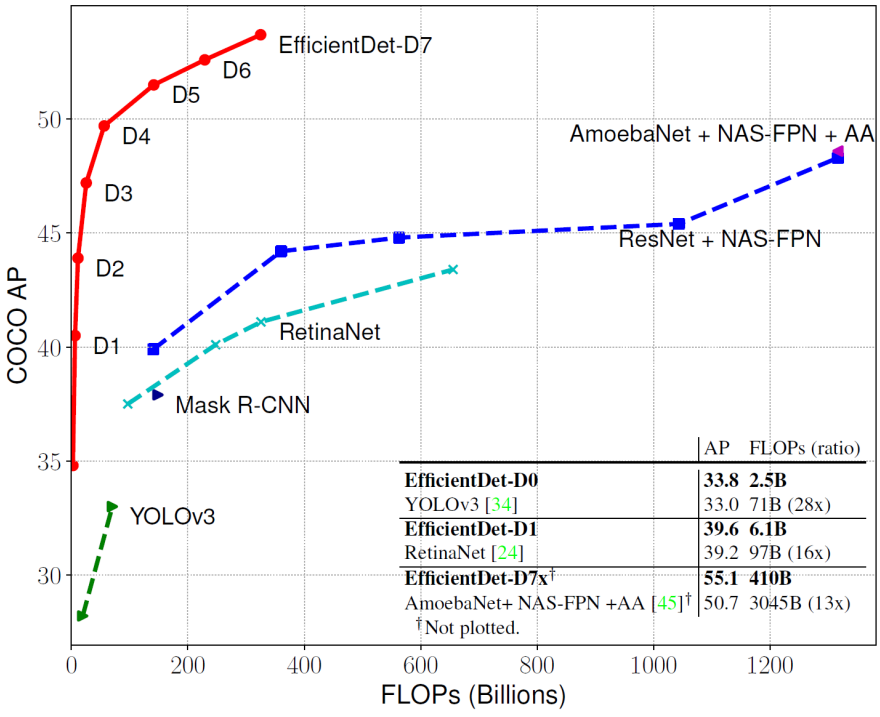
Liquid crystal droplets on PDMS substrate - day 8.
On the right, a magnification of an exemplary area of interest.



Liquid crystal droplets on PDMS substrate - day 14.
On the right, a magnification of an exemplary area of interest.

3. Tools and models used

During the project, various tools were used to perform the training of a neural network adapted for objects recognition. Going from the beginning - after selecting the appropriate photos, a simple script in Python (version 3.7) was used, which changed the resolution of the photos to a specific one used by the model. The next step was to mark the appropriate fragments of the photos with the "LabelImg" tool, which is a graphical image annotation tool. Annotations about 1 class "sediment" was saved as XML files in PASCAL VOC format and then combined into one CSV file using a Python script. Data preparation in this way allowed to use the TensorFlow 2.5 library and the neural network models available in this library. The "EfficientDet D0" model was used for the tests.



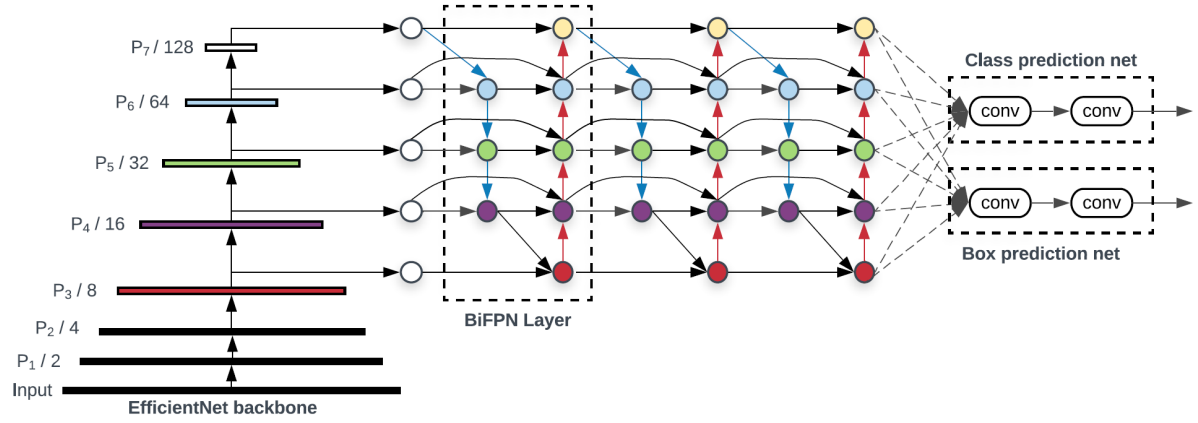
Model FLOPs vs. COCO accuracy *– All numbers are for single-model single-scale.*

Model	test-dev AP	test-dev AP_{50}	test-dev AP_{75}	val AP	Params	Ratio	FLOPs	Ratio	Latency (ms) TitianV	Latency (ms) V100
EfficientDet-D0 (512)	**34.6**	**53.0**	**37.1**	**34.3**	**3.9M**	**1x**	**2.5B**	**1x**	**12**	**10.2**
YOLOv3 [34]	33.0	57.9	34.4	-	-	-	71B	28x	-	-
EfficientDet-D1 (640)	**40.5**	**59.1**	**43.7**	**40.2**	**6.6M**	**1x**	**6.1B**	**1x**	**16**	**13.5**
RetinaNet-R50 (640) [24]	39.2	58.0	42.3	39.2	34M	6.7x	97B	16x	25	-
RetinaNet-R101 (640)[24]	39.9	58.5	43.0	39.8	53M	8.0x	127B	21x	32	-
EfficientDet-D2 (768)	**43.9**	**62.7**	**47.6**	**43.5**	**8.1M**	**1x**	**11B**	**1x**	**23**	**17.7**
Detectron2 Mask R-CNN R101-FPN [1]	-	-	-	42.9	63M	7.7x	164B	15x	-	56‡
Detectron2 Mask R-CNN X101-FPN [1]	-	-	-	44.3	107M	13x	277B	25x	-	103‡
EfficientDet-D3 (896)	**47.2**	**65.9**	**51.2**	**46.8**	**12M**	**1x**	**25B**	**1x**	**37**	**29.0**
ResNet-50 + NAS-FPN (1024) [10]	44.2	-	-	-	60M	5.1x	360B	15x	64	-
ResNet-50 + NAS-FPN (1280) [10]	44.8	-	-	-	60M	5.1x	563B	23x	99	-
ResNet-50 + NAS-FPN (1280@384)[10]	45.4	-	-	-	104M	8.7x	1043B	42x	150	-
EfficientDet-D4 (1024)	**49.7**	**68.4**	**53.9**	**49.3**	**21M**	**1x**	**55B**	**1x**	**65**	**42.8**
AmoebaNet+ NAS-FPN +AA(1280)[45]	-	-	-	48.6	185M	8.8x	1317B	24x	246	-
EfficientDet-D5 (1280)	**51.5**	**70.5**	**56.1**	**51.3**	**34M**	**1x**	**135B**	**1x**	**128**	**72.5**
Detectron2 Mask R-CNN X152 [1]	-	-	-	50.2	-	-	-	-	-	234‡
EfficientDet-D6 (1280)	**52.6**	**71.5**	**57.2**	**52.2**	**52M**	**1x**	**226B**	**1x**	**169**	**92.8**
AmoebaNet+ NAS-FPN +AA(1536)[45]	-	-	-	50.7	209M	4.0x	3045B	13x	489	-
EfficientDet-D7 (1536)	**53.7**	**72.4**	**58.4**	**53.4**	**52M**		**325B**		**232**	**122**
EfficientDet-D7x (1536)	**55.1**	**74.3**	**59.9**	**54.4**	**77M**		**410B**		**285**	**153**

We omit ensemble and test-time multi-scale results [30, 12]. RetinaNet APs are reproduced with our trainer and others are from papers.
‡Latency numbers with ‡ are from detectron2, and others are measured on the same machine (TensorFlow2.1 + CUDA10.1, no TensorRT).

TABLE 1. ***EfficientDet performance on COCO*** *– Results are for single-model single-scale. test-dev is the COCO test set and val is the validation set. Params and FLOPs denote the number of parameters and multiply-adds Latency is for inference with batch size 1. AA denotes auto-augmentation.*

The EfficientDet network was selected due to its efficiency and average precision, which is confirmed by the chart presented. Table 1 compares the individual EfficientDet models and the corresponding competitor models. The results of the EfficientDet neural network are better despite the smaller number of operations performed.



EfficientDet architecture *– It employs EfficientNet as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints*

The architecture of the EfficientDet network is complex. It is composed of different layers. First, an EfficientNet network was used as the backbone. The feature network is the next layer. Efficient bidirectional cross-scale connections and weighted feature fusion (BiFPN) were, in this case, used to resolve the multi-scale feature fusion problem. Another part of the model is class / box prediction net.

Due to problems related to the library installation on a stationary unit, Google Colab was used to train the neural network. During training some problems occurred due to allocated hardware that had insufficient memory to train the neural network. Several trials were carried out, of which only two allowed to obtain satisfactory results. The settings for the EfficientDet D0 512x512 neural network were as follows:

1) Model nr 1:

batch_size = 16
num_steps = 8000
num_eval_steps = 500

2) Model nr 2:

batch_size = 16
num_steps = 3000
num_eval_steps = 500

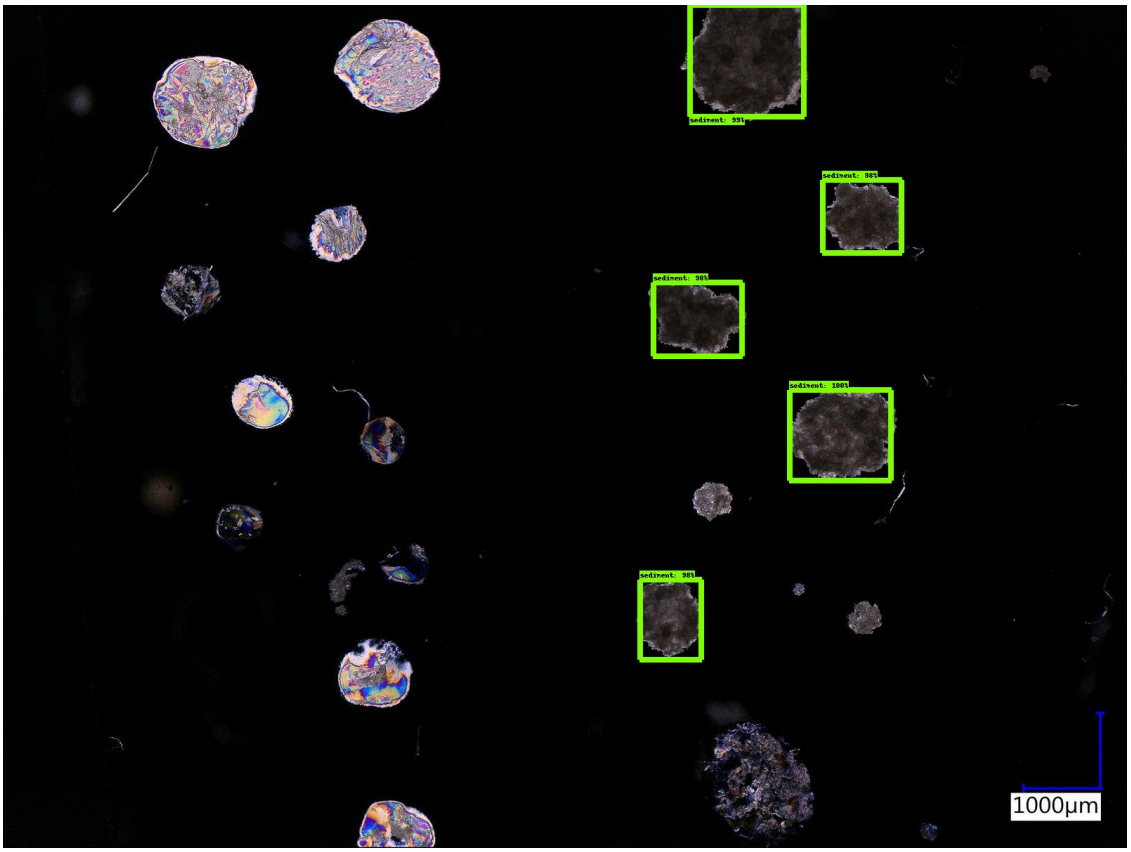
4. Results

The results for the model no. 1 for which 8000 steps were set are presented in the following photos. In 3 of the 7 photos selected for model evaluation, model no. 1 showed that there are areas with sediment. For the most part, these areas are marked with a probability of more than 80 percent, but there are also areas with a probability of 71%, 66% or 51%. The model didn't show all the sediment areas in the individual photos, but it's a good start.

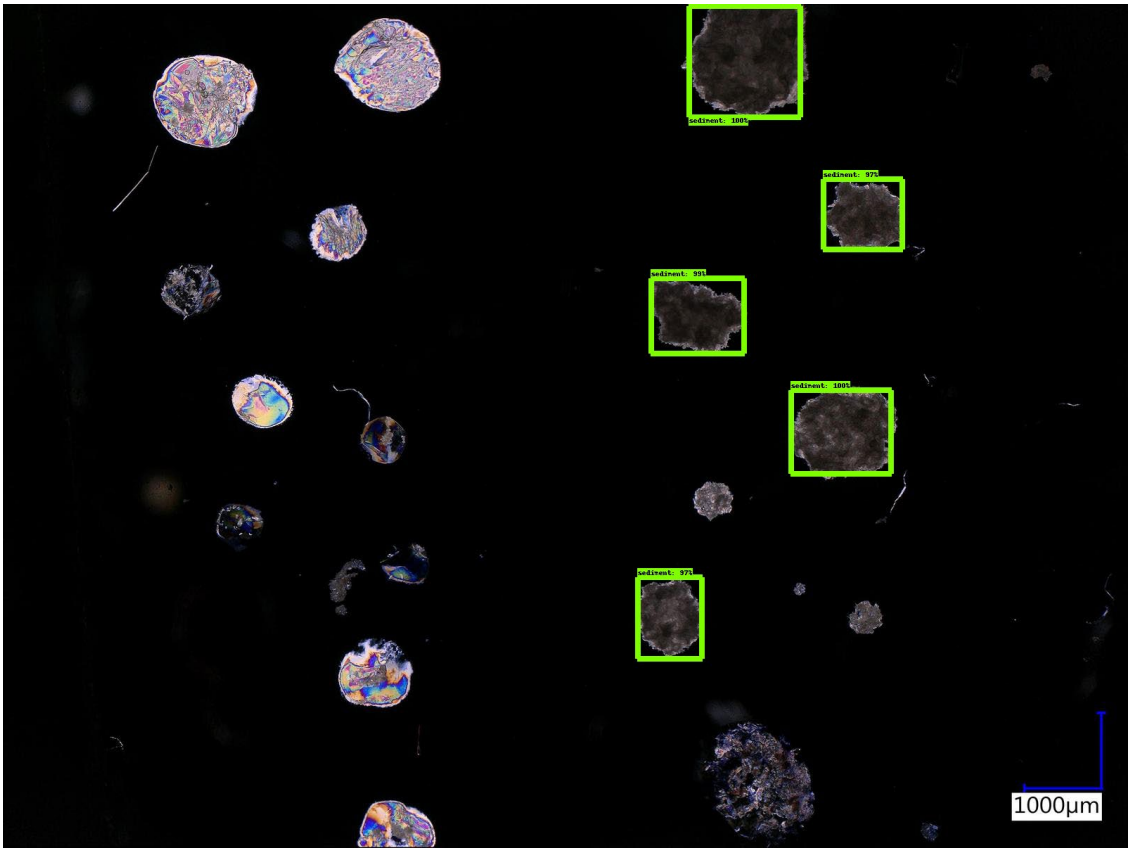
Model no. 2 with 3000 steps did much better. In this case, 5 out of 7 photos have marked areas with sediment. The marked areas had a probability of 50% to 100%. Marking areas with sediment in parallel channels was not a simple task, but the model detected a large portion of the sediment and marked it correctly. In no case did the model make a mistake and did not mark another part of the photo as sediment.

The results for model 2 are satisfactory and allow for the conclusion that using such models for this task may be a good path in the future. In the future, topics related to semantic segmentation will be explored. Semantic segmentation will allow to better understand which areas of the image are assigned to the sediment. But at the moment, correctly finding at least one area with the sediment in the photo allows to automate some of the tasks during the research work for my doctoral thesis.

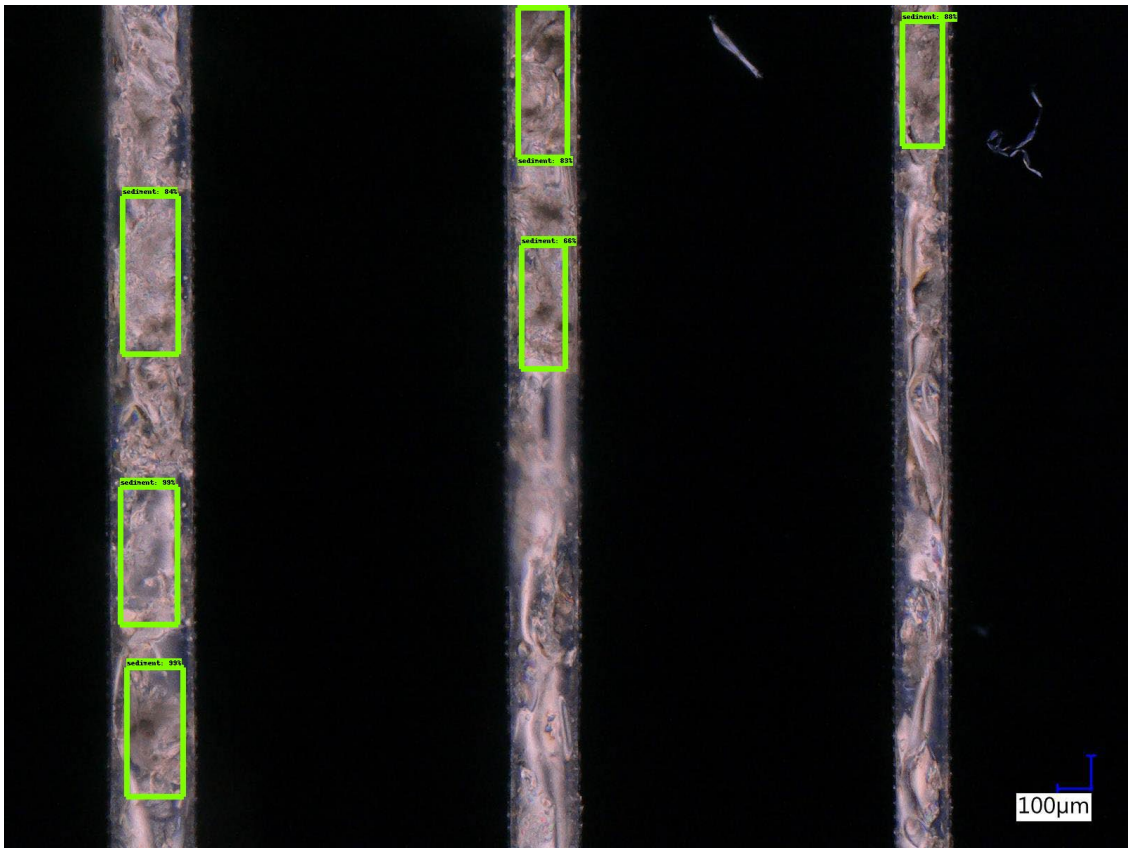
A comparison of the test photos, along with the marking of the sediment by both models, is presented on following pictures.



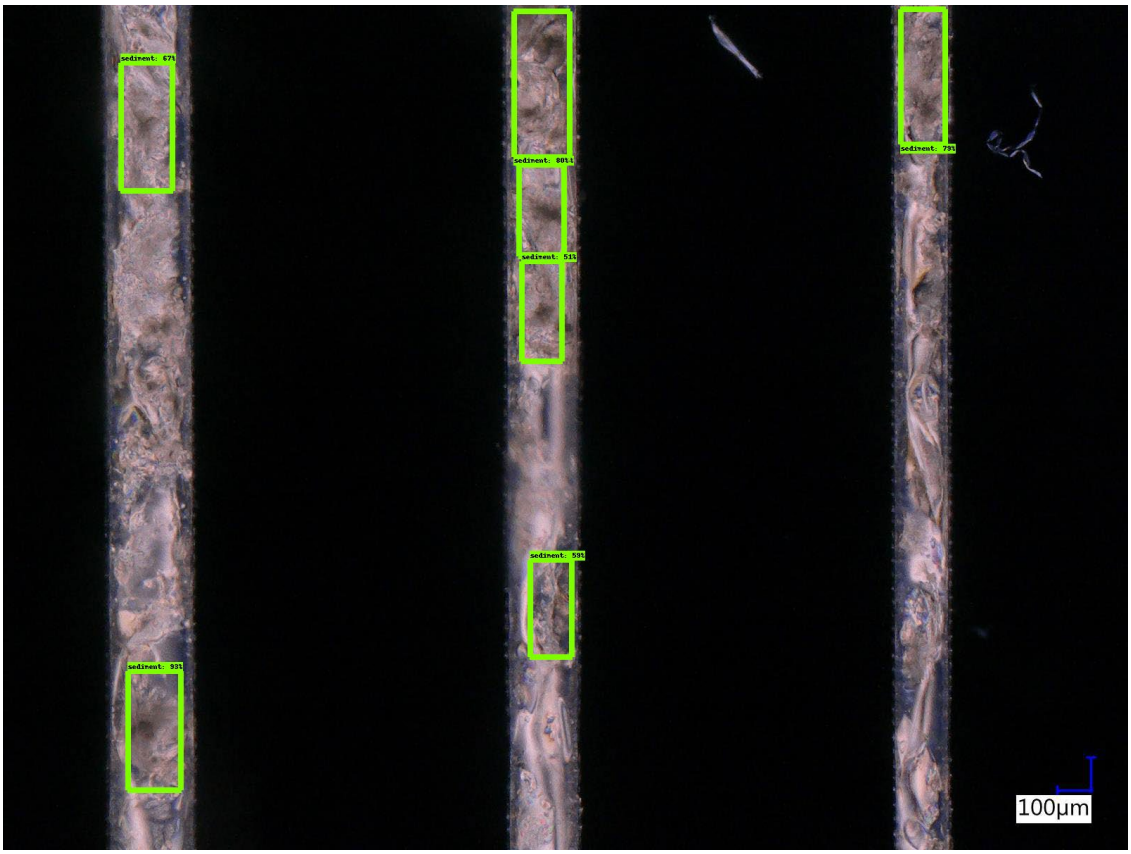
Model no. 1 (8000 steps) *– liquid crystal droplets with sediment*



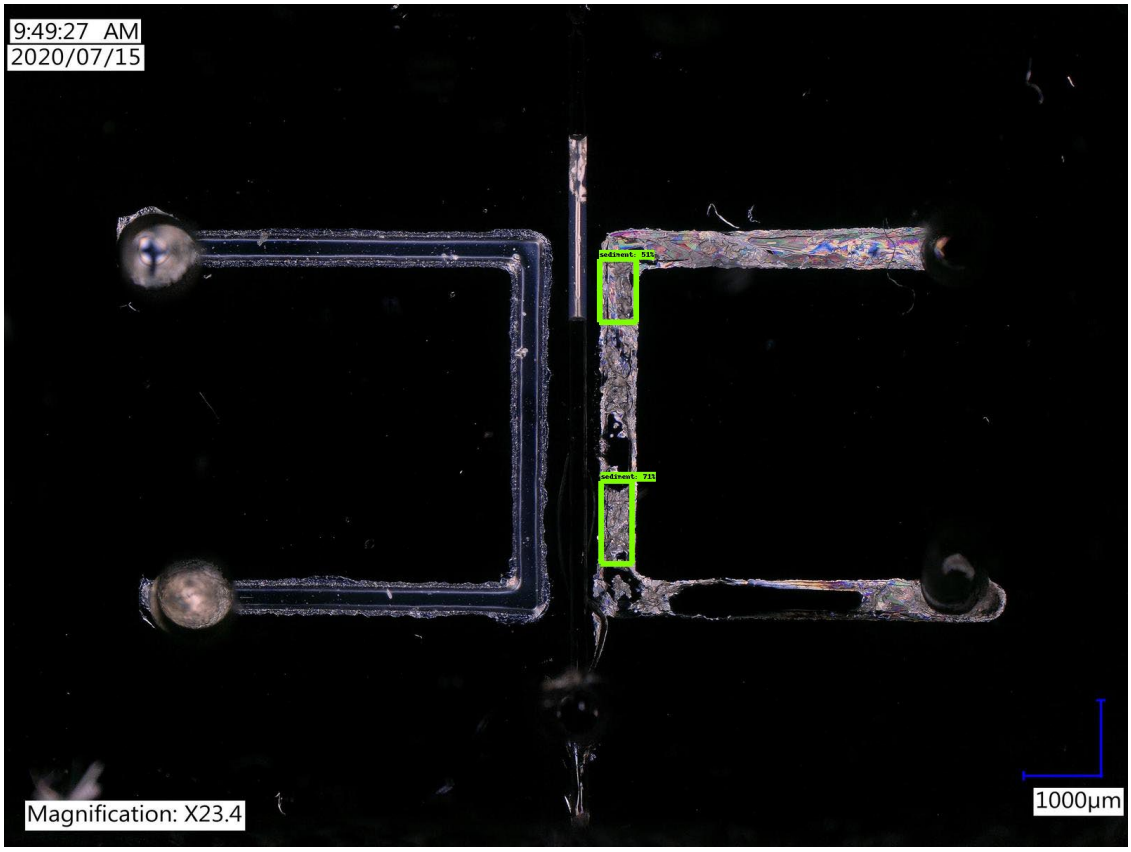
Model no. 2 (3000 steps) *– liquid crystal droplets with sediment*



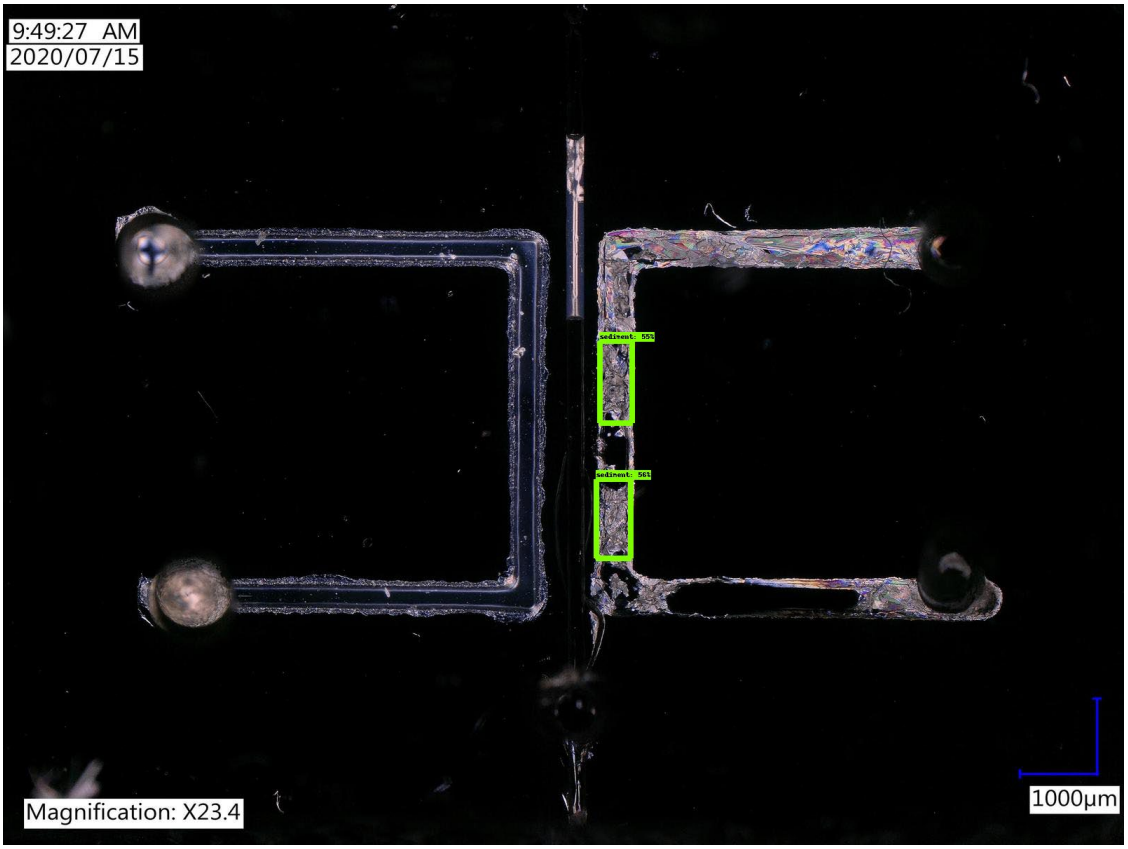
Model no. 1 (8000 steps) *– microchannels in PDMS*



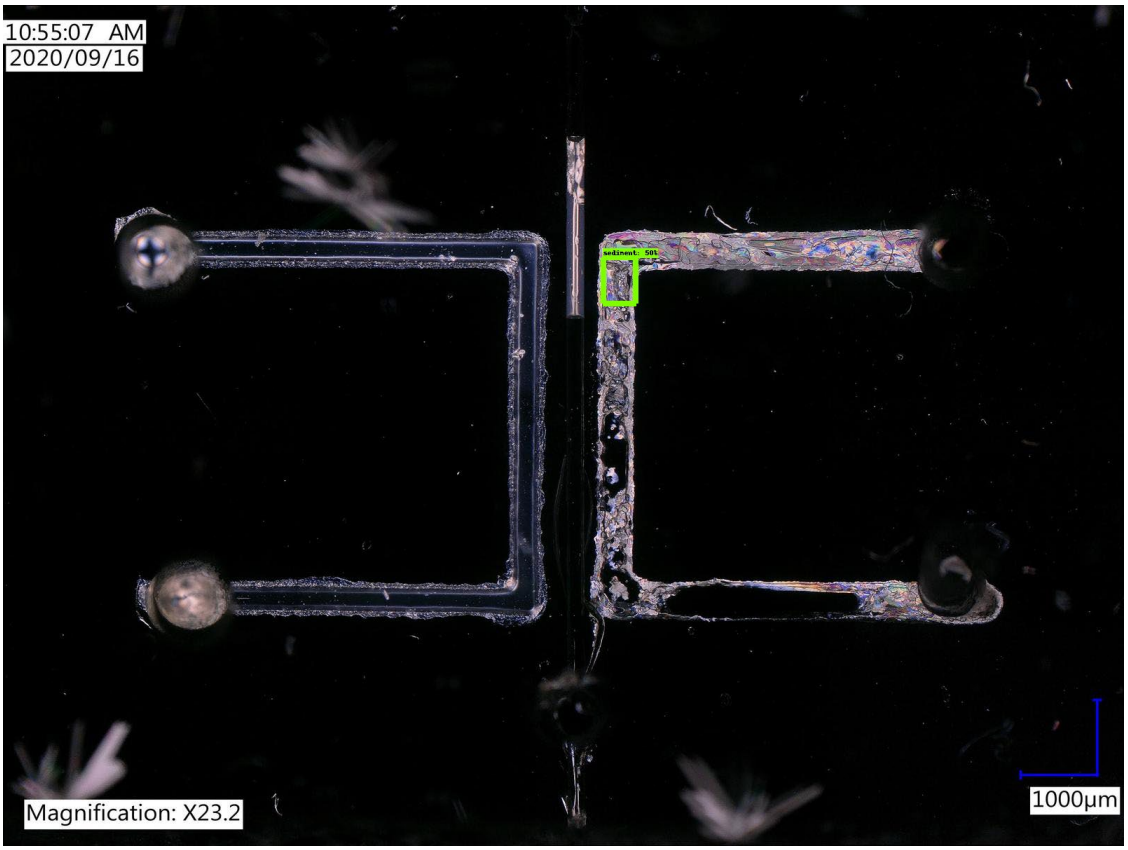
Model no. 2 (3000 steps) *– microchannels in PDMS*



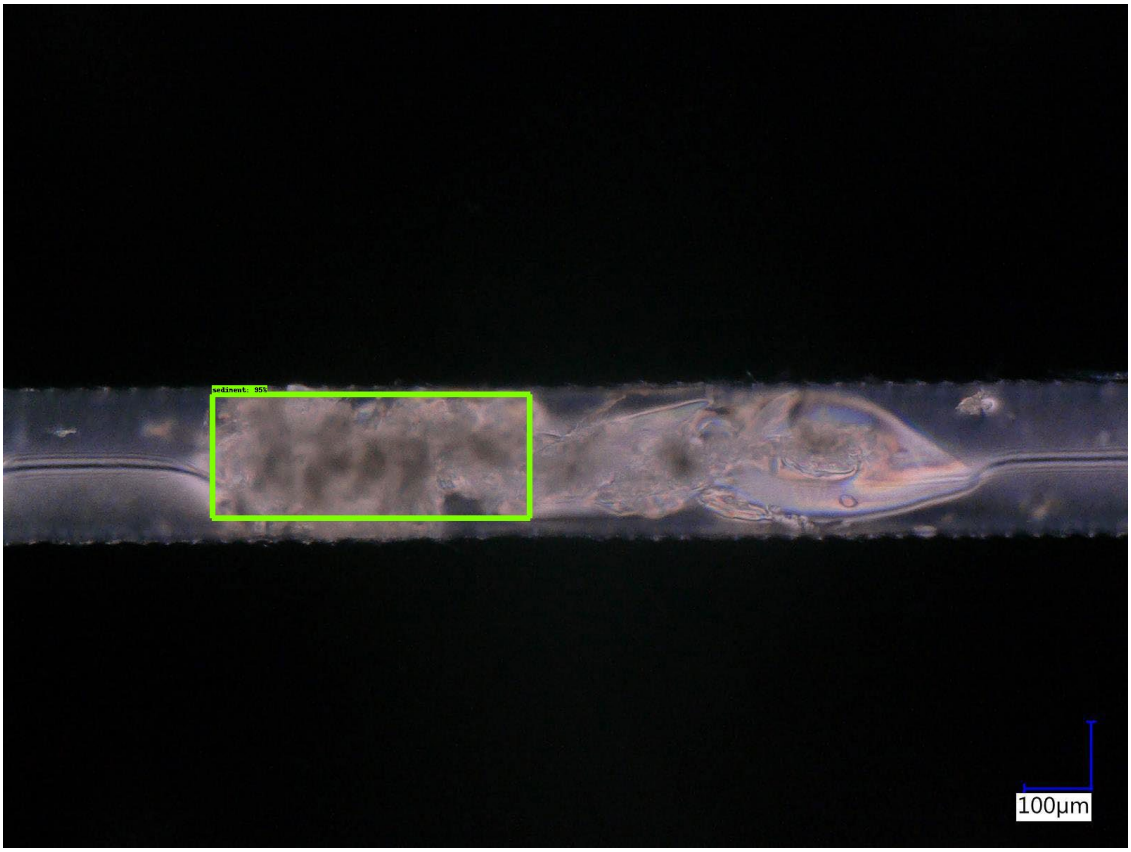
Model no. 1 (8000 steps) *– microchannels in PDMS*



Model no. 2 (3000 steps) *– microchannels in PDMS*



Model no. 2 (3000 steps) *– microchannels in PDMS*



Model no. 2 (3000 steps) *– microchannel in PDMS*

References

[1] Mingxing Tan, Ruoming Pang, Quoc V. Le, "*EfficientDet: Scalable and Efficient Object Detection*", https://arxiv.org/pdf/1911.09070.pdf

[2] LabelImg - graphical image annotation tool, https://github.com/tzutalin/labelImg

[3] TensorFlow 2 Object Detection API tutorial - Training Custom Object Detector, https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html

[4] How to train a custom object detection model with the Tensorflow Object Detection API, https://github.com/TannerGilbert/Tensorflow-Object-Detection-API-Train-Model

[5] TensorFlow 2 Detection Model Zoo, https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md