Adaptive Vision

Intuitive

Powerful

Adaptable

Future Processing Sp. z o. o.
ul. Bojkowska 37a, 44-100 Gliwice, Polska
Regon: 278263445 VatID: PL6342532128
Tel. +48 32 461 23 30
E-mail: info@adaptive-vision.com

Adaptive Vision Studio 4.10

Introduction

Created: 24.08.2018

Product version: 4.10.2.62669

Table of content:

Complexity Levels

Introduction

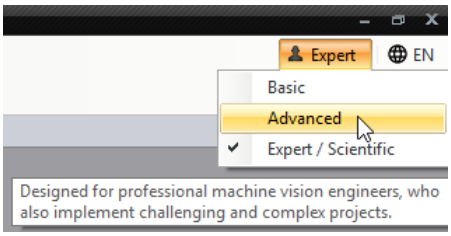
Adaptive Vision Studio has three levels of feature complexity. These levels are defined with regard to complexity of the project being created – at lower levels some features are hidden, so that the user is able to create projects more easily.

Available Levels

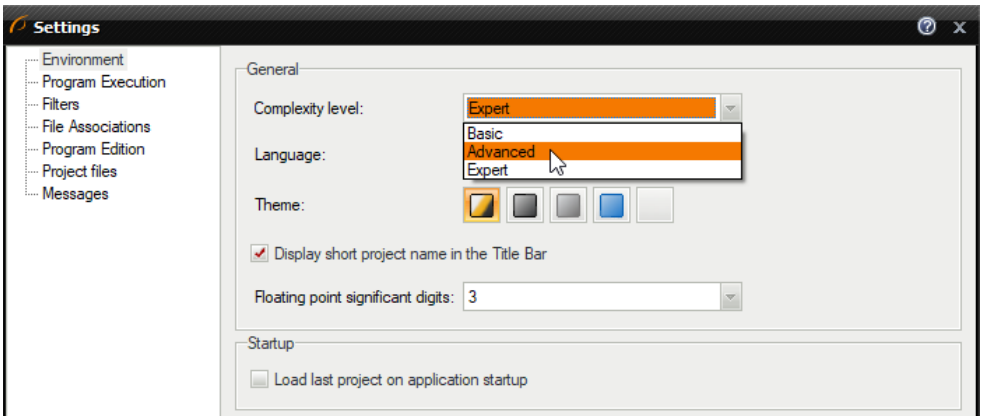
- **Basic** – Designed for production engineers who want to quickly build simple machine vision projects without devoting much time to learning the full capabilities of the software.
- **Advanced** – Designed for professional machine vision engineers, who also implement challenging and complex projects.
- **Expert / Scientific** – Gives access to experimental and scientific filters, which are not recommended for typical machine vision applications, but might be useful for research purposes.

Changing Complexity Level

Complexity Level can be changed at any time. It can be done by clicking on the level name in the upper-right corner of Adaptive Vision Studio:



Complexity Level can also be changed in the application settings:

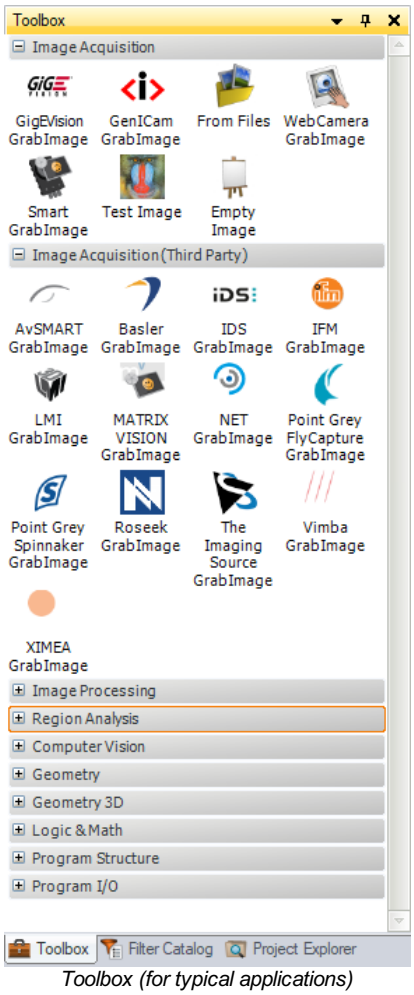


Finding Filters

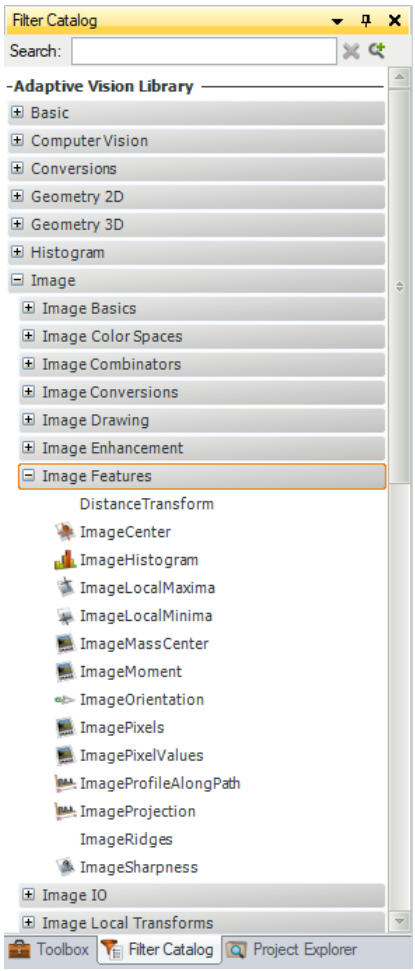
Introduction

There are many hundreds of ready-for-use filters in Adaptive Vision Studio implementing common image processing algorithms, planar geometry, specialized machine vision tools as well as things like basic arithmetics or operating system functions. On the one hand, it means that you get an instant access to results of tens of thousands of programmers' work hours, but it also means that there are quite a lot of various libraries and filter categories that you need to know. This article advises you how to cope with that multitude and how to find filters that you need.

The most important thing to know is that there are two different catalogs of filters, designed for different types of users:



Toolbox (for typical applications)



Filter Catalog (for advanced users)

Toolbox is designed for use in typical machine vision applications. It is task-oriented, most filters are grouped into tools and they are supported with intuitive illustrations. This makes it easy to find the filters you need the most. It does not, however, contain all the advanced filters, which might be required in more challenging applications. To access the complete list of filters, you should use the Filter Catalog. This catalog is organized in libraries, categories and subcategories. Due to its comprehensiveness it usually takes more time to find what you need, but there is also an advanced text-based search engine, which is very useful if you can guess a part of the filter name.

Toolbox

Sections

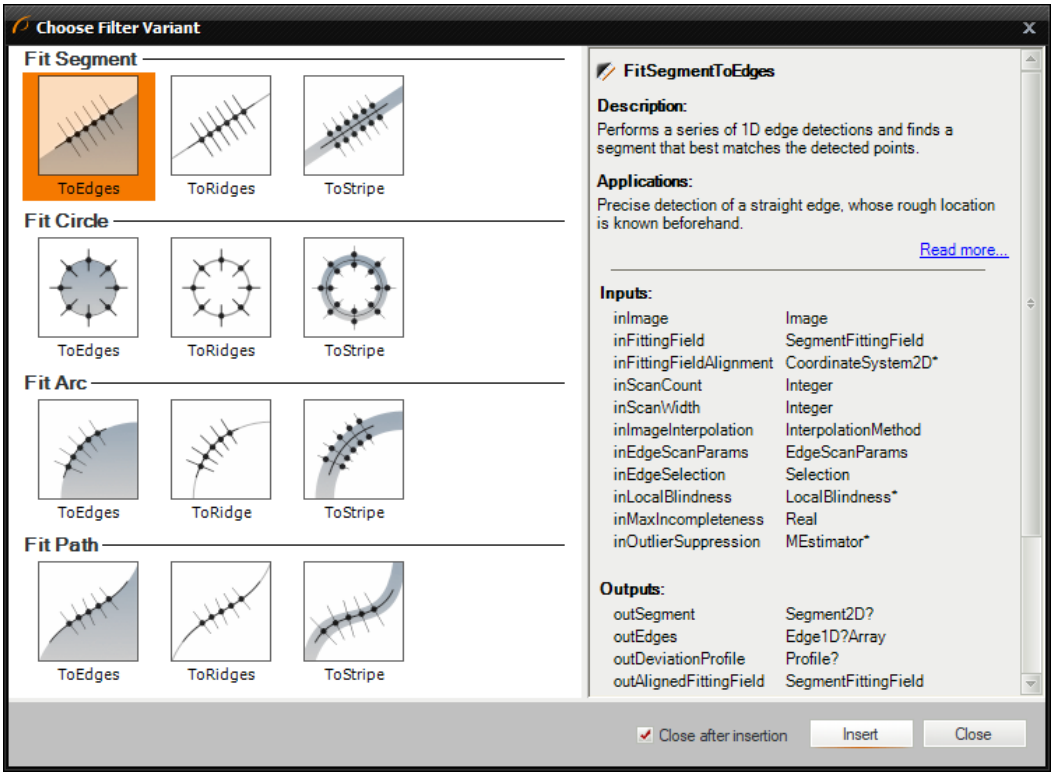
When you use the Toolbox, the general idea is that you will most probably start with a filter from the first section, Image Acquisition, and then follow with filters from consecutive sections:

1. **Image Acquisition**
   – Acquiring images from cameras, frame grabbers or files.
2. **Image Acquisition (Third Party)**
   – Acquiring images from third party cameras.
3. **Image Processing**
   – Image conversions, enhancements, transformations etc.
4. **Region Analysis**
   – Operations on pixel sets representing foreground objects (blobs).
5. **Computer Vision**
   – Specialized tools for object detection, recognition and measurements.
6. **Geometry**
   – Filters for constructing, transforming and analysing primitives of 2D geometry.
7. **Geometry 3D**
   – Filters for constructing, transforming and analysing primitives of 3D geometry.
8. **Logic and Math**
   – Numerical calculations and conditional data processing.
9. **Program Structure**
   – Category contains the basic program structure elements.
10. **Program I/O**
    – Filters for reporting inspection results.

Choosing Filter from Tools

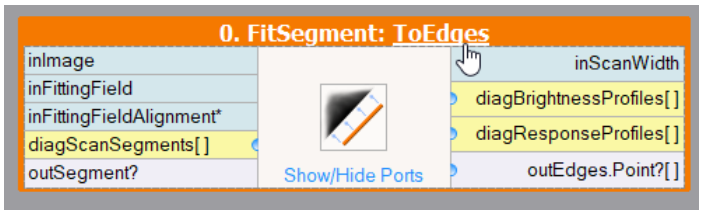
A tool is a collection of related filters. The process of selecting a filter from the Toolbox thus consists of two steps:

1. First you select a tool in the Toolbox, e.g. "Fit Shape".
2. Then you select a filter from the tool in the "Choose Filter of Group" window.



Choosing a filter from a tool (Toolbox).

This dialog can have several sections (e.g. "Fit Segment", "Fit Circle" etc.)



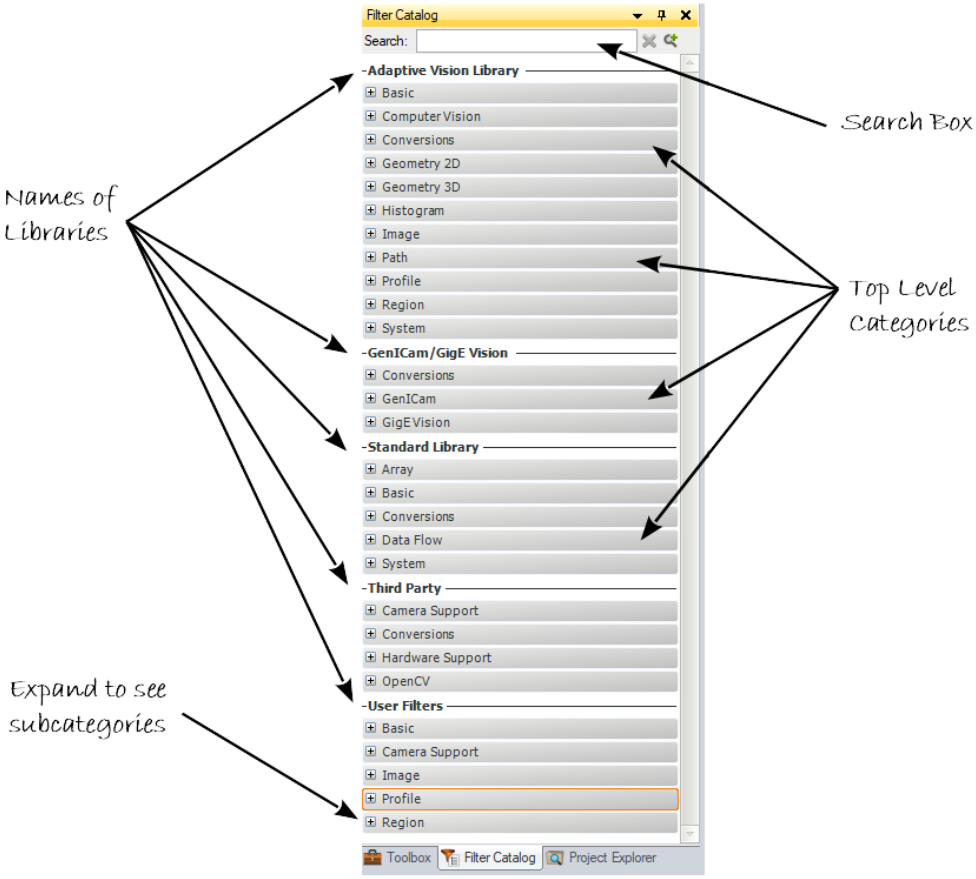
Changing filter variant.

Filter Catalog

Libraries

The Filter Catalog contains several libraries:

- **Adaptive Vision Library**
  – the most important library that contains all the image analysis filters.
- **GenICam/GigE Vision**
  – the library that contains filters providing standard interfaces to industrial cameras and frame grabbers.
- **Standard Library**
  – filters for general data processing and other low level operators (required, as pure computer vision is not enough for real-world applications).
- **Third Party**
  – this is a set of interfaces to APIs provided by other vendors – mainly for hardware support and for the OpenCV library.
- **User Filters**
  – contains various filters created by user –.



Categories in Adaptive Vision Library

The data processing filters of Adaptive Vision Library are organized into a clear two-dimensional structure:

- Categories correspond to the types of data:
  - **Image**
  - **Region** (a set of pixel locations)
  - **Geometry 2D** (point, segment, circle etc.)
  - **Geometry 3D** (surface, plane, point3D etc.)
  - **Path** (a sequence of points, a contour)
  - **Profile** (a 1D image, a sequence of real numbers)
  - **Histogram**

- Subcategories correspond to the types of operators:
  - **Basics** – trivial and technical filters (e.g. **GetImagePixel**, **CreateBoxRegion**)
  - **Features** – primitive properties and information retrieval (e.g. **ImageHistogram**, **RegionMassCenter**)
  - **Metrics** – data comparators and binary features (e.g. **ImageDifference**, **AngleBetweenSegments**)
  - **Relations** – relations and predicates on two or more objects (e.g. **TestImageEqualTo**, **TestPointInRegion**)
  - **Point Transforms** – transformations applied to each element separately (e.g. **NegateImage**, **AddToHistogram**)
  - **Combinators** – producing an object out of two or more other objects (e.g. **AverageImages**, **RegionUnion**)
  - **Local Transforms** – transformations based on information from a fixed neighbourhood of each given location (e.g. **SmoothImage_Gauss**, **ErodeRegion**)
  - **Spatial Transforms** – transformations changing the locations of the elements (e.g. **RotateImage**, **TranslatePoint**)

Now, if you for example have two line segments and need to find their intersection point, you know that you should start with the *Geometry 2D* category that corresponds to the type of data and then choose Geometry 2D Intersections that corresponds to the type of operation. The **SegmentSegmentIntersection** filter will be there.

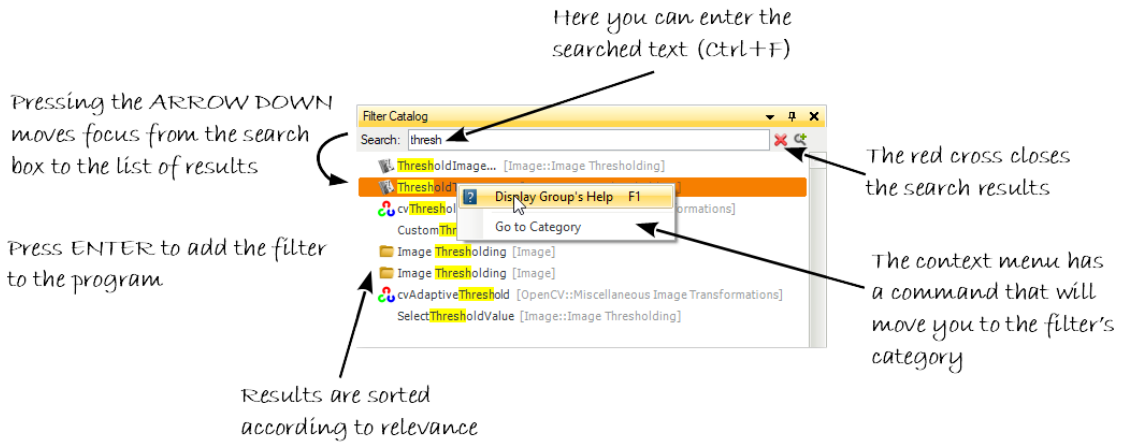
The Computer Vision Filters

A separate important set of filters in Adaptive Vision Library is in the *Computer Vision* category, which contains bigger and ready-for-use tools in several subcategories:

- **1D Edge Detection** – for measurements and positioning using one-dimensional scan paths
- **2D Edge Detection** – for measurements and shape recognition using 2D edge or ridge tracing methods
- **Barcodes** – for detection and reading of 1D barcodes
- **Camera Calibration** – for compensating lens and perspective distortions
- **Datacodes** – for detection and reading of 2D codes like DataMatrix or QRCodes
- **Fourier Analysis** – for analysing images in the frequency domain
- **Hough Transform** – for detecting analytic shapes like lines or circles
- **Image Analysis** – contains filters detecting connected regions that correspond to objects having uniform colors and filters for detection of feature points
- **Multilayer Perceptron** – for automated pattern recognition with artificial neural networks
- **Optical Character Recognition** – for recognizing printed characters in an image
- **Shape Fitting** – for 2D shape recognition and measurements
- **Template Matching** – for locating objects in an image using a predefined template

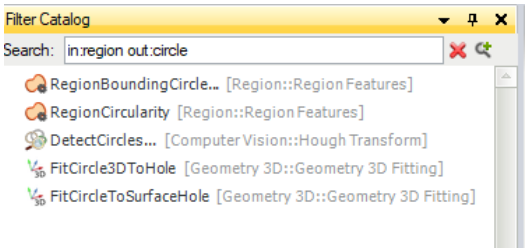
The Search Box

If you know a part of the name of the filter that you need, or if you can guess it, the Filter Search Box will make your work much more efficient. Just enter the searched text there and you will get a list of filters with most relevant matches at the top:



The Search Box and search results in the Filter Catalog.

If you can not guess the filter name, but you know what you expect on the inputs and outputs, you can use special queries with "in:" and "out:" operators as the image below depicts:



Advanced search with expected inputs and outputs.

As a matter of fact, some advanced users of Adaptive Vision Studio stop browsing the categories and just type the filter names in the Search Box to have them quickly added to the program.

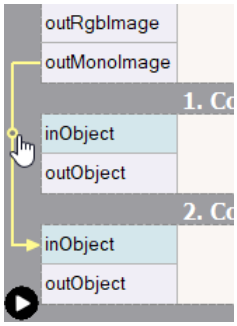
Connecting and Configuring Filters

After a filter is added to the program it has to be configured. This consists in setting its inputs to appropriate constant values in the Properties window or connecting them with outputs of other filters. It is also possible to make an input linked with an external AVDATA file, connected with HMI elements or with Global Parameters.

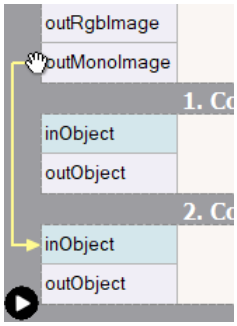
Connecting with Other Filters

Filters receive data from other filters through connections. To create a connection, drag an output of a filter to an input of a filter located below in the Program Editor (upward connections are not allowed). When you drag an output - possible destinations will be highlighted. It is basic way to create program's work-flow.

For convenience, new connections can also be created by forking existing connections (drag from the vertical part of the existing connection):



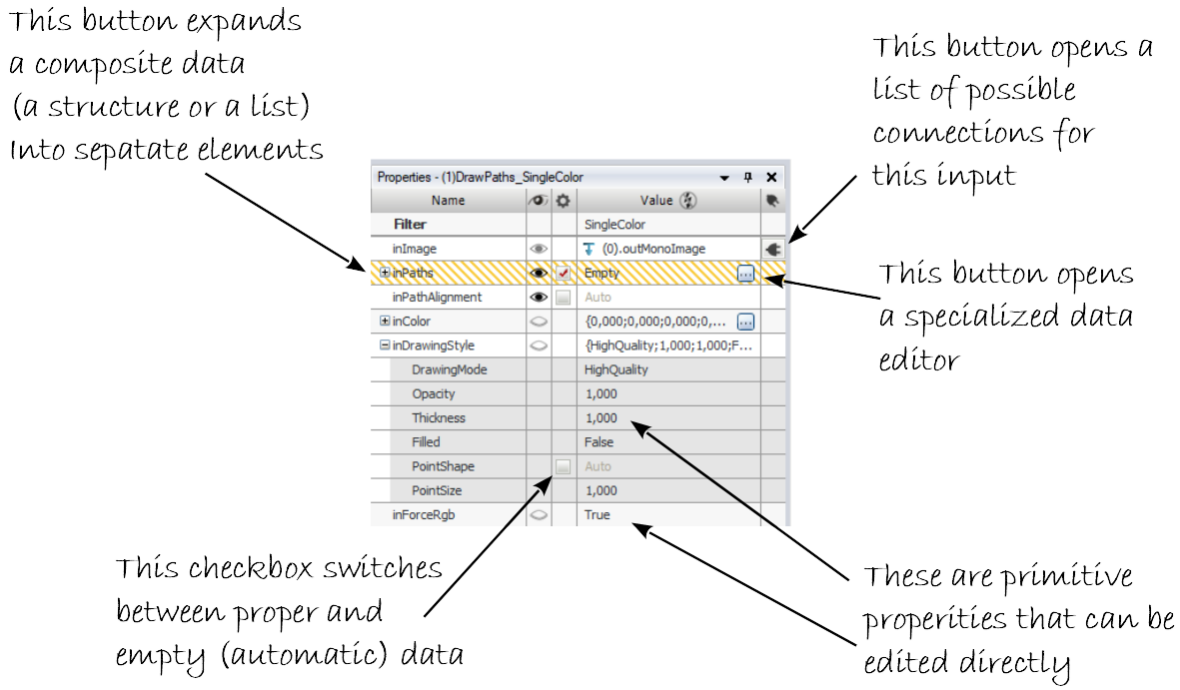
Moreover, it is possible to reconnect an existing connection to another output or input port by dragging the connection's head (near the destination port) or the connection's first horizontal segment (near the source port).



Another way to create connections between filters is by using only the Properties window. When you click the plug () icon in the right-most column, you get a list with possible connections for this input. When the input is already connected the plug icon is solid and you can change the connection to another one.

Setting Basic Properties

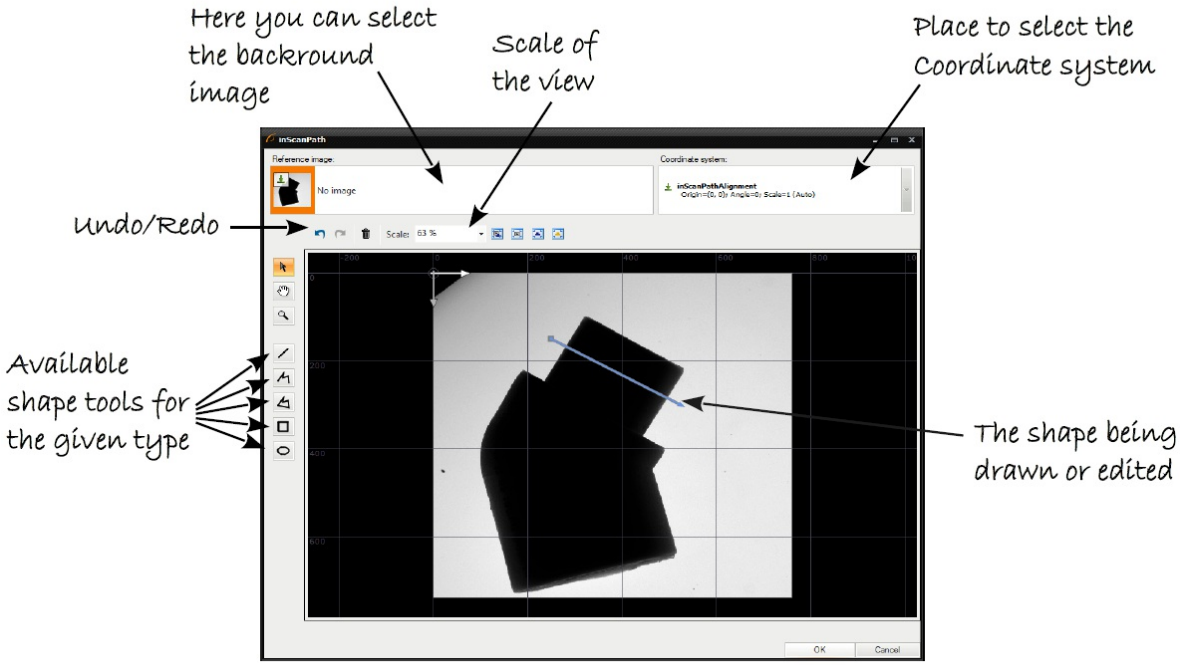
Most of the filters have inside parameters which you can set in Properties Box like on sample image below. Properties Box is a place where you can adjust the conditions of filter's work. It is very important to go through these parameters is order to get desired results. To start, first select a filter in the Program Editor window.



Note: After clicking on the header of the properties table it is possible to choose additional columns.

Editing Geometrical Primitives

To edit geometrical data, such as segments, circle, paths or regions, click the three dots button () in the Properties window at the input port you want to set or modify. A window similar to the one below will appear. The first thing you will usually need to do, when you open this window for the first time, is to select the background image from the combo-box at the top of the window. This will set a context for the data you edit.



Editing of a path in a context of an image.

Tips:

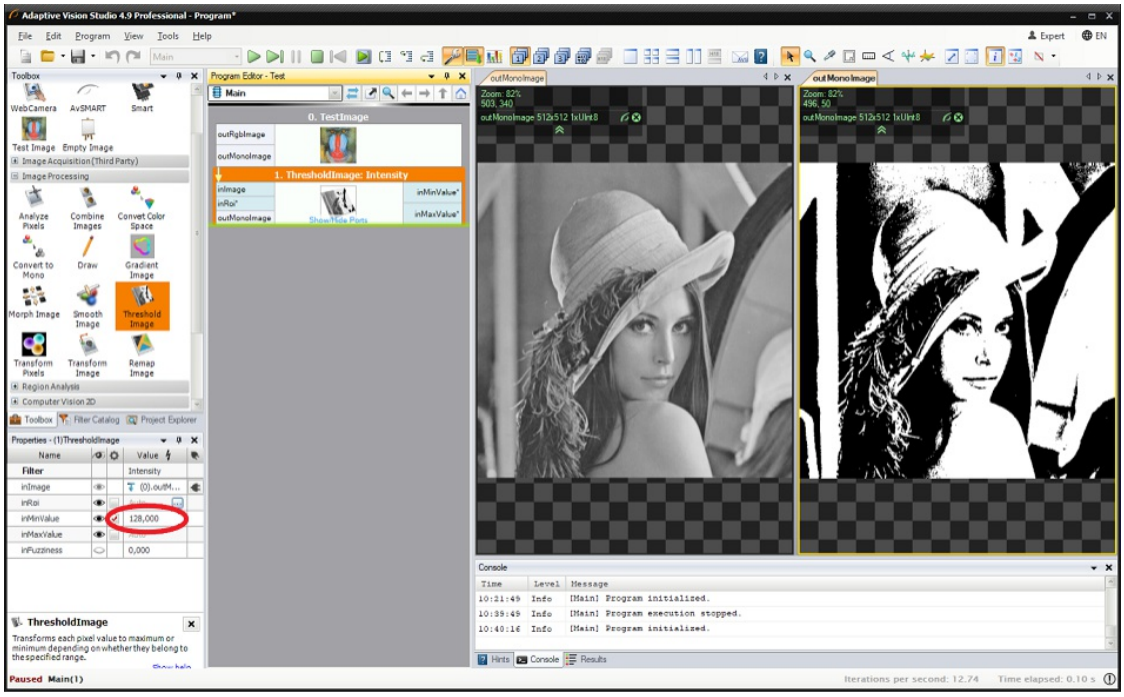
- Select a point and use its context menu to inspect or set the numeric coordinates.
- Use the mouse wheel to zoom the view without changing the tool.
- Hold 3rd mouse button and drag to move the view without changing the tool.
- Hold *Ctrl* to limit the segment angles to the multitudes of 15 degrees.

Testing Parameters in Real Time

One of the greatest features of Adaptive Vision Studio is its orientation on rapid development of algorithms. This encompasses the ability to instantly see how different values of parameters affect the results. Due to the dynamic nature of this feature it cannot be presented in a static picture, so please follow the instructions:

1. Create a simple program with a **TestImage** filter connected to a **ThresholdImage** filter.
2. Put the output of the **ThresholdImage** to a data preview.
3. Click *Iterate Current Macro* to execute the program to the last filter, but without ending it as the whole (the execution state will be: *Paused*).
4. Select the **ThresholdImage** filter in the Program Editor and change its **inMinValue** input in the Properties window.

Now, you will be able to see in real time how changing the value affects the result.



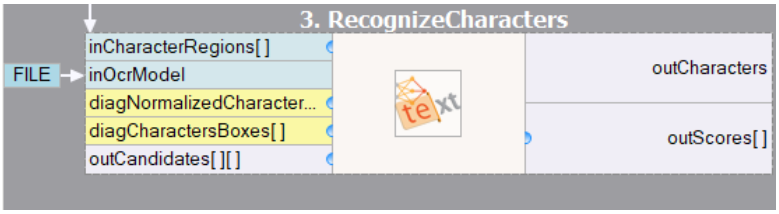
Re-executing a filter after a change of a parameter.

Remarks:

- Please note, that although being extremely useful, this is a "dirty" feature and may sometimes lead to inconsistencies in the program state. In case of problems, stop the program and run it again.
- This feature works only when the filter has already been executed and the program is *Paused*, but NOT *Stopped*.
- By default re-executed is the entire macrofilter. By unchecking the "Global Rerun Mode" setting the re-execution can be limited to a single filter. This can be useful when there are long-lasting computations in the current macrofilter.
- It is not possible to re-execute i/o filters, loop accumulators or loop generators (because this would lead to undesirable side effects). These filters are skipped when the entire macrofilter is getting re-executed.
- When re-executing a nested instance of another macrofilter, the previews of its internal data are NOT updated.
- Re-executing some filters, especially macrofilters, can take much time. Use the Stop command (Shift+F5) to break it, when necessary.
- If you set an improper value and cause a *Domain Error*, the program will stop and it will have to be started again to use the re-execution feature.
- The filter parameters can also be modified during continuous program execution (*F5*).

Linking or Loading Data From a File

Sometimes the data that have to be used on an input of a filter is stored in an .avdata file, that has been created as a result of some Adaptive Vision Studio program (for example creating a custom OCR model). It is possible to load such data to the program with the **LoadObject** filter, but most often it is more convenient and more effective to link the input port directly to the file. To create such a link choose *Link from AVDATA File...* from the context menu of the input (click with the right mouse button on an input port of a filter).

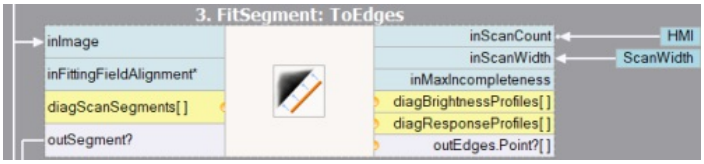


Input data linked from an AVDATA file.

It is also possible to load, not link, data from an .avdata file. This is done with the *Import from AVDATA File...* command in the context menu, which copies the data and makes them part of the current project.

Connecting HMI and Global Parameters

It is also possible to connect filter inputs from outputs of HMI elements and from Global Parameters. Both get displayed as rectangular labels at the sides of the Program Editor as can be seen on the image below:



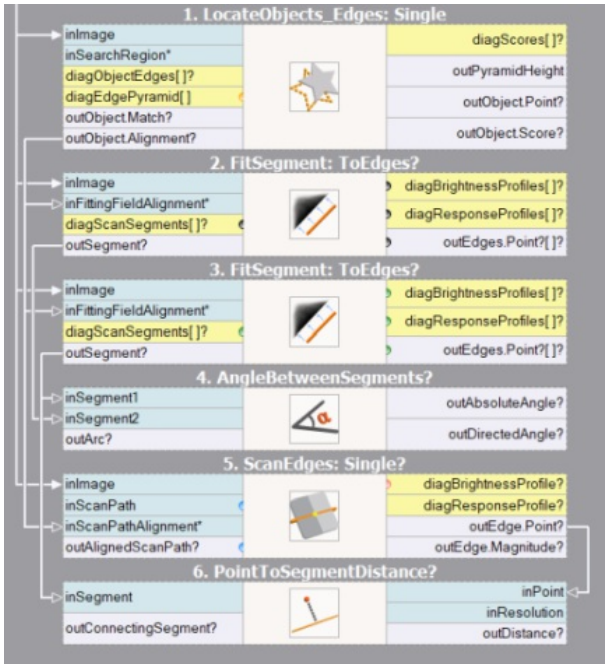
A filter with connections from HMI and from a Global Parameter.

For further details please refer to the documentation on the specific topics:

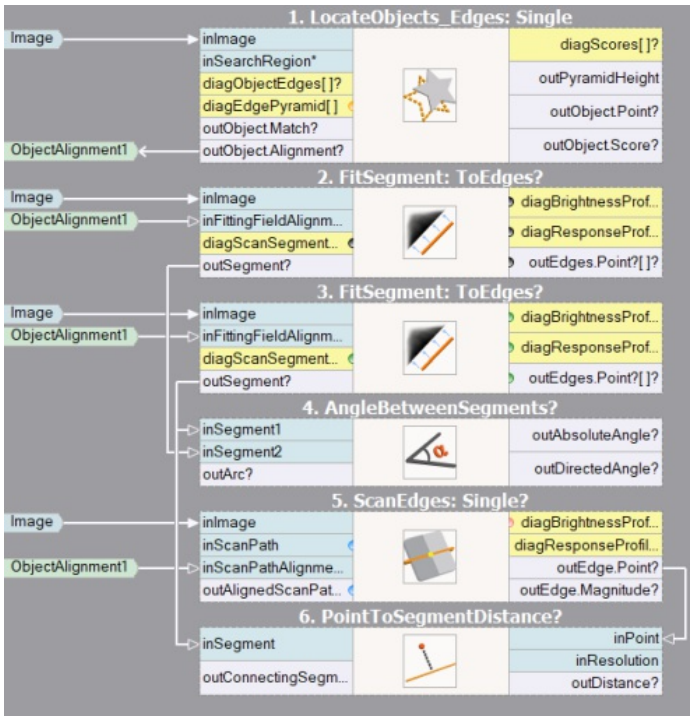
- HMI Designer
- Global Parameters

Labeling Connections

Connections are easier to follow than variables as long as there are not too many of them. When your program becomes complicated, with many intersecting connections, its readability may become reduced. To solve this problem Adaptive Vision Studio provides a way to replace some connections with labels. You can right-click on a connection, select "Label All Connections..." - when there is more then one connection or "Label Connection..." when only one connection is present. Then set a name of a label that will be displayed instead. The labels are similar to variables known from textual programming languages – they are named and can be more easily followed if the connected filters are far away from each other. Here is an example:



A program with some long connections becomes not easy to analyze.



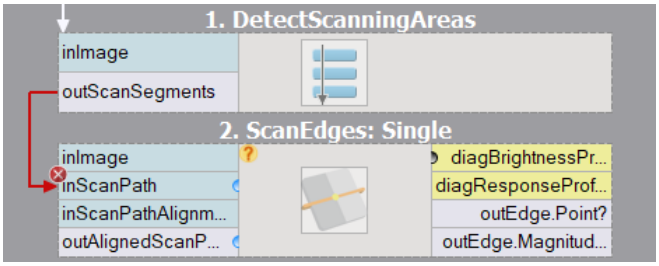
Long connections are replaced with labels for better readability.

Remarks:

- Labeled connections can be brought back (unlabeled) by using the "Un-label This Connection" or "Un-label All Connections" commands available in the context menu of a label.
- Please note, that when your program becomes complicated, the first thing you should consider is reducing its complexity by refactoring the macrofilter hierarchy or rethinking the overall structure. Labeling connections is only a way to visualize the program in a more convenient way and does not make its structure any simpler. It is the user's responsibility to keep it well organized anyway.
- Adaptive Vision Studio enforces that all connections between filters are clearly visualized, even if making them implicit would make programming easier in typical machine vision applications. This stems from our design philosophy that assumes that: (1) it is wrong to hide something that the user has to think about anyway, (2) the user should be able to understand all the details of a macrofilter looking at a static screen image.

Invalid Connections

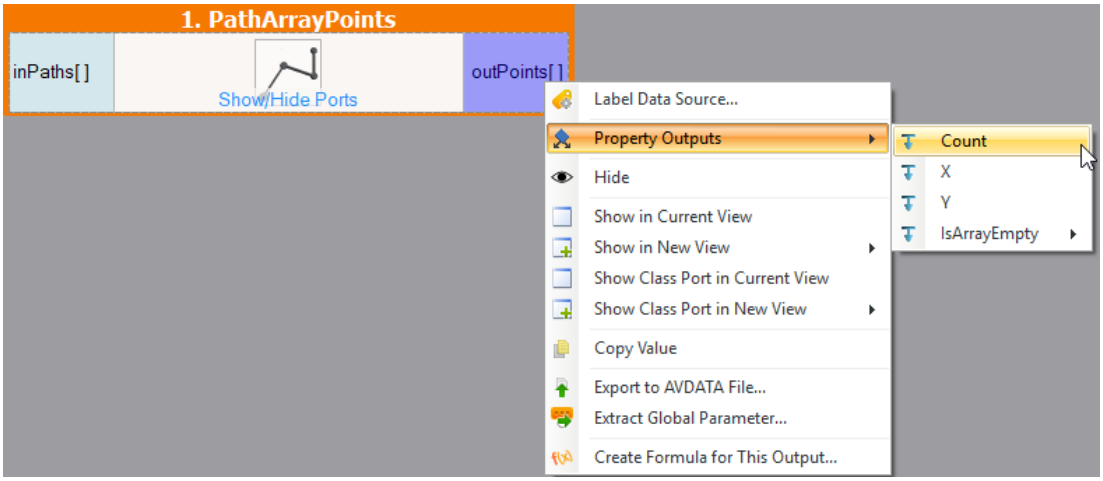
As types of ports in macrofilters and formulas can be changed after connections with other filters have been created, sometimes an existing connection may become invalid. Such an invalid connection will be marked with red line and white cross in the Program Editor:



Invalid connections do not allow to run the program. There are two ways to fix it: replace the invalid connections or revise the types of the connected ports.

Property Outputs

Many types of data can be decomposed into simpler elements called fields or properties. For example, Point2D is a structure composed of "X" and "Y" fields, whereas arrays have a "Count" property informing about the number of contained elements and "IsArrayEmpty" property checking whether there are any. Such properties can be exposed as additional filter outputs (or inputs), so that they will be directly available for creating connections with other filters. Accessing property outputs works also for ports on macrofilter input blocks inside of macrofilters.



Accessing fields of Point2DArray type.

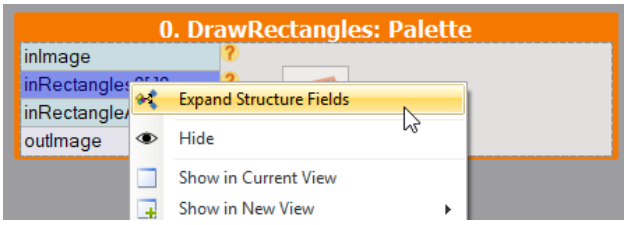
Not all structure fields are always available. For example Image type gives access only to information such as: Width, Height, Depth, but not to pixel data. The Count property is available only for output structures, but not for inputs.

The most common property outputs are:

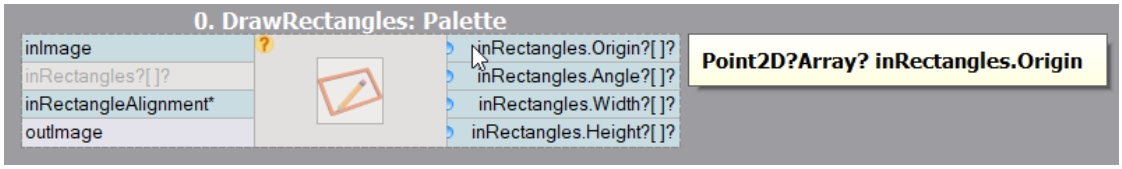
- **Count** – only for arrays; returns the number of elements
- **Length** – only for strings; returns the number of characters
- **Not** – only for logical values; returns the negation
- **IsNil** – only for conditional values; informs whether a value equals Nil
- **IsArrayNil, IsElementNil** – only for arrays with some conditions; inform whether an array or its elements equal Nil
- **IsArrayEmpty** – only for arrays; inform whether an array contains no elements
- **IsRegionEmpty** – only for regions; informs whether a region contains no pixels
- **IsPathEmpty** – only for paths; informs whether a path contains no points
- **Size** – only for paths; returns the number of points in a path

Expanded Input Structures

Analogously to expanding output properties, it is also possible to expand input structures. This works only for basic structures which allow access to all their fields (e.g. Point2D, but not Image). Please note, that the expanded input structure is replaced by the fields it consists of (unlike by adding property outputs, where the structure itself remains available).



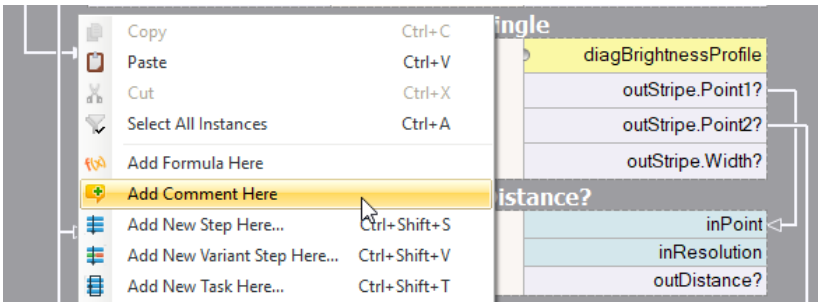
Expanding filter input of Rectangle2D?Array? type.



Result of expanding input of Rectangle2D?Array? type.

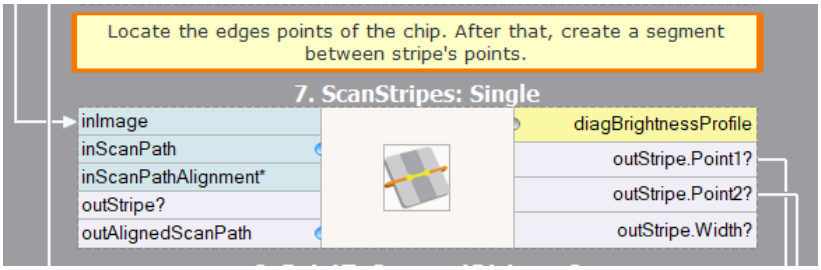
Comment Blocks

Comments are very important part of every computer program. They keep it clear and easy to understand during development and further maintenance. There are two ways to make a comment in Adaptive Vision Studio. One way is to add a special comment block. Another option is adding a comment directly in the filter. To add a new comment block to program click the right mouse button on the Program's Editor background and select the "Add Empty Comment" option like on the image below.



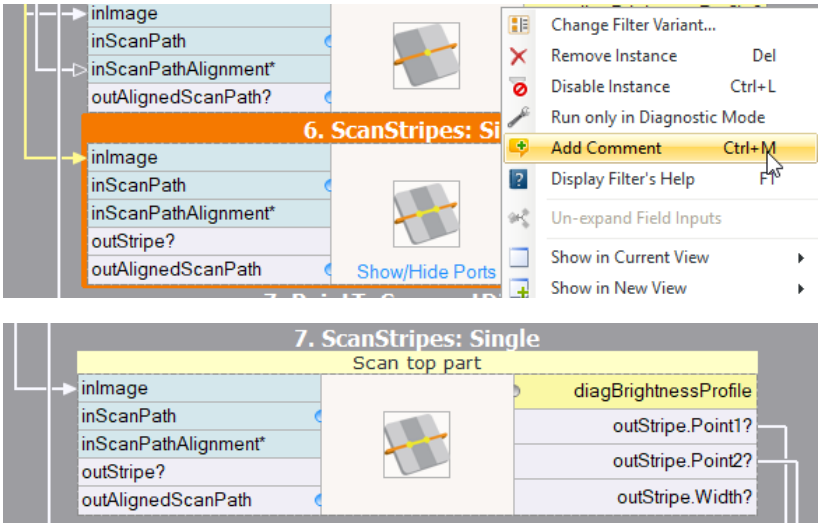
Adding a new comment.

Comment block can be very useful for describing the details of the algorithm.



Comments can be used for describing program steps.

But when you need just a simple tip or short remark, use a "Add Comment" after mouse right click on the filter:



New comment directly in the filter.

Creating Macrofilters

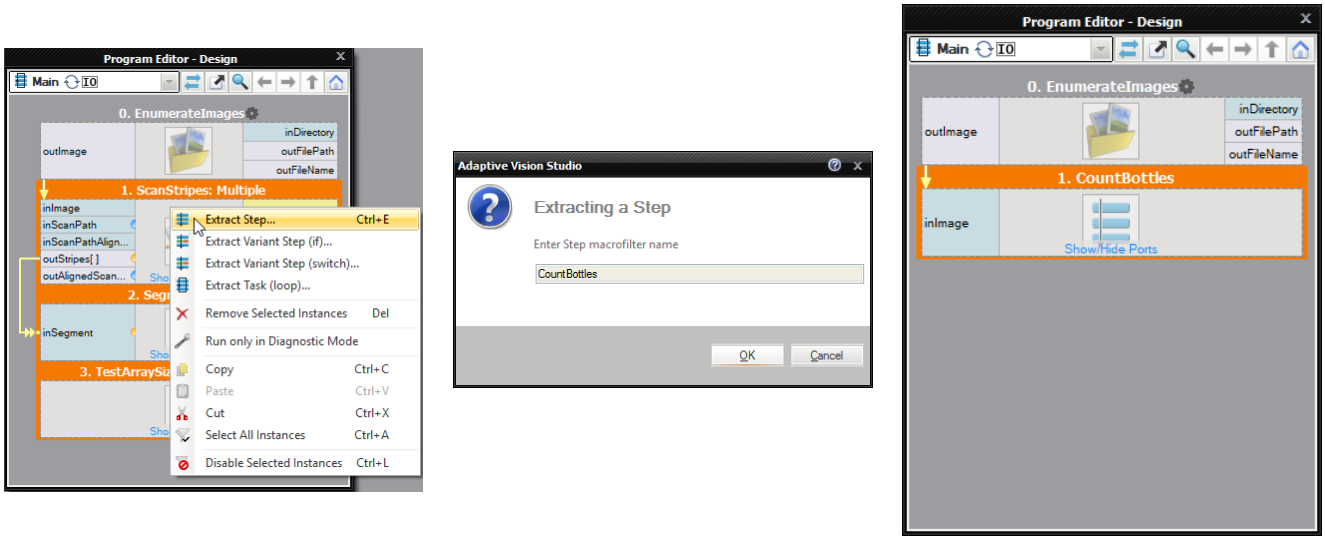
Introduction

Macrofilters play an important role in developing bigger programs. They are subprograms providing a means to isolate sequences of filters and re-use them in other places of the program. After creating a macrofilter, you can create its instances. From outside an instance looks like a regular filter, but by double-clicking on it, you navigate inside and can see its internal structure.

When you create a new project, it contains exactly one macrofilter, which is named "Main". This is the top level macrofilter from which the program execution will start. Further macrofilters can be added by extracting them from existing filters or by creating them in the Project Explorer window.

Extracting Macrofilters (The Quick Way)

The most straightforward way of creating a macrofilter is by extracting it from several filters contained in an existing macrofilter. This is done by selecting several filters in the Program Editor and choosing the *Extract Step...* command from the context menu, as depicted in the picture below:



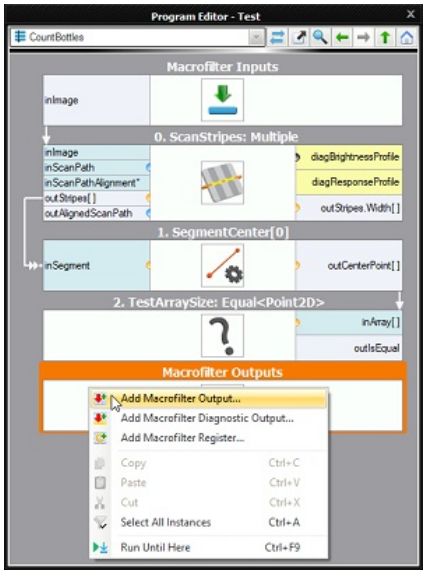
Extracting a macrofilter from three existing filters.

After completing this operation, a new macrofilter class is created and the previously selected filters are replaced with an instance of this macrofilter. Now, additional inputs and outputs of the new macrofilter can be created by dragging and dropping connections over the new macrofilter instance.

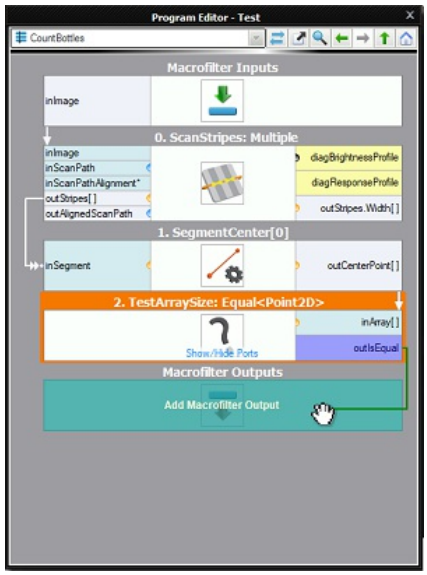
Remark: The "Extract Task (Advanced)..." command creates a new Task macrofilter which should be used only in special cases. Execution of the Task macrofilter is more complex than execution of the Step macrofilter. Usage of the Task macrofilter for reducing macrofilter complexity may be inappropriate. For more details please read section about Task macrofilters.

Defining the Interface

Being inside of a macrofilter other than "Main" you can see two special blocks: the *Macrofilter Inputs* and the *Macrofilter Outputs*. The context menus of these blocks allow to add new inputs or outputs. Another method of adding them is by dragging connections and dropping them over these blocks.

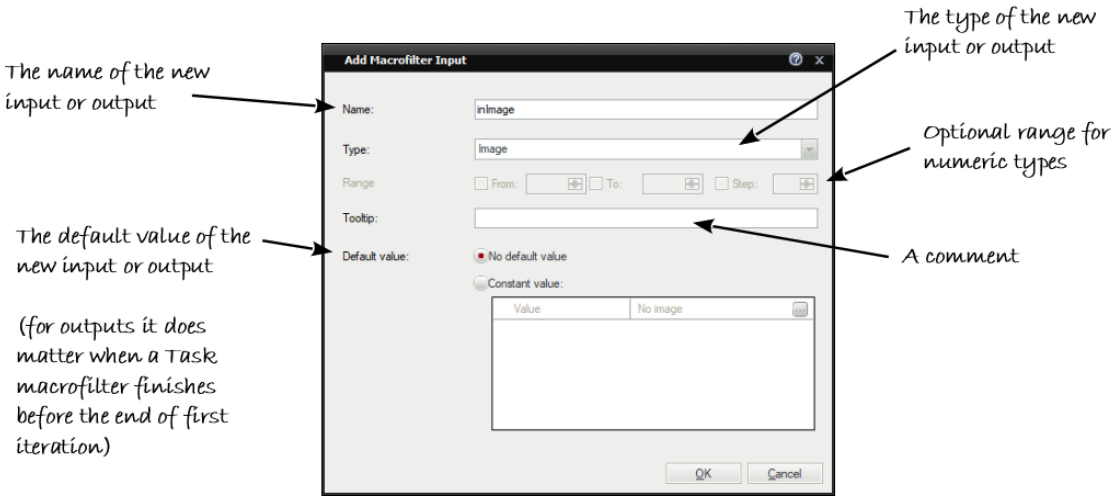


Adding a new output using the context menu of the Macrofilter Outputs block.



Adding a new output by dragging and dropping a connection.

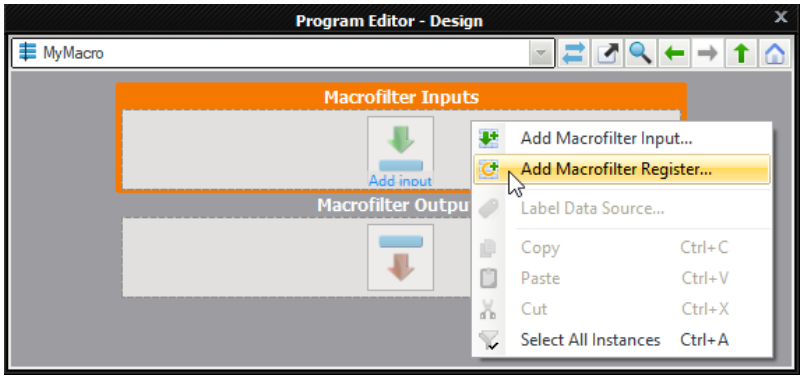
Before the new port is created you need to provide at least its name and type:



Defining a port of a macrofilter.

Adding Registers

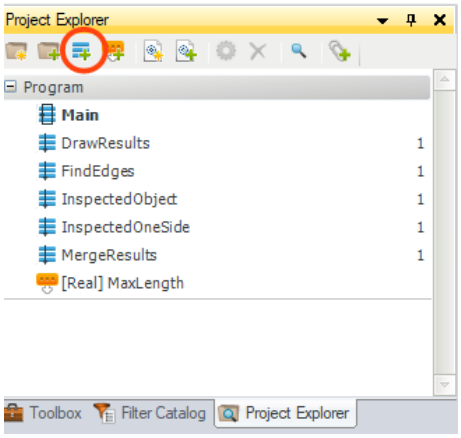
The context menus of macrofilter input and output blocks also contain a command for adding macrofilter registers, *Add Macrofilter Register...*:



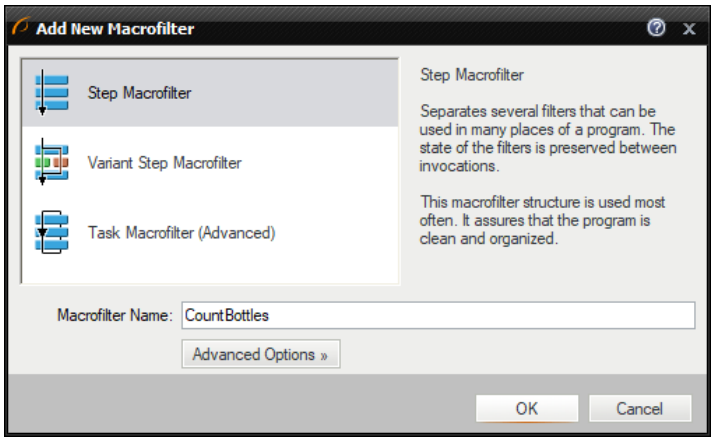
Creating Macrofilters in the Project Explorer

All macrofilter classes that are contained in the current project can be displayed in the Project Explorer window (by default it is behind the Toolbox on a tab page). Using this control, the user can create or edit macrofilters, but it also acts as a filter catalog from which instances can be created by dragging and dropping the items into the Program Editor.

To create a new macrofilter in the Project Explorer click the *Create New Macrofilter...* button. Then a new window will appear, when you will be able to select the name and the structure of the new macrofilter.



Macrofilters in the Project Explorer.



Selecting the name and the structure of a new macrofilter.

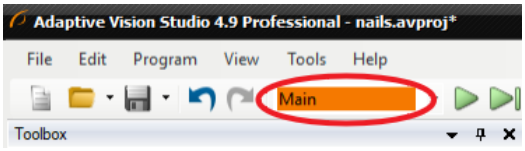
The StartUp Program

It is sometimes useful to have several programs in a single project. The most common scenarios are:

- When some more complex data, e.g. custom OCR models or calibrated reference images, need to be prepared in a separate program.
- When it is convenient to have multiple versions of a single program customized for different installations.

- When the highest reliability is important and sets of automated tests need to be performed on recorded images.

In Adaptive Vision Studio a program can be any Task macrofilter that does not have any inputs and outputs. The list of all macrofilters fulfilling this criterion can be seen in the StartUp Macrofilter combo box on the Application Toolbar. This control makes it possible to choose the program that will be run. The related macrofilter will be displayed in the Project Explorer with the bold font.



The StartUp Program Combo Box.

Macrofilter Guidelines

Macrofilters organize big projects, but it is the responsibility of the user to make this organization clean and effective. When a project grows, especially under the pressure of time, it is easy to forget this and create programs that are difficult to understand and maintain. To avoid this, please follow the following guidelines:

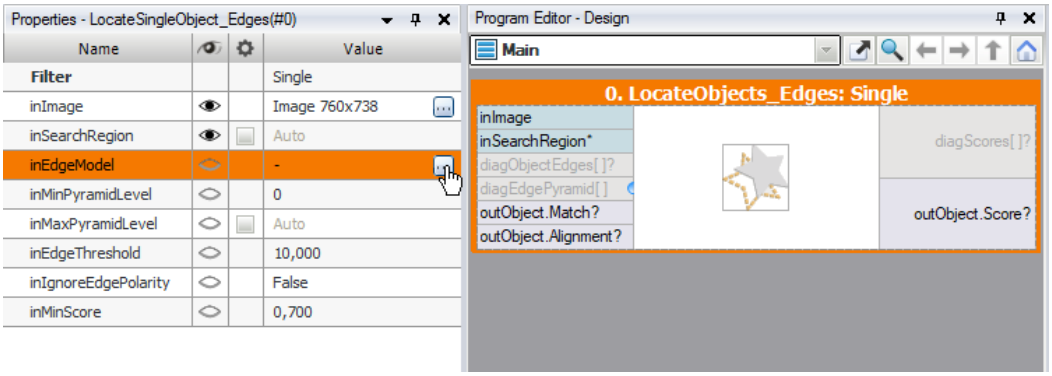
1. For program clarity it is recommend to create a macrofilter at least when there are more than 16 filters (about 2 screens) in a single view of the Program Editor.
2. Macrofilter names do not affect the program execution, but they are very important for effective maintenance. Choose precise and unambiguous English names, follow the *verb* + *noun* naming convention whenever possible and avoid abbreviations (this applies also to names of macrofilter ports).
3. A single macrofilter should have a single purpose. Avoid doing two unrelated things in a single macrofilter.
4. If data visualization is needed for HMI, try to compute results in one macrofilter and then prepare the visualization in another. This will make the data flow more clean.

Creating Models for Template Matching

Introduction

Template Matching tools are very often used as one of the first steps in industrial inspection applications. The goal is to detect the location of an object. Before this can be done the user has to create a model representing the expected object's shape or structure. To make this step straightforward, Adaptive Vision Studio provides an easy user interface. We call it a "GUI for Template Matching".

The GUI for Template Matching is an editor for values of two types: EdgeModel and GrayModel. This means that to open it the user has to select a template matching filter in the program and then click on the button at the **inGrayModel** or **inEdgeModel** input in the Properties window:



Opening GUI for Template Matching.

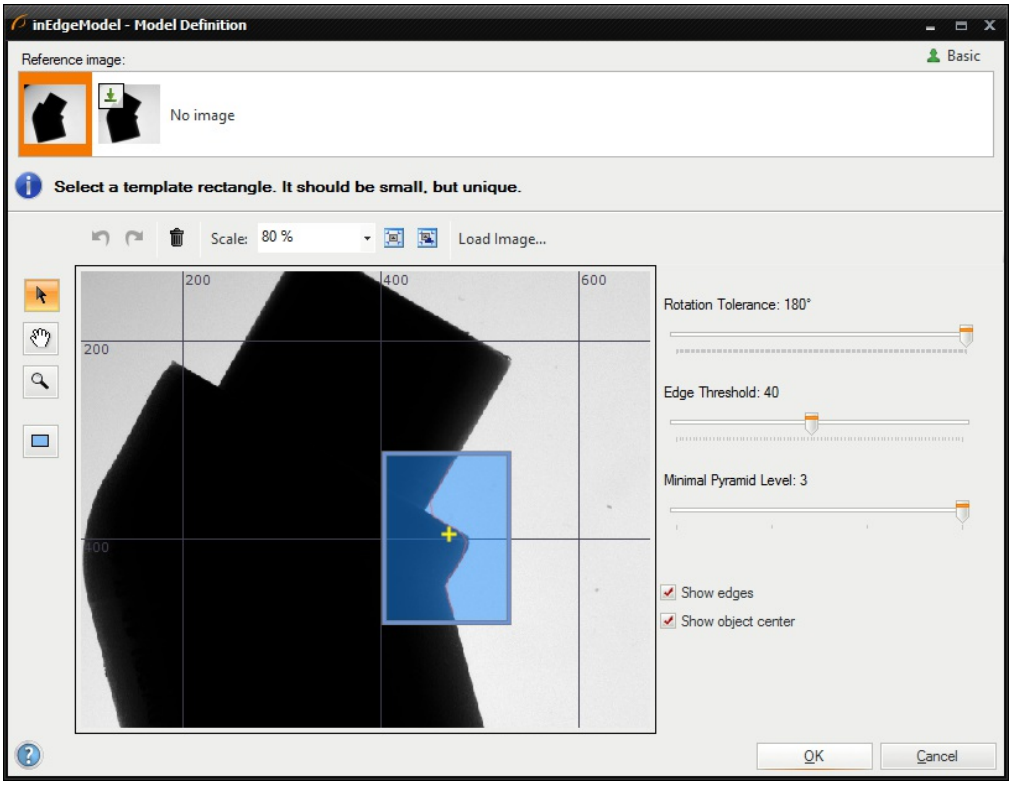
In majority of industrial applications Template Matching algorithms implemented in Adaptive Vision Studio easily detect the location of an object using default parameters. However, there are cases where it is needed to tune some of them, mainly in order to achieve higher reliability. It turned out that there is a group of parameters, which is used more often than others. Therefore GUI for Template Matching is available in two variants: Basic and Expert. This is related to Complexity Levels available in Adaptive Vision Studio.

Creating a Model

Basic

The Basic window contains the following elements:

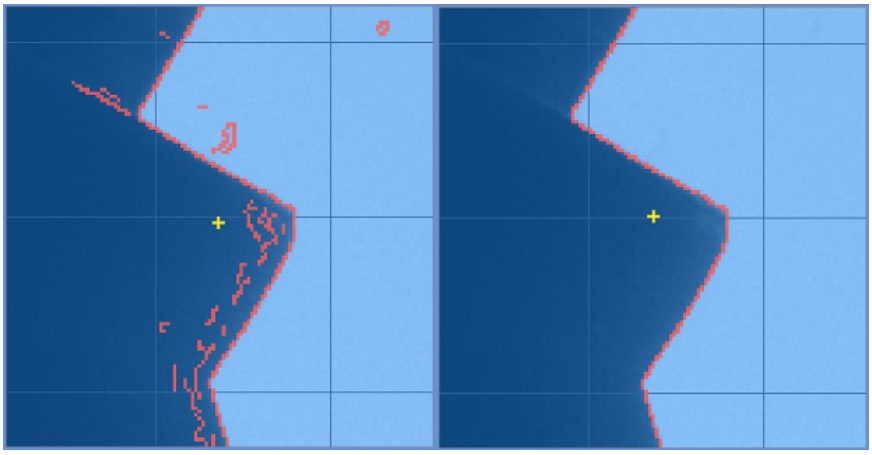
1. At the top: a list of possible template images
2. Below: a simple toolbar that also contains a button for loading a template image from a file
3. On the left: a tool for selecting a rectangular template region
4. On the right: track-bars for setting parameters and some options related to the view
5. In the center: an area for selecting the template region in the context of the selected template image



Basic GUI for Template Matching window

To create a template matching model you need to:

1. Choose a template image from the "Reference image" list.
2. Select a rectangular template region using  drawing tool. For maximum performance this rectangle should be as small as possible.
3. Edge-based matching only: Set the **Edge Threshold** parameter, which should be set to value that results in the best quality of the edges. **Edge Threshold** determines the minimum strength (gradient's magnitude) of pixels that can be considered as an edge.



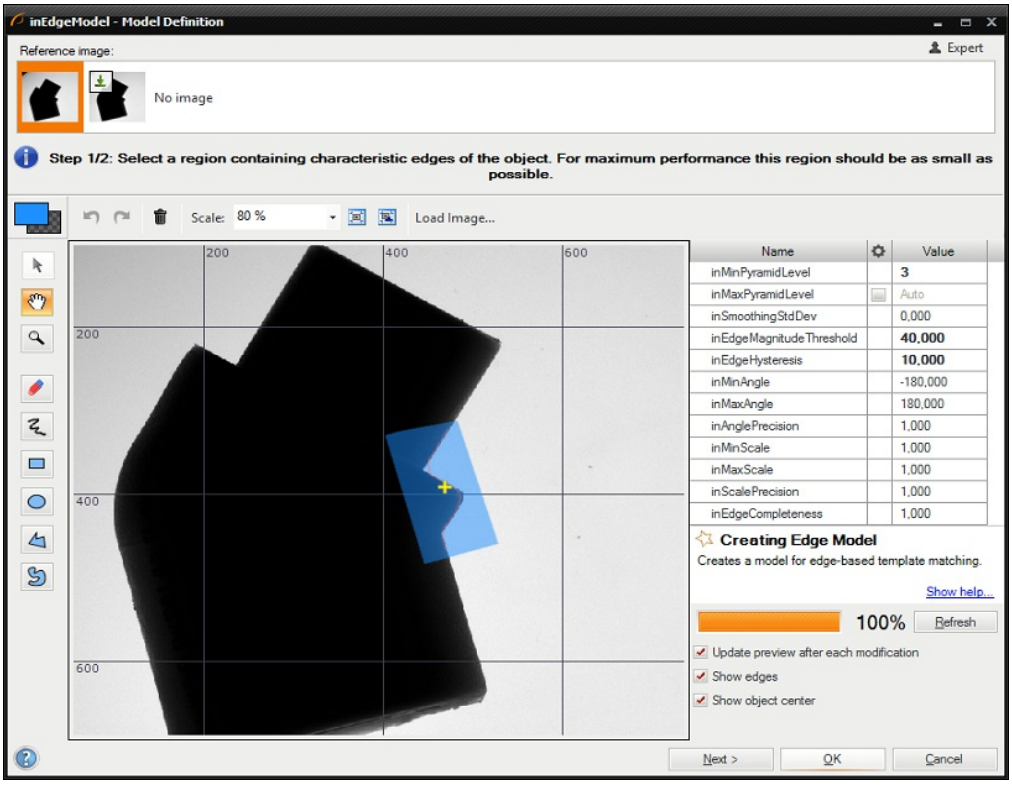
Low quality edges (Edge Threshold = 8) and high quality edges (Edge Threshold = 30)

4. Set the **Rotation Tolerance** in range from 0° to 360°. This parameter determines the maximum expected object rotations. Please note that the smaller the **Rotation Tolerance**, the faster the matching.
5. Set the **Minimal Pyramid Level** parameter which determines the lowest pyramid level used to validate candidates who were found on the higher levels. What is worth mentioning is that setting this parameter to a value greater than 0 may speed up the computation significantly, however, the accuracy of the matching can be reduced. More detailed information about Image Pyramid is provided in Template Matching document in our Machine Vision Guide.

Expert

The Expert window contains the following elements:

1. At the top: a list of possible template images
2. Below: a simple toolbar that also contains a button for loading a template image from a file
3. On the left: a tool for selecting a template region of any shape
4. On the right: parameters of the model, their description and some options related to the view
5. In the center: an area for selecting the template region in the context of the selected template image

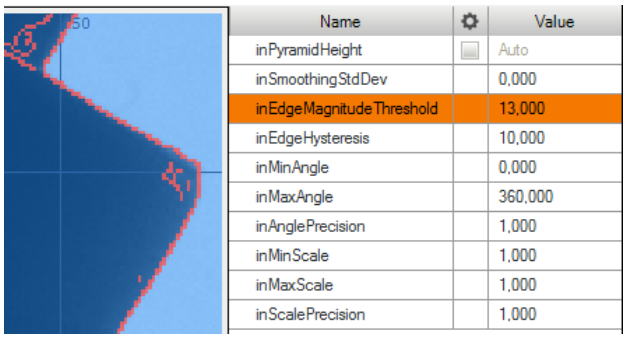


GUI for Template Matching window

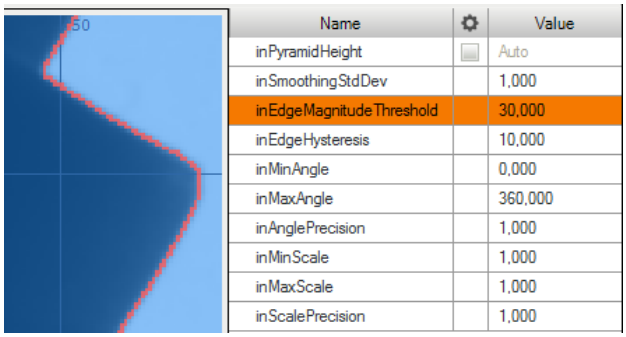
To create a template matching model you need to:

1. Choose a template image from the "Reference image" list.

2. Mark a template region using drawing tools (please note, that the button above the tools switches the current color, i.e. you can both draw or erase shapes). For maximum performance this region should be as small as possible.
3. Edge-base matching only: Set the **inSmoothingStdDev**, **inEdgeMagnitudeThreshold** and **inEdgeHysteresis** parameters to values that result in the best quality of the found edges. It is advisable to first set **inEdgeHysteresis** to zero, then choose a value for **inEdgeMagnitudeThreshold** that assures that all edges have some parts detected and then increase **inEdgeHysteresis**. Small noise might by removed by change the value of parameter **inSmoothingStdDev**. For example:

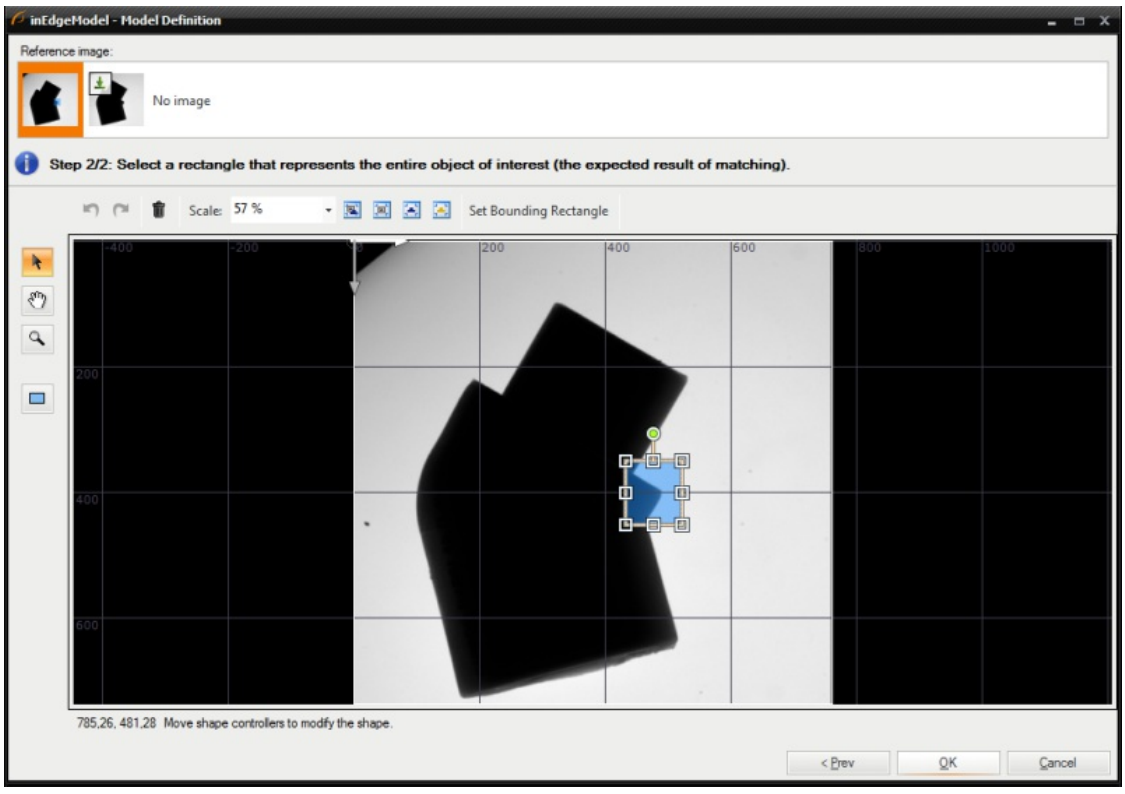


Low quality edges



High quality edges

4. Set the **inMinAngle** and **inMaxAngle** parameters accordingly to the expected range of object rotations (the smaller the range, the faster the matching).
5. Set the **inAnglePrecision** to a value lower that 1.0 if you prefer to lower the angular precision for the benefit of speed.
6. Set the **inMinScale**, **inMaxScale** and **inScalePrecision** to appropriate value if you need to detect objects in different scale. You should be aware of longer detection time when you detect objects in scale.
7. Click the "Refresh" button or check "Update preview after each modification" to review the results.
8. Click the "Next >" button to select template rectangle that represents the entire object of interest (the expected result of matching)

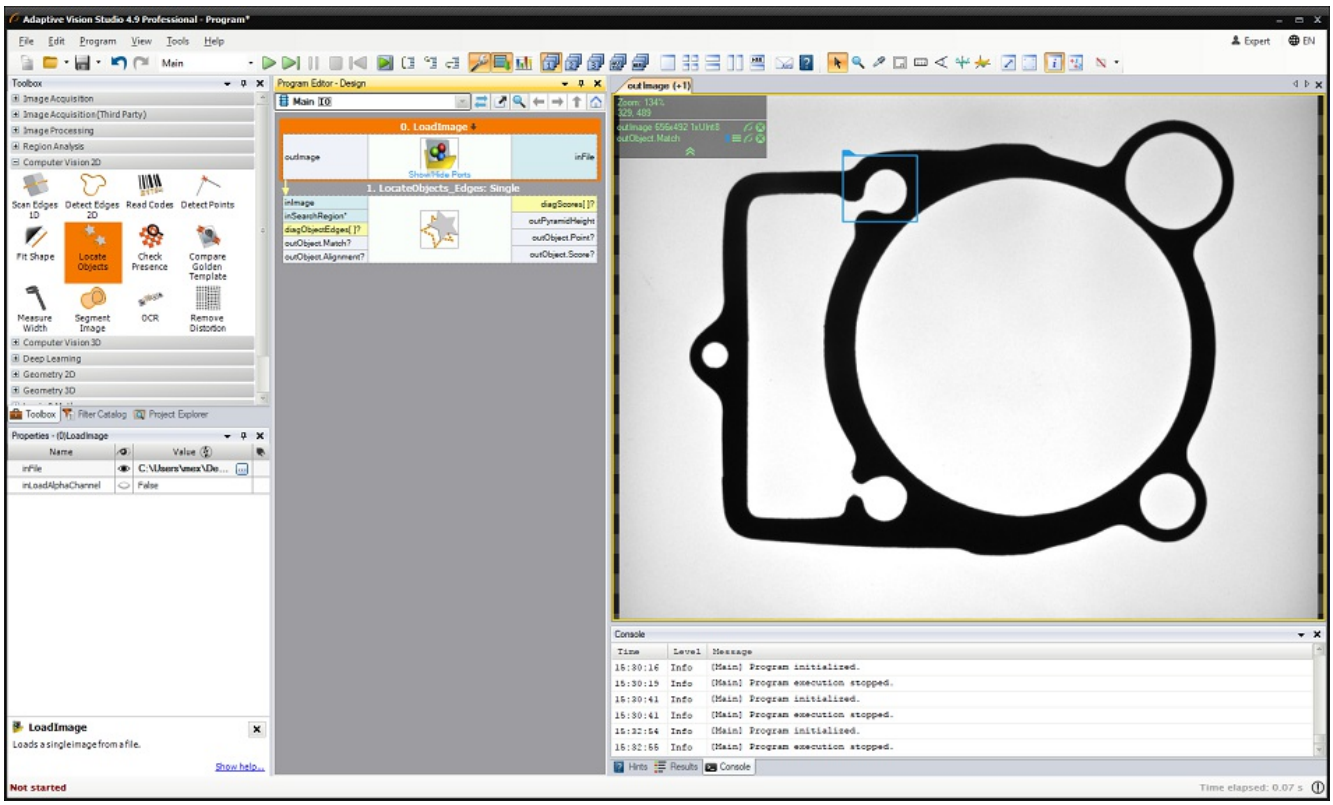


Template rectangle selection

9. Click "OK" to close the window and generate the model.

Performing Template Matching

When the model is ready, performing template matching in an application is straightforward – after connecting the filters and setting the matching parameters, the results are on the outputs of the **LocateSingleObject_Edges** filter (or similar):



Example result of template matching

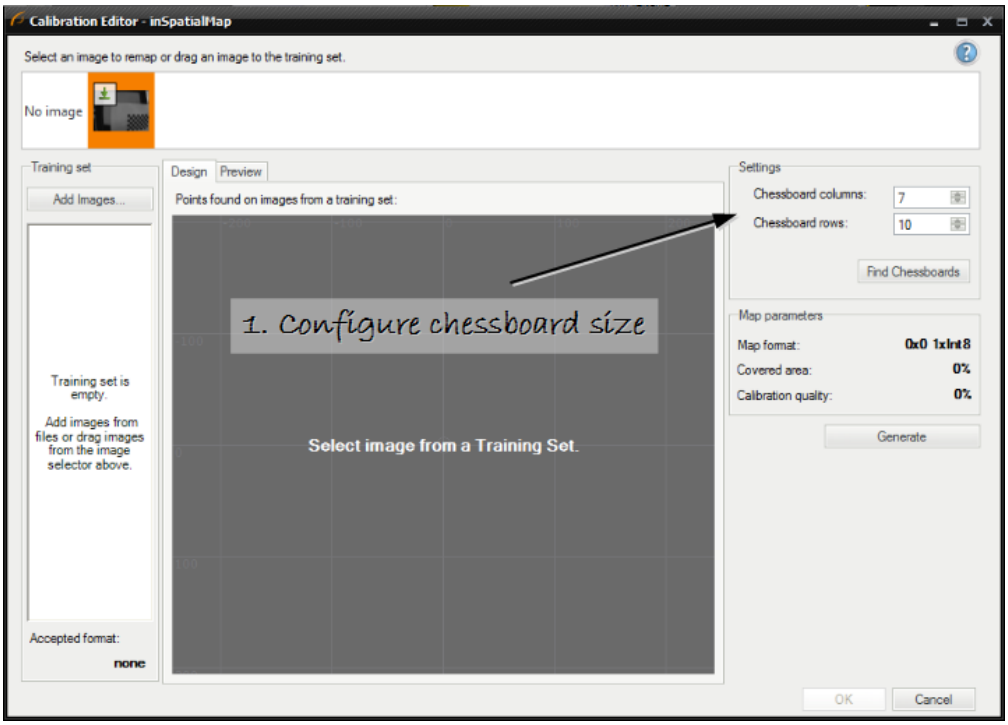
See also:

Template Matching guide, Template Matching filters

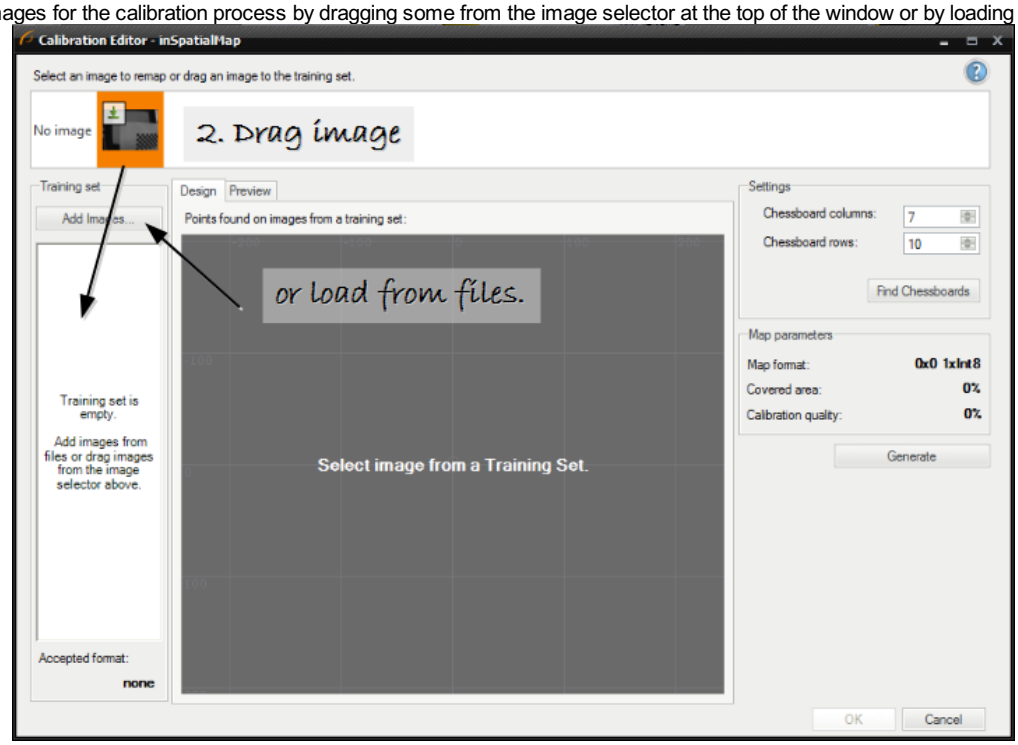
Creating Lens Undistortion Maps

To calibrate camera for lens distortion an easy graphical tool can be used. The following steps must be taken:

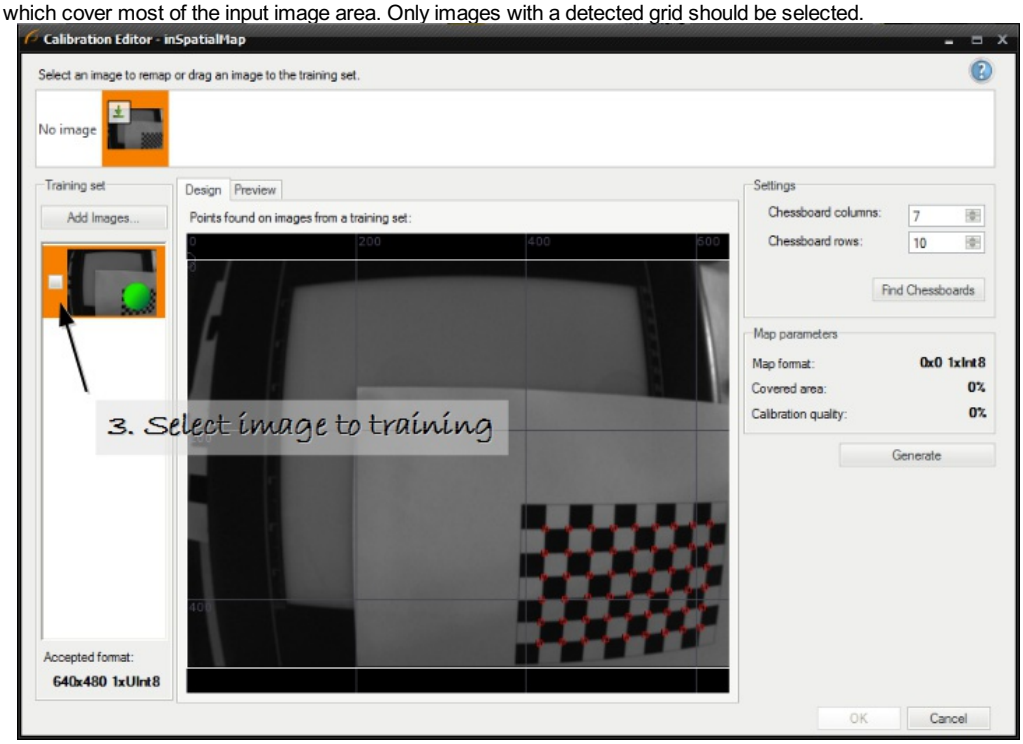
1. Add a **UndistortImage** filter to your program.
2. Select the filter and click the "..." button by the **inSpatialMap** input in the Properties window. The calibration editor will open up.
3. Set the number of rows and columns of the chessboard grid:



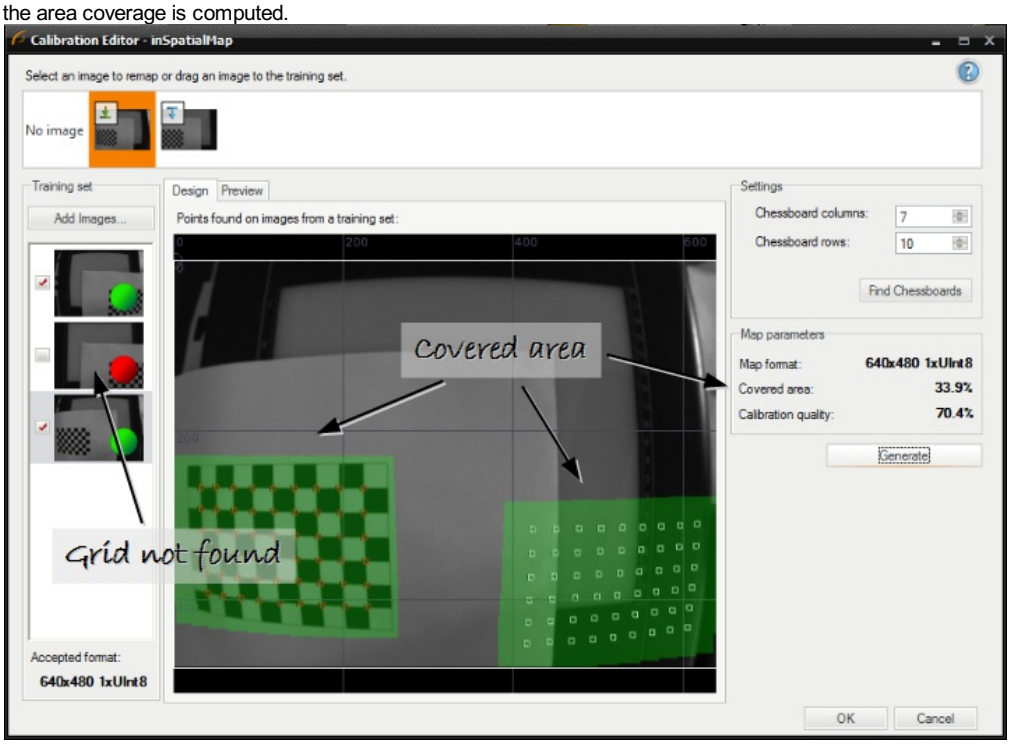
- Collect input images for the calibration process by dragging some from the image selector at the top of the window or by loading them from files.



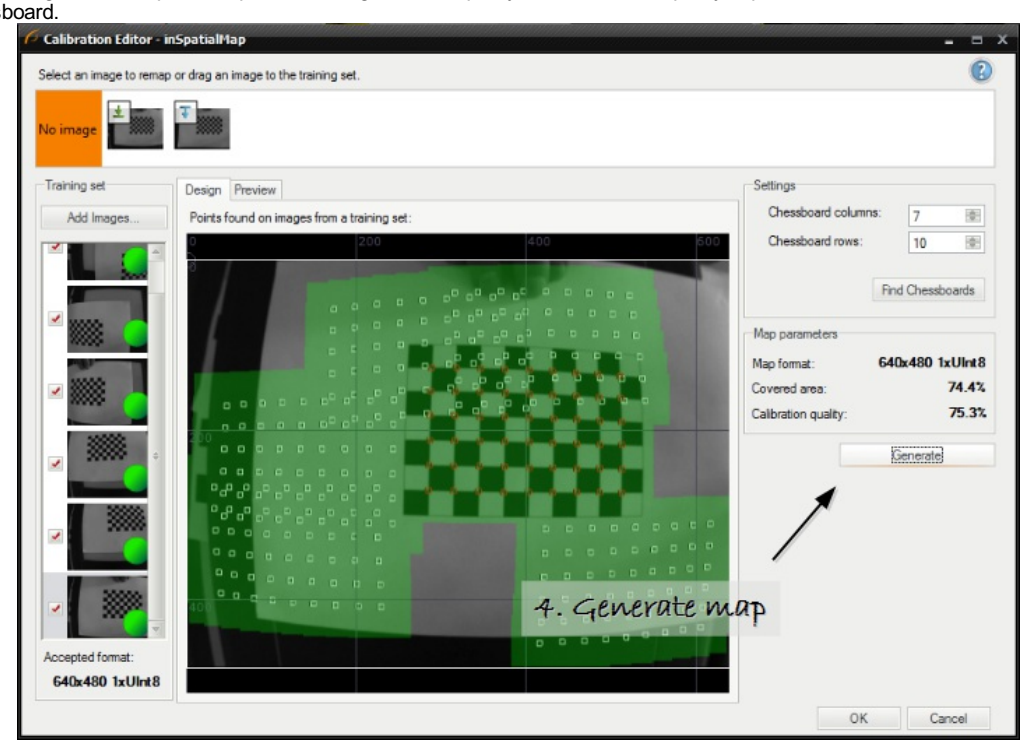
- Select images which cover most of the input image area. Only images with a detected grid should be selected.



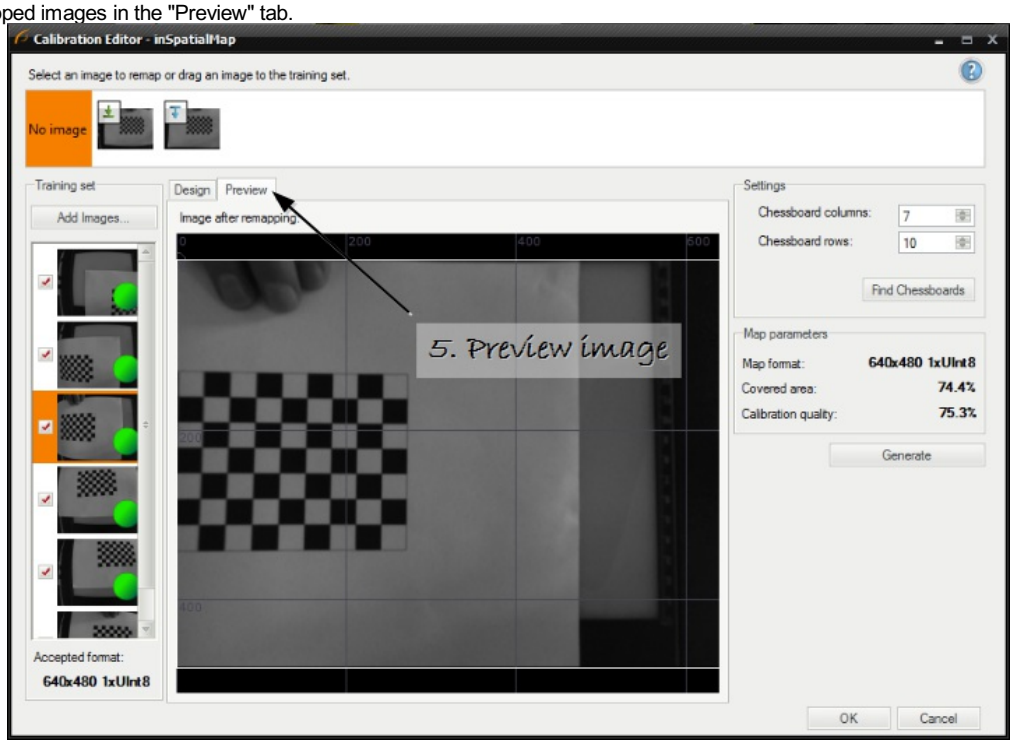
After the selection the area coverage is computed.



- Click "Generate" to generate a spatial map. Check the generation quality. The calibration quality depends on the covered area and the quality of the calibration chessboard.



- Preview remapped images in the "Preview" tab.



Tips on generating map:

- Try to cover at least 75% of the input image area. To get high quality results the chessboard grid should appear near all borders of the image.
- Avoid overlapping of the chessboard grids. These point will not improve the quality but will extend computation time.
- Put the chessboard not only at different locations, but also at different rotations.
- Reducing the number of images will reduce computation time.
- Always check results of remapping in the preview window. Provided invalid chessboard points can introduce a computation error, which can result in invalid calibration map.

Creating Text Segmentation Models

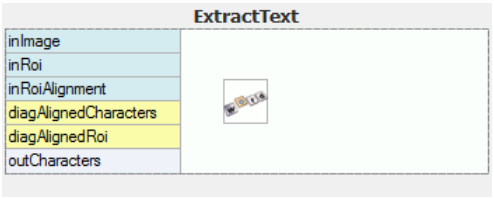
The graphical editor for text segmentation performs two operations:

1. **Thresholding** an image with one of several different methods to get a single foreground region corresponding to all characters.
2. **Splitting** the foreground region into an array of regions corresponding to individual characters.

Details about using OCR filters can be found in Machine Vision Guide: Optical Character Recognition.

To configure text extraction please perform the following steps:

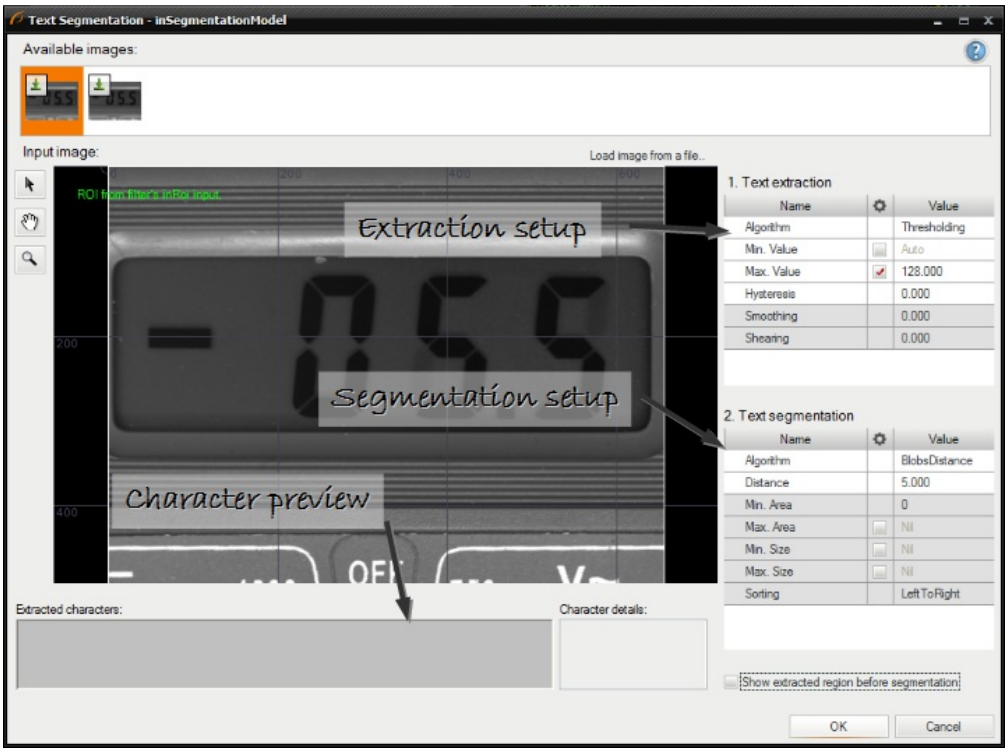
1. Add an **ExtractText** filter to the program.



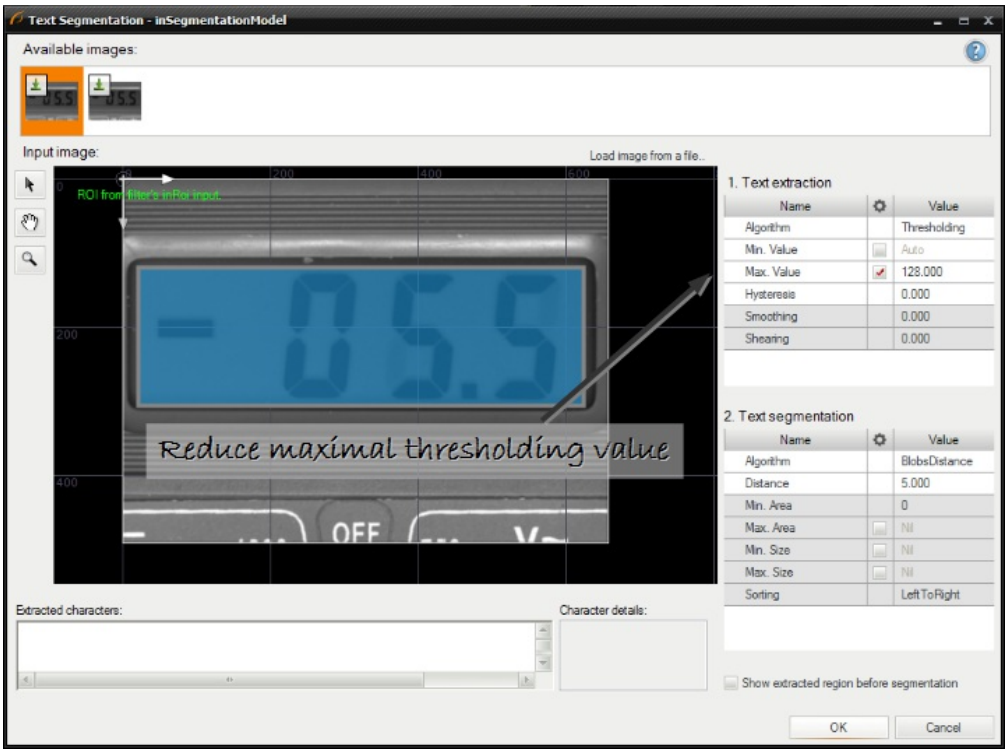
2. Set the region of interest on the inRoi input. This step is necessary before performing next steps. The image below shows how the ROI was selected in an example application:

3. Click on the "..." button at the **inSegmentationModel** input to enter the graphical editor.
4. When entering first time, complete the quick setup by selecting most common settings. In this example a black non-continuous text should be extracted from a uniform background. Configuration was set to meet these requirements.

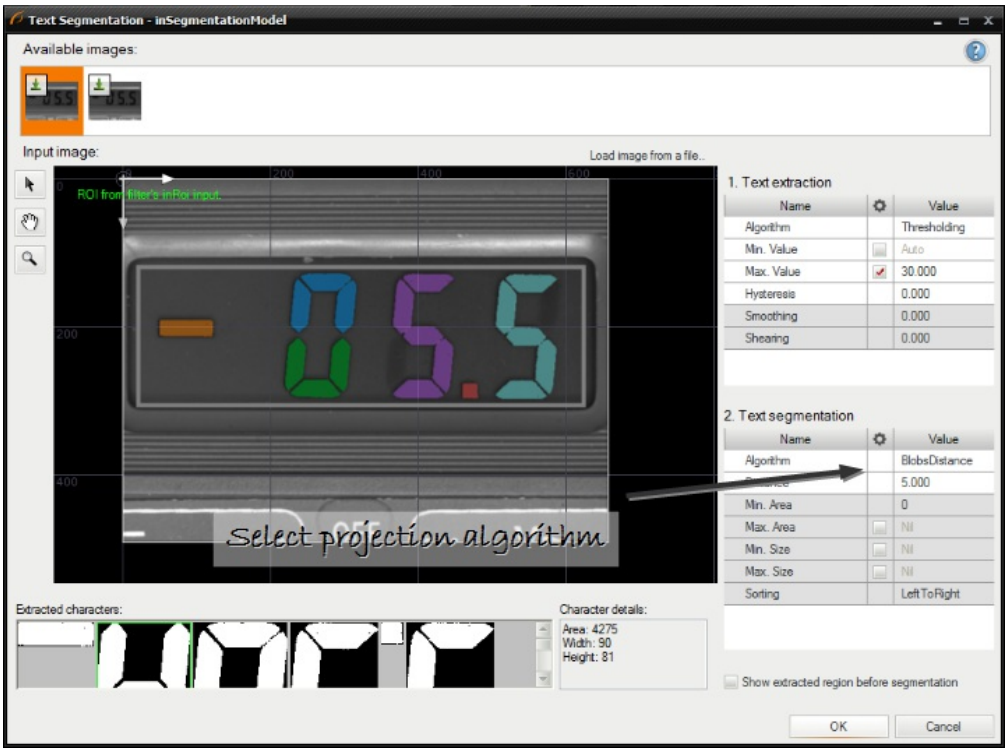
5. After the quick setup the graphical editor starts with some parameter set. Adjust the pre-configured parameters to get best results.



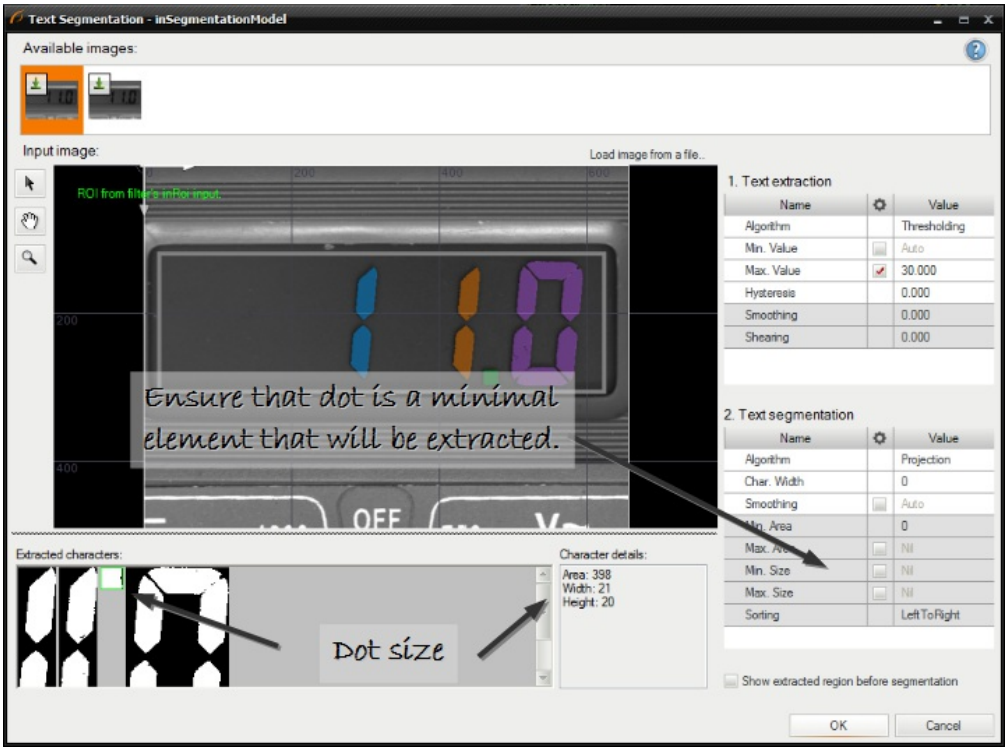
6. Configure a character extraction algorithm. In this case thresholding value is too high and must be reduced.



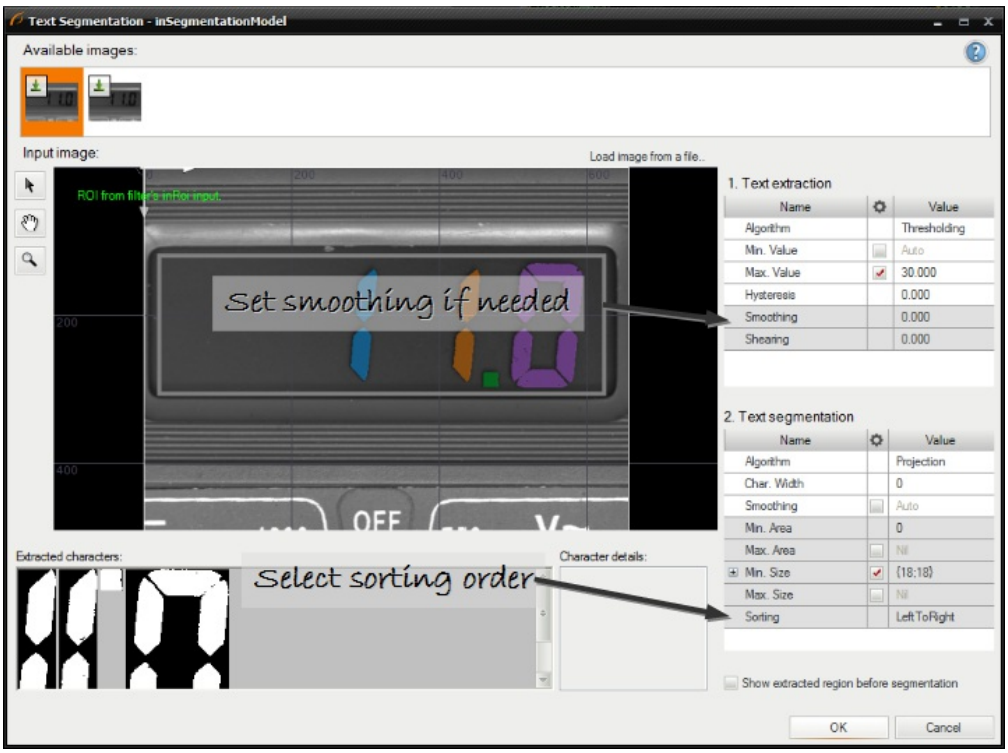
7. Select a character segmentation algorithm.



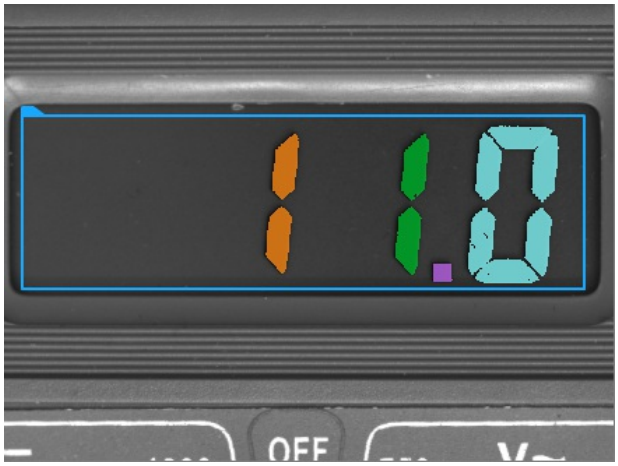
8. Set the minimal and the maximal size of a character. The editor shows the character dimensions when the character is selected in the list below.



9. Select a character sorting order, character deskewing (shearing) and image smoothing. Smoothing is important when images have low quality.

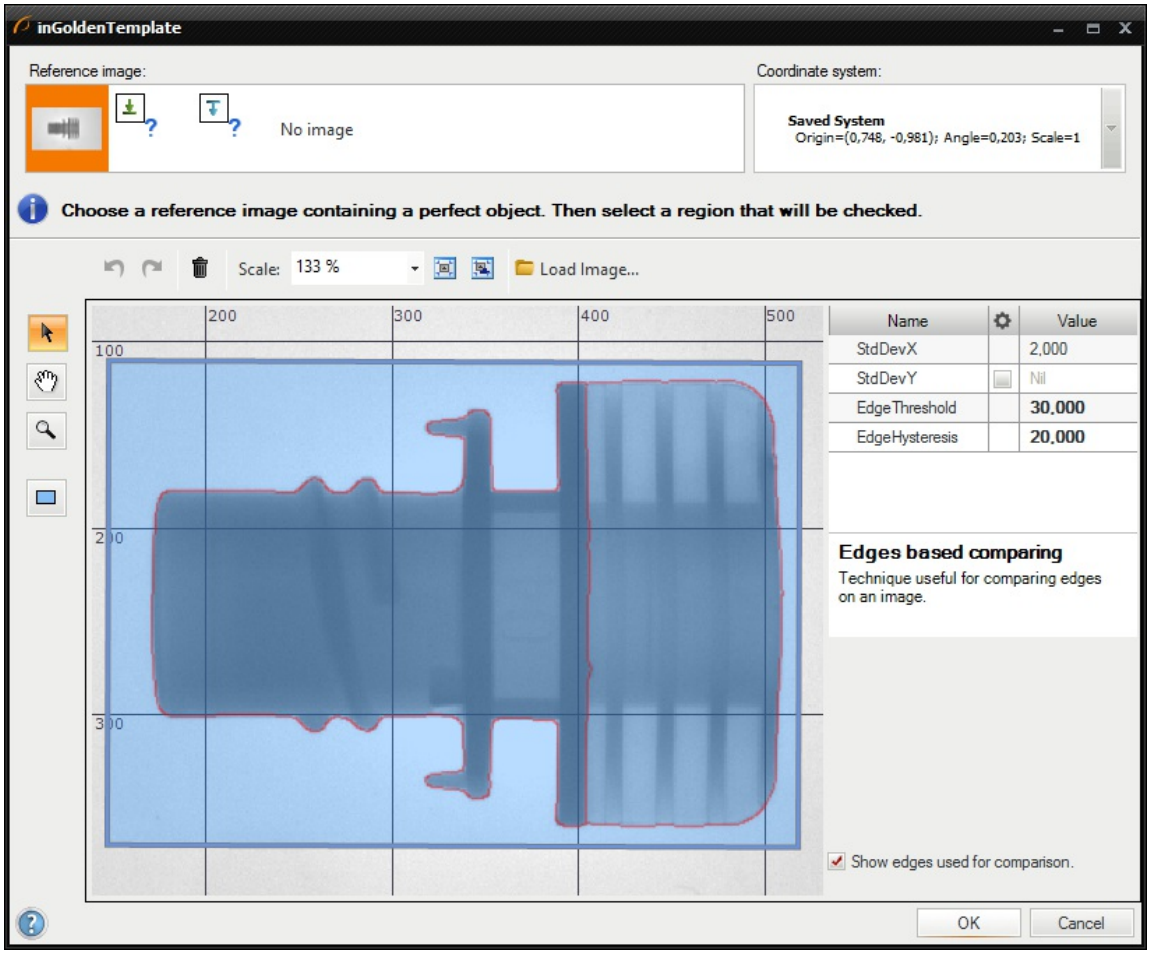


10. Check results using available images.



Creating Golden Template Models

Golden template technique is the most powerful method for finding objects' defects. Editor presented below is available in filters **CompareGoldenTemplate_Intensity** and in **CompareGoldenTemplate_Edges**.



To create golden template, select template region and configure its parameters.

Remarks:

- To reduce computation time try to select only necessary part of an object,
- For comparing both edges and surface use two **CompareGoldenTemplate** filters,
- To create mode programmatically use filter **CreateGoldenTemplate**.

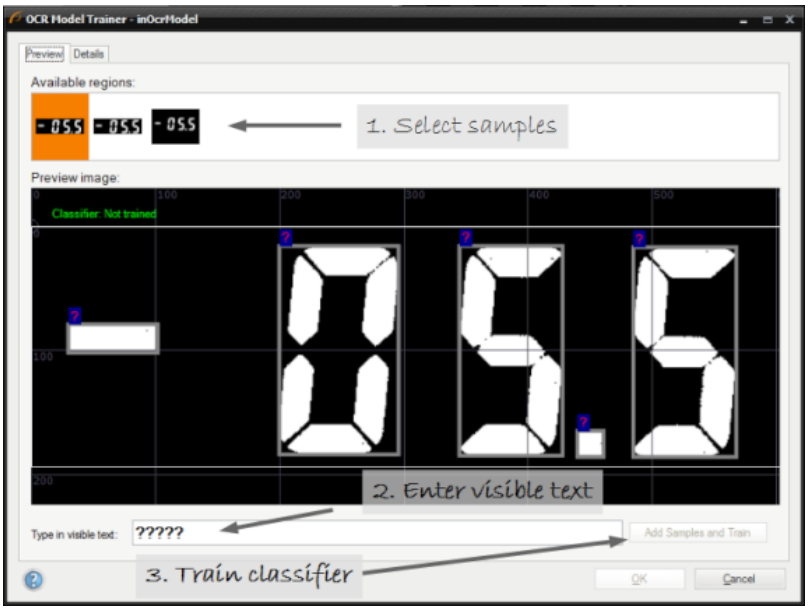
Creating Text Recognition Models

Text recognition editor creates an OCR model for getting text from regions. More details about the OCR technique can be found in Machine Vision Guide: Optical Character Recognition.

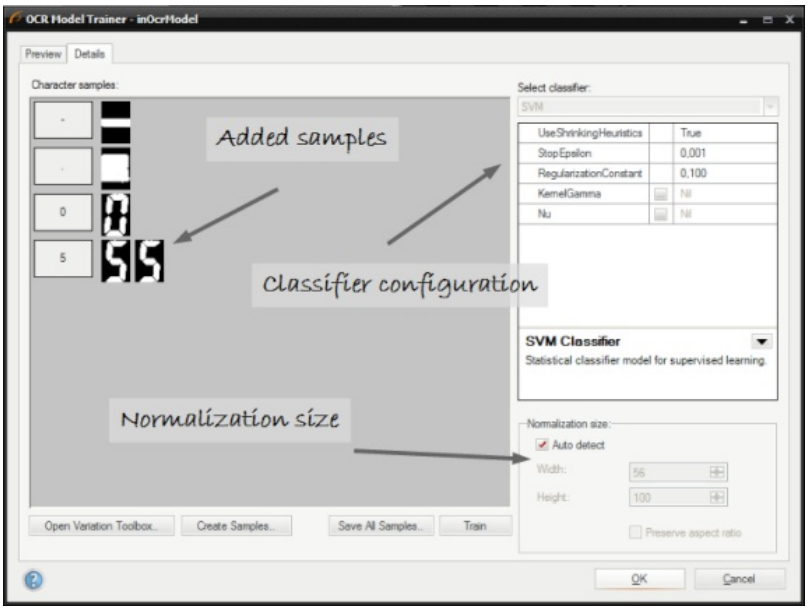
To create an OCR model a set of characters should be collected. If recognition score is low after training based on real samples then artificial character variations can created.

Creation of a model consists of following steps:

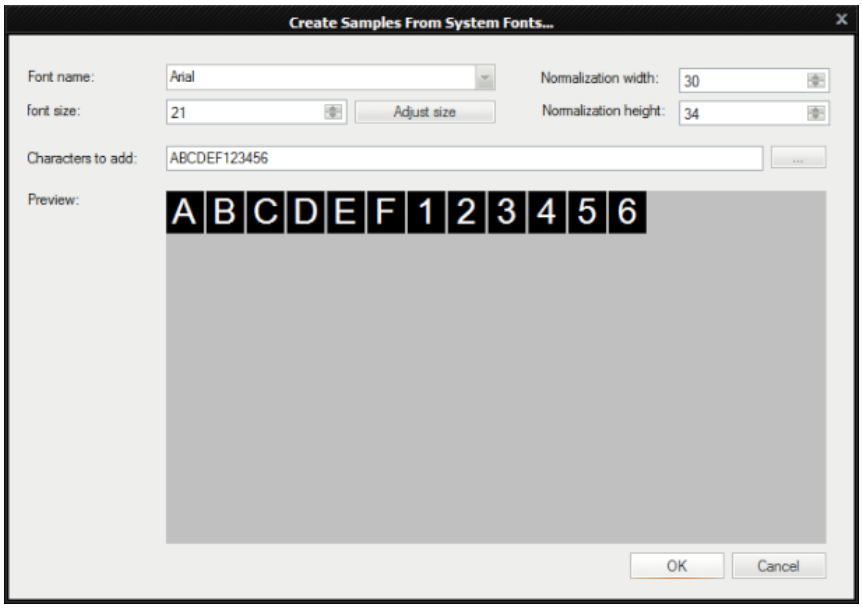
1. **Collecting real samples** - after opening the editor characters are visible and can be added to a training set.



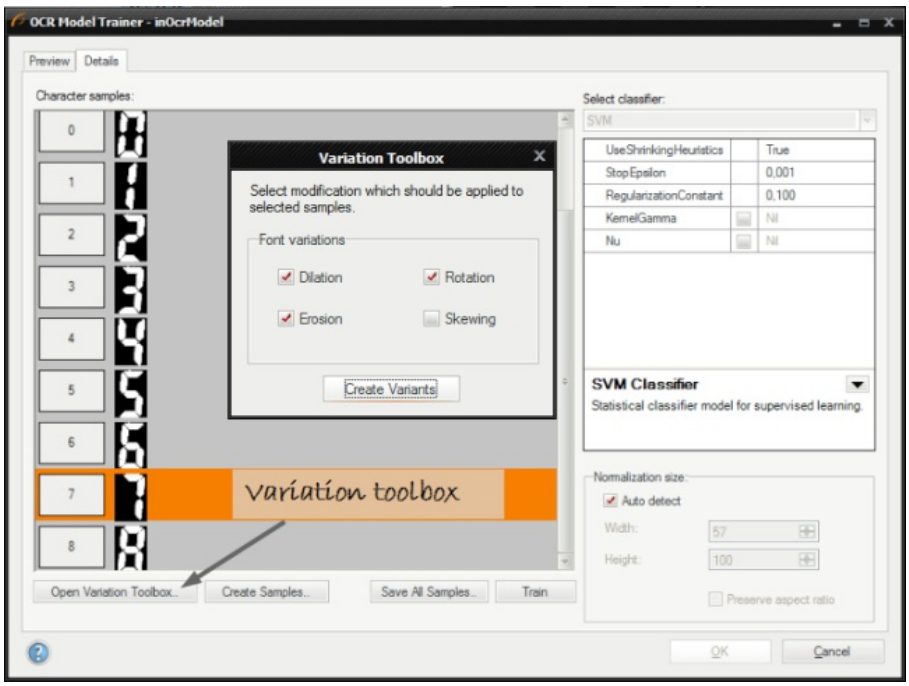
After the training character samples can be viewed in the details tab:



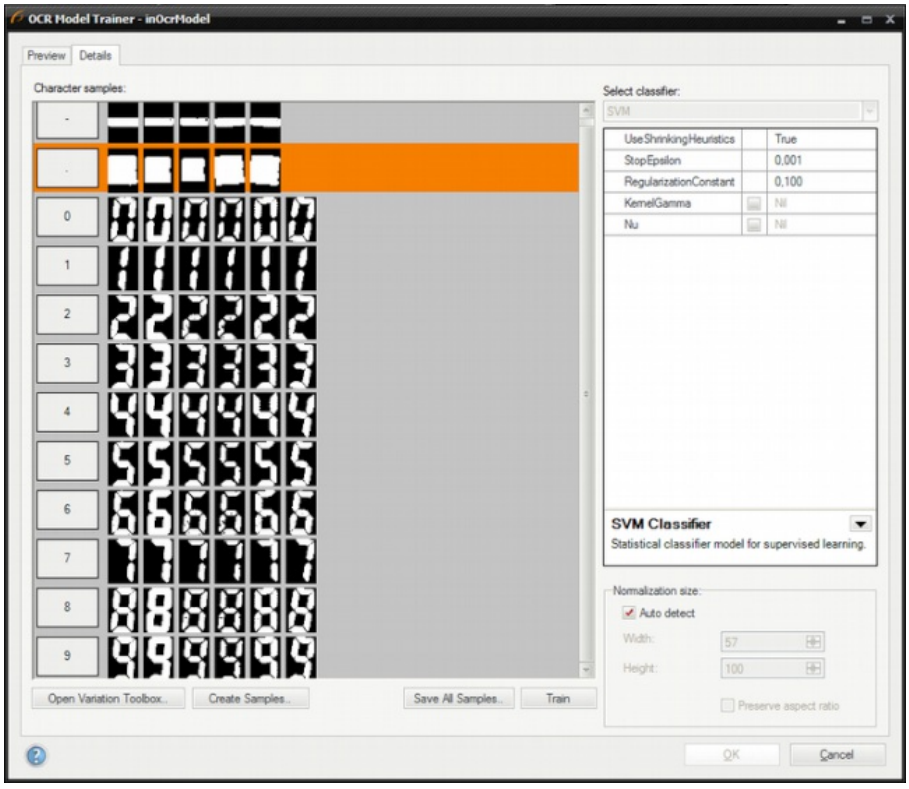
2. **Creating artificial samples** - when no samples are available user can create a training set using systems fonts.



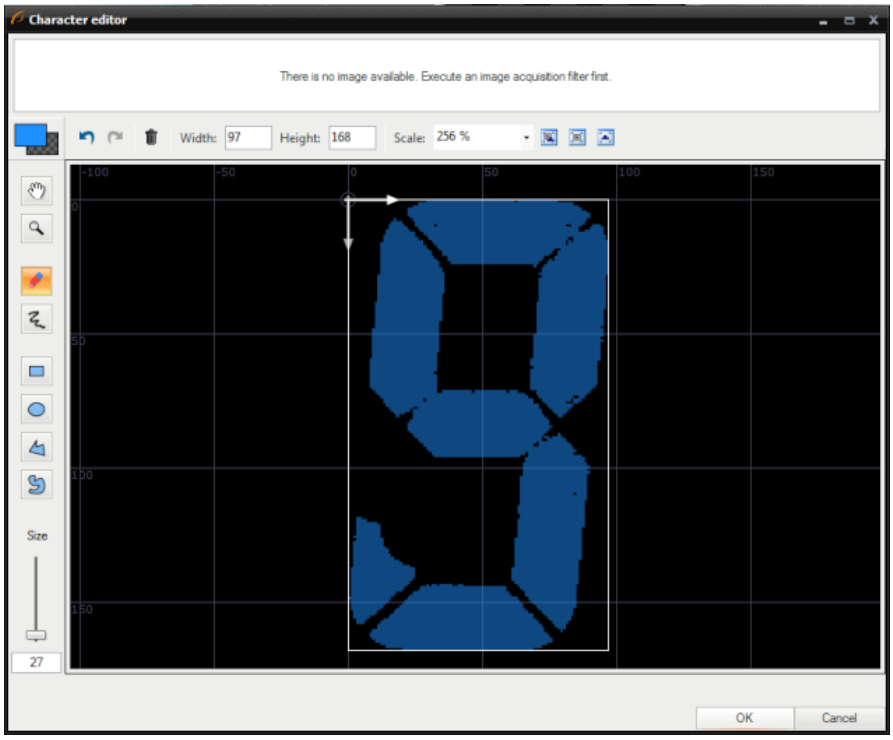
3. **Creating character variations** in case when no more samples are available and the training result is not fine the editor can modify existing samples to create a new set.



The training set after adding new samples variations:



4. **Editing samples** - in case when gathered samples contain noises, or its quality is low, user can edit them manually. The image below show how to edit a character '8' to get character '9'.

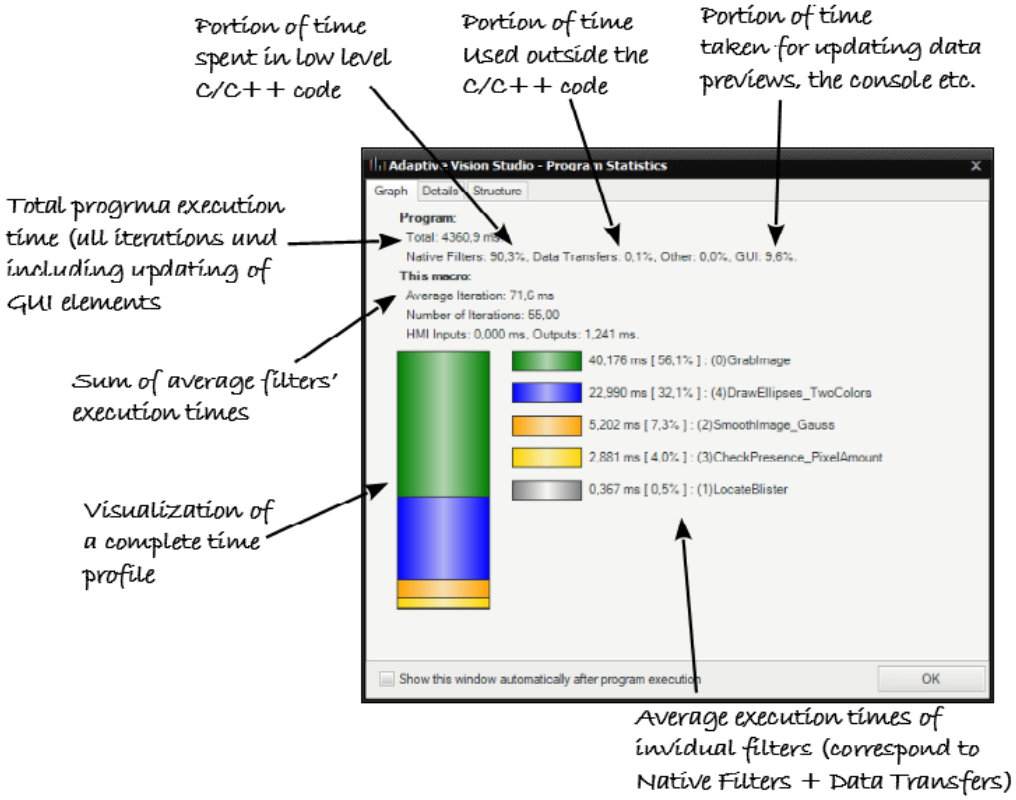


Note:

- Each training character should have this same number of samples.
- In cases when some characters are very similar number of samples can be increased to improve classification.
- Character samples can be stored in an external directory to perform experiments on them.

Analysing Filter Performance

The Program Statistics window contains information about the time profile of the selected macrofilter. This is very important as real vision algorithms need to be very fast. However, before starting program optimization we must know what needs to be optimized, so that later we optimize the right thing. Here is how the required information is presented:



As can be seen on the above illustration, program execution time is affected by several different factors. The most important is the "Native Filters" time, which corresponds to the core data processing tools. Additional time is consumed by "Data Transfers", which is related to everything that happens on connections between filters – this encompasses automatic conversions as well as packing and unpacking arrays on array and singleton connections. Another statistic called "Other" is related to the virtual machine that executes the program. If it value is significant, then C++ code generation might be worth considering. The last element, "GUI", corresponds to visualization of data and program execution progress. You can expect that this part is related only to the development environment and can possibly be reduced down to zero in the runtime environment.

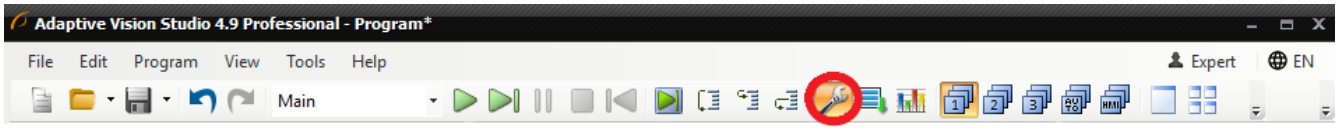
Remarks:

- In practice, performance statistics may vary significantly in consecutive program executions. It is advisable to run the program several times and check if the statistics are coherent. It might also be useful to add the **EnumerateIntegers** filter to your program to force a loop and collect performance statistics not from one, but from many program iterations.
- Turn off the diagnostic mode when testing performance.
- Data Preview Panels, animations in the Program Editor and even the Console Window can affect performance in Adaptive Vision Studio. Close these windows and use the "Update Data Previews Once an Iteration" option to test performance with minimal influence of the graphical environment.
- Even with all windows closed there are some background threads that affect performance. It will still be higher when you run the program with the Executor (runtime) application.
- Please note, that the first program iteration might be slower. This is due to the fact that in the first iteration memory buffers are allocated, filters are initialized, communication with external devices is established etc.

See also: Optimizing Image Analysis for Speed.

Seeing More in the Diagnostic Mode

Programs can be executed in two different modes: *Diagnostic* and *Non-Diagnostic*. The difference between them is in the computation of values on the diagnostic outputs. Values of this kind of outputs are computed only in the *Diagnostic* mode. They can be helpful in debugging programs but are not necessary in its final version. In the *Non-Diagnostic* mode, execution is faster because no diagnostic values are computed.



The Diagnostic Mode switch.

The execution mode can be easily changed in Adaptive Vision Studio using a button on the Application Toolbar. Outside of Adaptive Vision Studio, programs are always executed in the *Non-Diagnostic* mode to provide the highest performance.

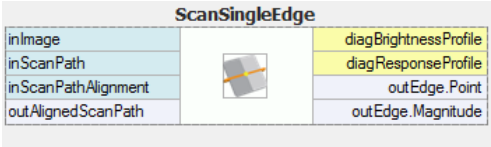
Diagnostic Filter Instances

Filters that have inputs connected to diagnostic outputs of some filters above them, are said to be diagnostic too. They are executed only when the program runs in the *Diagnostic* mode.

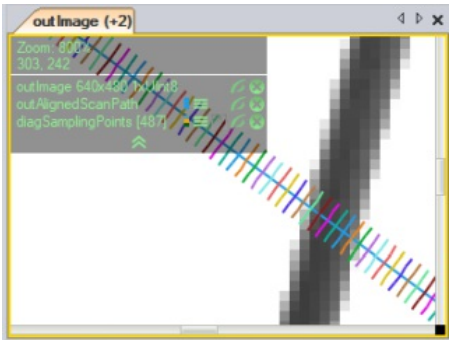
Example

The **ScanSingleEdge** filter has three diagnostic outputs:

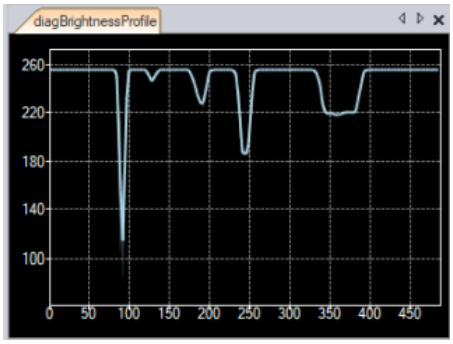
- **diagBrightnessProfile** is the profile of image brightness sampled along the scan path.
- **diagResponseProfile** is the profile derivative after preprocessing.
- **diagSamplingPoints** visualizes the points on the input image from where the brightness samples were taken.



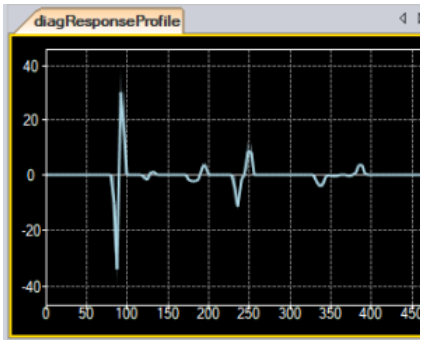
*The **ScanSingleEdge** filter.*



Input image with the scan path and the diagnostic sampling points.



The brightness profile.



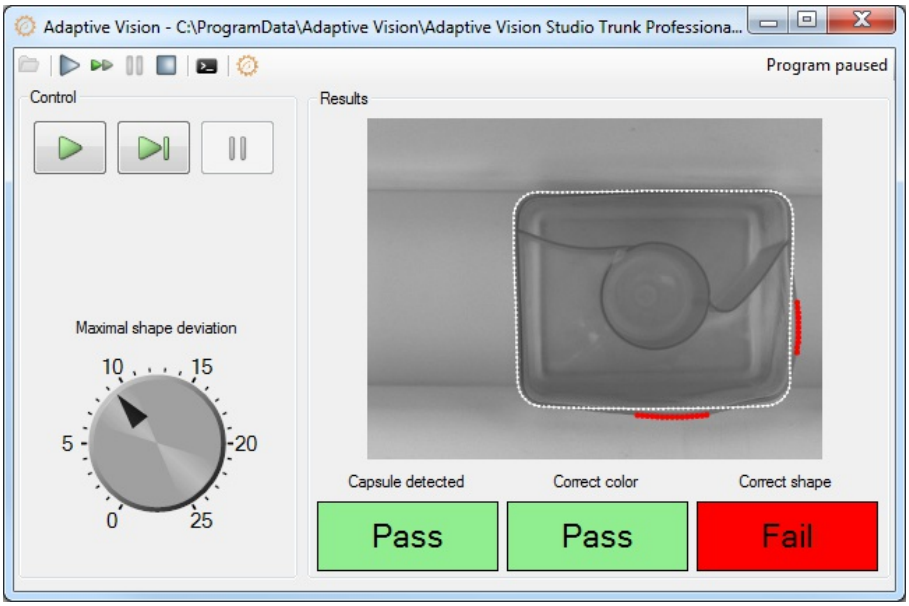
The response profile.

Deploying Programs with the Runtime Application

Introduction

Adaptive Vision Executor is a lightweight application that can run programs created with Adaptive Vision Studio. The GUI controls that appear in this application are the ones that have been created with the HMI Designer. The end user can manipulate the controls to adjust parameters and can see the results, but he is not able change the project.

Adaptive Vision Executor application is installed with the Adaptive Vision Studio Runtime package. It can be used on computers without the full development license. Only a runtime license is required. What is more, programs executed in Adaptive Vision Executor usually run significantly faster, because there is no overhead of the advanced program control and visualization features of the graphical environment of Adaptive Vision Studio.



The screen of Adaptive Vision Executor.

Usage

Open a project from a file and use standard buttons to control the program execution. A file can also be started using the Windows Explorer context menu command *Run*, which is default for computers with Adaptive Vision Studio Runtime and no Adaptive Vision Studio installed.

Please note, that Adaptive Vision Executor can only run projects created in exactly the same version of Adaptive Vision Studio.

Console mode

It is possible to run the Adaptive Vision Executor in the console mode. To do so, the `--console` argument is needed to be passed. Note, that this mode makes the `--program` argument required so the application will know which program to run at startup.

Adaptive Vision Executor is able to open a named pipe where it's log will be write into. This is possible with `--log-pipe` argument which accepts a pipe name to be opened. One may then connect to the pipe and process Adaptive Vision Executor log live. This can be easily done e.g. in C#:

```
var logPipe = new NamedPipeClientStream(".", "myProjectPipe", PipeDirection.In);
logPipe.Connect();

byte[] buffer = new byte[1024];
int count = 0;
while (logPipe.IsConnected && (count = logPipe.Read(buffer, 0, 1024)) > 0)
{
 Console.WriteLine(Encoding.UTF8.GetString(buffer, 0, count));
}
```

Available Adaptive Vision Executor arguments are as follows:

--program
: Path to the program to be loaded

--log-level
: Sets the logged information level

--console
: Runs the application in the console mode

--auto-close
: Automatically closes the application when program is finished. Meaningful only in console mode.

--language
: Specifies the language code to use as the user interface language.

--attach
: Attaches application process to the calling process console.

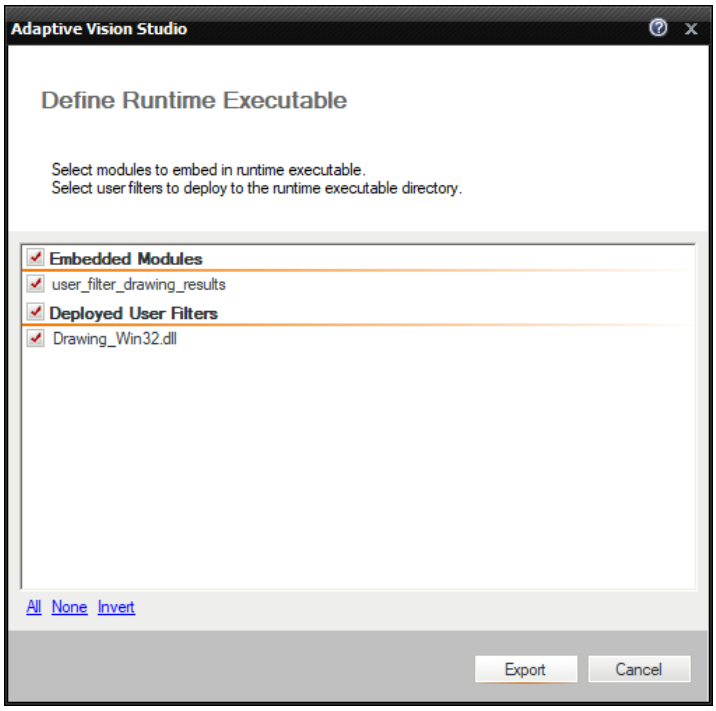
--log-pipe
: Creates a named pipe which will be populated with log entries during application lifetime. Meaningful only in console mode.

--help
: Displays help

Runtime Executables

Adaptive Vision Executor can open .avproj files, the same as Adaptive Vision Studio, however it is better to use .avexe files here. Firstly one can have a single binary executable file for the runtime environment. Secondly this file is encrypted so that nobody is able to look at project details. To create it open project in Adaptive Vision Studio and use *File » Export to Runtime Executable...*. This will produce an .avexe file that can be executed directly from the Windows Explorer.

If Adaptive Vision project contains any User Filter libraries, it is crucial to put their *.dll files into the appropriate directory when running in Adaptive Vision Executor. This is when exporting to .avexe file might also be a handy option. While defining the .avexe contents, it is possible to select all the User Filters libraries, the exporting Adaptive Vision project depends from. Selected libraries are deployed then to the same directory as generated .avexe file and the .avexe itself is set to use all User Filter libraries from its directory.



Defining the Runtime Executable.

In case there are any other dependencies, e.g. exposed by used User Filter libraries, one can add them into the Adaptive Vision project as an attachment in Project Explorer and also deploy with .avexe file during export.

Trick: INI File as a Module Not Exported to AVEXE

It is often convenient to have an *INI* file separated from the executable, so that various parameters can be adjusted for a particular installation (but not made available to the end user). There is no such feature as INI files in Adaptive Vision Studio, but it can be easily implemented with a simple programming idiom:

1. Use global parameters in your project for values that might require adjusting.
2. Place the global parameters in a separate module.
3. Exclude the module when exporting the .avexe file.
4. In the runtime environment copy the .avexe file and the module (as a separate file) with global parameters.
5. Open the INI module in the Notepad to edit it when needed.

Other Runtime Options

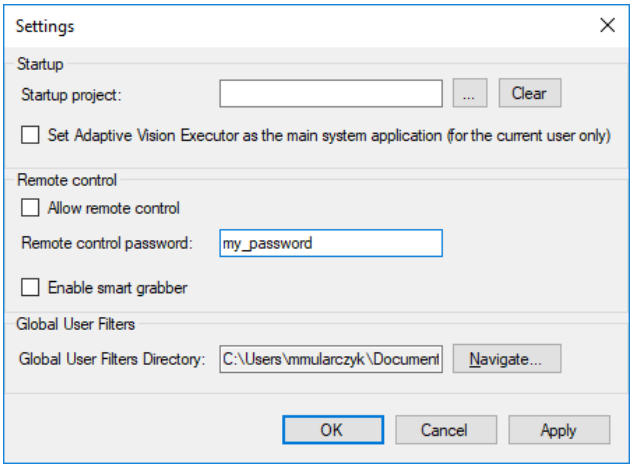
Please note, that Adaptive Vision Executor is only one of several options for creating end-user's applications. Other available options are:

- .NET Macrofilter Interface Generator – generates a native .NET assembly (a DLL file) and makes it possible to invoke macrofilters created in Adaptive Vision Studio as simple class methods. Internally the execution engine of Adaptive Vision Studio is used, so modifying the related macrofilters does not require to re-compile the .NET solution. The HMI can be implemented with WinForms, WPF or similar technologies.
- C++ Code Generator – generates a native C++ code (a CPP file) that is based on Adaptive Vision Library C++. This code can be integrated with bigger C++ projects and the HMI can be implemented with Qt, MFC or similar libraries. Each time you modify the program in Studio, the C++ code has to be re-generated and re-compiled.

Remote Access to the Runtime Application

Introduction

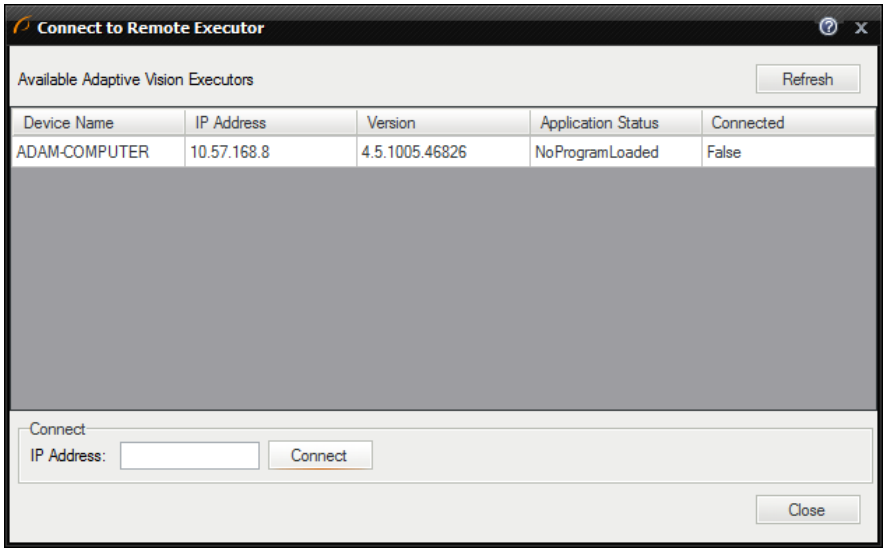
It is possible to use Adaptive Vision Studio to control an Adaptive Vision Executor running on a remote computer, if only the two computers are in the same Ethernet subnet. To enable remote access in Adaptive Vision Executor, the option "Allow remote control" should be enabled. For security reasons, it is also possible to protect connection with a password:



Enable remote executor.

Usage

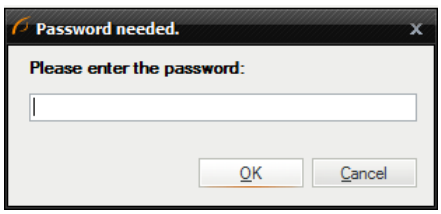
In Adaptive Vision Studio the list of available remote systems is accessible in the Connect to Remote Executor window which opens up after choosing the File › Connect to Remote Executor menu command:



Browse remote systems.

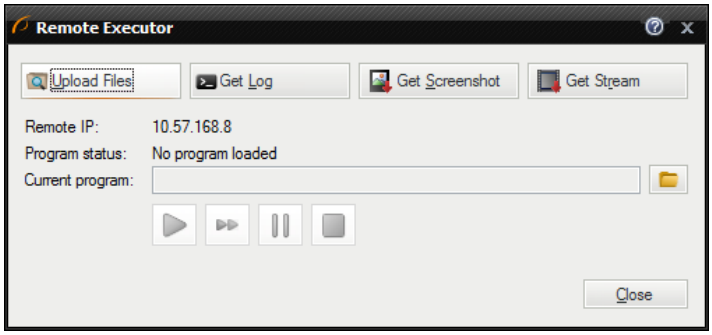
If an expected remote system is not visible on the list, please verify that: (1) it is configured for the same local area network, and (2) it can access Ethernet through its firewall (in the first place verify that in the Windows' Control Panel, Windows Firewall section, the option "Notify me when Firewall blocks a new program" is enabled).

When connecting to a selected device, the program will ask for the password (if it has been set):



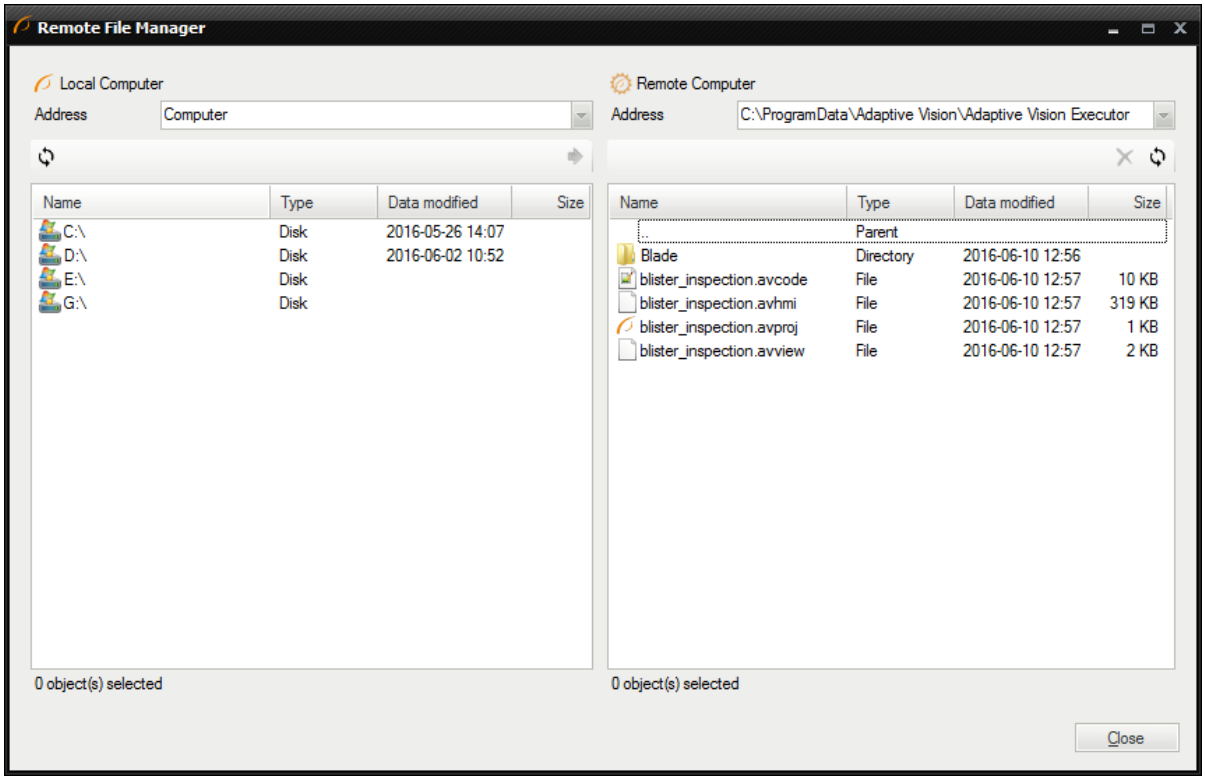
Password protection window.

After successful connection, management of the selected executor becomes possible. Now you are able to do several actions: Upload Files, Control Program and Get Diagnostics.



Remote executor window.

Upload Files:

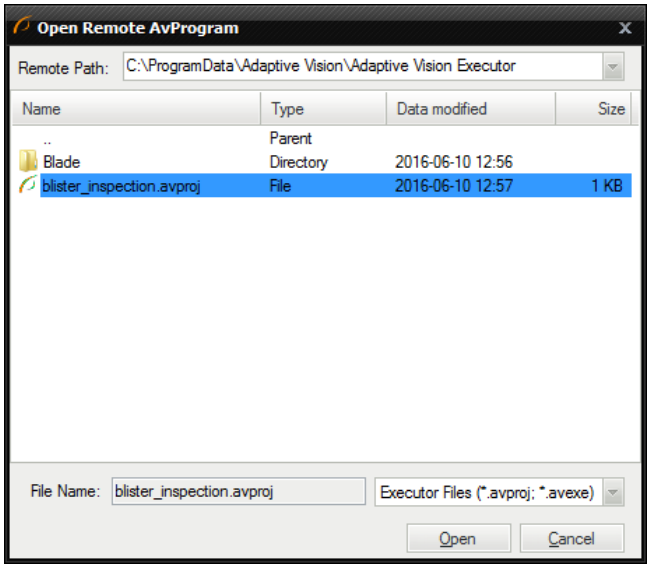


Upload files window.

To Manage Files on remote executor, click Upload Files button. In this window it is possible to send program files to the remote executor. By default all files are stored in the directory <CommonApplicationData>\Adaptive Vision\Adaptive Vision Executor.

Program Control:

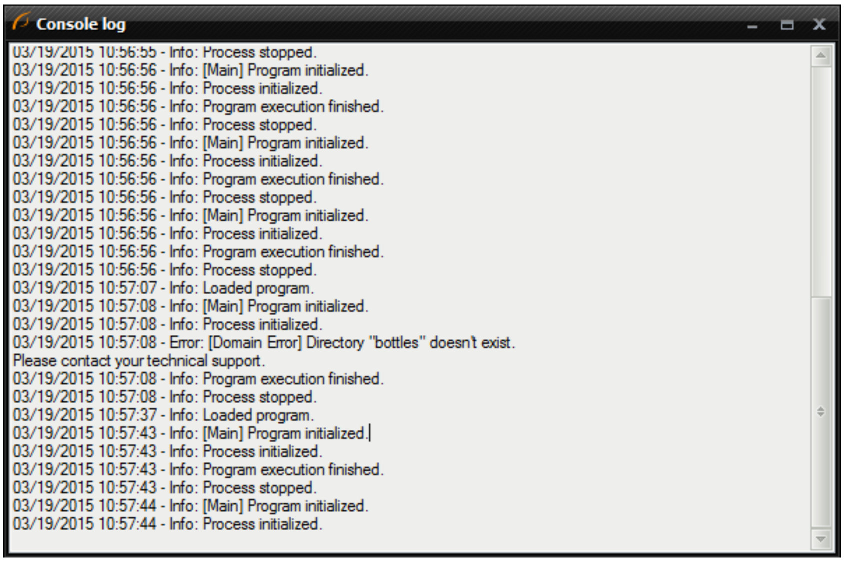
In the Remote executor window you are able to control currently executed program. In the middle of the window we can see the current program status and the path to the currently loaded program. Next there are four buttons controlling program execution. Finally there is an option to open another program.



Remote file dialog window.

Diagnostics:

In the Remote Executor window there is available a few diagnostic tools. Here, it is possible to preview the execution log and a screen preview from the currently connected executor. All diagnostic options are located at the top of the Remote executor window.



Log preview window.

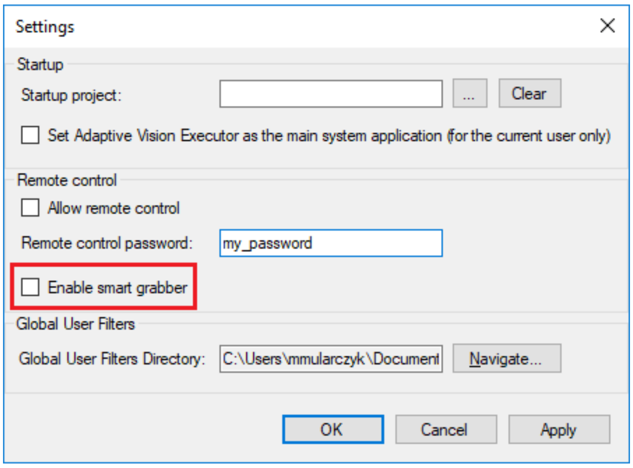
Remote Image Acquisition

Introduction

It is also possible to get images from a connected device using the remote executor. This method offers an easy way to acquire images using a different image protocols. Remote image acquisition filters enable to run this same code on the smart cameras (client side) as good as on developer computers without any additional modifications.

Usage

To enable remote image acquisition it is necessary to stop the executor program and enable the "Enable smart grabber" option:



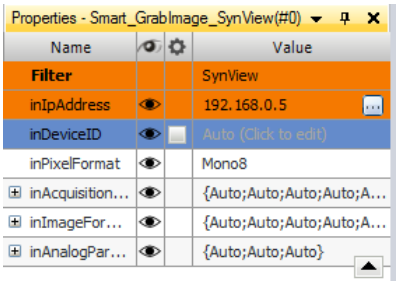
Enable Remote Image Acquisition.

In Adaptive Vision Studio it is possible to use the filters from the category Camera Support\Smart to get images from a device connected to the remote Executor system.

Currently the available image acquisition protocols are:

- **AvSMART** – access to AvSMART,
- **GenICam** – access to GenICam complaint devices,
- **Roseek** – access to Roseek cameras,
- **SynView** – access to NET GmbH cameras,
- **WebCamera** – access to DirectX complaint image sources like web cameras or frame grabbers.

To grab images using one of these protocols it is necessary to set **inIpAddress** in the filter input. The IP Address can be checked in the Connect to Remote Executor window when "Allow remote control" option is enabled in the Executor.

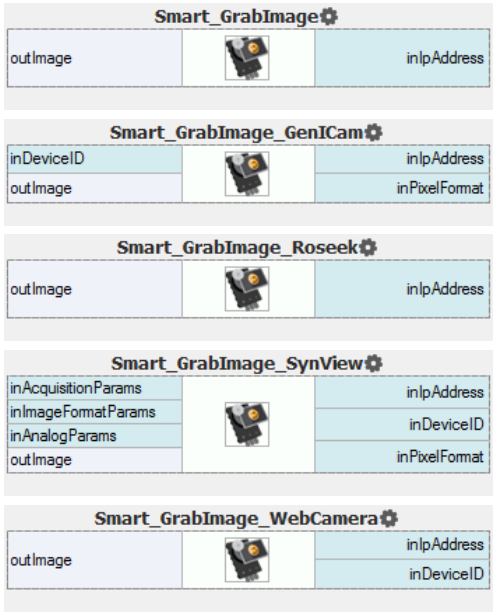


Smart_GrabImage_SynView *filter properties.*

If system detect that the provided IP address describes local machine the **Smart_GrabImage** filters will perform all operation on local computer. So it is no need to perform any changes during transferring project from developers machine to the client side device.

If the program is deployed to a device with another IP address then the input **inIpAddress** should be changed to the IP of the Executor or left empty. If the input **inIpAddress** is empty then the image is always grabbed using an appropriate local image acquisition protocol.

Available filters for the remote image acquisition:



If **inDeviceID** is set to*NIL*the first found device will be used.

Performing General Calculations

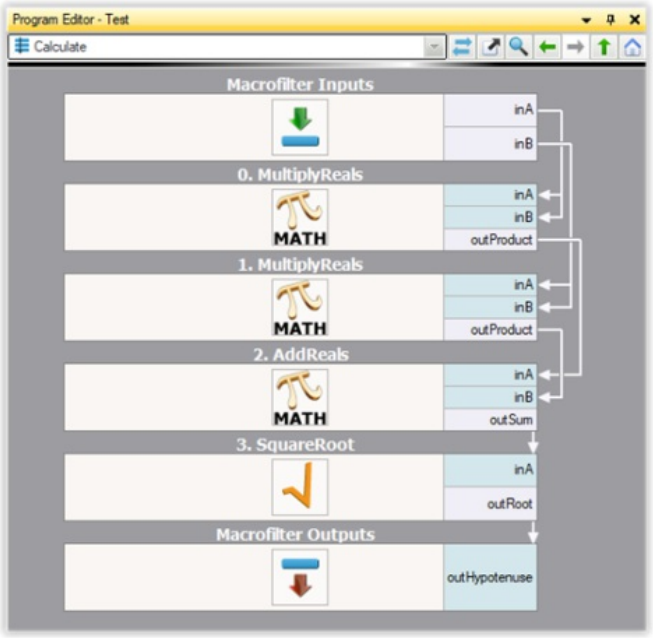
Introduction

Apart from using the image processing or computer vision tools, most often it is also necessary to perform some general calculations on numeric values or geometrical coordinates. There are two ways to do this in Adaptive Vision Studio:

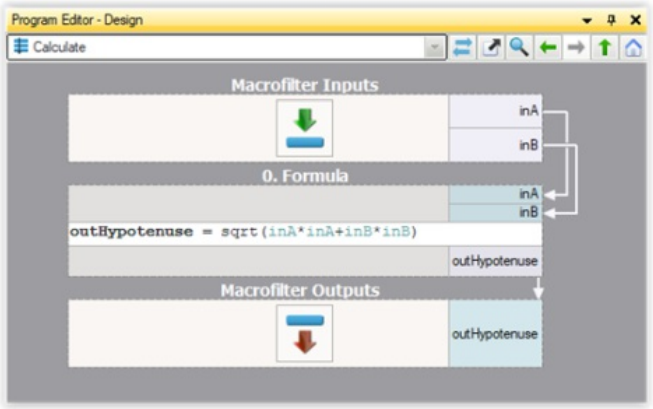
1. With the filters of the Standard Library.
2. With special formula blocks.

Example

Let us assume that we need to compute the hypotenuse $\sqrt{a^2+b^2}$. Here are the two possible solutions:



Calculations with standard filters.



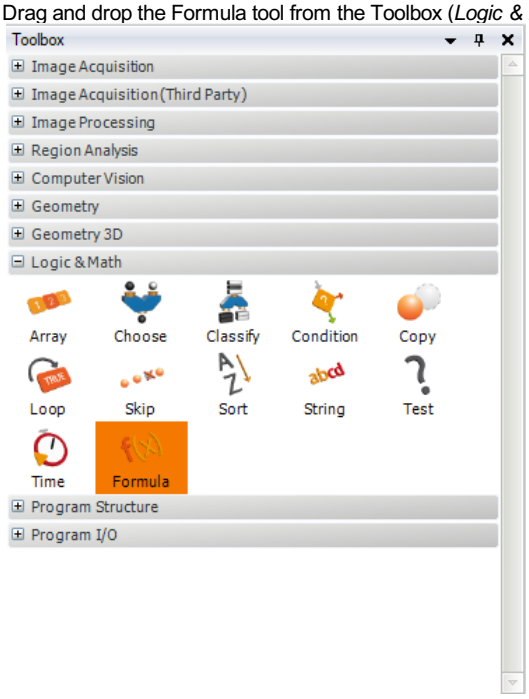
Calculations with formula blocks.

The second approach, with formula blocks, is the most recommended. Data flow programming is just not well suited for numerical calculations and standard formulas, that can be defined directly in formula blocks, are much more readable. You can also think about this feature as an ability to embed little spreadsheets in your machine vision algorithms.

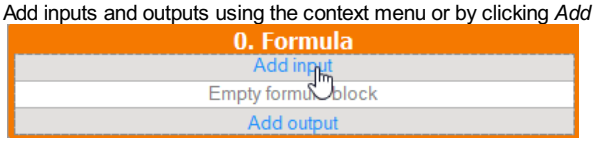
Creating Formula Blocks

To create a formula block:

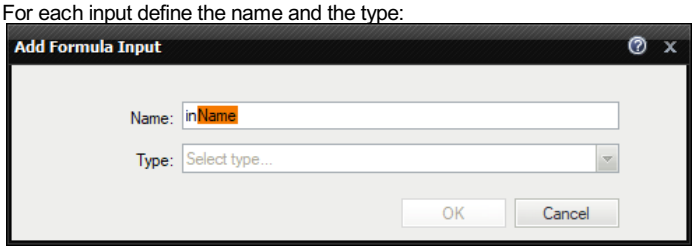
1. Drag and drop the Formula tool from the Toolbox (*Logic & Math* section) to the Program Editor:



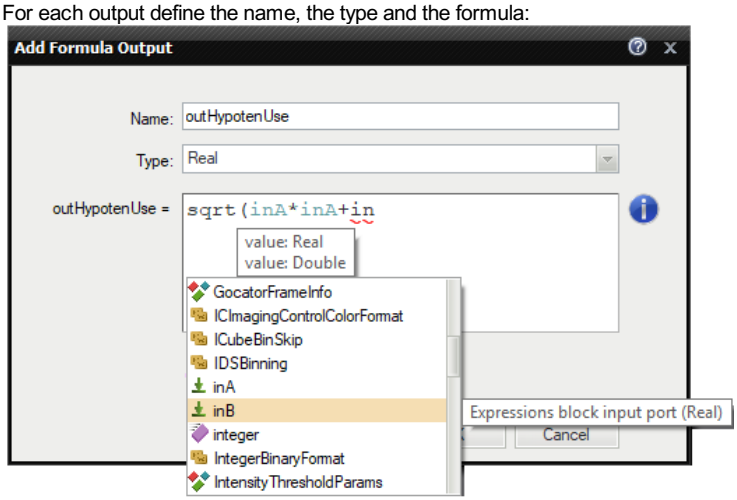
2. Add inputs and outputs using the context menu or by clicking *Add Input* and *Add Output* links:



3. For each input define the name and the type:



4. For each output define the name, the type and the formula:



Remarks

- Existing formulas can also be edited directly on a formula block in the Program Editor.
- It is also possible to create new inputs and outputs by dragging and dropping connections onto a formula block.
- Formula blocks containing incorrect formulas are marked with red background. Programs containing such filters can not be run.
- When defining a formula for an output it is possible to use other outputs, provided that they are defined earlier. The order can be changed through the context menu of the outputs.
- For efficiency reasons it is advisable not to use "heavy" objects in formulas, such as images or regions.

Syntax and Semantics

For complete information about the syntax and semantics please refer to the Formulas article in the Programming Reference.

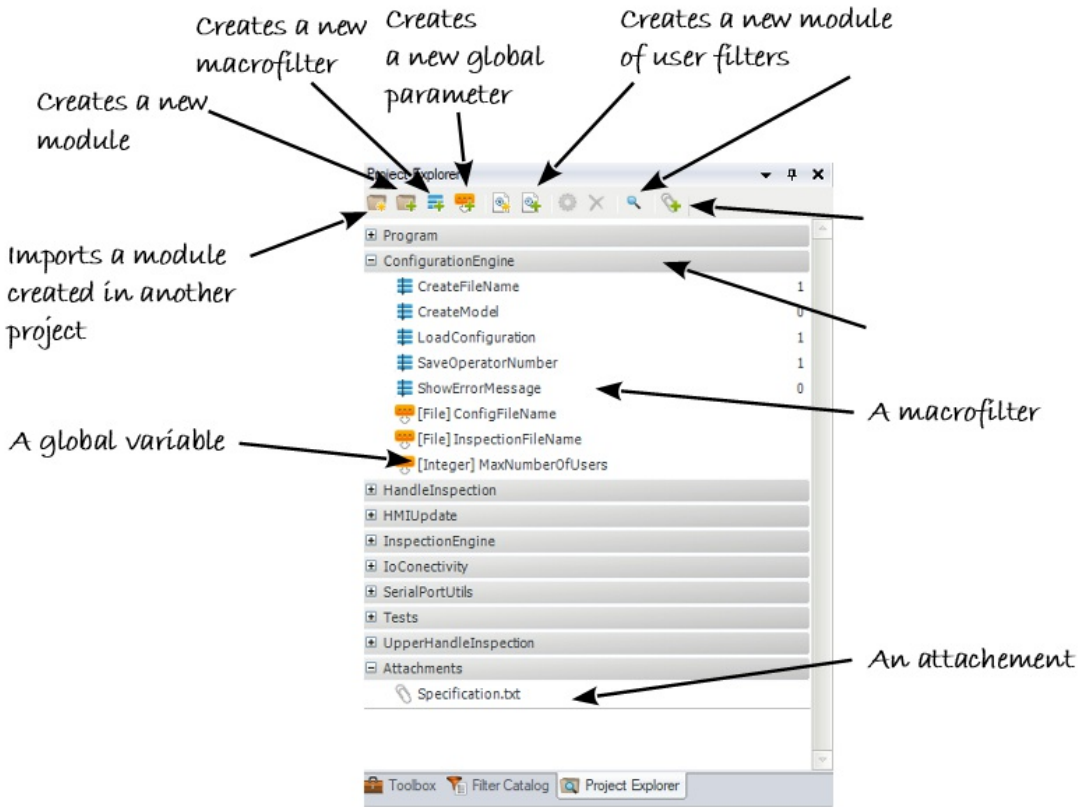
Managing Projects with Project Explorer

Introduction

Project Explorer is a window displaying elements which are contained in the currently opened project:

- Modules
  - Macrofilters (definitions)
  - Global Parameters
- User Filter libraries
- Attachments

Its main goal is to provide a single place to browse, add, remove, rename or open the items, which are grouped into categories in the same way as the filters in the Filter Catalog. A category may correspond to a standard module or to a module of a User Filter library. There is also one special category, Attachments, which appears when the user adds an external file to the project (it might be for example a text document with a piece of documentation).



Modules in the Project Explorer.

Opening Macrofilters

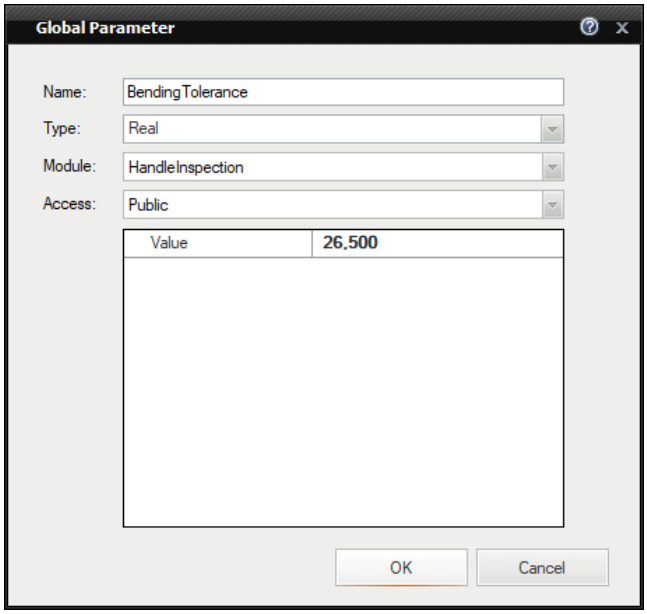
As described in Running and Analysing Programs, there are two ways of navigating through the existing macrofilters. One of them is with the Project Explorer window, which displays *classes* (definitions) of macrofilters, not the instances. After double-clicking on a macrofilter in the Project Explorer, however, a macrofilter instance is opened in the Program Editor. As one macrofilter class can have zero, one or many instances, some special rules apply to which of the instances it is:

- If possible, the most recently executed instance is opened.
- If no instance has been executed yet, the most recently created one is opened.
- If there are no instances at all the "ghost instance" is presented, which allows editing the macrofilter, but will never have any data on the output ports.

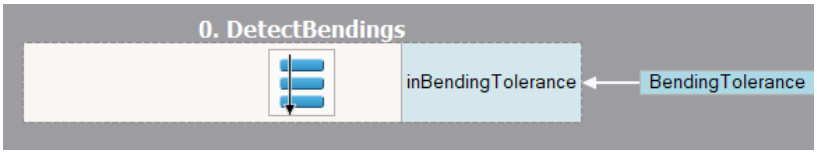
Global Parameters

If some value is used many times in several different places of a program, then it should be turned into a global parameter. Otherwise, consecutive changes to the value will require the error-prone manual process of finding and changing all the occurrences. It is also advisable to use global parameters to clearly distinguish the most important values from the project specification – for example the expected dimensions and tolerances. This will make a program much easier to maintain in future.

In Adaptive Vision Studio global parameters belong to specific modules and are managed in the Project Explorer. To create one, click the *Create New Global Parameter...* button and then a dialog box will appear, where you will provide the name, the type and the value of the new item. After a global parameter is created it can be dragged-and-dropped on filter inputs and appropriate connections will be created with a visual label displaying the name of the parameter.



Creating a global parameter.



Global parameter used in a program.

Global parameters contained in a project can also be edited in the Properties window after being selected in the Project Explorer or in the Program Editor.

Remarks:

- Connected filters are not re-executed after the global parameter is changed. This is due to the fact, that many filters in different parts of the program can be connected to one global parameters. Re-executing all of them could cause unexpected non-local program state changes and thus is forbidden.

Modules

When a project grows above 10-20 macrofilters it might be appropriate to divide it into several separate modules, each of them would correspond to some logical part. It is advisable to create separate modules for things like i/o communication, configuration management or for automated unit testing. Macrofilters and global variables will be then grouped in a logical way and it will be easier to browse them.

Modules are also sometimes called "libraries of macrofilters". This is because they provide a means to develop sets of common user's tools that can be used in many different projects. This might be very handy for users who specialize in specific market areas and who find some standard tasks appearing again and again. Here are some guidelines on how to use modules in such situations:

- Create a separate module for each set of related, standard macrofilters.
- Give each module a unique and clear name.
- Use the English language and follow the same naming conventions as in the native filters.
- Create the common macrofilters in such a way, that they do not have to be modified between different projects and only the values of their parameters have to be adapted.
- If some of the macrofilters are intended as implementation only (not to be used from other modules), mark them as *private*.

See also: Trick: INI File as a Module Not Exported to AVEXE.

Creating Deep Learning Model

Contents:

Introduction

Deep Learning Editor allows the user to create **DeepModel** objects for Deep Learning tasks.

Requirements:

- Deep Learning license is required to use Deep Learning Editor and filters.
- Deep Learning Service must be up and running to perform model training.

Currently four deep learning tools are available:

1. **Anomalies detection** - technique which enables the user to find defects based on the provided samples. The user only has to mark which images are good and which are bad.
2. **Features detection** - technique in which the user has to select which kind of objects should be found on the provided samples.
3. **Object classification** - technique which enables the user to identify images. The user only has to label provided samples with respect to desired number of classes. Images are identified with respect to those classes.
4. **Instance segmentation** - technique which enables a user to locate, segment and classify multiple objects in images. The user only draws regions to mark objects of proper classes.

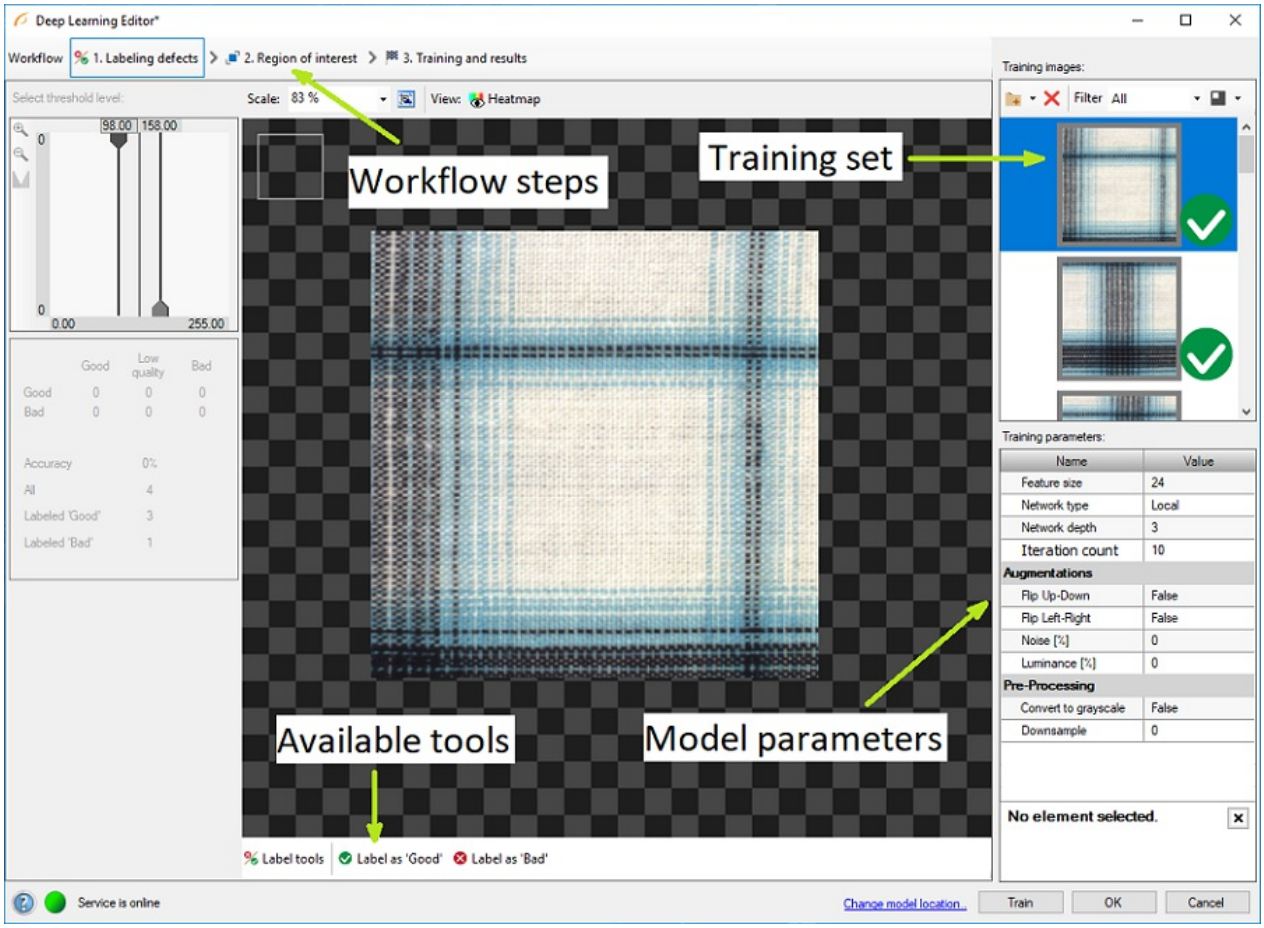
Technical details about these tools are available at Machine Vision Guide: Deep Learning

Workflow

To open the **Deep Learning Editor**, place the relevant Deep Learning filter in the Program Editor, go to its Properties and click on the button next to the **inDeepModel** parameter.

The Deep Learning model preparation process is usually split into the following steps:

1. **Loading images** - load the training images from disk
2. **Labeling images** - mark features or attach labels on each training image
3. **Setting the region of interest (optional)** - select the area of the image to be analyzed
4. **Adjusting training parameters** - select training parameters, preprocessing steps and augmentations for the particular application
5. **Training the model and analyzing results**



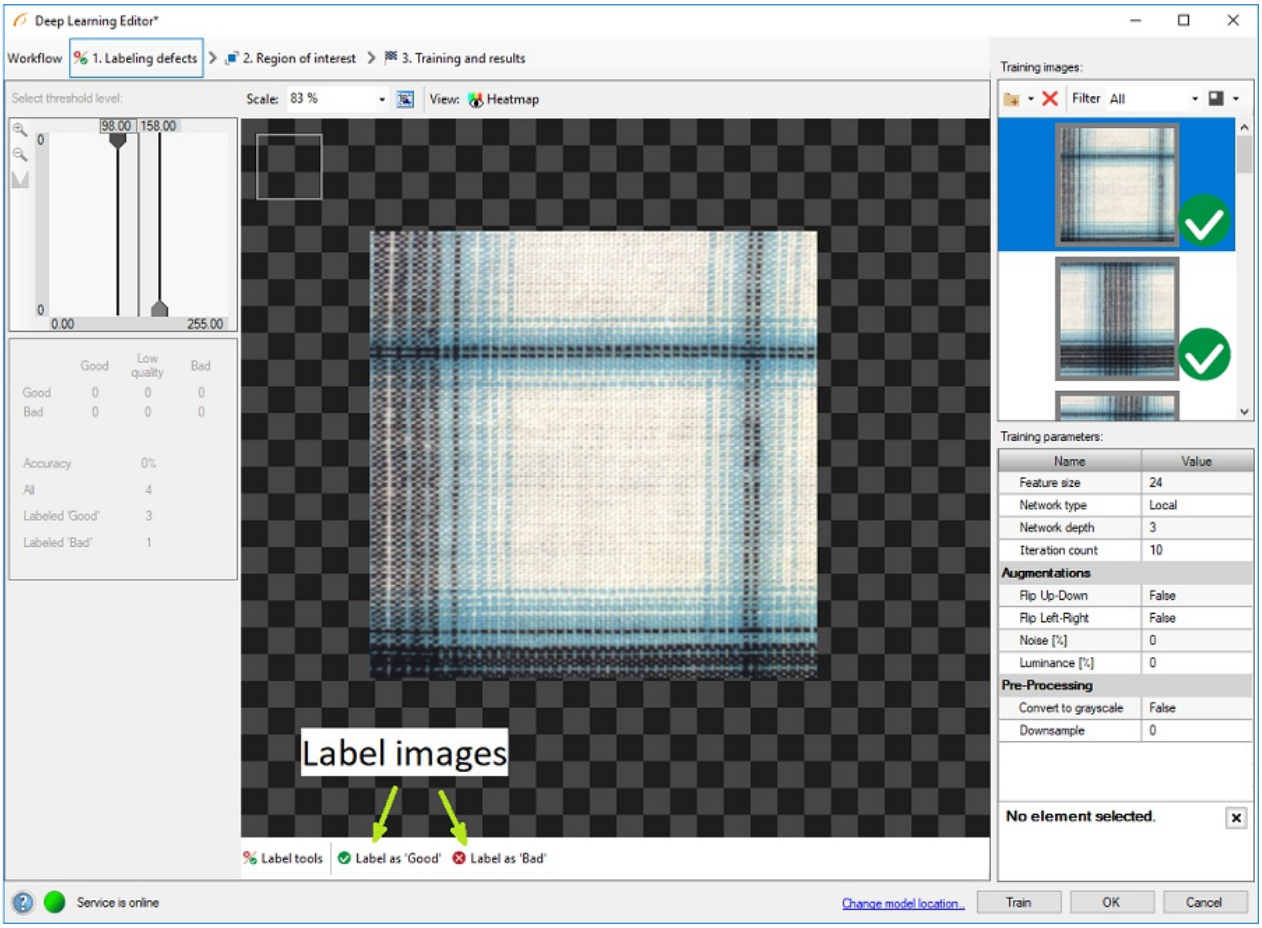
The Deep Learning Editor overview.

Detecting Anomalies

In this algorithm variant, the user only need to mark which images contain correct cases (good) or incorrect ones (bad).

1. Marking good and bad samples

Use **Label as 'Good'** or **Label as 'Bad'** buttons to label each image in the training set. Green and red icons on the right side of the training images indicate to which set an image was assigned.



Labeled images in Deep Learning Editor

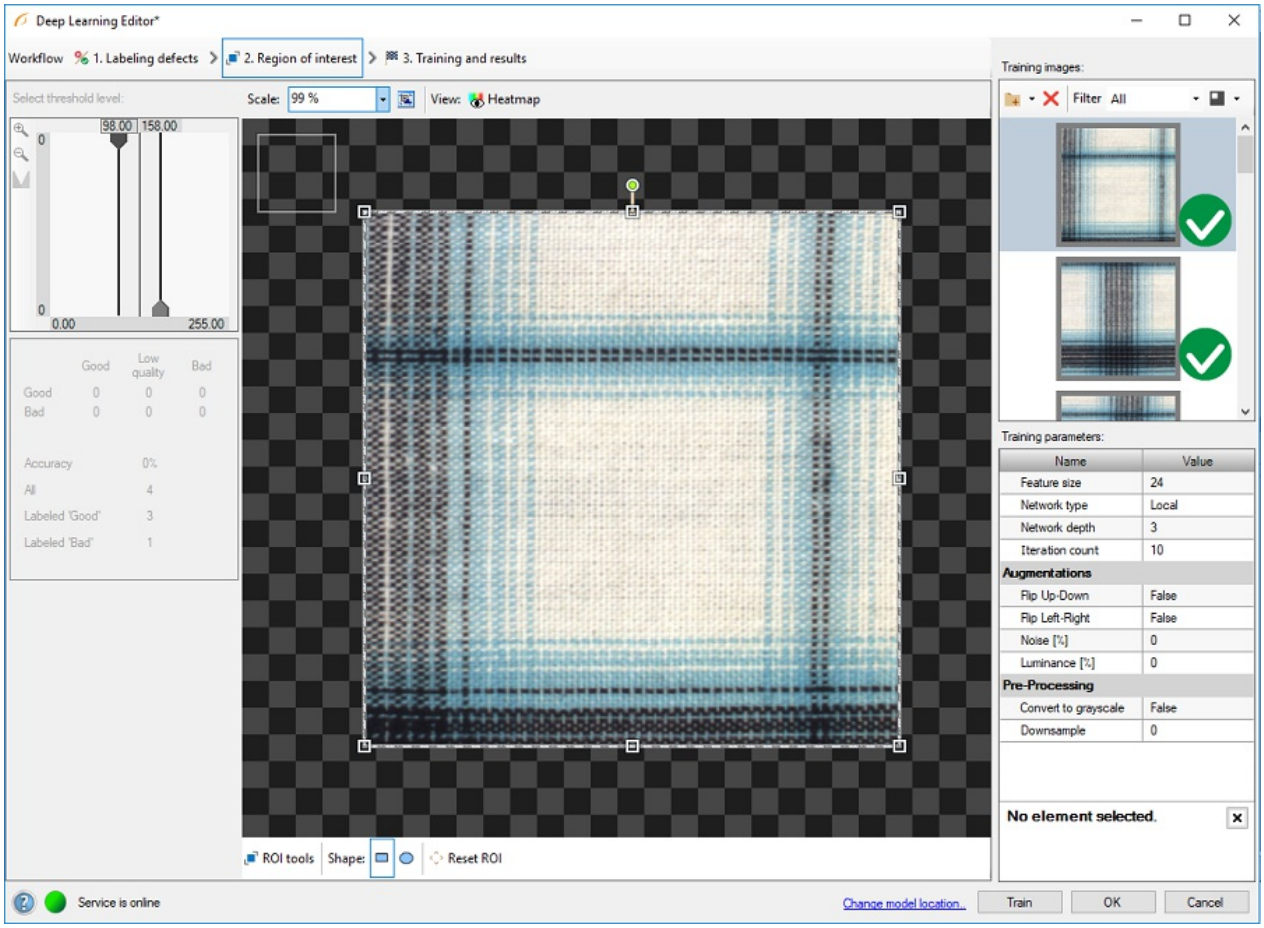
2. Configuring additional samples augmentation

Additional sample augmentation can be turned on in case when only few samples are available. For example the user can add additional changes in color intensity to prepare the model for different lighting conditions. Refer to augmentation section for detailed description of parameters. Deep Learning - Augmentation

3. Reducing region of interest

Reduce region of interest to focus only on the important part of the image. Reducing region of interest will speed up the training and model usage.

To get the best results, use the same region of interest for training and evaluation of samples.



By default region of interest contains the whole image.

4. Setting training parameters

- **Network type** - selecting different network architecture.
- **Density** - defines area of neighboring tiles overlap. In general, the higher option is selected, the better results are achieved at the expense of longer computation time.
- **Network depth** - predefined network architecture parameter. For more complex problems higher depth might be necessary.
- **Iteration count** - defines maximal number of times that all samples will be used to train network.
- **Feature size** - defines size of feature. Generally defect should cover about 30% of feature area.

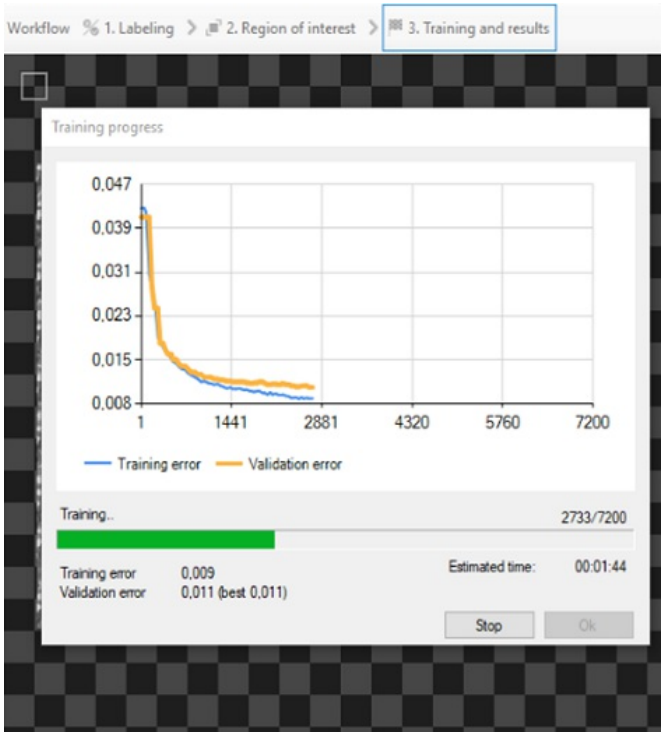
For more details read Deep Learning - Setting parameters.

5. Performing training

During training, two series are visible: training error and validation error. Both charts should have a similar pattern.

More detailed information is displayed below the chart:

- current training statistics (training and validation),
- number of processed samples (depends on the number of images and feature size),
- estimated training time.

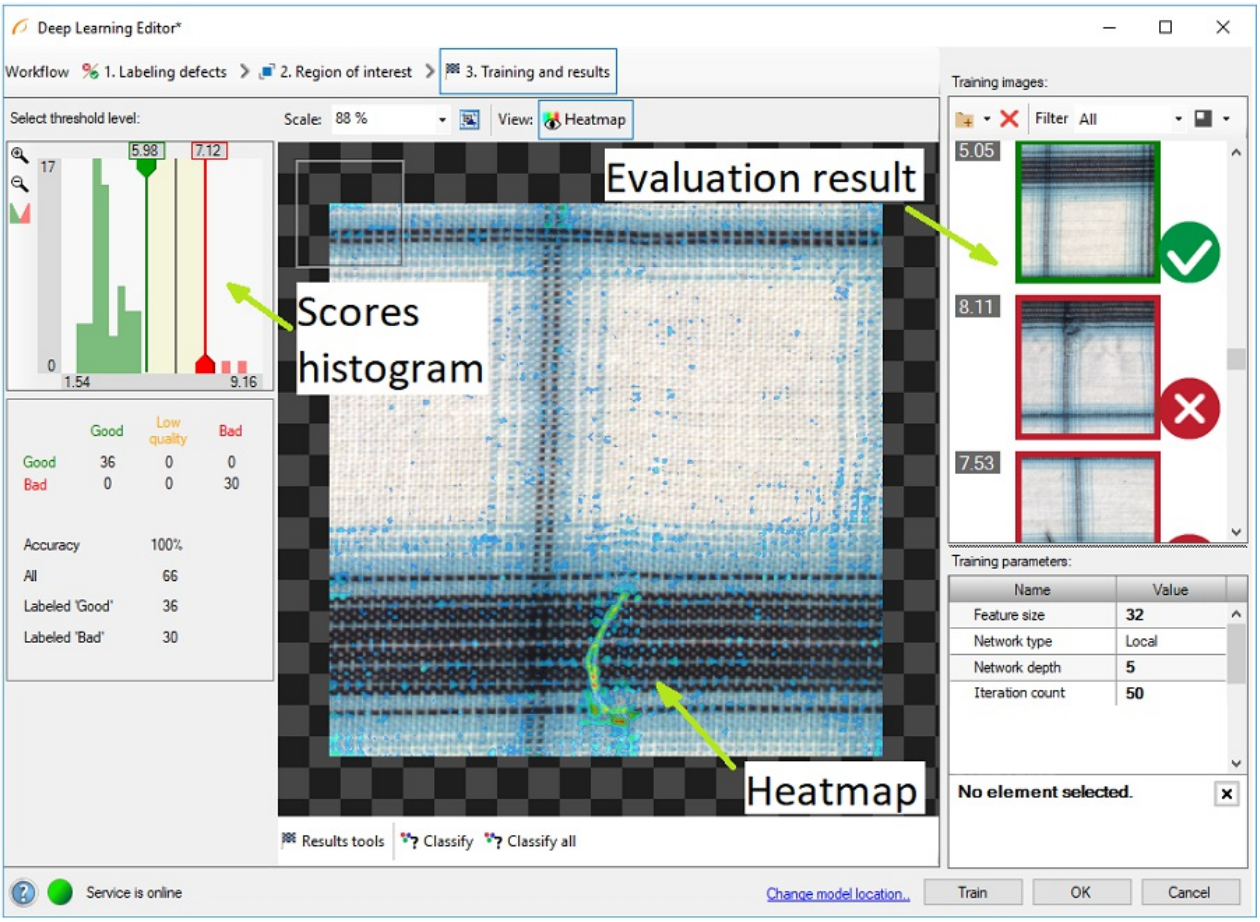


Training process can take a longer time. During this time, training can be cancelled. If no model is present (first training attempt) model with best validation accuracy will be saved. Consecutive training attempts will prompt user about old model replacement.

6. Analyzing results

The window shows a histogram of sample scores and a heatmap of found defects. Left column contains histogram of scores computed for each image in the training set. Additional statistics are displayed below the histogram.

Classify and **Classify All** buttons can be used to evaluate trained model. It can be useful after adding new images to the data set or after changing the area of interest.



After training, two border values are computed:

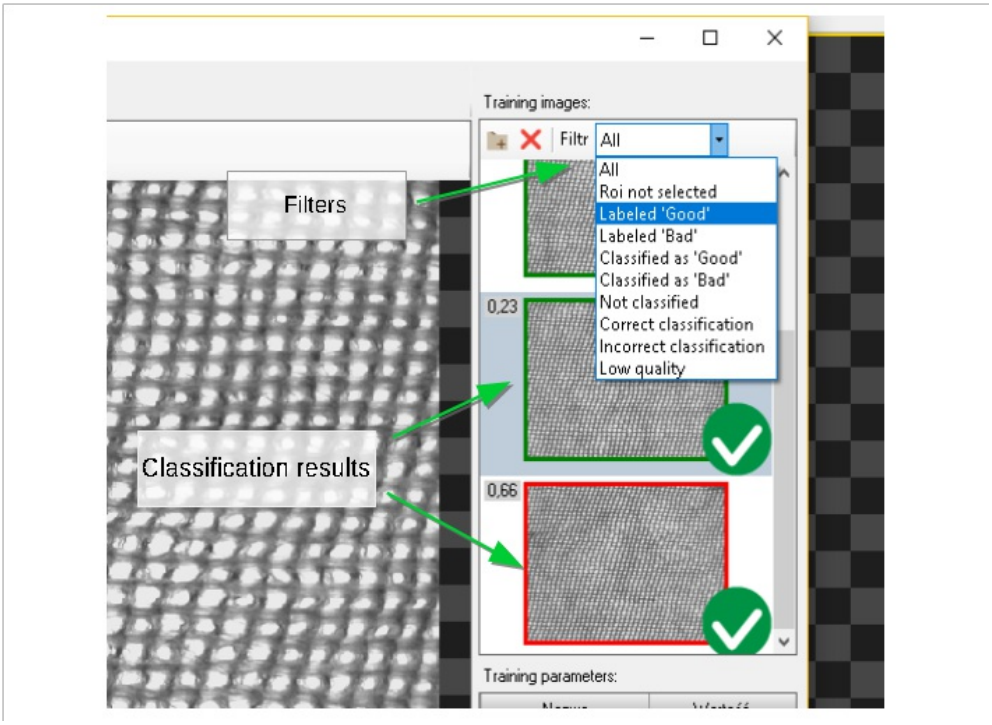
1. Maximum good sample score (T1) - all values from 0 to T1 are marked as good.
2. Minimum bad sample score (T2) - all values greater than T2 are marked as bad.

All scores between T1 and T2 are marked as "low-quality" scores. Results in this range are uncertain and may not be correct. Filters contain additional output

outIsConfident which determines the values which are not in the T1-T2 range.

In the top left corner of the editor, a small rectangle visualizes the selected feature size.

After evaluation, additional training set images filter can be applied.



Detecting features (segmentation)

In this algorithm variant, the user has to define each feature class and then mark features on each image in the training set. This technique is used to find object defects like scratches or color changes, and for detecting image parts trained on a selected pattern.

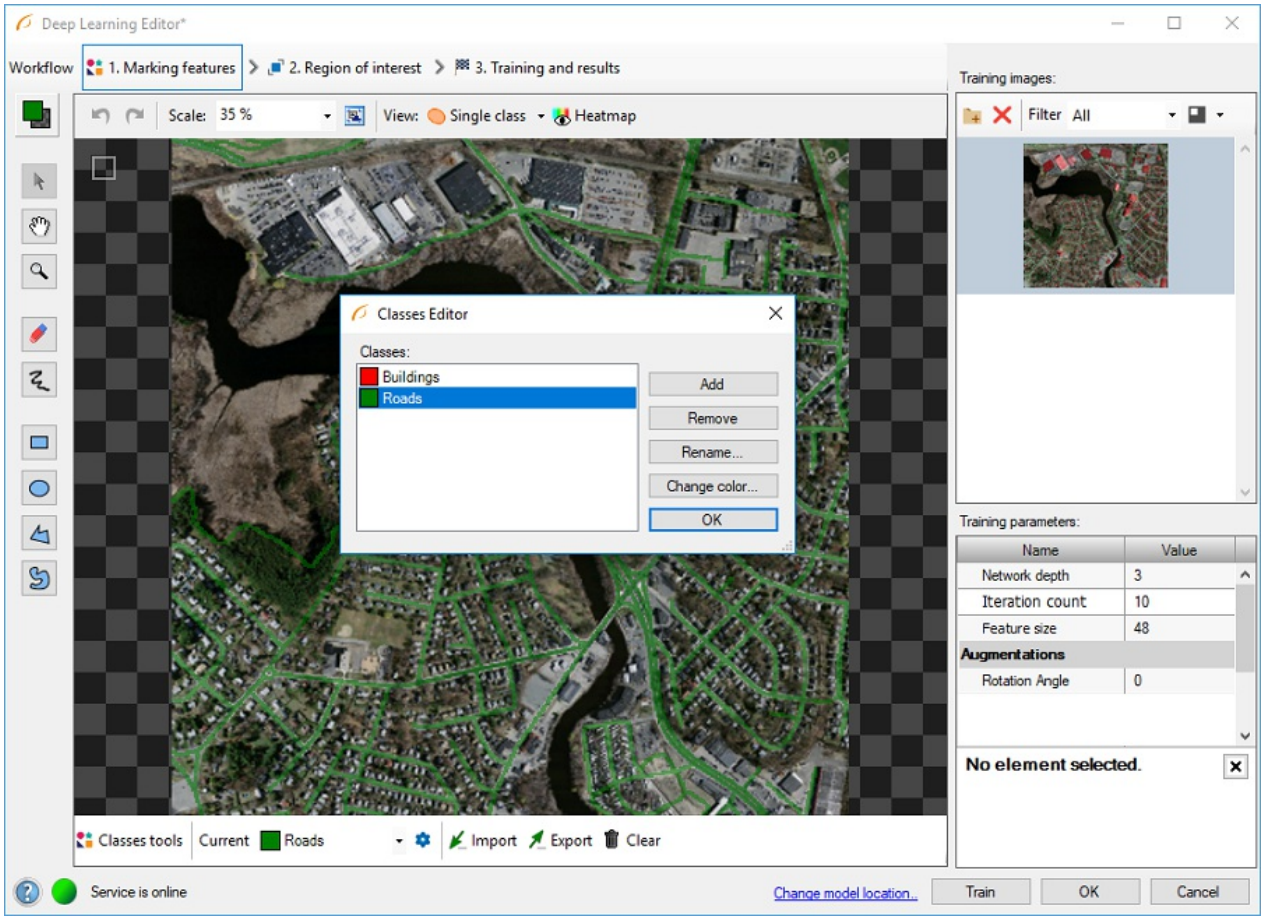
This technique is a very good solution for problems which are too complex for classical vision algorithms.

1. Defining feature classes (Marking class)

First, the user has to define classes of defects. Generally, they should be features which user would like to check on images. Multiple different classes can be defined but it is not recommended to use more than a few.

Class editor is available under sprocket wheel icon on the bottom bar.

To manage classes, **Add**, **Remove** or **Rename** buttons can be used. To customize appearance, color of each class can be changed using **Change Color** button.

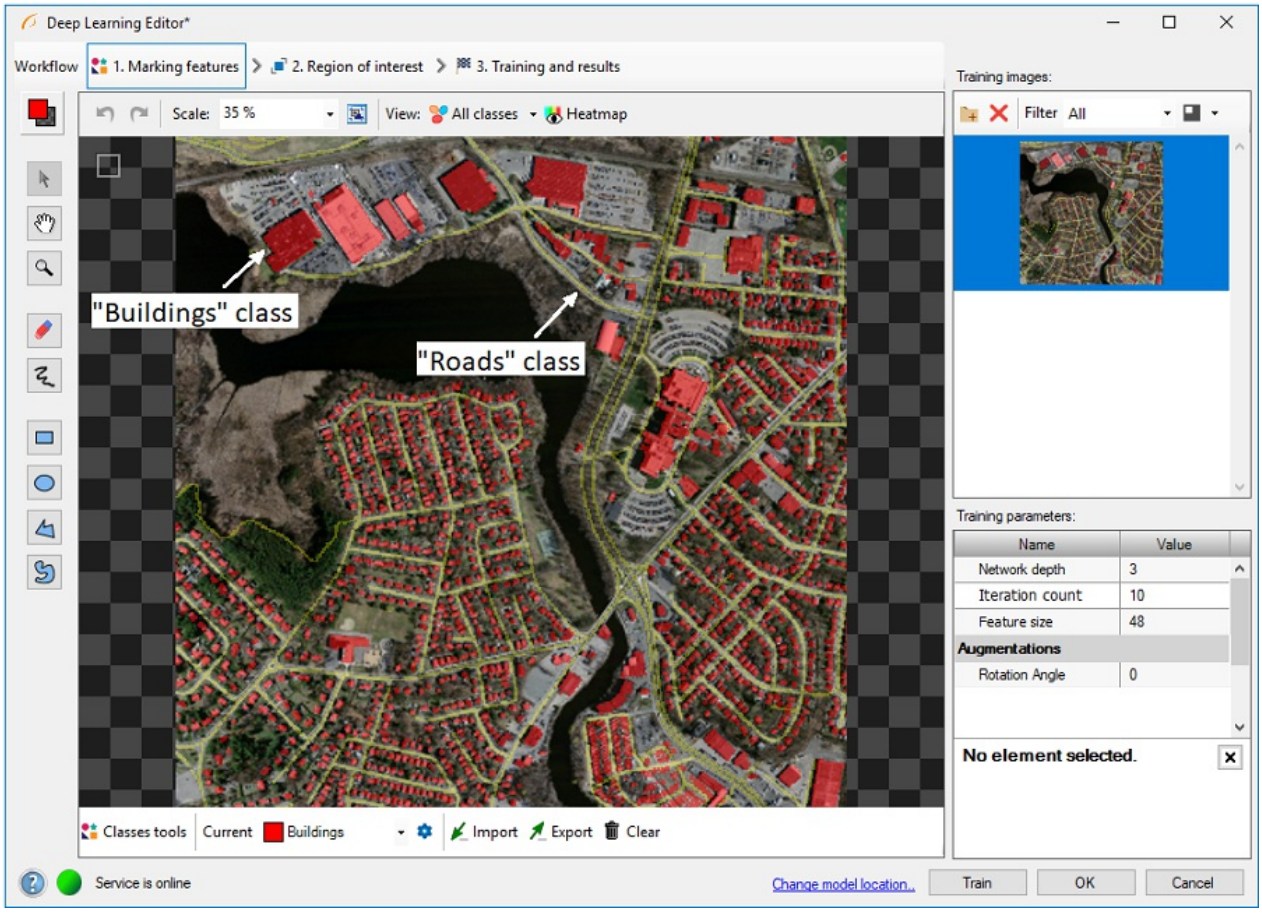


Current class for editing is displayed on the left, the user can select different class after click.

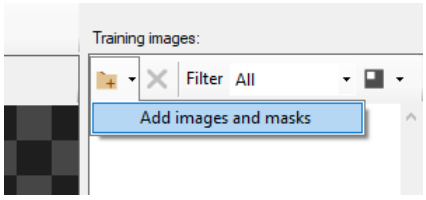
Use drawing tool to mark features on the input images. Tools such as **Brush** or **Rectangle** can be used for selecting features.

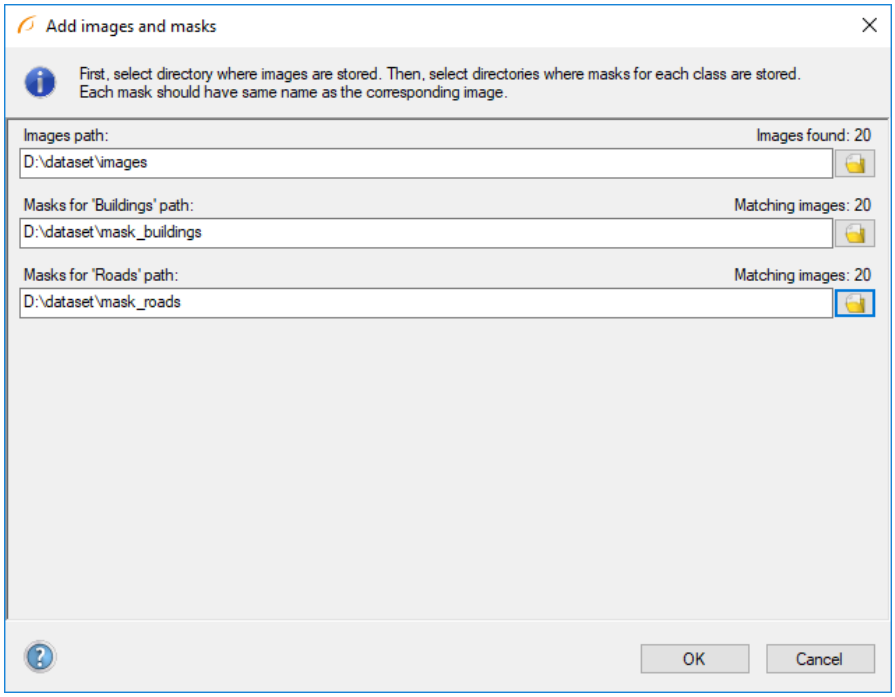
In addition, class masks can be imported from external files. There are buttons for **Import** and **Export** created classes so that the user can create a image of mask automatically prior to Deep Learning model.

The image mask should have the same size as the selected image in the input set. When importing an image mask, all non-black pixels will be included in the current mask.



The user can also load multiple images and masks at the same time, using **Add images and masks** button.



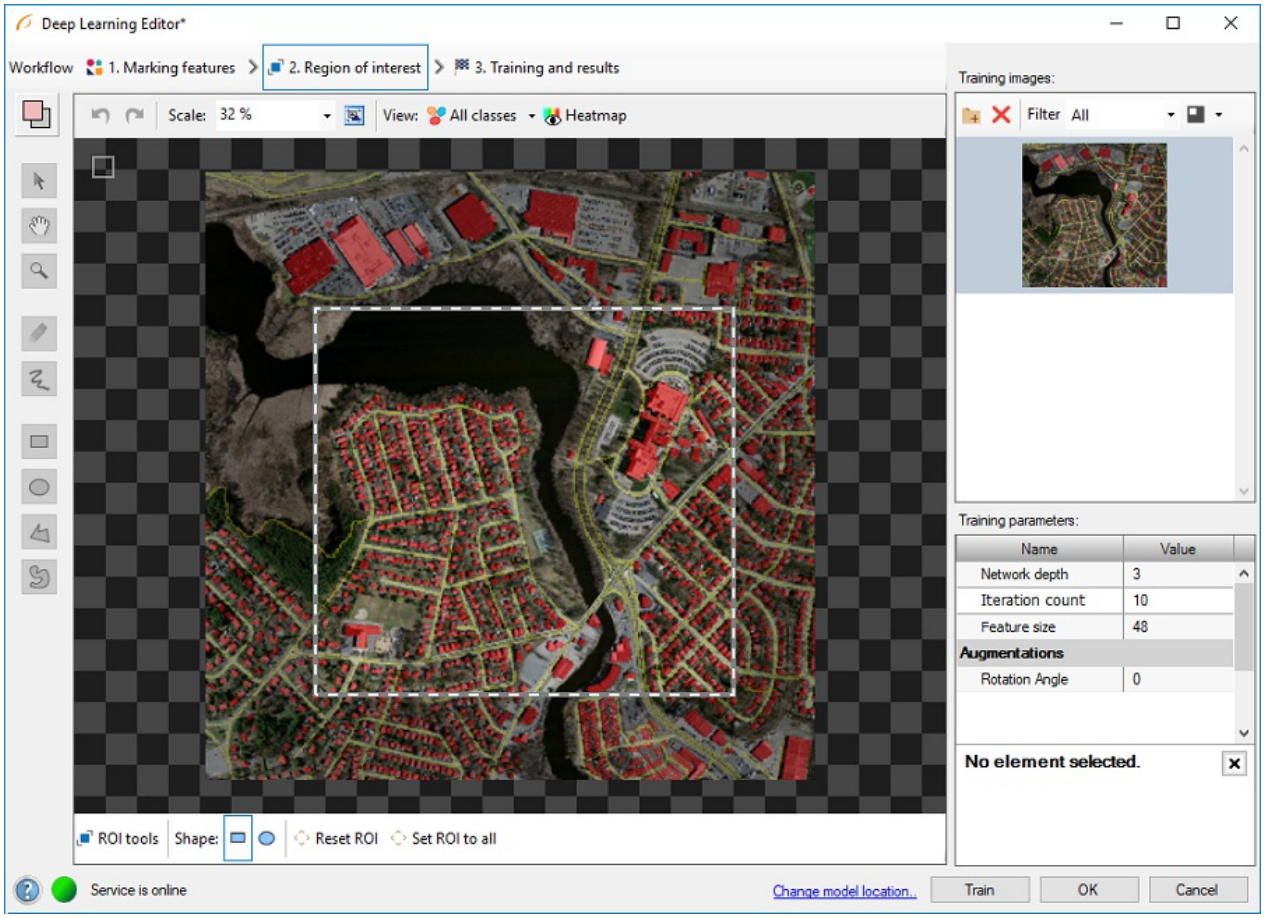


Directory containing input images should be selected first. Then, directories for each feature class can be selected below. Images and masks are matched automatically using their file names. For example, let's assume that "images" directory contains images 001.png, 002.png, 003.png; "mask_class1" directory contains 001.png, 002.png, 003.png; and "mask_class2" directory contains 001.png, 002.png, 003.png. Then "images\001.png" image will be loaded together with "mask_class1\001.png" and "mask_class2\001.png" masks.

2. Reducing region of interest

The user can reduce input image size to speed up the training process. In many cases, number of features on an image is very large and most of them are the same. In such case region of interest also can be reduced.

On the bottom bar there are tools for applying current ROI to all images as well as resetting the ROI.



3. Setting training parameters

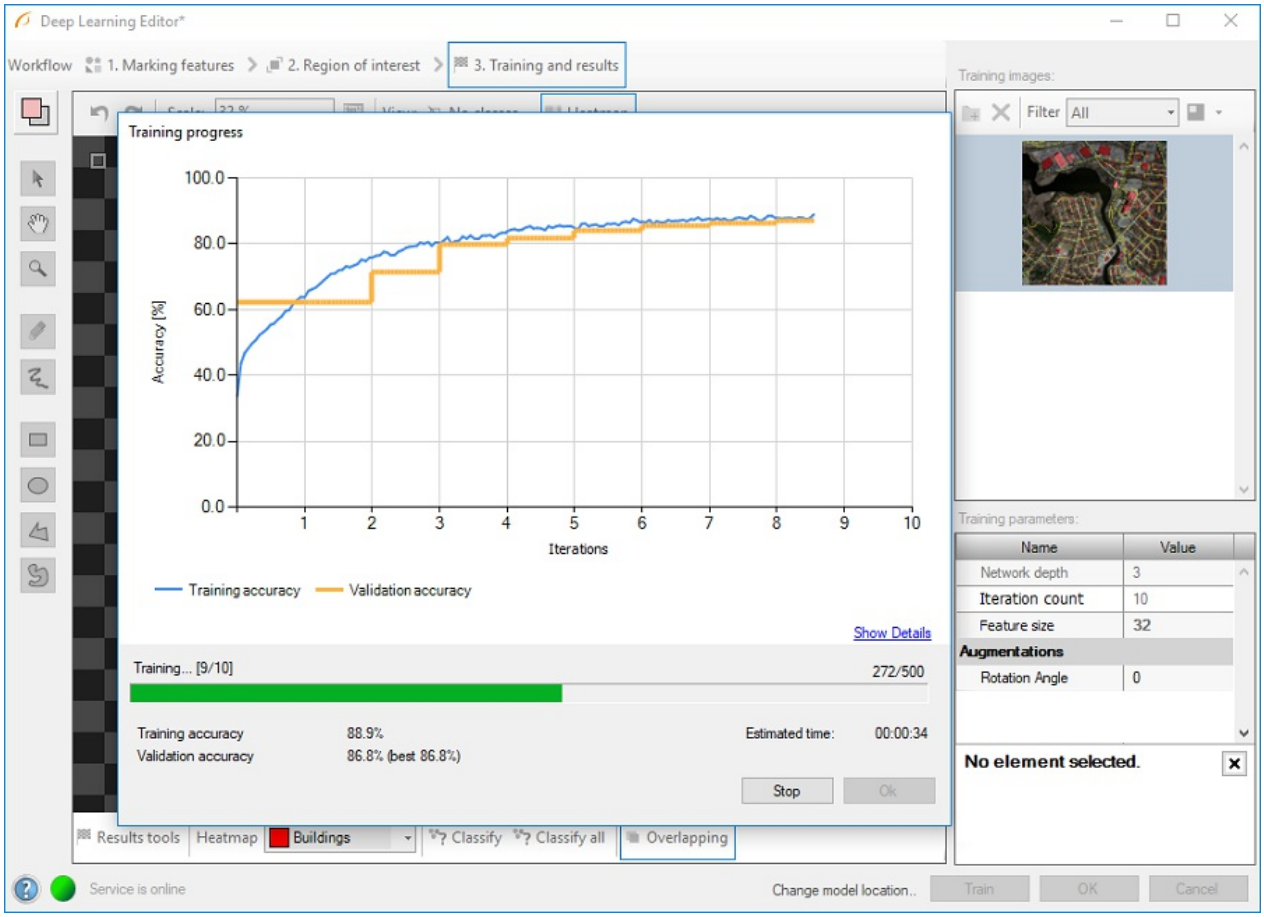
- **Network depth** - predefined network architecture parameter. For more complex problems higher depth might be necessary.
- **Iteration count** - defines maximal number of times that all samples will be used to train network.
- **Feature size** - defines size of feature. Generally defect should cover about 30% of feature area.

For more details read Deep Learning - Setting parameters.

Details regarding augmentation parameters Deep Learning - Augmentation.

4. Model training

The chart contains two series: training and validation score. Higher score value leads to better results.



5. Result analysis

Image scores (heatmaps) are presented in blue-yellow-red colors palette after using model to evaluation of image. The color represents the probability of the element belonging to the currently selected feature class.

Classify and **Classify All** buttons can be used to evaluate images. It can be useful after adding new images to the data set or after changing the area of interest.

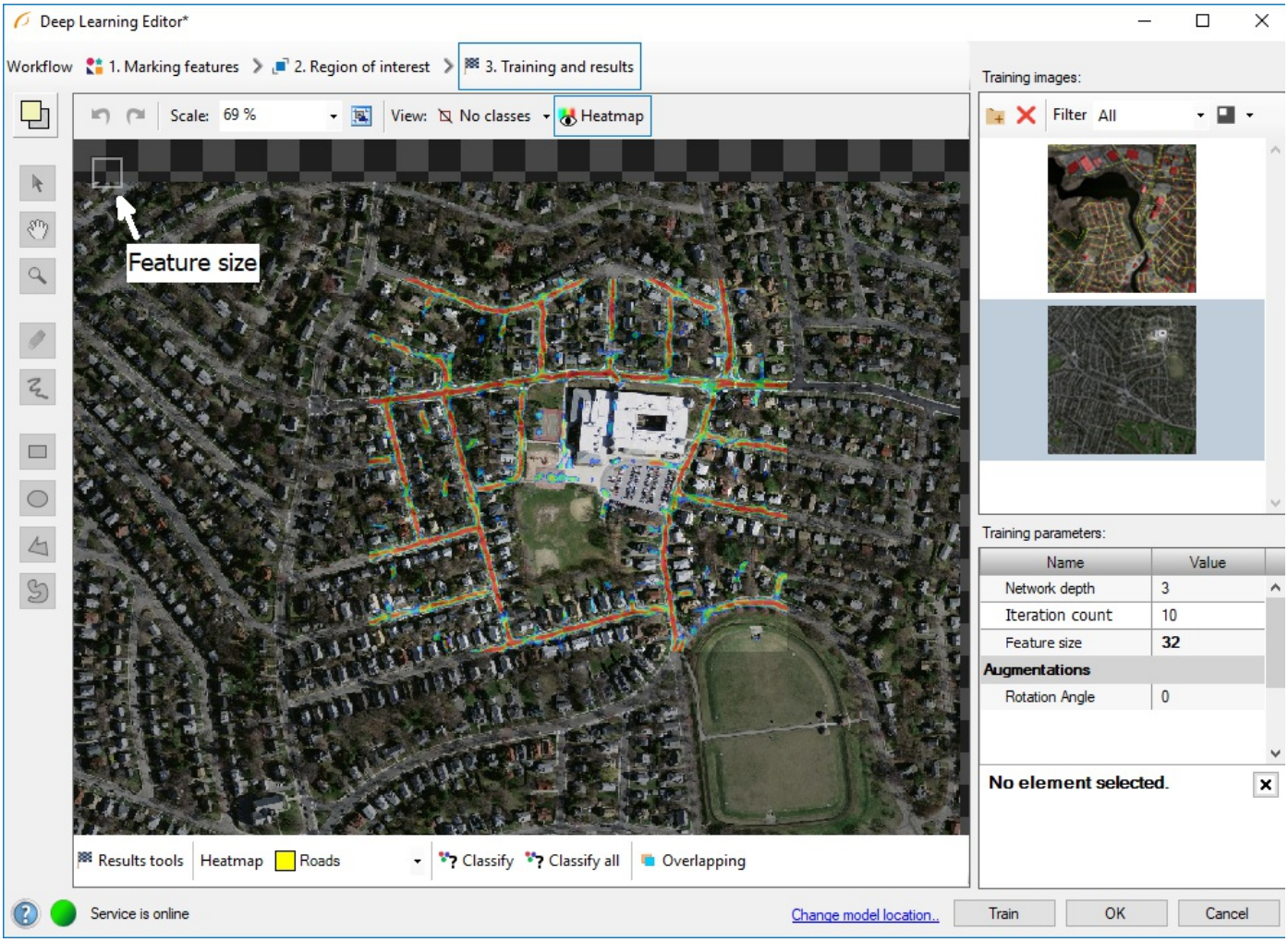


Image after classification performed using reduced region of interest.

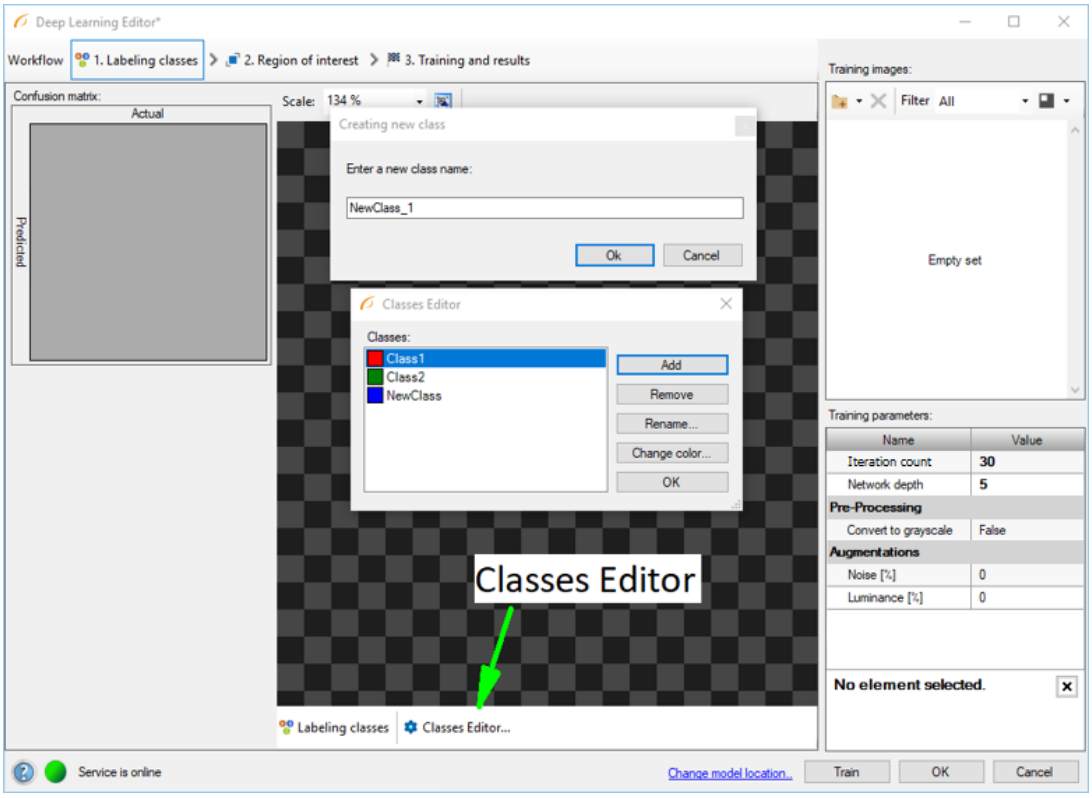
In the top left corner of the editor, a small rectangle visualizes the selected feature size.

Object classification

In this algorithm variant, the user only has to label images with respect to desired number of classes. Labeled images will allow to train model and determine features, which will be used to evaluate new samples and assign them to proper class.

1. Editing number of classes

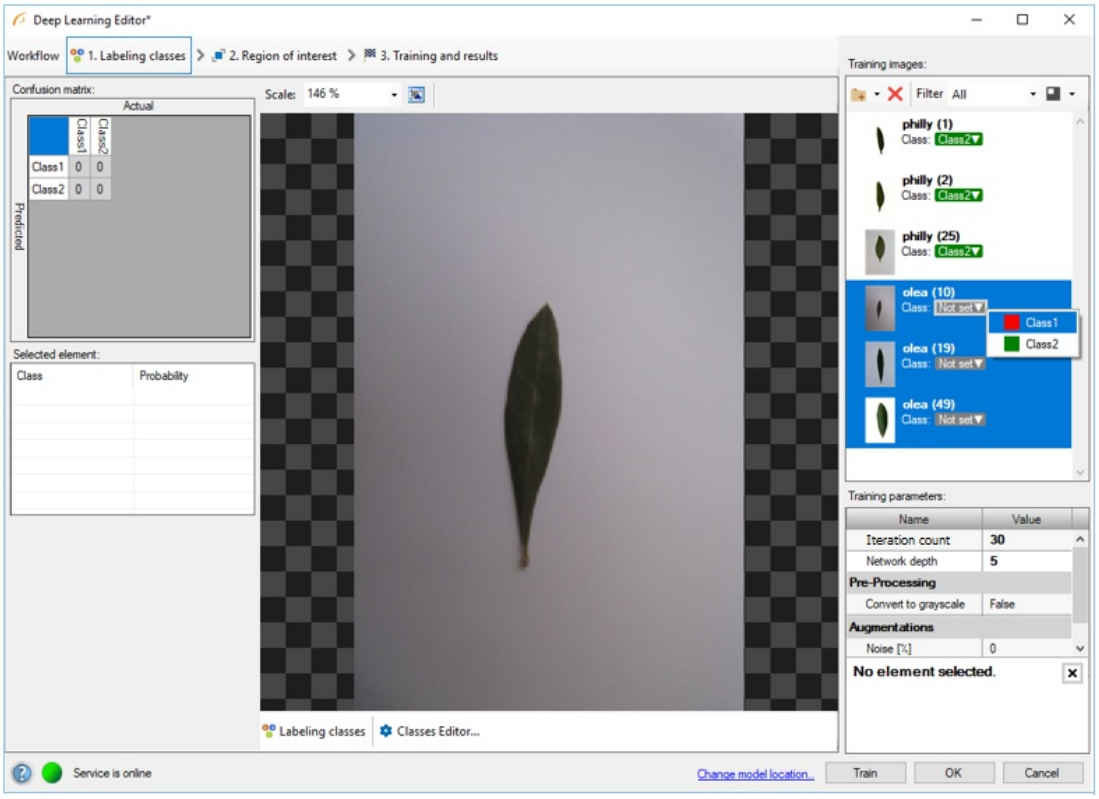
By default two classes are defined. If the problem is more complex than that, user can edit classes, and define more if needed. Once user is ready with definition of classes, images can be labeled.



Using Classes Editor.

2. Labeling samples

Labeling of samples is possible after adding training images. Each image has corresponding drop-down list which allows to assign specific class. It is possible to assign single class to multiple images by selecting desired images in Deep Learning Editor.

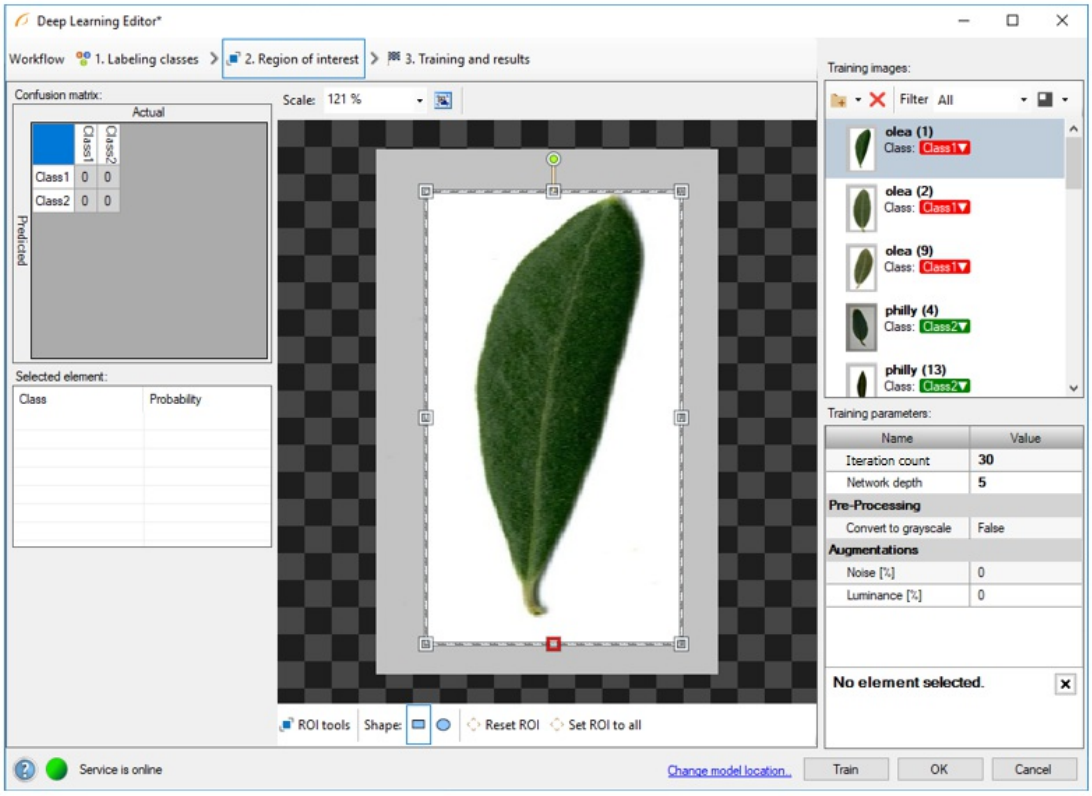


Labeling images with classes.

3. Reducing region of interest

Reduce region of interest to focus only on the important part of the image. Reducing region of interest will speed up the training and classification. By default region of interest contains the whole image.

To get the best classification results, use the same region of interest for training and classification.



Changed region of interest.

4. Setting training parameters

- **Iteration count** - defines maximal number of times that all samples will be used to train network.
- **Network depth** - predefined network architecture parameter. For more complex problems higher depth might be necessary.

For more details read Deep Learning - Setting parameters.

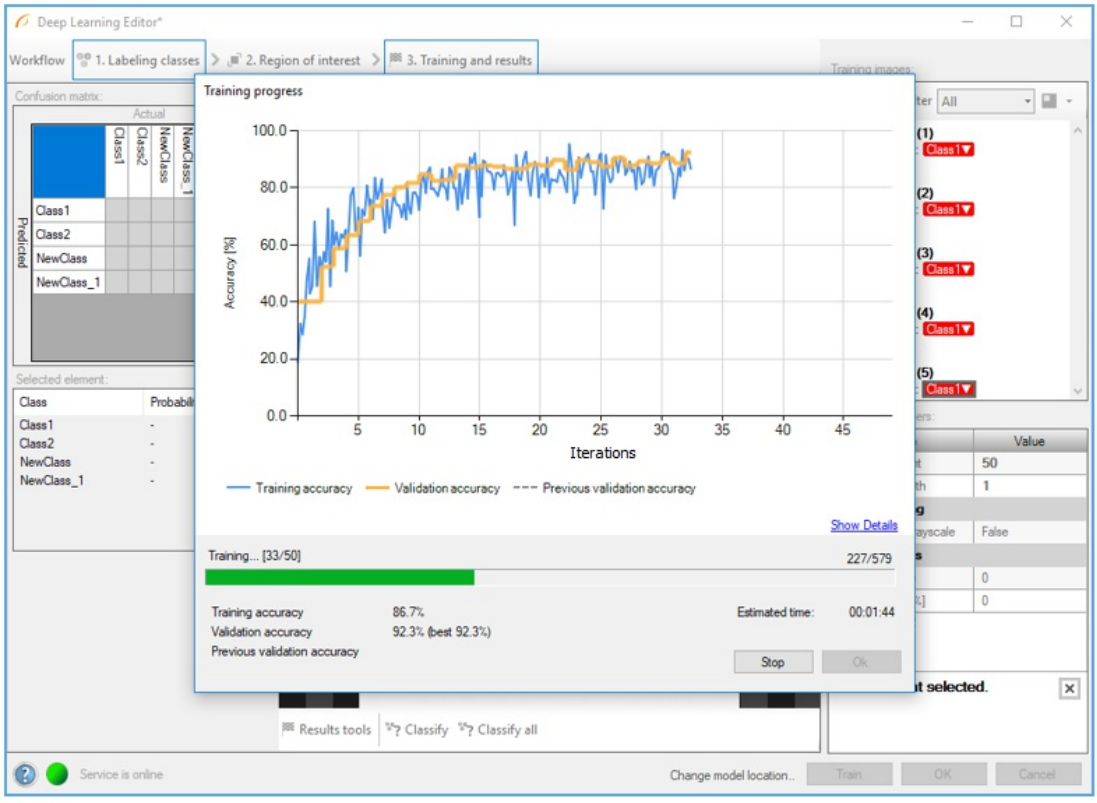
Details regarding augmentation parameters. Deep Learning - Augmentation

5. Performing training

During training, two series are visible: training accuracy and validation accuracy. Both charts should have a similar pattern.

More detailed information is displayed below the chart:

- current training statistics (training and validation accuracy),
- number of processed samples (depends on the number of images),
- estimated training time.



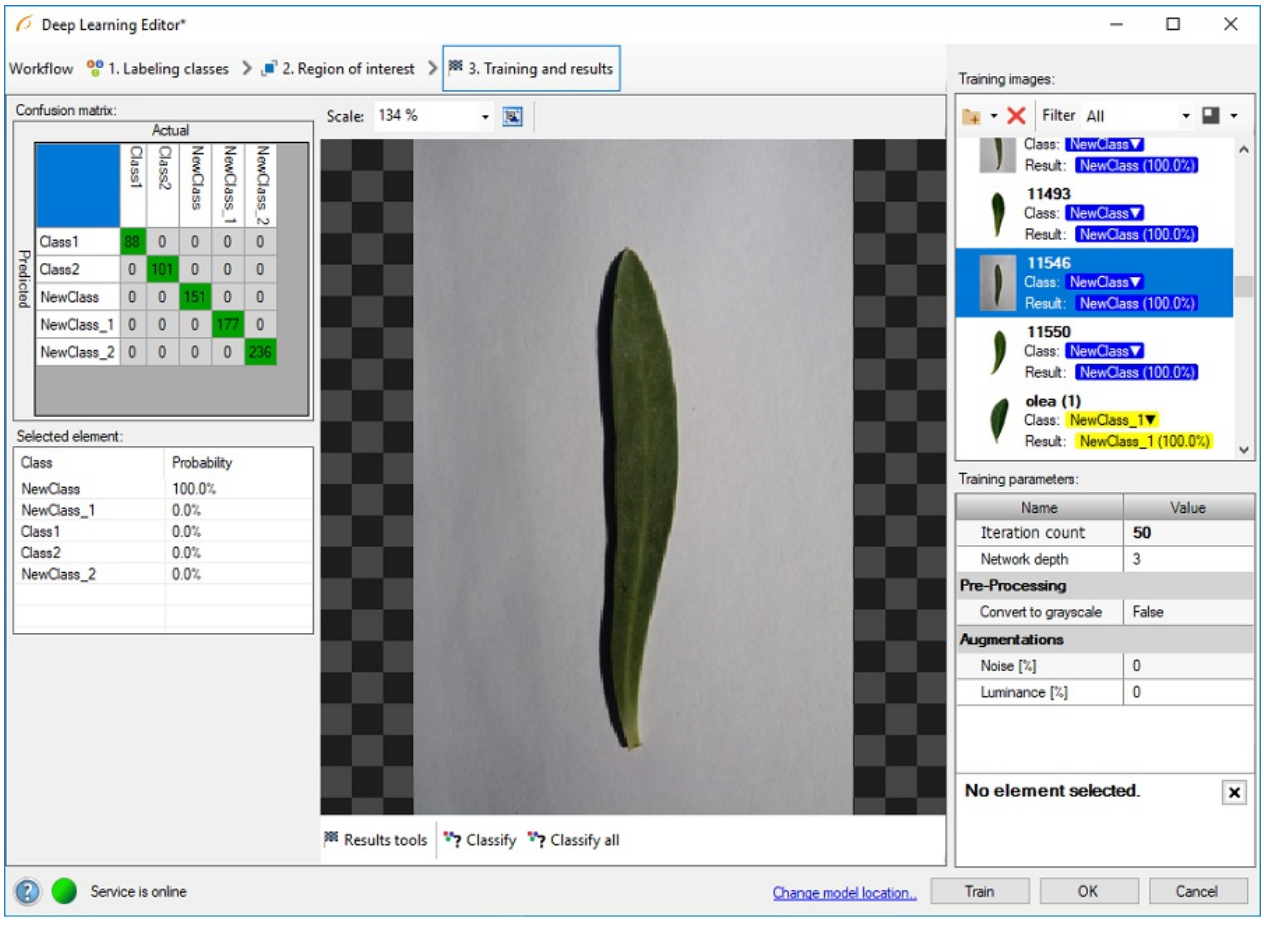
Training object classification model.

Training process can take a longer time. During this time, training can be cancelled. If no model is present (first training attempt) model with best validation accuracy will be saved. Consecutive training attempts will prompt user about old model replacement.

6. Analysing results

The window shows a confusion matrix, which indicates how well training samples have been classified.

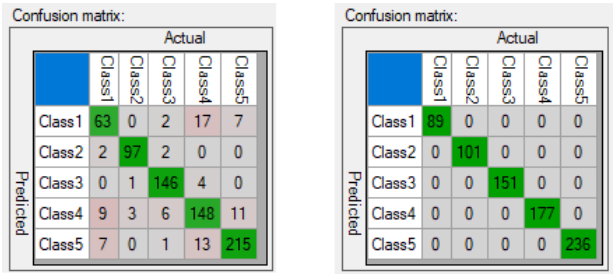
Classify and **Classify All** buttons can be used to classify images. It can be useful after adding new images to the data set or after changing the area of interest.



Confusion matrix and class assignment after the training.

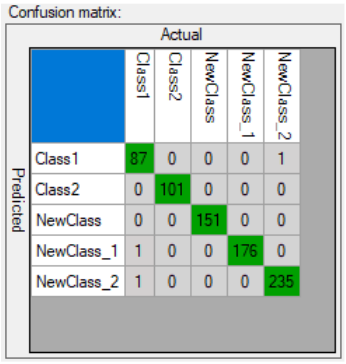
Sometimes it is hard to guess the right parameters in the first attempt. Picture below shows confusion matrix that indicates inaccurate classification during the

training (left).



Confusion matrices for model that needs more training (left) and for model well trained (right).

It is possible that confusion matrix indicates that trained model is not 100% accurate with respect to training samples (numbers assigned exclusively on main diagonal represent 100% accuracy). User needs to properly analyze this data, and use to his advantage.



Confusion matrix indicating good generalization

Too many erroneous classifications indicate poor training. Few of them may indicate that model is properly focused on generalization rather than exact matching to training samples (possible overfitting). Good generalization can be achieved if images used for training are varied (even among single class). If provided data is not varied within classes (user expects exact matching), and still some images are classified outside main diagonal after the training, user can:

- increase the network depth,
- prolong training by increasing number of iterations,
- increase amount of data used for training,
- use augmentation.

Instance segmentation

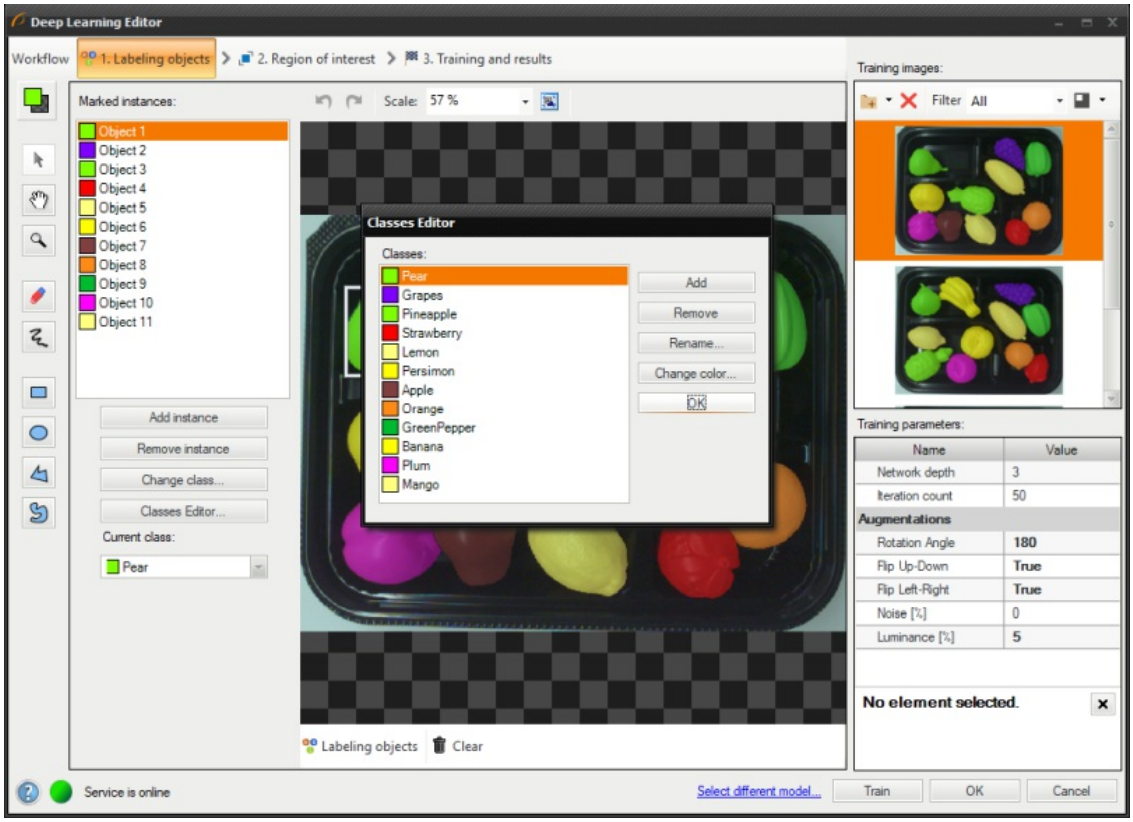
In this algorithm variant, a user needs to draw regions (masks) corresponding to objects in the image and select their classes. These images and masks are used to train a model which then will be used to locate, segment and classify objects in images.

1. Defining object classes

First, a user needs to define classes of objects that the model will be trained on and later will be used to detect. Instance segmentation model can deal with single as well as multiple classes of objects.

Classes editor is available under the Classes Editor button.

To manage classes, Add, Remove or Rename buttons can be used. To customize appearance, color of each class can be changed using Change Color button.



Using Classes Editor.

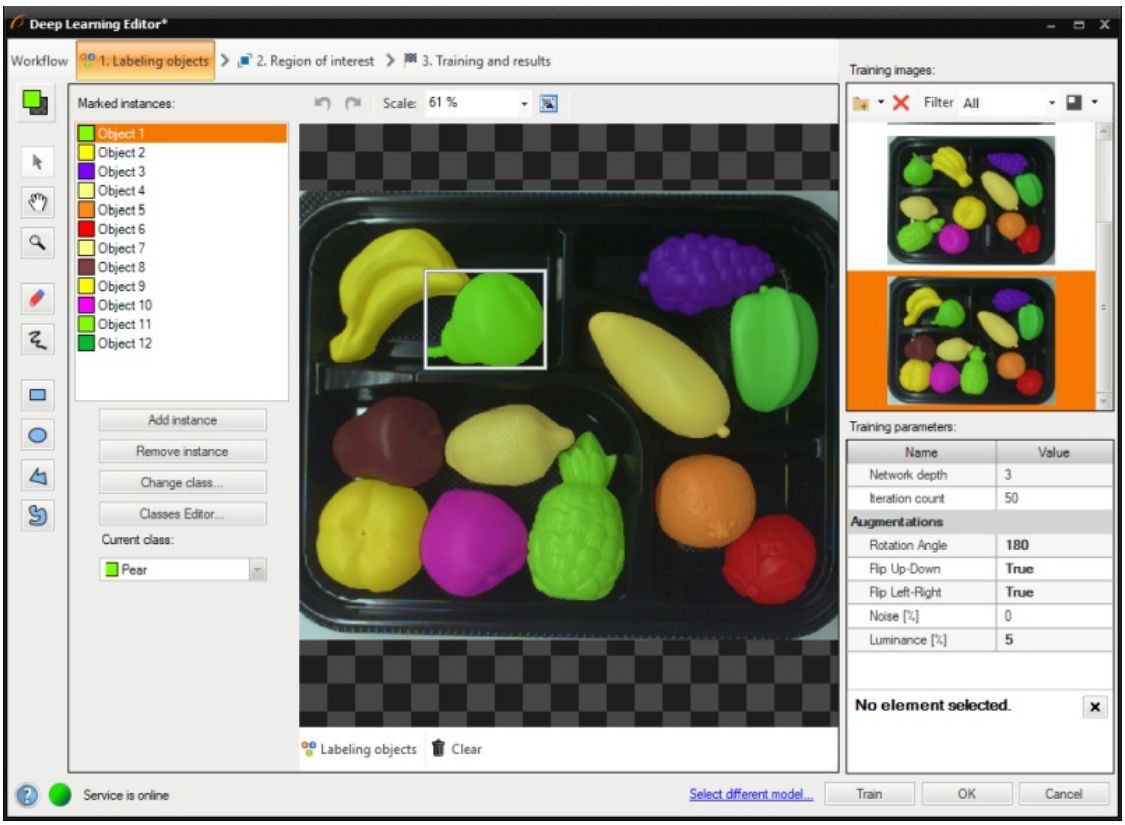
2. Labeling objects

After adding training images and defining classes a user needs to draw regions (masks) to mark objects in images.

To mark an object a user needs to select a proper class in the Current Class drop-down menu and click the Add Instance button.

Use drawing tool to mark objects on the input images. Multiple tools such as brush and shapes can be used to draw object masks. Masks are the same color as previously defined for the selected classes.

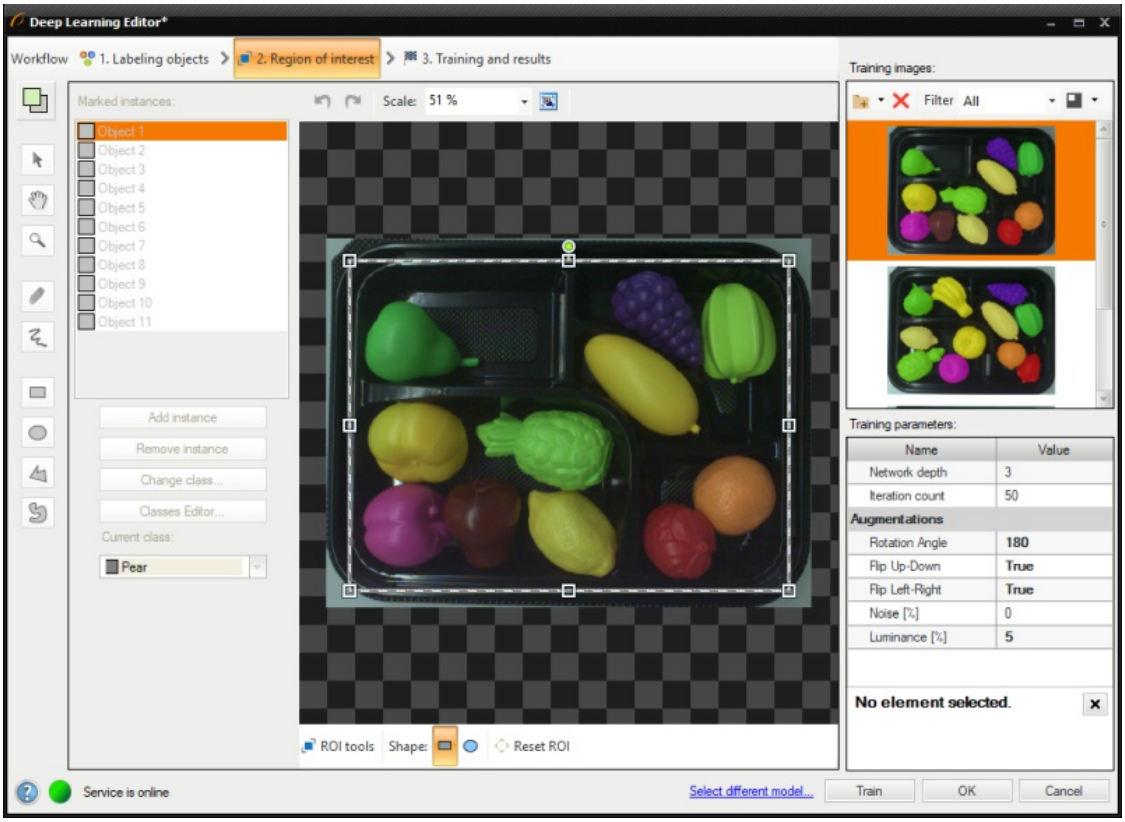
The Marked Instances list in the top left corner displays a list of defined objects for the current image. If an object does not have a corresponding mask created in the image, it is marked as "(empty)". When an object is selected, a bounding box is displayed around its mask in the drawing area. Selected object can be modified in terms of a class (Change Class button) as well as a mask (by simply drawing new parts or erasing existing ones). Remove Instance button allows to completely remove a selected object.



Labeling objects.

3. Reducing region of interest

Reduce region of interest to focus only on the important part of the image. By default region of interest contains the whole image.



Changing region of interest.

4. Setting training parameters

- **Iteration count** - defines maximal number of times that all samples will be used to train network.
- **Network depth** - predefined network architecture parameter. For more complex problems higher depth might be necessary.

For more details read Deep Learning - Setting parameters.

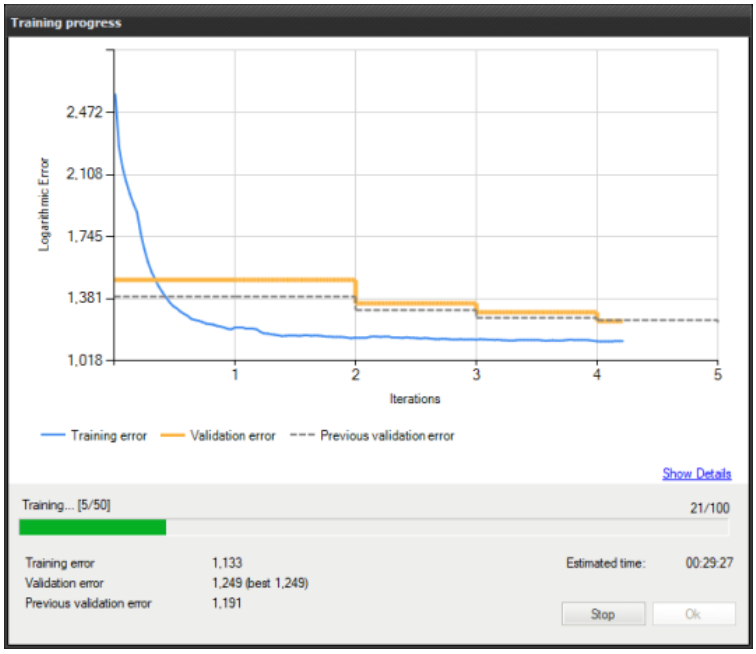
Details regarding augmentation parameters. Deep Learning - Augmentation

5. Performing training

During training, two main series are visible: training error and validation error. Both charts should have a similar pattern. If a training was run before the third series with previous validation error is also displayed.

More detailed information is displayed below the chart:

- current iteration number,
- current training statistics (training and validation error),
- number of processed samples,
- estimated training time.



Training instance segmentation model.

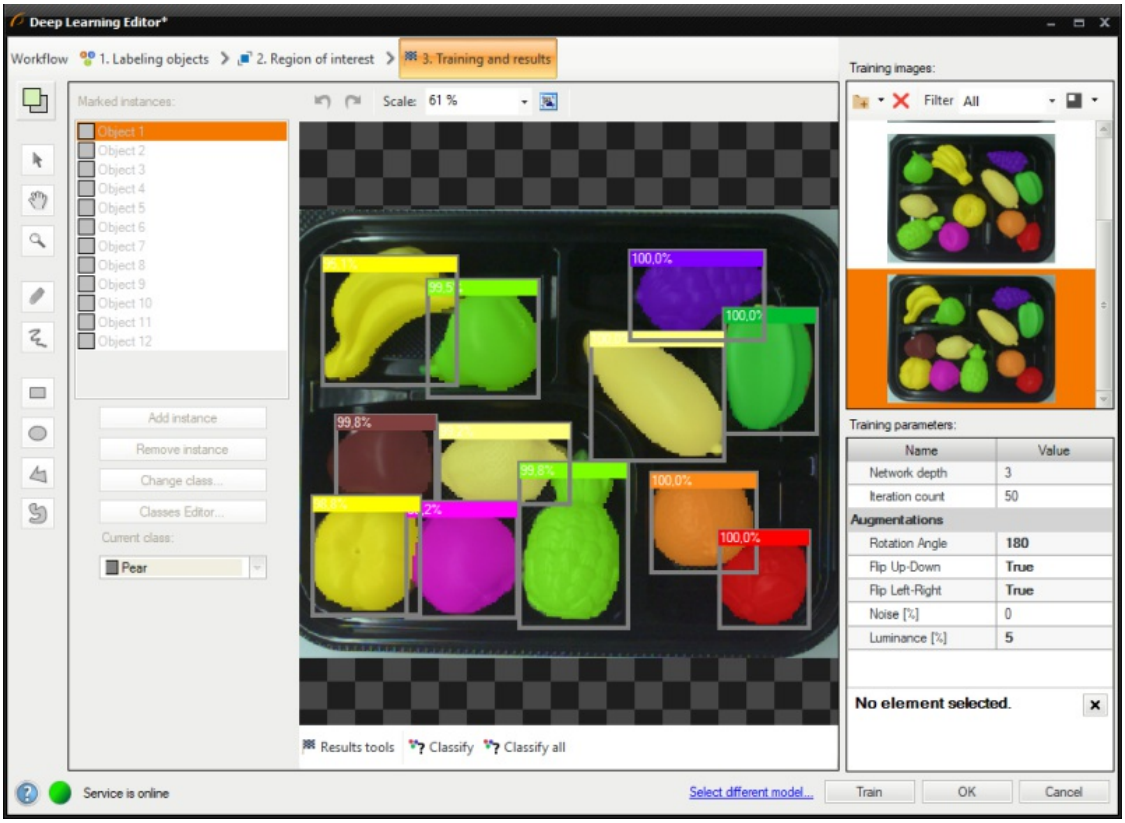
Training may be a long process. During this time, training can be stopped. If no model is present (first training attempt) model with best validation accuracy will be saved. Consecutive training attempts will prompt user whether to replace the old model.

6. Analysing results

The window shows results of instance segmentation. Detected objects are displayed on top of the images. Each detection consists of following data:

- class (identified by a color),
- bounding box,
- model-generated instance mask,
- confidence score.

Classify and **Classify All** buttons can be used to perform instance segmentation on the provided images. It can be useful after adding new images to the data set or after changing the area of interest.



Instance segmentation results visualized after the training.

Instance segmentation is a complex task therefore it is highly recommended to use data augmentations to improve network's ability to generalize learned information. If results are still not satisfactory the following standard methods can be used to improve model performance:

- providing more training data,
- increasing number of training iterations,

- increasing the network depth.

See also:

- Machine Vision Guide: Deep Learning - Deep Learning technique overview,
- Deep Learning Service Configuration - installation and configuration of deep learning service.

Adaptive Vision
This article is valid for version 4.10.2
©2007-2018 Future Processing