Sofia University "St. Kliment Ohridski"

Faculty of Physics

Major: "Computer Engineering",

Thesis

for

acquiring the educational and qualification degree

"bachelor"

by

Petko Petkov, faculty №15037

Topic: Determining the parameters of capillary bridges using machine vision

Scientific Supervisor:

/Assoc. Prof. Dr. Hristo Iliev/

Sofia, July 2024

Contents

1 Introduction

Machine vision is a field of computer science and engineering that deals with the automated processing and analysis of visual information by computer systems. Its main goal is to give computers the ability to "see" and understand the surrounding environment in a way similar to human vision. Machine vision has many applications. Some of them are industrial automation (quality control, assembly, and sorting), medical diagnostics (analysis of X-rays, MRI scans, disease detection), security (facial recognition, surveillance, and access control), transportation (autonomous cars, driver assistance and traffic systems), and others.

Machine vision can be considered separately from computer vision ([1]), which is a part of computer science. Some of the problems this field deals with are classification (assigning a category to an image based on its features), object recognition (detecting and identifying objects in an image), image segmentation (dividing an image into individual objects or regions of interest), object tracking (tracking the movement of objects in a sequence of images), and others.

Manual measurement of parameters from experimental photos of capillary bridges is inaccurate and slow. Automating the measurements increases accuracy, simplifies the work, and saves time, allowing for more experiments and more work on the theory. The goal of this thesis is to develop software that implements various models to automate the measurement of the parameters (Chapter 2) of capillary bridges with minimal error ($\sim 2\%$) compared to manual measurements. The software must be able to control basic camera parameters (exposure, frame rate) and capture photos and videos of the capillary bridges whose parameters are being measured. The graphical interface of the software, shown in Chapter 7.2, allows for better control over the entire process of creating the structures, optimizing the experiment, capturing the measured parameters and comparing them with the raw images, controlling the camera's frame rate and exposure time, and measuring the parameters of a video in real-time or from a single photo. Real-time image processing is used for optimization, system setup, and parameter monitoring. In this mode of operation, the accuracy of the measurements may be lower in many cases.

After capturing images of a capillary bridge using the experimental setup shown in Chapter 2, the parameters visualized in Fig. 2 must be determined. The images of the capillary bridges from the experiment are not identical. They depend on the execution of the experiment itself, lighting, reflections, camera settings, etc. Also, the structures are not symmetrical. These differences between experiments do not allow the use of an analytical model for the automatic measurement of the parameters. It is necessary to use other models that work correctly for the different images.

2 Experimental Setup

The experimental setup [2] consists of mounted equipment, which includes a specially designed cell. It is composed of two support plates (upper and lower). The lower one is fixed (point 1 in Fig. 1), and the upper one (point 2 in Fig. 1) can move controllably up and down (point 5 in Fig. 1) via a micrometer screw. Support tables are placed on the plates (upper shown at point 3 in Fig. 1 and lower shown at point 4 in Fig. 1). On them, pre-cleaned cover glasses $22 \times 22mm$ with a thickness of 0.13 to 0.16 mm (Deltalab microscopy cover glass) are placed. The cover glasses are replaced each time a new drop is placed to create a capillary bridge.

The video camera used is a Basler ace U acA800-510uc. It uses a USB 3.0 interface. The maximum frame rate it can achieve is 511 frames per second. The minimum exposure time is 59 µs, and the maximum is 1 second. The camera's resolution is 800×600 pixels. The recording speed and exposure time can be changed through the software. The optical system allows for recording images in polarized light, which significantly increases the contrast of the images. All equipment is installed on a four-axis precision positioning system, which allows for very high accuracy in positioning the camera relative to the analyzed structures.

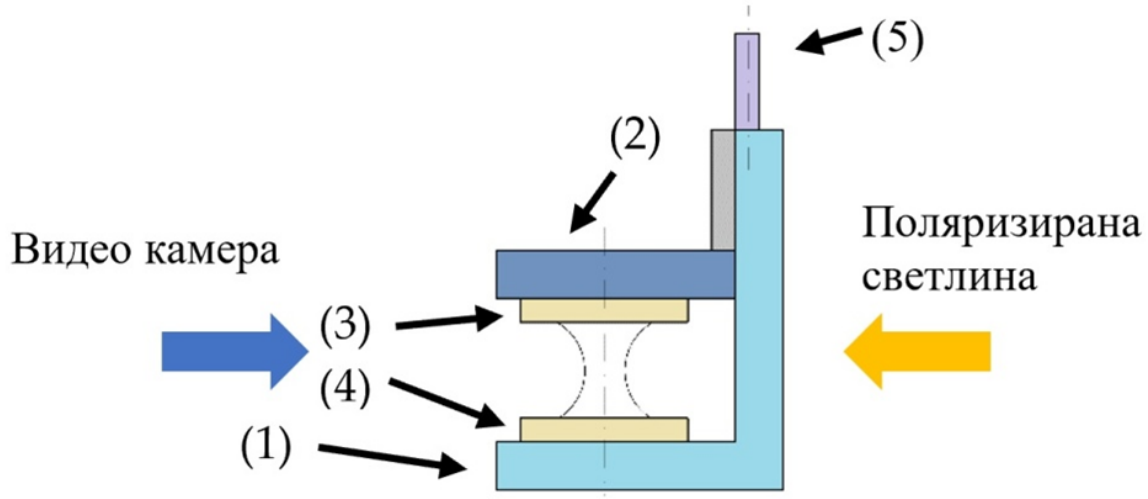


Fig. 1: Diagram of the experimental setup (adapted from [2])

3 Searched Parameters

The parameters to be determined are as follows (Fig. 2):

- neck length - the distance between the two closest points
- upper boundary length
- lower boundary length
- left boundary length
- right boundary length
- left ellipse axes lengths
- right ellipse axes lengths

In the middle of the example image in Fig. 2 is the liquid used for the experiment. The indentations represent reflections, and they are used to find the parameters.

The final result of the program is the lengths of the parameters measured in µm. To achieve this, the lengths of the lines are measured, which represent a number of pixels. Then, the number of pixels is converted to micrometers using the following function:

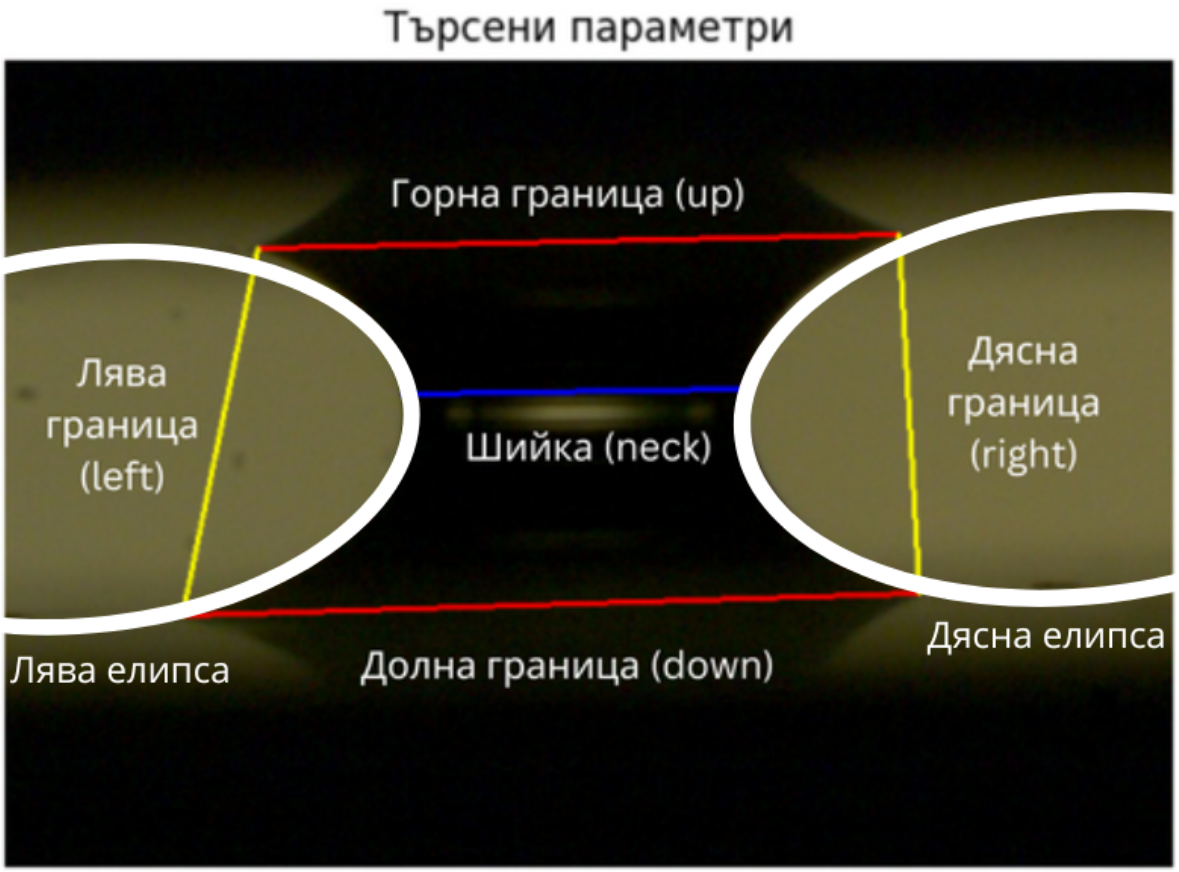


Fig. 2: Example image of a capillary bridge with visualization of the searched parameters

$$c(p) = \frac{p \times 3659.269}{1920}$$

where 1920 is the number of pixels corresponding to 3659.269 µm, and p is the pixels of the current image being measured.

To measure all these parameters, it is necessary to find the boundaries of the objects in the image. This problem is called segmentation. The space to the left and right of the liquid is the region that needs to be segmented. The two regions (left and right) will be called objects for segmentation, or objects for short. Then, the starting and ending points of the lines representing the lengths of the searched parameters must be found. To find the neck, a line must be drawn between the closest points of the object boundaries; for the upper boundary - between the two upper points; for the lower boundary - between the two lower points; for the left boundary - between the upper and lower left points; and for the right boundary - between the upper and lower right points.

The parameters are needed to calculate certain ratios that are used for theory development (Chapter 5.5). The measurement of the parameters and the ratios represents the final result of the program.

4 Segmentation Models

Segmentation is the task of classifying each pixel of an image. In this specific case, the pixels of each image are divided into two categories - objects and background. For the segmentation of the objects in the image, two main models and several of their variations with different configurations have been implemented and compared:

- Image thresholding with Otsu's method (Chapter 4.1) [3]
    1. Without a contrast enhancement filter
    2. With a contrast enhancement filter (Chapter 4.1.1)
- Neural network "Segment Anything" (Chapter 4.2)([4])
    1. Baseline "Segment Anything" model (Chapter 4.2.3)
    2. Neural network "FastSAM" (a variant of "Segment Anything") (Chapter 4.2.4) ([5])
    3. "Segment Anything" with fine-tuning (Chapter 4.2.5)

Before segmentation, the image is converted from the RGB color model to a grayscale image as follows:

$$I_{\text{gray}} = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B$$

where
- R is the red channel
- G is the green channel
- B is the blue channel
- the constants 0.2989, 0.5870, 0.1140 are standard and represent the relative perception of the brightness of red, green, and blue by humans

4.1 Image thresholding with Otsu's method

Thresholding is one of the most commonly used methods for segmenting objects in an image. The basic idea is to choose a threshold value T that divides the pixels into two groups: those with an intensity less than or equal to T, and those with an intensity greater than T.

$$I'(x,y) = \begin{cases} 0, & \text{if } I(x,y) \leq T \\ 1, & \text{if } I(x,y) > T \end{cases}$$

where:
- $I(x,y)$ is the intensity of the pixel at coordinates (x,y) in the original image.
- $I'(x,y)$ is the intensity of the pixel at coordinates (x,y) in the resulting binary image.
- T is the threshold value.

Here, the threshold value T is a constant:

$$T = \text{const}$$

This version of the algorithm works for simpler tasks and for images where there is no noise and the lighting is evenly distributed. These conditions are not met for all images

of capillary bridges whose parameters need to be determined. To properly segment the objects in the image, an algorithm with an adaptive threshold value is needed. In this case, the value of T changes depending on the local properties of the image. The algorithm used is Otsu's method ([3]):

Image histogram (number of pixels vs. intensity):

$$h(z) = \#\left\{I(x,y) \mid I(x,y) = z\right\} \quad z \in [0, L]$$

Number of pixels of the object:

$$n_o(T) = \sum_{z=T+1}^{L} h(z)$$

Number of pixels of the background:

$$n_b(T) = \sum_{z=0}^{T} h(z)$$

Mean intensity of the object:

$$m_o = \frac{\sum_{z=T+1}^{L} zh(z)}{n_o(T)}$$

Mean intensity of the background:

$$m_b = \frac{\sum_{z=0}^{T} zh(z)}{n_b(T)}$$

Between-class variance:

$$\sigma(T) = n_b(T)n_o(T)\Big[m_b(T) - m_o(T)\Big]^2$$

Optimal threshold value:

$$T : \sigma(T) \to \max$$

Otsu's method works better than thresholding with a constant threshold value, but it does not give good results for images whose histograms do not contain clearly separated classes of regions for segmentation (objects and background), which can occur in images with reflections and unevenly distributed lighting.

Despite inaccuracies in measuring some of the parameters, Otsu's method allows for real-time image processing ($\sim$ 30 frames per second on the computer used for the experiment). A visualization of an image after processing can be seen in Fig. 3.

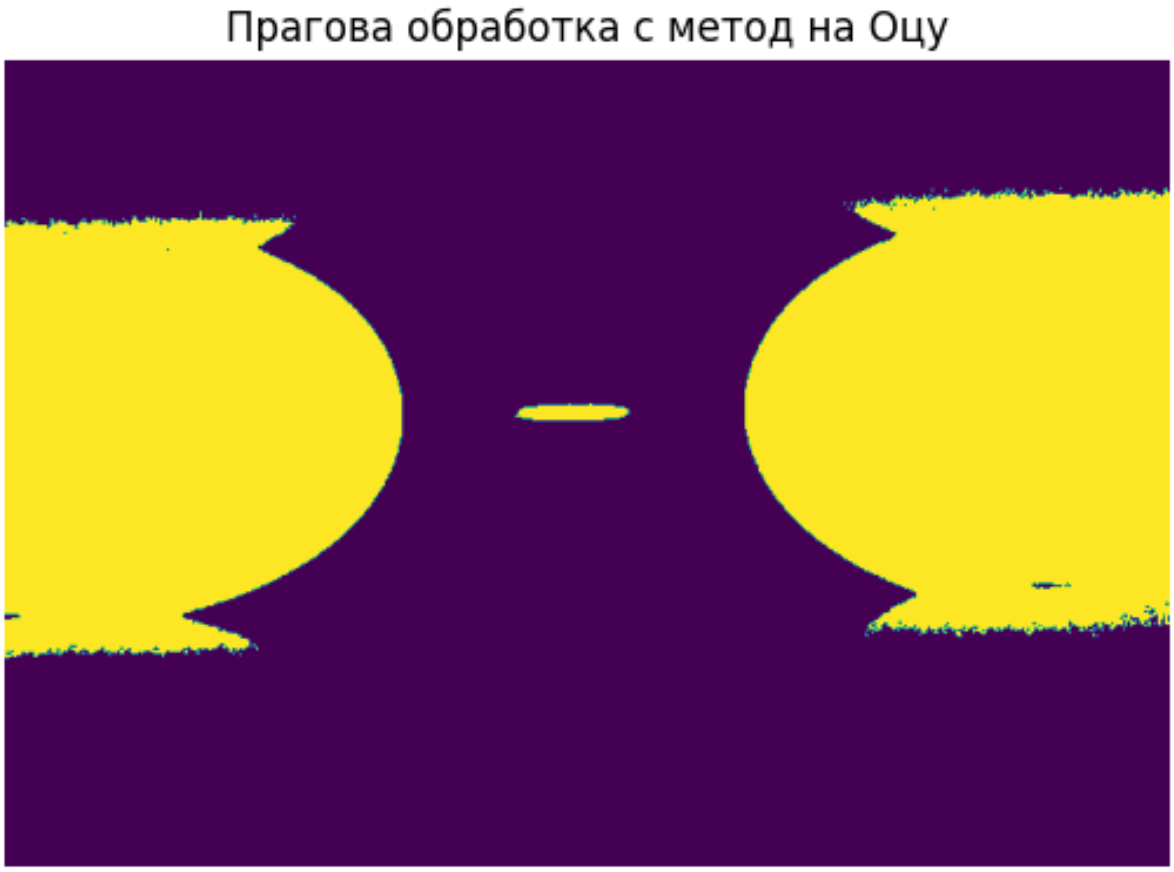


Fig. 3: Image after thresholding with Otsu's method

4.1.1 Contrast enhancement filter

Because some of the images do not have high contrast between the objects and the background (the difference between the intensity of the objects and the background is small), proper segmentation is not achieved when using Otsu's method. This problem can be solved by adding a filter to increase the contrast between the objects and the background (it increases the intensity only of the parts with higher intensity and preserves the intensity of those with lower intensity) before applying the thresholding.

$\mathbf{I}$ is the input image with dimensions $m \times n \times 3$, where the third dimension represents the three RGB color channels (red, green, and blue), and $\mathbf{I_o}$ is the output image after applying the filter.

The two parameters β and γ are in the following intervals:

$$\beta \in [-255, 255]$$
$$\gamma \in [-127, 127]$$

If $\beta \neq 0$

$$\mathrm{s} = \begin{cases} \beta & \text{if } \beta > 0 \\ 0 & \text{if } \beta \leq 0 \end{cases}$$

$$\mathrm{h} = \begin{cases} 255 & \text{if } \beta > 0 \\ 255 + \beta & \text{if } \beta \leq 0 \end{cases}$$

$$\alpha_b = \frac{\mathrm{h} - \mathrm{s}}{255}$$

$$\gamma_b = \mathrm{s}$$

Applying brightness to the image:

$$\mathbf{I}_o = \alpha_b \cdot \mathbf{I} + \gamma_b$$

If $\beta = 0$, the input image remains unchanged:

$$\mathbf{I}_o = \mathbf{I}$$

If $\gamma \neq 0$

$$f = \frac{131 \cdot (\gamma + 127)}{127 \cdot (131 - \gamma)}$$

$$\alpha_c = f$$

$$\gamma_c = 127 \cdot (1 - f)$$

Applying contrast to the image:

$$\mathbf{I}_o = \alpha_c \cdot \mathbf{I} + \gamma_c$$

where:
- β is the brightness
- γ is the contrast

Applying this filter leads to more accurate measurements of the parameters for some of the images (Chapter 6.1.2).

A comparison between an image without a contrast enhancement filter and an image with a filter can be seen in Fig. 4.

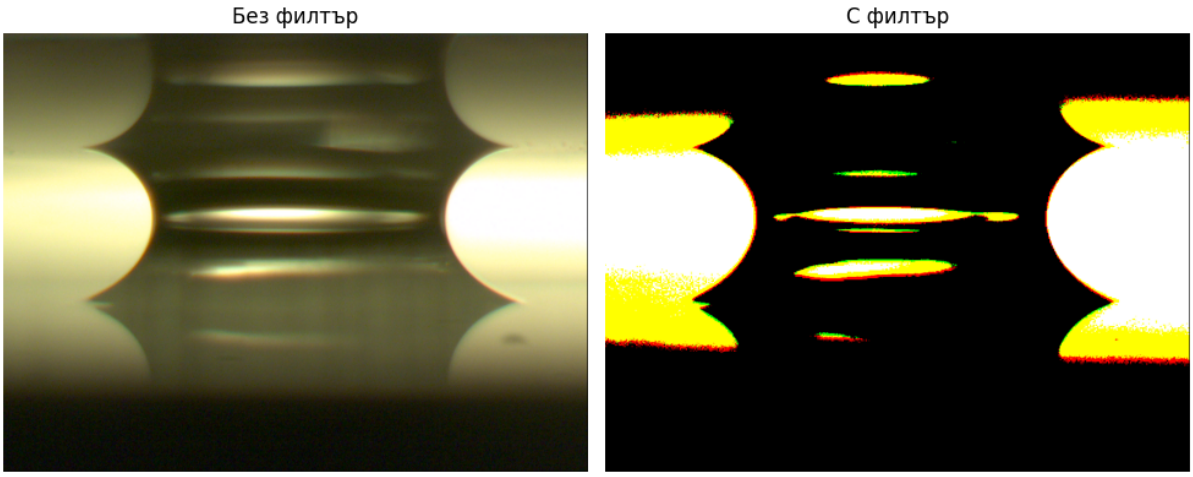


Fig. 4: Comparison between an image without a contrast enhancement filter and an image with a filter. A value of 127 is used for the γ parameter.

4.2 Neural network "Segment Anything"

Thresholding with Otsu's method does not correctly segment all images because some of them have unevenly distributed lighting or too many reflections, which makes it difficult to find the upper and lower ends of the objects. Another approach, which is more universal than thresholding with Otsu's method and allows for the segmentation of images with unevenly distributed lighting and reflections, is the use of neural networks [6] for segmentation. The results of three neural networks, which are variants of "Segment Anything," have been compared. The baseline model of "Segment Anything" (Chapter 4.2), an optimized architecture with a convolutional neural network "FastSAM" (Chapter 4.2.4), and a model with fine-tuning (Chapter 4.2.5) were used.

4.2.1 Basics of neural networks

Artificial neural networks are a simplified model of biological neural networks. They consist of layers of "neurons" that are connected to each other. Each "neuron" receives input data, processes it, and passes the result to the next layer of neurons.

1. Each input x_i is multiplied by a corresponding weight w_i.

$$z = \sum_{i=1}^{n} w_i x_i + b$$

where z is the sum of the input data multiplied by the weights and a bias b.

2. The result z is passed to an activation function $\sigma(z)$, which adds non-linearity to the model. An activation function that is often used is the sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

or ReLU (Rectified Linear Unit):

$$\sigma(z) = \max(0, z)$$

3. The output of the neuron is $\hat{y} = \sigma(z)$.

$$\hat{y} = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

These operations are repeated for each "neuron" and each layer in the network. The network produces an output that can be used for classification (including segmentation), regression, or other tasks.

Example of a fully connected layer with 3 inputs and 1 output:

$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

$$\hat{y} = \sigma(z)$$

This process is repeated for all layers in the network, with the output of one layer becoming the input for the next. At the end of the network, a final output is obtained, which represents the model's prediction for the corresponding input data.

4.2.2 Training a neural network

Training a neural network is the process of optimizing the weights and biases so that the network can make more accurate predictions for the corresponding input data. The main training method is the backpropagation algorithm ([7]), which uses gradient descent.

1. The weights w and biases b are initialized with random values.

2. The input data x passes through the network to calculate the prediction $\hat{y}$.

$$\hat{y} = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

3. The error between the prediction $\hat{y}$ and the correct value y is calculated. A function often used to calculate the error is the mean squared error:

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

4. The gradients of the error with respect to each weight w and bias b are calculated by differentiating complex functions using the chain rule.

For the weight w:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_i}$$

For the bias b:

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b}$$

5. The weights and biases are updated with a small step in the direction opposite to the gradient of the error to reduce the error. In the simplest case, the step size is a constant. For the weight w:

$$w_i := w_i - \eta \frac{\partial L}{\partial w_i}$$

For the bias b:

$$b := b - \eta \frac{\partial L}{\partial b}$$

where η is the learning rate.

This process is repeated for each sample in the training set. One full pass through all samples in the training set is called an epoch. The entire training process usually runs for several epochs until the error function is minimized to the desired level.

Example of a gradient descent step:

$$w_i := w_i - \eta \, (\hat{y} - y) \cdot x_i \cdot \sigma'(z)$$

$$b := b - \eta \, (\hat{y} - y) \cdot \sigma'(z)$$

where $\sigma'(z)$ is the derivative of the activation function.

4.2.3 Baseline "Segment Anything" model

Unlike traditional segmentation models that require specific modeling for each individual task, "Segment Anything" [4] is a universal model that performs well on a wide range of segmentation tasks. The model can make predictions by specifying a prompt, which can be in the form of points, boxes, and masks that denote the object to be segmented, or in the form of text describing the object. "Segment Anything" is divided into three main smaller models:

- image encoder - a neural network with a vision transformer architecture ([8]), trained using the masked auto-encoding method ([9]). The input image to the "Segment Anything" model is transformed into a lower-dimensional vector by this part (image encoder) of the model. This is called an image embedding. It represents a general form of image representation that is robust to variations such as changes in lighting, scale, and rotation.
- prompt encoder - creates an embedding of the prompt (points, boxes, text, and masks), which is in the form of a lower-dimensional vector. If text is used as a prompt, it also has an embedding, which is created using CLIP ([10]).
- mask decoder - takes the results from the image encoding and the prompt encoding to make a prediction in the form of a mask, which represents the result of the segmentation.

A simplified diagram of the "Segment Anything" architecture can be seen in Fig. 5.

The baseline model is the neural network that is pre-trained and used directly on the images of capillary bridges without having been trained on similar images before. There are three main pre-trained models, which differ in the number of parameters (weights

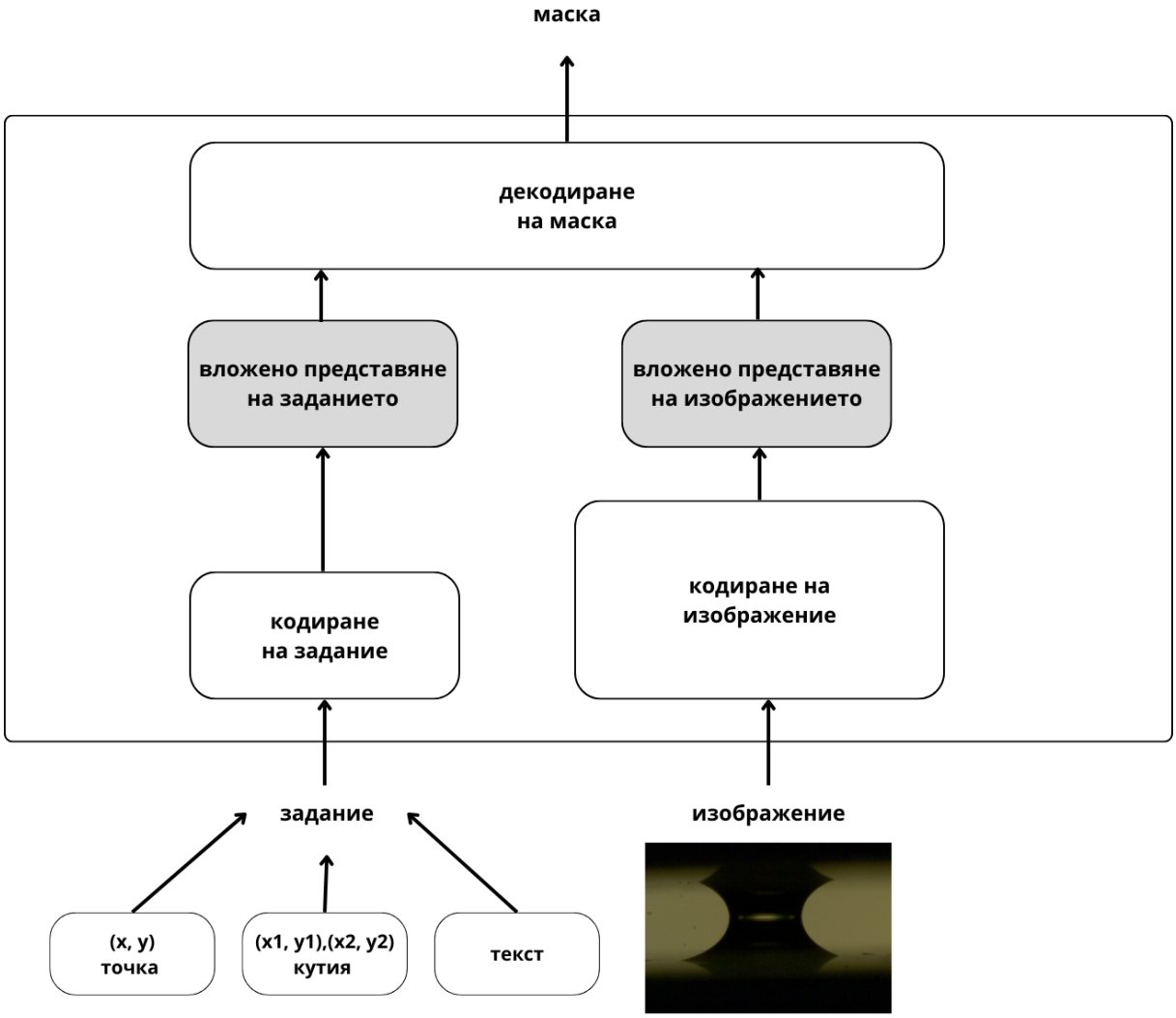


Fig. 5: Architecture of "Segment Anything"

and biases). The model used for the segmentation of the capillary bridge images is the smallest of the three pre-trained ones (91 million parameters - weights and biases). Two boxes b_l and b_r defined by the following points are used as prompts:

$$x_l = I_w - (I_w - 50)$$
$$x_r = I_w - 50$$
$$y = \frac{I_h}{2} + 50$$
$$b_l = (x_l, y)$$
$$b_r = (x_r, y)$$

where:
- I_w is the width of the image
- I_h is the height of the image

A visualization of segmentation with the baseline "Segment Anything" model is in Fig. 6.

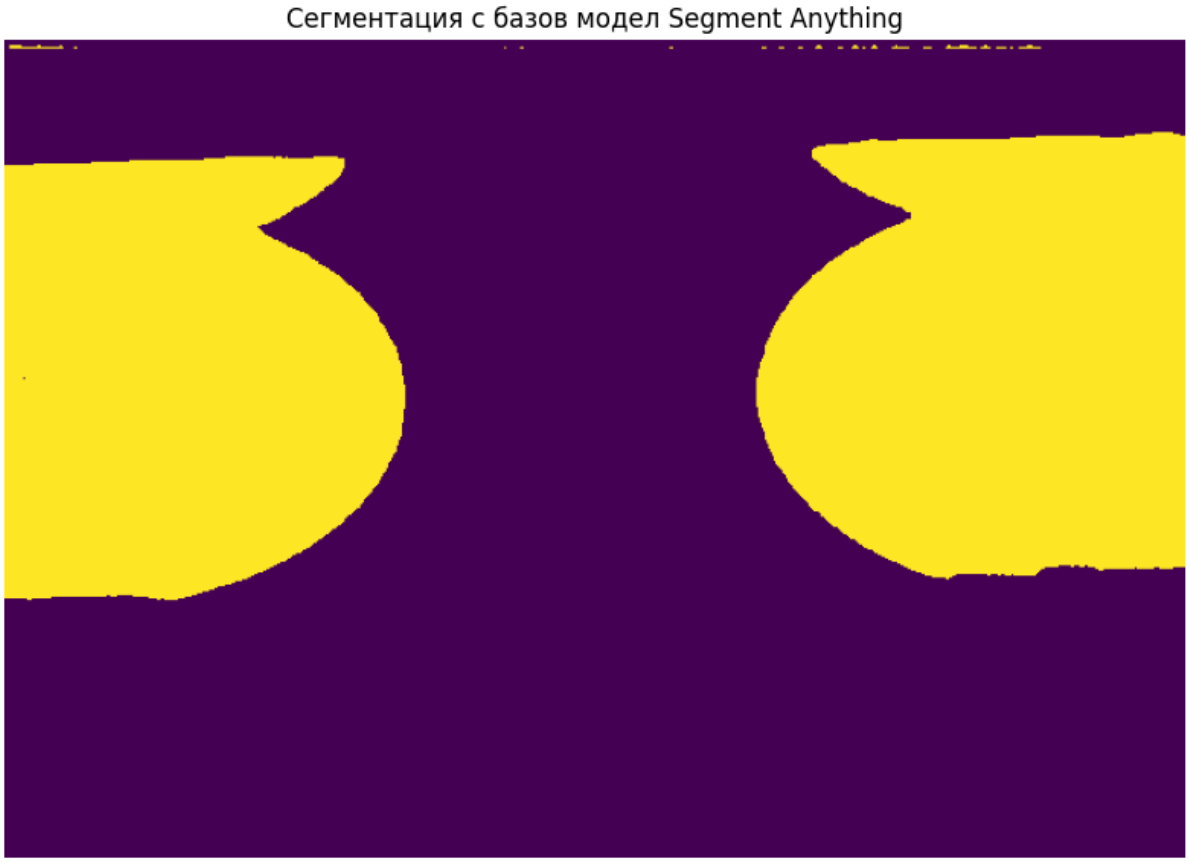


Fig. 6: Segmentation with the baseline "Segment Anything" model

4.2.4 Neural network "FastSAM"

The Vision Transformer model [8], which is the main part of the "Segment Anything" neural network architecture, requires a large amount of computational resources. This makes the model unsuitable for tasks that need to be performed on computers with limited computational resources or tasks that need to run in real-time. "FastSAM" is a more optimized and faster version of "Segment Anything". The main reason for this is the use of a convolutional neural network instead of a vision transformer architecture.

A convolutional neural network [11] is a type of neural network in which the connection between neurons is similar to the organization of the human visual system. It is composed of convolutional layers that contain filters. The filters in each layer traverse the input data for that layer. A dot product of the filter and the input data is performed. Usually, a pooling layer is added after each convolutional layer. Pooling layers compress the result of the convolutional layers (they reduce the dimensionality but try to preserve important information). The most commonly used types of pooling are average pooling, sum pooling, and max pooling. After the convolutional and pooling layers, fully connected layers are added (each "neuron" in the layer is connected to every "neuron" in the previous layer). The purpose of the fully connected layers is to classify the features that were obtained as a result of the previous layers into one of the categories. A Softmax activation function is used:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, 2, \ldots, K$$

The Softmax function is often used in multi-class classification tasks. It converts a vector of real numbers z into a probability distribution. z_i is the i-th element of the input vector z, and K is the number of categories. For each element z_i of the input vector, the exponential function e^{z_i} is calculated. This step increases the values of the input numbers and makes all values positive. All exponentiated values are summed, and each element e^{z_i} is divided by this sum. This ensures that the resulting values are in the interval $[0, 1]$ and their sum is equal to 1, which makes them suitable for interpretation as probabilities. Thus, the input vector z is transformed into an output vector $\sigma(z)$, where each value $\sigma(z_i)$ represents the probability that the input vector belongs to the i-th class.

An example of a simplified convolutional network architecture can be seen in Fig. 7.

Fig. 7: Simplified architecture of a convolutional neural network. The convolutional layer contains a filter of size 5×5, and the pooling is of size 2×2.

"FastSAM" consists of two main components. Instead of a vision transformer, the "Yoloact" architecture [12] is used, which is a convolutional neural network. It is trained to make predictions for object masks for only two categories - mask and background (in this part of the model, there is no distinction between categories as in the original "Segment Anything" neural network). The second component of the architecture is the selection of masks based on the prompt, which is the final result of the model. For the prompt encoder, a similar model to the one in the "Segment Anything" neural network is used, which is also based on CLIP [10]. The results of image processing after segmentation with "FastSAM" (Chapter 6.2.2) are in some cases more accurate than "Segment Anything".

A visualization of segmentation with "FastSAM" is in Fig. 8.

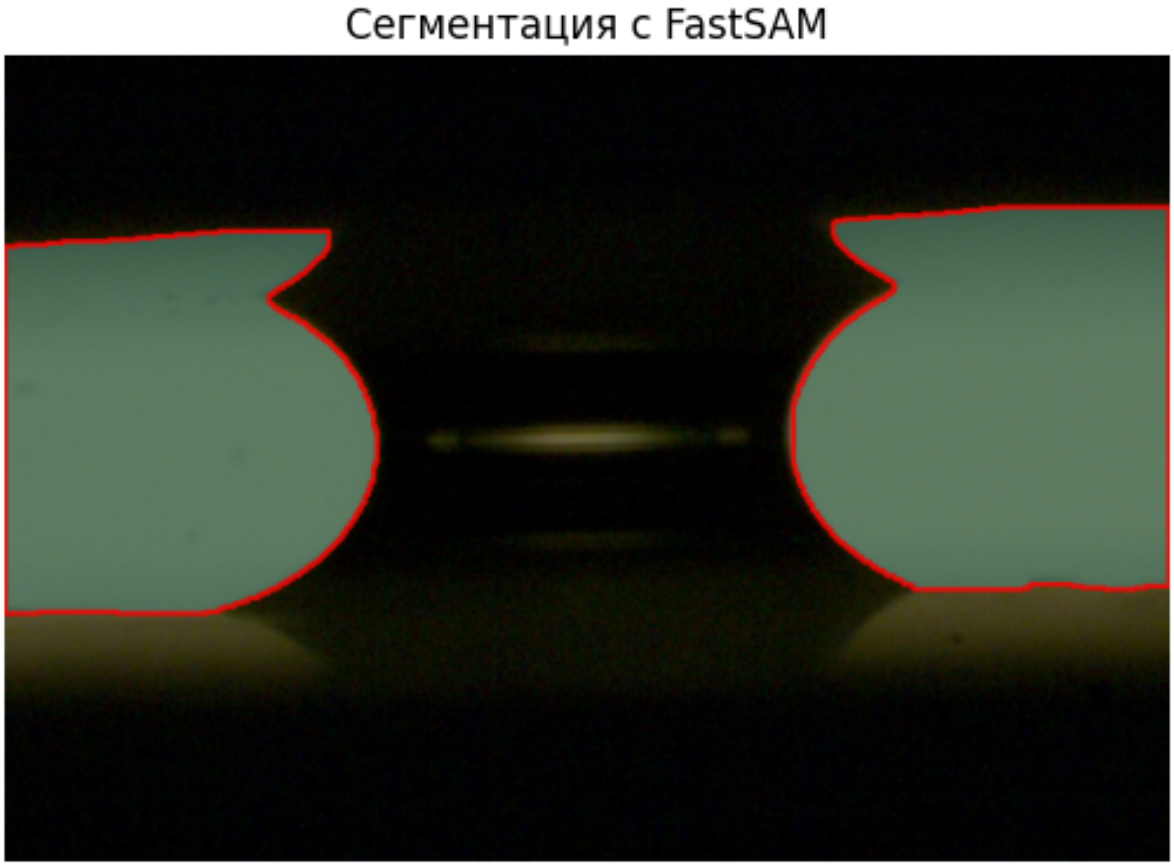


Fig. 8: Segmentation with "FastSAM"

4.2.5 Neural network "Segment Anything" with fine-tuning

The neural network, which is not trained to segment images of capillary bridges, does not perform accurately enough on some of them. To solve this problem, instead of training a completely new model, one can take a model that is pre-trained on a large number of different images and can already approximate objects well in a large portion of images it has not been trained on. Then this model can be trained on the specific images of capillary bridges, thereby improving the model's performance on this type of image without the need to train a completely new model.

For this purpose, a dataset is needed, which consists of different images of capillary bridges and the parts of the images with the correct segmented regions (masks). The creation of the masks involves manually marking the edges of the objects in each image in the dataset. The software VGG Image Annotator (VIA) ([13]) was used for creating the masks. Then the masks are converted to a black-and-white image. A visualization of a created mask can be seen in Fig. 9.

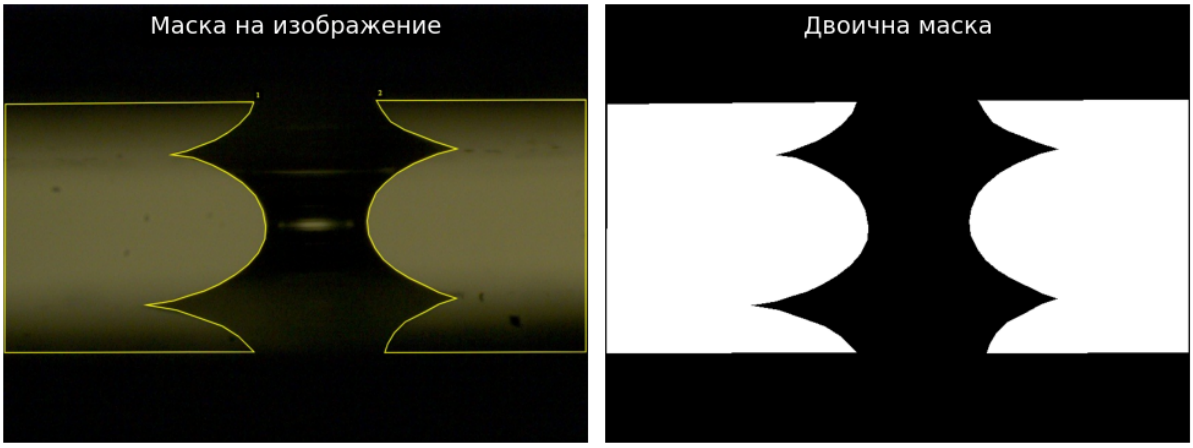


Fig. 9: Mask of an example image

After masks have been created for all images, the dataset is divided into two parts - a training set and a test set. The neural network is trained on the training set and then tested on the test set to check the effectiveness of the model on data it has not seen before. 115 images were used for training and 50 images for testing. The PyTorch library [14], an NVIDIA Titan V graphics card at the Faculty of Physics, and the Compute Unified Device Architecture (CUDA) library, which provides an application programming interface for NVIDIA graphics cards, are used.

The full fine-tuning process can be very slow, especially if the model has many parameters. A more efficient alternative to training is the use of an adapter. In this case, the model's parameters are "frozen" (they do not change during training). The adapters are added as separate layers to the model and are then trained. In this case, the LoRA (Low-Rank Adaptation) adapter ([15]) was chosen. In traditional fine-tuning (Fig. 10), the parameters W of a pre-trained neural network are changed to adapt to the new task. The changes made to W during fine-tuning are represented as ΔW, so the updated parameters can be expressed as $W + \Delta W$. Instead of modifying W directly, LoRA (Fig. 11) aims to decompose ΔW. This decomposition is a crucial step in reducing the computational costs associated with fine-tuning large models. With LoRA, ΔW is proposed to be represented as the product of two smaller, lower-rank matrices, A and B. The two matrices have specific dimensions $d \times r$ and $r \times d$. By specifying a rank $r < d$, we can reduce the size of the parameters and represent the task with a smaller rank. The product $B \times A$ results in a matrix with dimensions $d \times d$, so no information is lost, but the model learns a new representation during training. Thus, the matrix with the updated parameters W' becomes:

$$W' = W + B \cdot A$$

The matrix W remains "frozen" (it is not updated during training). The matrices B and A have lower dimensionality, and their product $B \cdot A$ represents a low-rank approximation of ΔW.

By using matrices A and B with a lower rank r, the number of trainable parameters is significantly reduced. For example, if W is a $d \times d$ matrix, traditionally updating W would involve d^2 parameters. Instead, using B and A with dimensions $d \times r$ and $r \times d$

respectively, the total number of parameters reduces to $2 \cdot d \cdot r$, which is much smaller when $r \ll d$.

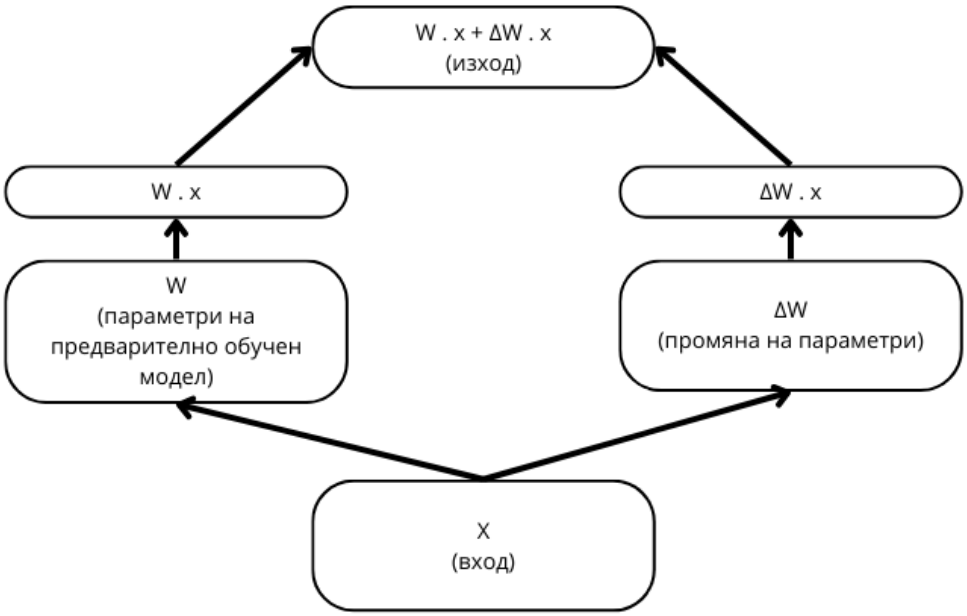


Fig. 10: Traditional fine-tuning. W does not change during training, ΔW changes during training

A visualization of the training error is illustrated in Fig. 12. The neural network is trained for 74 epochs. A learning rate of 1×10^{-4} and a LoRA rank of 512 are used. Training with these parameters takes about 1 hour.

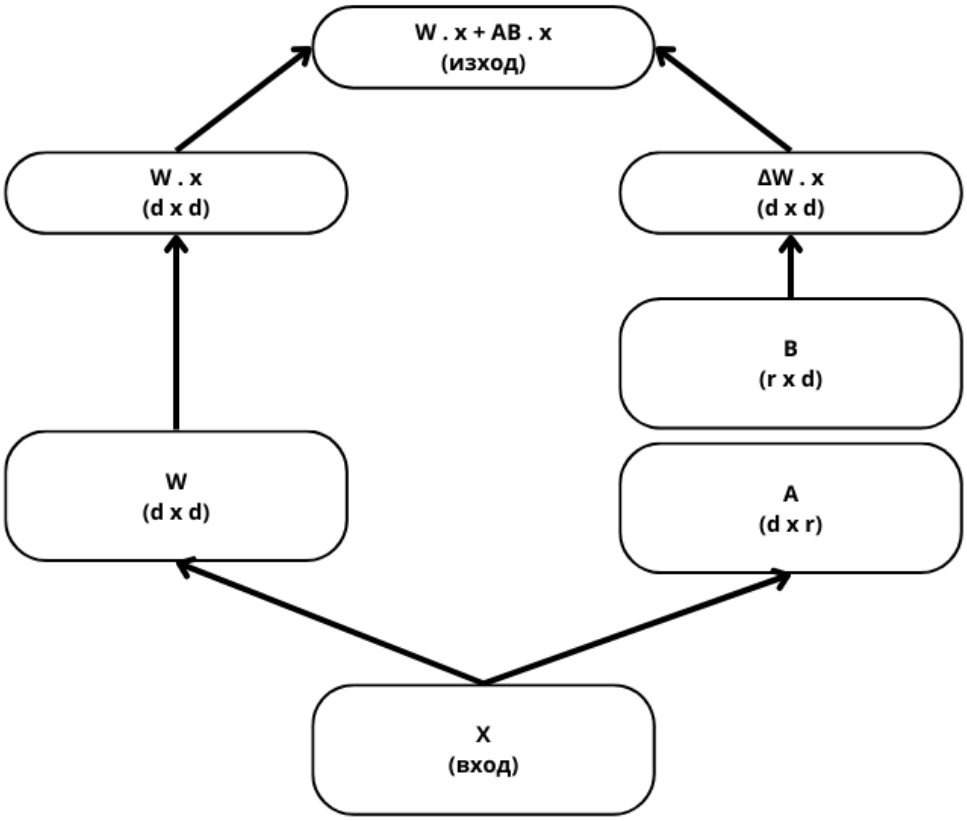
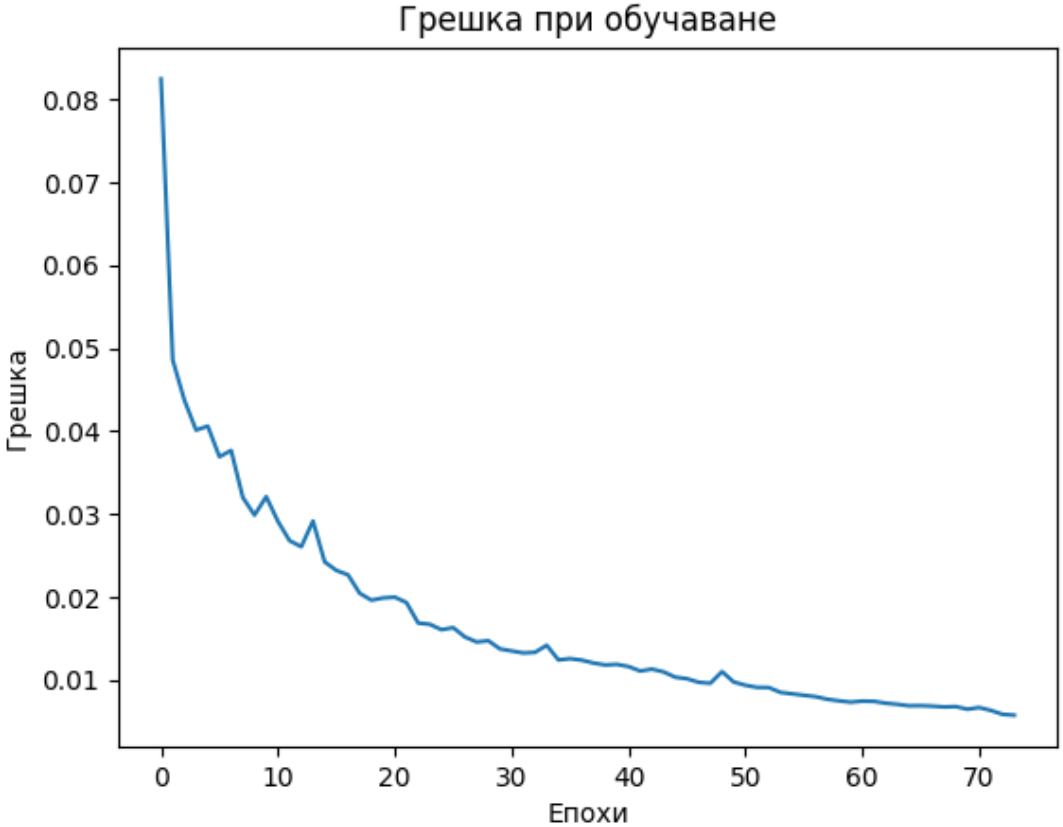


Fig. 11: Fine-tuning with LoRA. ΔW is decomposed into two matrices A and B, which have a lower dimensionality than $d \times d$



A visualization of an image after segmentation by a fine-tuned neural network can be seen in Fig. 13.

Fig. 13: Image after segmentation by a fine-tuned neural network

The measurement results after segmentation with a fine-tuned neural network (Chapter 6.2.3) are better than those after segmentation with thresholding using Otsu's method (Table 8). Despite the more accurate measurements, segmentation with a neural network is slower and does not allow for real-time processing of images from the camera.

5 Parameter Measurement

After the image has been segmented (the objects and the background are separated), the points necessary for measuring the parameters must be found.

5.1 Contour Extraction

The pixels on the boundary between the object and the background are called a contour. It is necessary to find the contours that describe the boundaries of the left object and the right object. The algorithm used to find the two contours (left and right) is Topological structural analysis by border following ([16]). The implementation from the OpenCV

library ([17]) is used. The algorithm can find more than two contours, so sorting by the area of the contours is applied, and the two contours with the largest area are taken. The result is a list of pixels that form the contour.

The left contour is denoted by:

$$C_l = \{(x_i, y_i) \mid i = 1, 2, \ldots, n_l\}$$

The right contour is denoted by:

$$C_r = \{(x'_j, y'_j) \mid j = 1, 2, \ldots, n_r\}$$

A visualization of an image after contour extraction can be seen in Fig. 14.

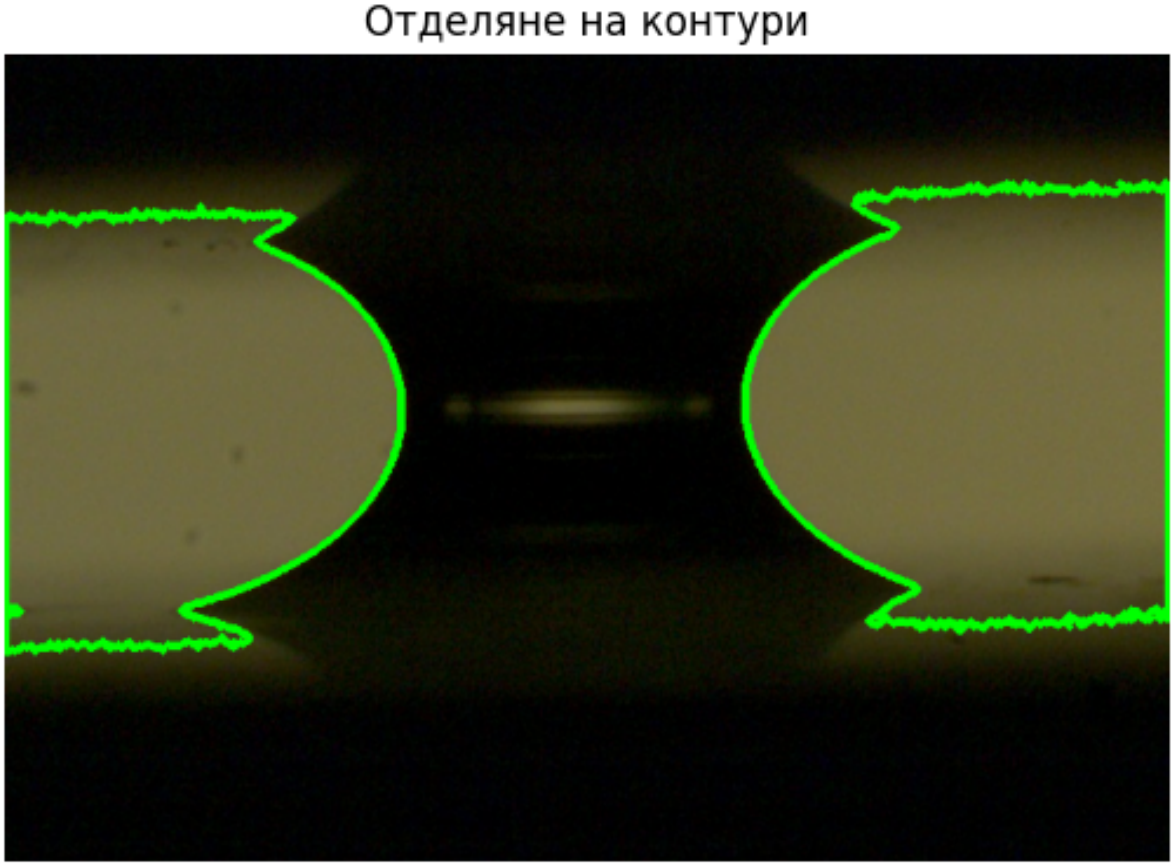


Fig. 14: Image after contour extraction

5.2 Neck Measurement

To measure the neck length, it is necessary to find the closest points from both sides (the rightmost point of the left contour and the leftmost point of the right contour).

For the contours $C_l = \{(x_i, y_i) \mid i = 1, 2, \ldots, n_l\}$ and $C_r = \{(x'_j, y'_j) \mid j = 1, 2, \ldots, n_r\}$, we can calculate the neck length as follows:

1. Calculate the Euclidean distance between all points:

$$d_{i,j} = \sqrt{(x_i - x'_j)^2 + (y_i - y'_j)^2}$$

where (x_i, y_i) are the points in C_l and (x'_j, y'_j) are the points in C_r.

2. Find the indices of the minimum distance:

$$(i, j) = \arg\min_{i,j} d_{i,j}$$

The closest points are $(x_i, y_i), (x'_j, y'_j)$

A visualization of the measurement can be seen in Fig. 15.

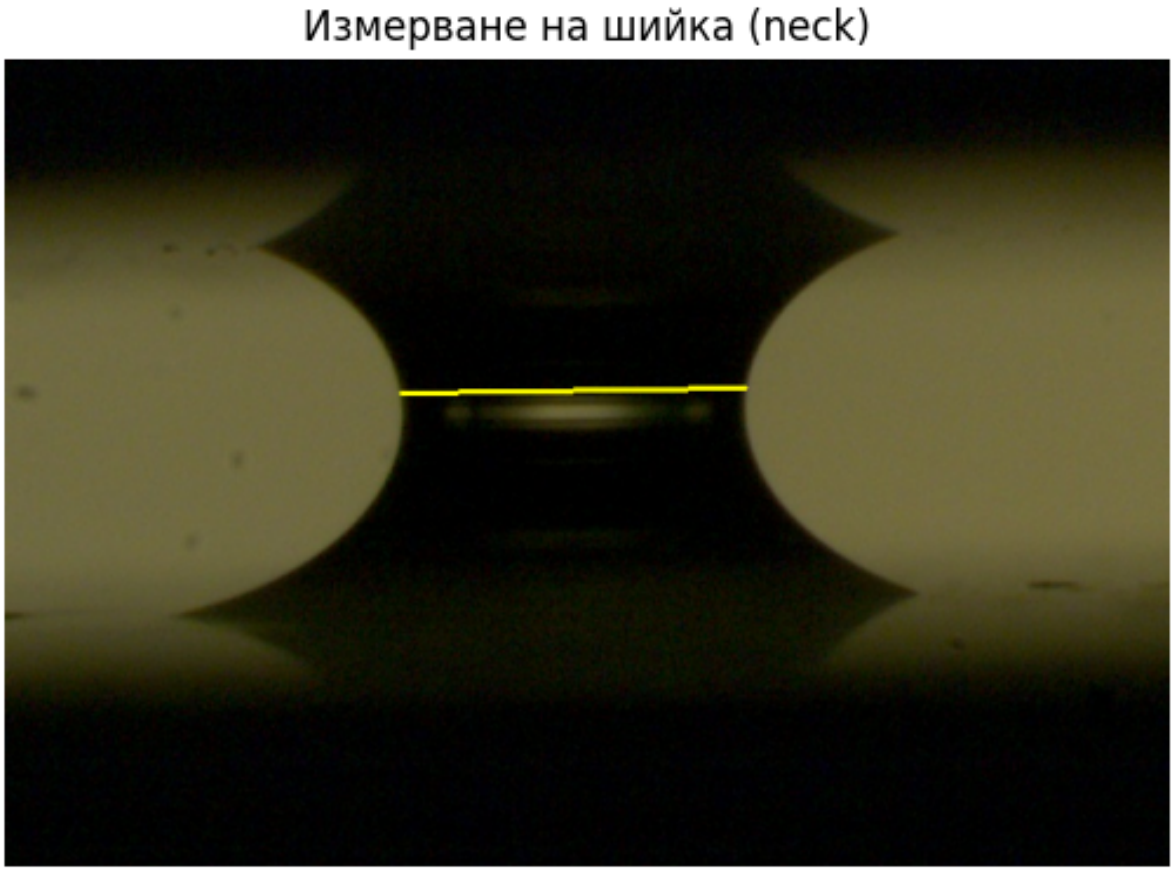


Fig. 15: Image with measured neck

5.3 Measurement of upper, lower, left, and right boundaries

The endpoints of the upper and lower boundaries are taken as the most concave points on the right side of the left contour and the most concave points on the left side of the right contour. To find them, the convex hull of the two contours is measured. The convex hull is the smallest set of points that encloses the entire contour, forming the smallest convex outer set. The implementation from the OpenCV library ([17]) is used, which employs the following algorithm [18].

For the two contours

$$C_l = \{(x_i, y_i) \mid i = 1, 2, \ldots, n_l\}$$

$$C_r = \{(x'_j, y'_j) \mid j = 1, 2, \ldots, n_r\}$$

we obtain the convex hulls

$$H_l = H(C_l)$$

$$H_r = H(C_r)$$

For C_l, the convex hull H_l is represented by the points $\{(x_i, y_i) \mid i = 1, 2, \ldots, n_l\}$.
For C_r, the convex hull H_r is represented by the points $\{(x'_j, y'_j) \mid j = 1, 2, \ldots, n_r\}$.

A visualization of a found convex hull can be seen in Fig. 16.

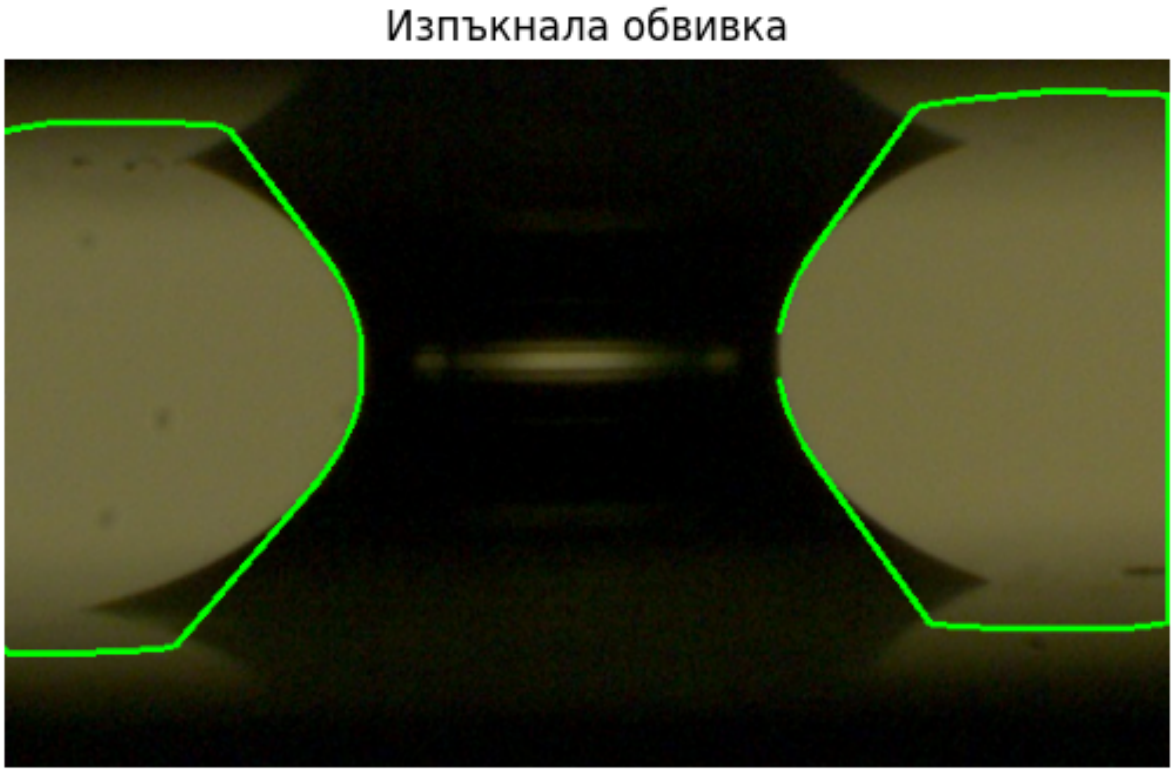


Fig. 16: Image with found convex hulls

Then, the convexity defects are found. They represent any deviation of the contour from its convex hull. The largest deviation is the most concave point. An algorithm from the OpenCV library ([17]) is used.

For the contours C_l, C_r and their convex hulls H_l, H_r, the convexity defects are obtained from the contours and the convex hulls

$$D_l = D(C_l, H_l)$$

$$D_r = D(C_r, H_r)$$

and contain a set of defects, where each defect d_i represents an ordered pair (f, d)

where
- f is the index of the point farthest from the convex hull
- d is the distance between the farthest point and the convex hull

A visualization of the convexity defects can be seen in Fig. 17.

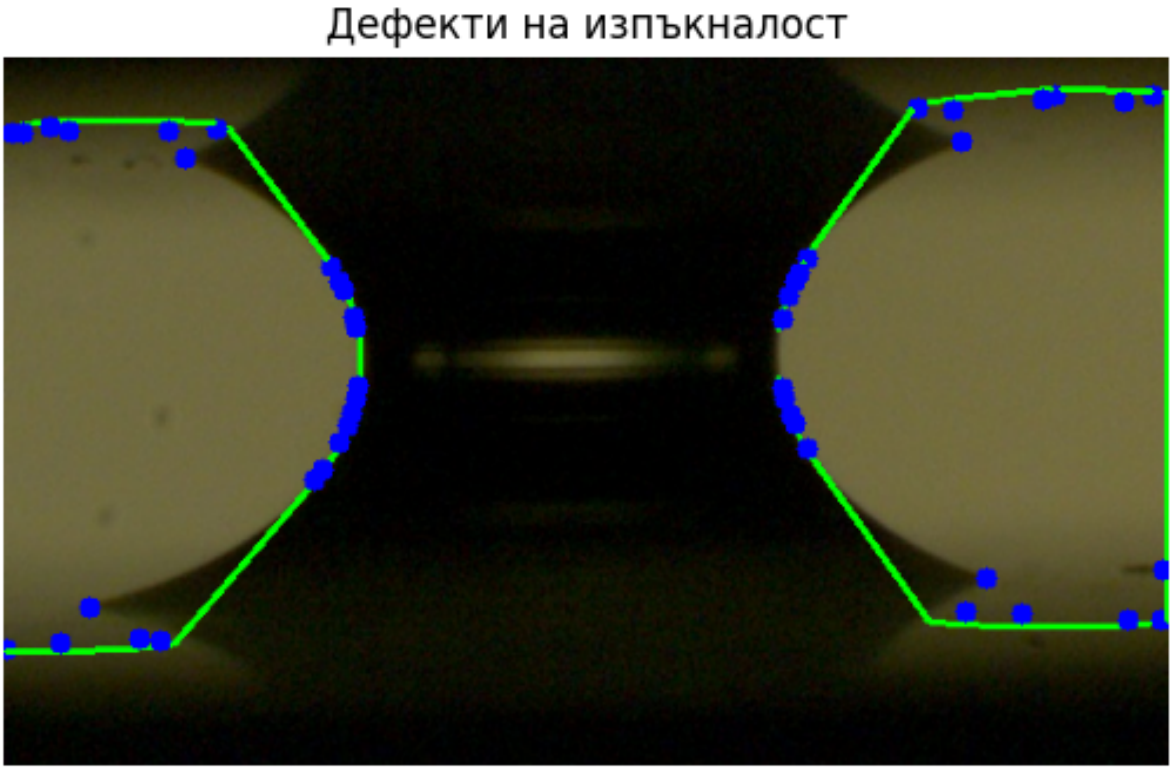


Fig. 17: Image with found convexity defects

Let p_u and p_d be the two most distant (most concave) points of the contour, and max be an ordered pair (max_0, max_1) of the two largest distances.

For each distance between the farthest point and the convex hull d_i in D, the corresponding point in the contour C is found by the index f_i and is denoted by C_{f_i}. Then the following steps are performed:

$$\text{if } d_i > max_0 :$$

$$\begin{cases} max_1 \leftarrow max_0 \\ max_0 \leftarrow d_i \\ p_u \leftarrow p_d \\ p_d \leftarrow C_{f_i} \end{cases}$$

$$\text{or if } d_i > max_1 :$$

$$\begin{cases} max_1 \leftarrow d_i \\ p_u \leftarrow C_{f_i} \end{cases}$$

The same steps are repeated for both contours, thus obtaining the four most concave points, through which the desired parameters can be found:

To find the lengths of the parameters U, D, L, R using the points U_l, U_r, D_l, D_r, where
- U is the length of the upper boundary
- D is the length of the lower boundary
- L is the length of the left boundary
- R is the length of the right boundary
- U_l is the upper point of the left contour
- U_r is the upper point of the right contour
- D_l is the lower point of the left contour
- D_r is the lower point of the right contour

the Euclidean distance is calculated for the respective boundaries

$$E(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

$$U = E(U_l, U_r)$$

$$D = E(D_l, D_r)$$

$$L = E(U_l, D_l)$$

$$R = E(U_r, D_r)$$

A visualization of the measured boundaries can be seen in Fig. 18.

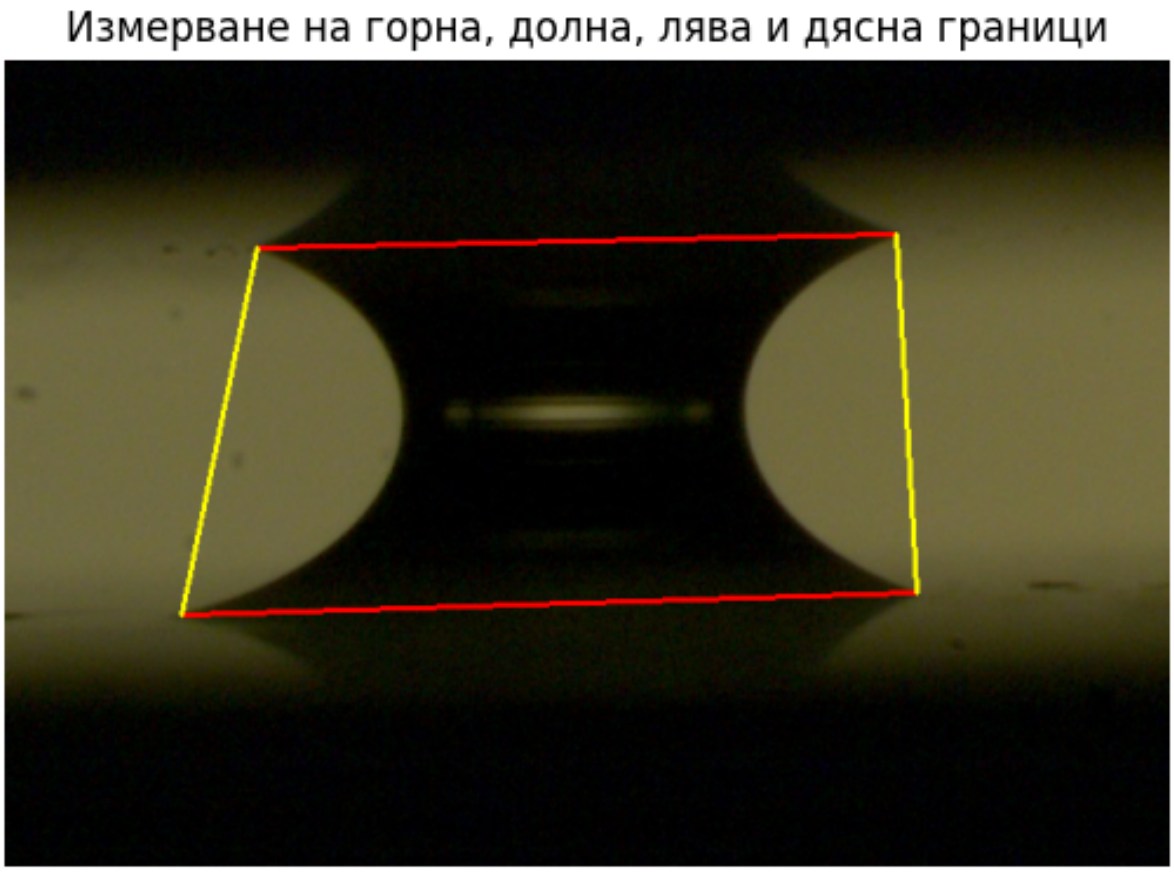


Fig. 18: Image with measured upper, lower, left, and right boundaries

5.4 Ellipse Fitting

For fitting the ellipses, an implementation from the OpenCV library [17] is used. First, the contours need to be found. The points of the contours are used to fit the corresponding ellipses. The implementation is based on the algorithm from [19]. Then, the major and minor axes of each of the two ellipses are found.

The fitted ellipse has the following parameters:

$$\mathrm{E} = ((x_c, y_c), (d_1, d_2), \theta)$$

where:
- (x_c, y_c) are the coordinates of the center of the ellipse
- d_1 and d_2 are the lengths of the major and minor axes
- θ is the angle of rotation of the ellipse with respect to the horizontal axis

The length of the major semi-axis is determined:

$$r_{\text{major}} = \frac{\max(d_1, d_2)}{2}$$

The angle is corrected:

$$\text{if } \theta > 90^\circ \text{ then } \theta = \theta - 90^\circ \text{ else } \theta = \theta + 90^\circ$$

The endpoints of the major axis are found:

$$\begin{aligned} x_1 &= x_c + r_{\text{major}} \cos(\theta) \\ y_1 &= y_c + r_{\text{major}} \sin(\theta) \\ x_2 &= x_c + r_{\text{major}} \cos(\theta + 180^\circ) \\ y_2 &= y_c + r_{\text{major}} \sin(\theta + 180^\circ) \end{aligned}$$

The length of the minor semi-axis is determined:

$$r_{\text{minor}} = \frac{\min(d_1, d_2)}{2}$$

The angle is corrected again:

$$\text{if } \theta > 90^\circ \text{ then } \theta = \theta - 90^\circ \text{ else } \theta = \theta + 90^\circ$$

The endpoints of the minor axis are found:

$$\begin{aligned} x_1 &= x_c + r_{\text{minor}} \cos(\theta) \\ y_1 &= y_c + r_{\text{minor}} \sin(\theta) \\ x_2 &= x_c + r_{\text{minor}} \cos(\theta + 180^\circ) \\ y_2 &= y_c + r_{\text{minor}} \sin(\theta + 180^\circ) \end{aligned}$$

A visualization of fitted ellipses and their axes can be seen in Fig. 19.

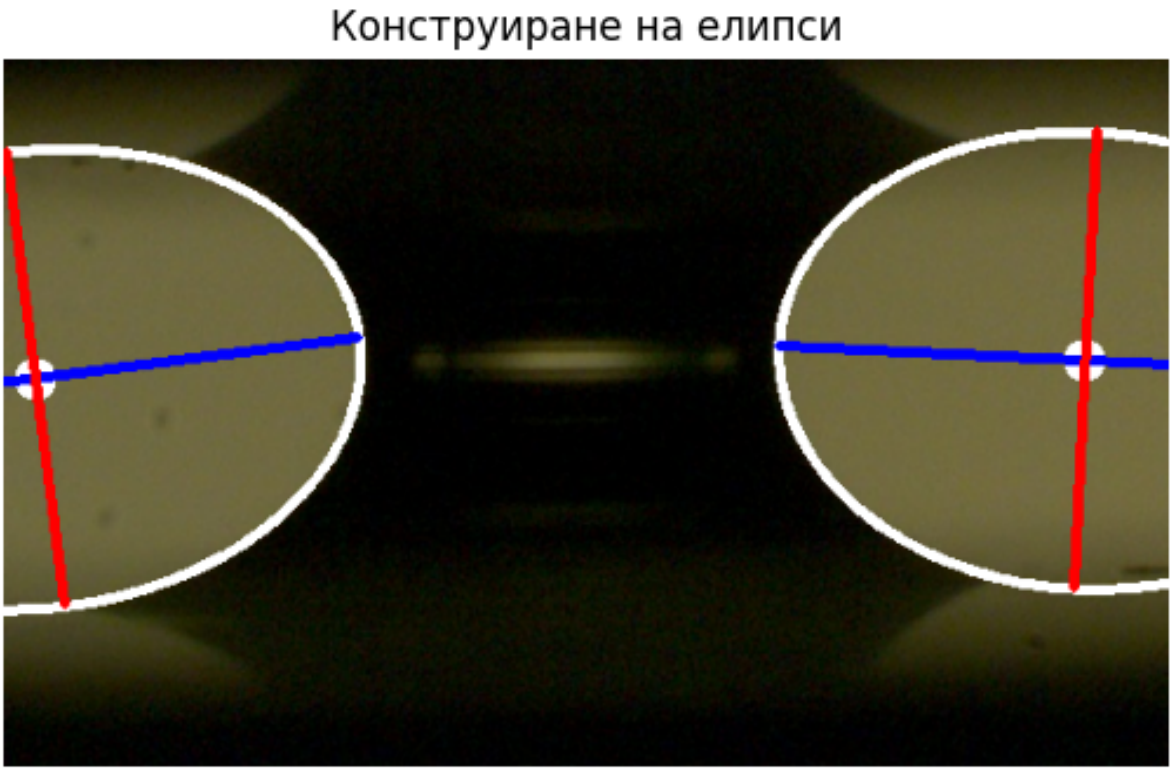


Fig. 19: Fitting ellipses and finding major and minor axes

5.5 Ratio Calculation

After we have all the necessary measurements, we can calculate the ratios. We calculate them using the following formulas, using the parameters defined in Chapter 5.2 and Chapter 5.3:

Calculating the base:

$$b = \frac{U + D}{2}$$

Calculating the height:

$$h = \frac{L + R}{2}$$

Calculating x:

$$x = \frac{b}{n}$$

Calculating y:

$$y = \frac{h}{n}$$

6 Experimental Results

The results from the program are compared with the manual processing of the images, and the error in the program's measurements is shown. The error is measured using the mean absolute percentage error (MAPE):

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{A_i - F_i}{A_i} \right| \times 100$$

where:
- n is the number of observations.
- A_i is the theoretical (actual) value for the i-th observation.
- F_i is the experimental (predicted) value for the i-th observation.

6.1 Results from Image Thresholding with Otsu's Method

The results from image thresholding with Otsu's method are not accurate enough for some of the images due to the limitations of the algorithm. However, the algorithm can process images from the camera in real-time, which allows for adjusting the experimental setup in such a way that the algorithm makes more accurate measurements.

6.1.1 Without Contrast Enhancement Filter

Table 7 shows the measurement results from image thresholding with Otsu's method without a contrast enhancement filter. The desired accuracy ($\sim 2\%$ error) is achieved for a large portion of the test images. The algorithm fails to correctly segment certain images due to software limitations (Chapter 6.5). This leads to high error percentages for some of the parameters and increases the mean error. The mean error is shown in Table 1.

Error	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
Mean	3.16 %	5.44 %	2.85 %	9.91 %	5.29 %

Tab. 1: Mean error for image thresholding with Otsu's method without a contrast enhancement filter

6.1.2 With Contrast Enhancement Filter

Table 8 shows the measurement results from image thresholding with Otsu's method with a contrast enhancement filter applied. Applying this filter leads to incorrect segmentation of some of the images due to failure to find the two separate contours, but it improves the accuracy for some of the parameters in certain images. The neck measurement is performed on the image before the filter is applied, which leads to identical results as those from segmentation with Otsu's method without a contrast enhancement filter. A value of 127 is used for the parameter γ, which is defined in Chapter 4.1.1. The mean error is shown in Table 2.

Error	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
Mean	4.63 %	9.45 %	2.85 %	18.51 %	4.75 %

Tab. 2: Mean error for image thresholding with Otsu's method with a contrast enhancement filter applied. Only successfully segmented images are included in the mean result.

6.2 Results from Neural Networks

Neural networks improve the measurements for some of the parameters but require more computational resources.

6.2.1 Baseline "Segment Anything" Model

Table 9 shows the measurement results from the baseline "Segment Anything" model. The baseline neural network manages to accurately segment a portion of the images without being previously trained on similar examples. The mean error for the neck is lower than with Otsu's method. Also, the difference in the error for measuring the upper and lower boundaries, compared to segmentation with Otsu's method, is less than 1 %. The mean error is shown in Table 3.

Error	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
Mean	6.76 %	21.33 %	1.20 %	24.24 %	5.93 %

Tab. 3: Mean error for the baseline "Segment Anything" model

6.2.2 Neural Network "FastSAM"

Table 10 shows the measurement results from the convolutional neural network "FastSAM". This neural network requires fewer computational resources than "Segment Anything" but does not achieve the same accuracy. The mean error is shown in Table 4.

Error	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
Mean	8.07 %	15.15 %	9.65 %	16.31 %	7.69 %

Tab. 4: Mean error for the neural network "FastSAM"

6.2.3 Neural Network with Fine-Tuning

Table 11 shows the measurement results from the neural network with fine-tuning. This model achieves the highest accuracy for 3 of the parameters - lower boundary, left boundary, and neck. The difference in error when measuring the right boundary is less than 1 % compared to the most accurate model for this parameter. The segmentation of the area where the upper boundary is located is not as accurate as the segmentation using Otsu's method. The mean error is shown in Table 5.

Error	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
Mean	1.93 %	3.35 %	1.03 %	10.49 %	9.10 %

Tab. 5: Results from the neural network with fine-tuning

6.3 Comparison of Results

Model	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
Otsu's Method (no filter)	3.16 %	5.44 %	2.85 %	**9.91** %	5.29 %
Otsu's Method (with filter)	4.63 %	9.45 %	2.85 %	18.51 %	**4.75** %
Baseline "Segment Anything"	6.76 %	21.33 %	1.20 %	24.24 %	5.93 %
"FastSAM"	8.07 %	15.15 %	9.65 %	16.31 %	7.69 %
Fine-tuned	**1.93 %**	**3.35 %**	**1.03** %	10.49 %	9.10 %

Tab. 6: Comparison of results

6.4 Detailed Results

The measurement results for each of the images in the test dataset are shown for each model.

Image	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
1	9.58 %	80.97 %	1.84 %	22.18 %	44.19 %
2	6.47 %	3.73 %	2.24 %	77.85 %	70.87 %
3	9.83 %	27.99 %	1.18 %	52.97 %	79.45 %
4	13.83 %	1.36 %	34.31 %	53.73 %	2.90 %
5	2.41 %	1.24 %	0.74 %	0.34 %	4.70 %
6	1.95 %	0.65 %	0.68 %	1.64 %	3.20 %
7	30.20 %	2.97 %	73.99 %	69.14 %	0.87 %
8	0.38 %	7.61 %	0.23 %	5.57 %	0.14 %
9	1.36 %	1.26 %	0.95 %	0.62 %	2.16 %
10	0.15 %	1.14 %	1.09 %	0.07 %	1.44 %
11	0.67 %	0.34 %	1.39 %	4.04 %	0.59 %
12	0.36 %	1.70 %	0.55 %	1.61 %	0.28 %
13	0.67 %	0.73 %	0.25 %	5.07 %	0.70 %
14	0.01 %	0.33 %	0.51 %	1.90 %	0.68 %
15	1.34 %	0.36 %	0.49 %	5.59 %	0.18 %
16	0.84 %	1.05 %	0.25 %	2.39 %	0.34 %
17	0.67 %	0.09 %	0.24 %	3.74 %	1.02 %
18	0.50 %	1.09 %	0.48 %	2.89 %	0.51 %
19	0.18 %	1.56 %	0.23 %	1.93 %	0.34 %
20	4.44 %	0.33 %	0.74 %	4.21 %	0.34 %
21	0.68 %	73.61 %	0.47 %	85.93 %	1.14 %
22	0.51 %	2.54 %	0.03 %	3.33 %	1.56 %
23	0.34 %	0.59 %	0.26 %	1.99 %	0.68 %
24	0.01 %	0.57 %	0.24 %	3.33 %	0.01 %
25	0.33 %	2.82 %	0.26 %	0.37 %	0.68 %
26	1.02 %	0.37 %	0.52 %	4.26 %	0.17 %
27	0.68 %	1.57 %	0.24 %	2.85 %	1.90 %
28	1.36 %	0.82 %	0.53 %	2.86 %	1.21 %
29	0.51 %	0.11 %	0.02 %	2.44 %	1.05 %
30	3.18 %	1.87 %	3.41 %	1.34 %	0.91 %
31	2.11 %	0.51 %	0.57 %	1.00 %	1.96 %
32	3.25 %	1.03 %	0.00 %	1.51 %	0.93 %
33	3.82 %	8.25 %	2.04 %	6.15 %	0.01 %
34	1.04 %	0.91 %	0.51 %	2.85 %	1.04 %
35	6.99 %	0.11 %	0.01 %	0.32 %	0.01 %
36	7.59 %	0.62 %	0.01 %	8.73 %	1.60 %
37	0.53 %	1.13 %	0.80 %	5.43 %	2.18 %
38	1.59 %	1.60 %	0.80 %	5.83 %	3.98 %
39	1.95 %	3.06 %	1.08 %	4.05 %	4.87 %
40	2.48 %	2.14 %	1.33 %	3.71 %	5.25 %
41	3.02 %	3.05 %	1.06 %	2.23 %	5.45 %
42	6.19 %	3.94 %	0.53 %	2.08 %	0.88 %
43	0.87 %	3.05 %	0.25 %	4.43 %	1.94 %
44	0.02 %	2.92 %	0.28 %	0.55 %	2.01 %
45	6.57 %	1.25 %	2.08 %	2.11 %	2.86 %
46	6.96 %	2.70 %	0.79 %	1.80 %	2.34 %
47	7.95 %	1.10 %	1.05 %	4.70 %	0.91 %
48	0.71 %	5.18 %	0.27 %	4.62 %	0.75 %
49	0.01 %	7.88 %	0.48 %	2.66 %	0.36 %
50	0.00 %	0.43 %	0.01 %	4.55 %	0.87 %
Mean	3.16 %	5.44 %	2.85 %	9.91 %	5.29 %

Tab. 7: Results from image thresholding with Otsu's method without a contrast enhancement filter

Image	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
1			1.84 %		
2			2.24 %		
3			1.18 %		
4			34.31 %		
5			0.74 %		
6			0.68 %		
7			73.99 %		
8			0.23 %		
9	6.55 %	5.04 %	0.95 %	93.95 %	15.28 %
10	10.79 %	7.71 %	1.09 %	0.92 %	12.87 %
11			1.39 %		
12			0.55 %		
13	0.51 %	1.65 %	0.25 %	4.57 %	0.35 %
14	0.18 %	0.77 %	0.51 %	0.97 %	0.17 %
15	0.72 %	0.09 %	0.49 %	97.26 %	5.33 %
16	1.12 %	0.55 %	0.25 %	23.94 %	1.86 %
17	1.28 %	0.39 %	0.24 %	25.82 %	0.36 %
18	1.36 %	0.67 %	0.48 %	2.90 %	2.25 %
19	0.16 %	1.11 %	0.23 %	1.93 %	0.68 %
20	2.61 %	0.36 %	0.74 %	10.03 %	0.87 %
21	5.24 %	9.22 %	0.47 %	26.81 %	1.85 %
22	3.09 %	1.97 %	0.03 %	3.32 %	3.99 %
23	1.19 %	0.17 %	0.26 %	2.94 %	2.24 %
24	1.42 %	0.53 %	0.24 %	25.26 %	0.54 %
25	0.49 %	27.41 %	0.26 %	5.05 %	1.06 %
26	3.41 %	1.91 %	0.52 %	1.82 %	2.97 %
27	2.91 %	0.64 %	0.24 %	2.36 %	4.68 %
28	3.24 %	0.79 %	0.53 %	3.30 %	4.34 %
29	2.77 %	0.13 %	0.02 %	1.93 %	4.39 %
30	6.01 %	3.79 %	3.41 %	1.83 %	4.75 %
31	4.94 %	0.01 %	0.57 %	0.01 %	4.47 %
32	6.67 %	1.53 %	0.00 %	1.95 %	5.04 %
33	7.83 %	7.28 %	2.04 %	7.05 %	4.39 %
34	3.41 %	0.39 %	0.51 %	77.61 %	5.88 %
35	6.29 %	0.62 %	0.01 %	0.83 %	0.01 %
36	3.04 %	9.77 %	0.01 %	90.90 %	7.17 %
37	3.40 %	2.03 %	0.80 %	3.59 %	5.47 %
38	5.86 %	2.40 %	0.80 %	4.09 %	6.88 %
39	9.26 %	1.90 %	1.08 %	3.19 %	8.12 %
40	23.64 %	90.45 %	1.33 %	3.75 %	13.04 %
41	37.68 %	60.98 %	1.06 %	97.21 %	33.13 %
42	0.88 %	3.92 %	0.53 %	0.48 %	1.94 %
43	0.70 %	3.59 %	0.25 %	2.56 %	3.36 %
44	0.17 %	0.82 %	0.28 %	0.89 %	3.30 %
45	9.65 %	97.54 %	2.08 %	52.04 %	5.45 %
46	5.40 %	12.85 %	0.79 %	28.95 %	7.25 %
47	0.57 %	2.52 %	1.05 %	15.60 %	0.54 %
48	0.53 %	6.48 %	0.27 %	5.06 %	1.51 %
49	0.17 %	7.83 %	0.48 %	3.60 %	1.09 %
50	0.18 %	0.03 %	0.01 %	4.10 %	1.23 %
Mean	4.63 %	9.45 %	2.85 %	18.51 %	4.75 %

Tab. 8: Results from image thresholding with Otsu's method with a contrast enhancement filter applied. Empty cells in the table indicate failed image segmentation. Only successfully segmented images are included in the mean result.

Image	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
1	2.54 %	1.60 %	0.79 %	0.25 %	0.21 %
2	0.98 %	1.15 %	0.74 %	4.97 %	1.54 %
3	2.47 %	3.18 %	2.40 %	1.39 %	0.89 %
4	19.41 %	12.51 %	1.71 %	1.89 %	0.20 %
5	34.64 %	11.11 %	1.28 %	11.89 %	3.02 %
6	19.13 %	10.20 %	1.45 %	1.64 %	0.83 %
7	16.61 %	15.33 %	0.65 %	0.32 %	1.06 %
8	16.21 %	32.90 %	1.20 %	40.39 %	6.66 %
9	10.88 %	0.81 %	2.19 %	9.64 %	0.07 %
10	7.03 %	1.24 %	2.61 %	36.96 %	8.46 %
11	3.00 %	1.16 %	2.26 %	5.30 %	0.07 %
12	2.27 %	1.76 %	2.84 %	0.04 %	2.27 %
13	2.58 %	0.69 %	0.50 %	2.13 %	0.35 %
14	8.51 %	45.19 %	2.06 %	41.92 %	3.63 %
15	1.78 %	1.88 %	0.50 %	49.81 %	9.65 %
16	1.19 %	1.85 %	1.00 %	0.55 %	2.41 %
17	9.39 %	1.31 %	0.75 %	40.74 %	11.26 %
18	5.46 %	41.41 %	1.01 %	47.85 %	4.17 %
19	1.03 %	0.92 %	0.50 %	0.43 %	3.63 %
20	0.86 %	2.76 %	0.50 %	0.32 %	3.46 %
21	0.29 %	1.35 %	0.75 %	82.39 %	8.74 %
22	3.95 %	39.46 %	2.02 %	43.26 %	2.09 %
23	4.81 %	45.94 %	1.28 %	45.98 %	3.81 %
24	1.88 %	1.60 %	0.81 %	51.16 %	16.08 %
25	7.34 %	46.28 %	1.02 %	91.98 %	1.21 %
26	4.56 %	4.19 %	1.27 %	46.33 %	11.07 %
27	1.04 %	4.32 %	1.77 %	7.50 %	6.92 %
28	0.69 %	5.81 %	2.04 %	19.75 %	9.61 %
29	10.58 %	39.07 %	2.05 %	38.20 %	4.40 %
30	14.23 %	94.53 %	2.89 %	21.86 %	23.81 %
31	0.83 %	37.81 %	1.90 %	0.49 %	8.63 %
32	2.64 %	46.41 %	1.11 %	1.13 %	16.09 %
33	0.17 %	53.57 %	0.01 %	49.68 %	7.87 %
34	27.18 %	37.92 %	0.28 %	39.86 %	9.77 %
35	8.35 %	37.45 %	0.00 %	89.96 %	12.70 %
36	7.90 %	72.33 %	0.01 %	11.39 %	13.11 %
37	11.15 %	89.03 %	0.28 %	13.98 %	8.55 %
38	16.18 %	80.11 %	0.55 %	92.99 %	17.07 %
39	13.56 %	6.60 %	0.81 %	22.51 %	12.99 %
40	10.89 %	68.67 %	1.65 %	42.49 %	11.12 %
41	8.02 %	42.55 %	2.18 %	85.60 %	10.56 %
42	3.18 %	3.40 %	0.52 %	0.58 %	3.02 %
43	0.90 %	0.87 %	0.79 %	3.15 %	1.96 %
44	2.33 %	0.51 %	1.34 %	0.86 %	2.21 %
45	0.19 %	0.90 %	1.56 %	0.80 %	3.40 %
46	1.42 %	1.54 %	0.25 %	1.06 %	0.54 %
47	0.71 %	2.89 %	0.52 %	3.03 %	0.56 %
48	1.63 %	3.37 %	1.93 %	0.97 %	0.36 %
49	3.01 %	7.34 %	0.52 %	3.29 %	0.55 %
50	2.47 %	1.85 %	0.80 %	1.29 %	3.70 %
Mean	6.76 %	21.33 %	1.20 %	24.24 %	5.93 %

Tab. 9: Results from the baseline "Segment Anything" model

Image	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
1	0.61 %	1.17 %	6.51 %	1.10 %	3.10 %
2	2.22 %	3.53 %	1.12 %	3.53 %	0.39 %
3	28.59 %	8.53 %	54.29 %	1.33 %	36.77 %
5	2.41 %	1.97 %	0.74 %	0.74 %	4.70 %
6	1.27 %	2.67 %	1.70 %	4.11 %	3.53 %
7	15.96 %	24.47 %	1.13 %	1.53 %	0.25 %
8	1.16 %	11.59 %	0.70 %	1.60 %	1.64 %
9	3.59 %	1.26 %	0.43 %	0.44 %	4.40 %
10	0.15 %	2.60 %	1.99 %	1.45 %	2.56 %
11	0.64 %	1.07 %	1.68 %	1.18 %	1.85 %
12	2.68 %	0.60 %	0.05 %	1.04 %	3.66 %
13	44.42 %	92.14 %	93.35 %	93.54 %	39.28 %
14	6.13 %	0.85 %	0.25 %	91.54 %	11.62 %
15	1.68 %	2.55 %	0.69 %	4.22 %	1.90 %
16	2.01 %	3.78 %	0.52 %	0.80 %	3.08 %
17	5.90 %	77.65 %	0.01 %	55.10 %	0.13 %
18	0.68 %	1.37 %	1.50 %	2.43 %	0.34 %
19	58.08 %	74.15 %	95.50 %	21.73 %	77.89 %
20	0.69 %	0.80 %	0.51 %	2.99 %	0.34 %
21	46.04 %	157.35 %	83.04 %	15.10 %	14.62 %
22	0.70 %	2.39 %	1.77 %	2.87 %	0.87 %
23	0.50 %	0.62 %	1.02 %	3.41 %	1.37 %
24	0.18 %	1.26 %	0.29 %	3.32 %	0.17 %
25	10.03 %	36.15 %	1.76 %	0.75 %	3.70 %
26	0.33 %	0.82 %	1.05 %	3.31 %	0.34 %
27	0.18 %	2.26 %	1.52 %	0.34 %	2.59 %
28	0.51 %	2.66 %	1.78 %	2.34 %	1.91 %
29	0.53 %	5.12 %	2.05 %	0.25 %	1.92 %
30	8.31 %	0.39 %	2.89 %	28.02 %	6.19 %
31	1.41 %	0.00 %	1.69 %	0.85 %	2.86 %
32	4.15 %	3.07 %	0.28 %	3.44 %	2.05 %
33	8.03 %	5.78 %	1.44 %	11.03 %	0.74 %
34	8.26 %	0.83 %	1.82 %	59.41 %	0.80 %
35	6.99 %	0.11 %	0.01 %	0.32 %	0.01 %
36	7.59 %	0.62 %	0.01 %	8.73 %	1.60 %
37	5.03 %	1.79 %	0.84 %	41.48 %	11.26 %
38	1.06 %	2.24 %	0.17 %	5.32 %	2.89 %
39	8.85 %	32.17 %	1.62 %	85.02 %	4.86 %
40	0.55 %	2.54 %	0.73 %	80.51 %	4.02 %
41	7.82 %	76.13 %	1.08 %	34.78 %	4.10 %
42	6.19 %	3.94 %	0.53 %	2.08 %	0.88 %
43	0.70 %	0.98 %	1.34 %	0.35 %	2.64 %
44	0.35 %	2.27 %	1.61 %	0.73 %	2.92 %
45	2.83 %	1.30 %	2.60 %	1.15 %	6.98 %
46	0.89 %	1.73 %	0.80 %	1.88 %	4.87 %
47	74.35 %	71.09 %	95.79 %	97.09 %	82.44 %
48	2.15 %	3.16 %	0.37 %	7.97 %	2.46 %
49	0.88 %	8.59 %	0.29 %	4.87 %	3.64 %
50	1.40 %	2.47 %	0.01 %	2.14 %	3.51 %
Mean	8.07 %	15.15 %	9.65 %	16.31 %	7.69 %

Tab. 10: Results from the neural network "FastSAM"

Image	Lower boundary	Left boundary	Neck	Right boundary	Upper boundary
1	2.66 %	1.32 %	0.03 %	0.91 %	0.06 %
2	0.02 %	2.10 %	0.46 %	4.68 %	1.82 %
3	2.31 %	3.15 %	1.57 %	0.51 %	1.04 %
4	1.20 %	1.33 %	2.38 %	3.28 %	2.70 %
5	0.27 %	2.10 %	2.09 %	11.81 %	21.87 %
6	1.04 %	0.82 %	1.70 %	13.44 %	20.12 %
7	1.70 %	1.12 %	1.32 %	2.34 %	1.38 %
8	1.92 %	5.85 %	1.18 %	8.89 %	0.08 %
9	2.52 %	1.13 %	1.94 %	0.99 %	0.53 %
10	3.02 %	0.55 %	2.29 %	2.52 %	1.68 %
11	7.29 %	52.66 %	1.39 %	1.83 %	2.14 %
12	12.64 %	31.14 %	2.96 %	6.31 %	14.74 %
13	3.43 %	1.76 %	0.01 %	11.00 %	8.03 %
14	1.54 %	2.32 %	1.79 %	11.38 %	8.90 %
15	0.16 %	3.54 %	0.51 %	19.17 %	10.04 %
16	0.67 %	0.87 %	1.00 %	15.59 %	7.38 %
17	0.85 %	0.04 %	0.25 %	8.28 %	6.87 %
18	1.54 %	0.71 %	0.51 %	26.50 %	11.88 %
19	0.52 %	0.96 %	0.75 %	13.58 %	8.07 %
20	2.05 %	2.16 %	0.26 %	16.13 %	9.63 %
21	0.00 %	0.87 %	0.29 %	8.60 %	8.10 %
22	1.56 %	1.99 %	1.26 %	20.61 %	11.07 %
23	1.72 %	0.53 %	1.03 %	13.73 %	8.59 %
24	1.71 %	2.98 %	0.76 %	15.52 %	13.81 %
25	3.57 %	1.34 %	0.78 %	19.70 %	5.45 %
26	2.05 %	1.35 %	1.02 %	1.31 %	4.90 %
27	1.89 %	2.98 %	1.27 %	21.85 %	15.25 %
28	1.37 %	0.02 %	1.78 %	23.69 %	15.60 %
29	3.30 %	1.59 %	1.80 %	22.91 %	15.23 %
30	2.13 %	0.64 %	2.10 %	16.32 %	19.61 %
31	1.60 %	1.16 %	1.92 %	2.19 %	10.74 %
32	0.37 %	0.01 %	0.52 %	0.43 %	10.66 %
33	0.74 %	8.52 %	1.73 %	7.85 %	5.74 %
34	3.14 %	1.27 %	0.27 %	15.65 %	9.62 %
35	0.53 %	0.47 %	0.81 %	14.07 %	11.43 %
36	0.17 %	0.09 %	0.83 %	14.53 %	10.72 %
37	3.23 %	0.93 %	0.55 %	15.07 %	11.82 %
38	1.61 %	0.46 %	0.55 %	7.96 %	11.64 %
39	1.79 %	0.96 %	0.27 %	7.31 %	11.24 %
40	1.96 %	2.32 %	0.55 %	7.96 %	11.44 %
41	1.43 %	1.40 %	0.55 %	6.14 %	11.87 %
42	0.36 %	1.09 %	0.84 %	10.30 %	10.03 %
43	0.89 %	0.29 %	1.06 %	11.91 %	9.83 %
44	2.16 %	1.41 %	1.37 %	3.35 %	9.74 %
45	0.89 %	0.86 %	1.04 %	4.53 %	7.01 %
46	1.25 %	0.50 %	0.29 %	6.57 %	9.25 %
47	0.89 %	2.91 %	0.26 %	12.57 %	9.23 %
48	3.06 %	4.41 %	0.77 %	8.85 %	6.26 %
49	1.60 %	7.00 %	0.01 %	4.79 %	10.95 %
50	2.11 %	1.35 %	1.06 %	18.95 %	9.17 %
Mean	1.93 %	3.35 %	1.03 %	10.49 %	9.10 %

Tab. 11: Results from the neural network with fine-tuning

6.5 Software Limitations

The software approaches the desired accuracy ($\sim 2\%$ error compared to manual measurements) for a large portion of the images but does not achieve sufficient accuracy for some of them.

The main step in the overall algorithm that is crucial for the accurate measurement of the parameters is the image segmentation. Finding the contours, the convex hull, and the convexity defects (Chapter 5), which are needed to find the start and end points of the parameters, largely depends on the segmentation. The main limitation in segmentation is the low pixel intensity in the areas of the image that contain the start and end points of the parameters. This problem can be solved by increasing the camera's exposure time or by applying a contrast enhancement filter. Another limitation in segmentation is reflections that are located near the lower or upper boundary and whose intensity is similar to the intensity of the objects to be segmented. In some cases, the algorithm cannot separate the left and right objects, and in other cases, it cannot find the start and end points of the lower or upper boundary. An example of an image with such characteristics, which pose a problem for the algorithm, is in Fig. 20.

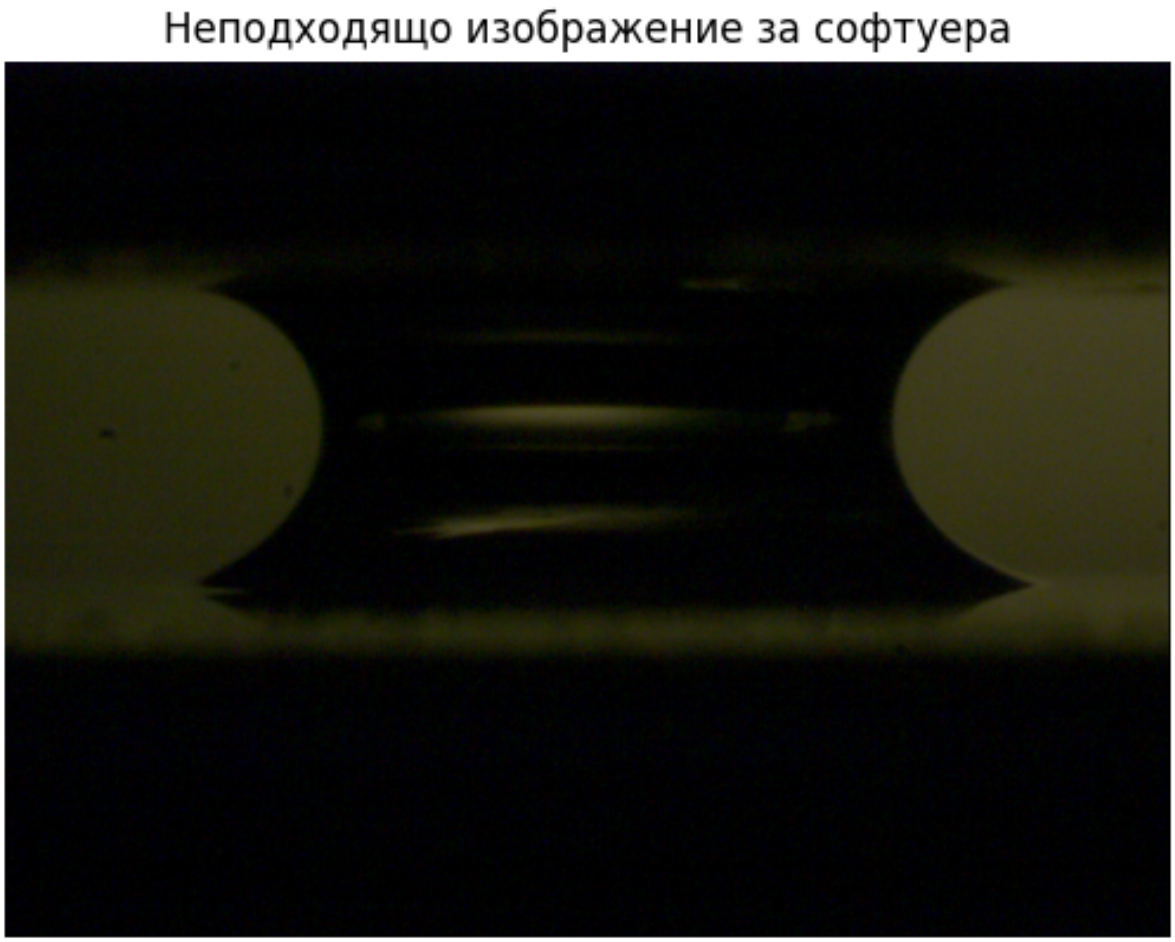


Fig. 20: Unsuitable image for the algorithm

The algorithm achieves the desired accuracy for images that have a high contrast between the objects and the background (high intensity for the objects and low intensity

for the background). Despite the presence of multiple reflections in the center of the image or near the lower and upper boundaries, the algorithm copes with the segmentation of the objects if the reflections do not overlap with the objects or if they overlap but the difference in intensity between them and the objects is high. An example of an image with such characteristics, which the algorithm processes correctly, is in Fig. 21.

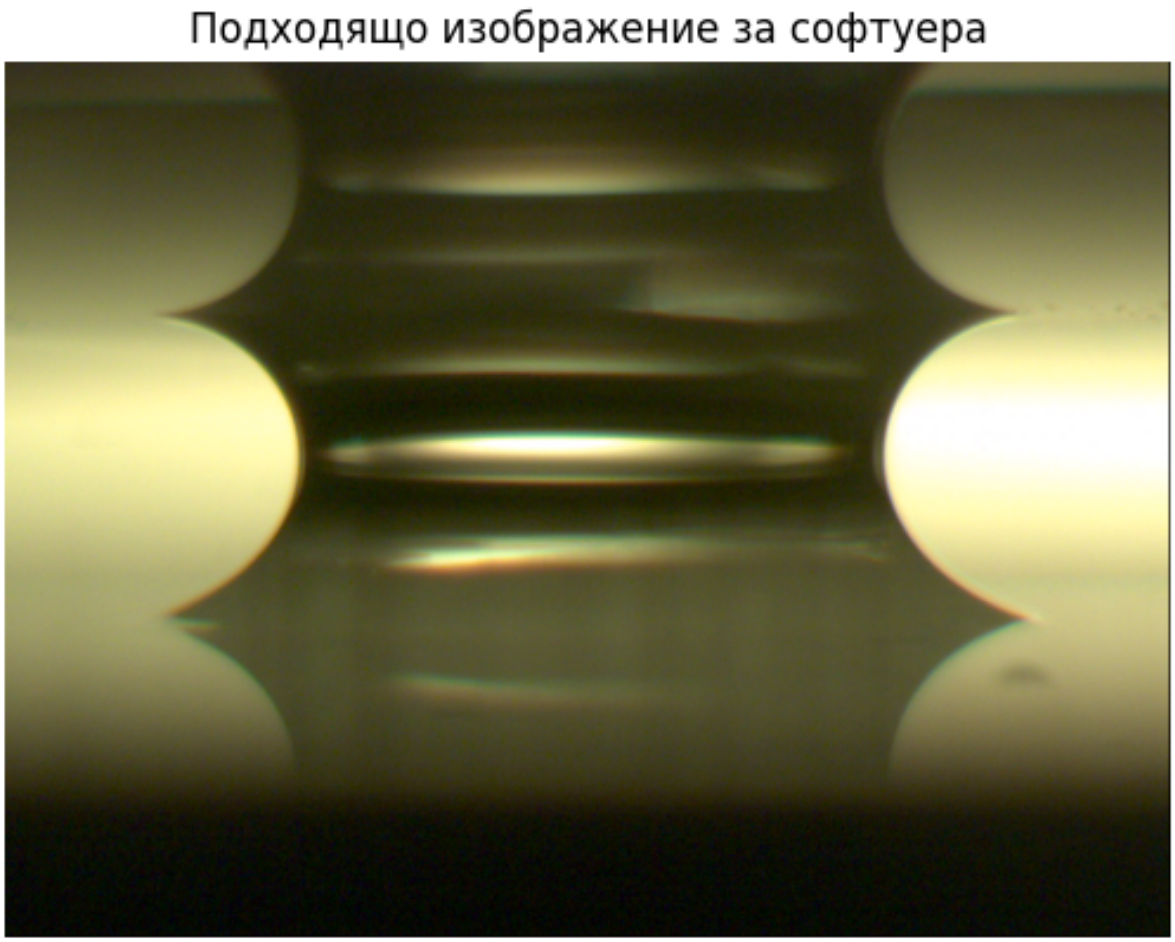


Fig. 21: Suitable image for the algorithm

7 Software

The software is written entirely in Python. A module for testing and comparing different models for parameter measurement has been created, which includes adding, removing, and changing the test images on which the models are tested. This module compares the model's measurements with the manual measurements, which are entered in advance for each image. The test results are saved as images with the measured parameters, as well as the measurements themselves in the form of a table. Functionalities for comparing the measured parameters between the different models have been added. The software can work with standard USB cameras as well as with Basler cameras.

7.1 Used Libraries

The main libraries used are "NumPy" [20], "OpenCV" [17], and "Matplotlib" [21]. The measured parameters are saved in Excel files using the "openpyxl" library. The "pypylon" library is used for working with the Basler camera. For training and using the neural networks, the "PyTorch" library [14] is used. An executable file is created using the "PyInstaller" library, which compiles the Python scripts and the libraries needed by the program.

7.2 Graphical Interface

The graphical interface provides a more convenient and intuitive way to use the program. It is implemented using the Tkinter library [22]. There are options for real-time video processing, saving individual frames from the video and their measurement results in an Excel file, recording video, processing automatically separated frames from the video, adding a contrast enhancement filter, and changing the exposure time and frame rate. A visualization of the interface can be seen in Fig. 22.

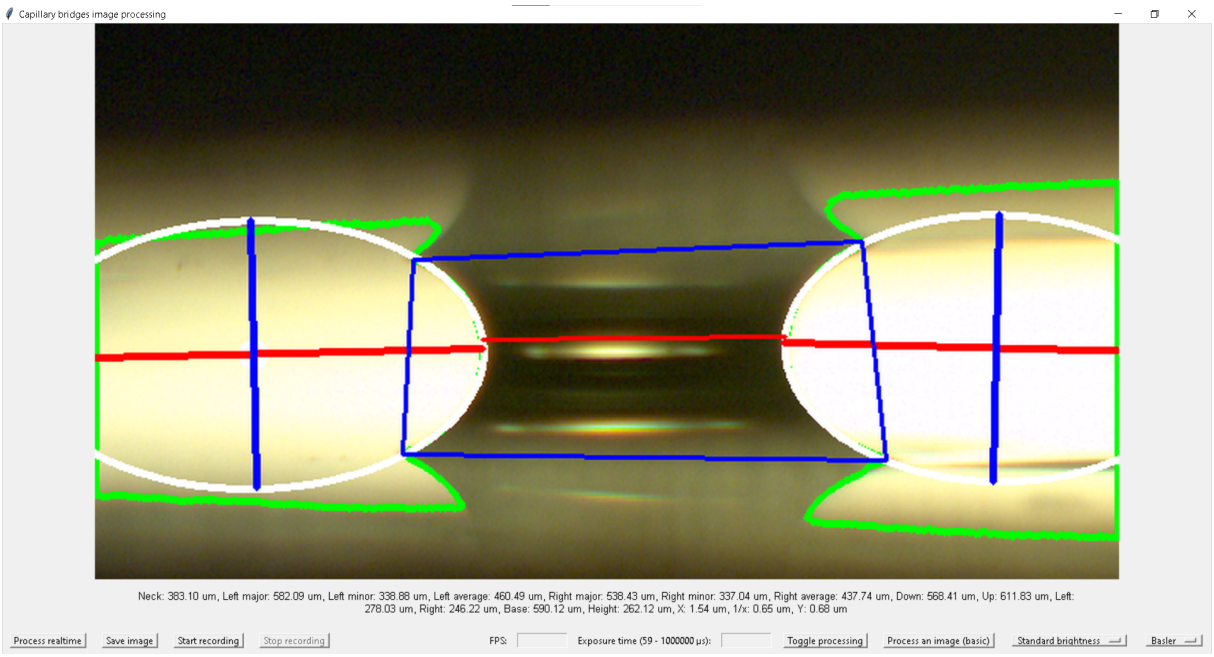


Fig. 22: Graphical interface of the program

8 Conclusion

Two main models for the segmentation of objects in images of capillary bridges were examined and compared - thresholding with Otsu's method and the "Segment Anything" neural network. The measurements after segmentation with a fine-tuned neural network are more accurate for most parameters, but this model does not allow for real-time image processing and requires more memory, unlike segmentation via thresholding with Otsu's method. A possible improvement in the accuracy of segmentation with a neural network is training on more images. Preserving accuracy and overcoming the requirement for more

memory and computational resources of the neural network can be achieved by applying the fine-tuning method to a smaller and faster model with fewer parameters, such as "FastSAM". In addition to the segmentation models, as part of the thesis, an algorithm was developed that automatically measures the desired parameters after the image has been segmented. A graphical user interface has also been developed, through which the models and algorithms can be easily used.

As a result of this thesis, software has been created that can automate the manual measurement of the parameters of capillary bridges from experimental images with an accuracy close to manual measurements. This facilitates work with the experimental setup, helps to control and optimize the experiment, and reduces the time for processing data from the experiment.

The code is uploaded to `https://github.com/petkokp/capillary-bridges-image-analysis`

References

[1] R. Szeliski, *Computer vision: algorithms and applications*. Springer Nature, 2022.

[2] N. Borisova, P. Petkov, and H. Iliev, "Study of binary liquid capillary bridges stretched between two solid flat surfaces," 10 2022.

[3] N. Otsu *et al.*, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[4] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, "Segment anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4015–4026.

[5] X. Zhao, W. Ding, Y. An, Y. Du, T. Yu, M. Li, M. Tang, and J. Wang, "Fast segment anything," *arXiv preprint arXiv:2306.12156*, 2023.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[9] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 16 000–16 009.

[10] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[11] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[12] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "Yolact: Real-time instance segmentation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 9157–9166.

[13] A. Dutta and A. Zisserman, "The VIA annotation software for images, audio and video," in *Proceedings of the 27th ACM International Conference on Multimedia*, ser. MM '19. New York, NY, USA: ACM, 2019. [Online]. Available: https://doi.org/10.1145/3343031.3350535

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[15] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[16] S. Suzuki, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[17] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[18] J. Sklansky, "Finding the convex hull of a simple polygon," *Pattern Recognition Letters*, vol. 1, no. 2, pp. 79–83, 1982.

[19] A. W. Fitzgibbon, R. B. Fisher *et al.*, *A buyer's guide to conic fitting.* Citeseer, 1996.

[20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[21] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[22] F. Lundh, "An introduction to tkinter," *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.