perClass Mira 2.0 User's Guide

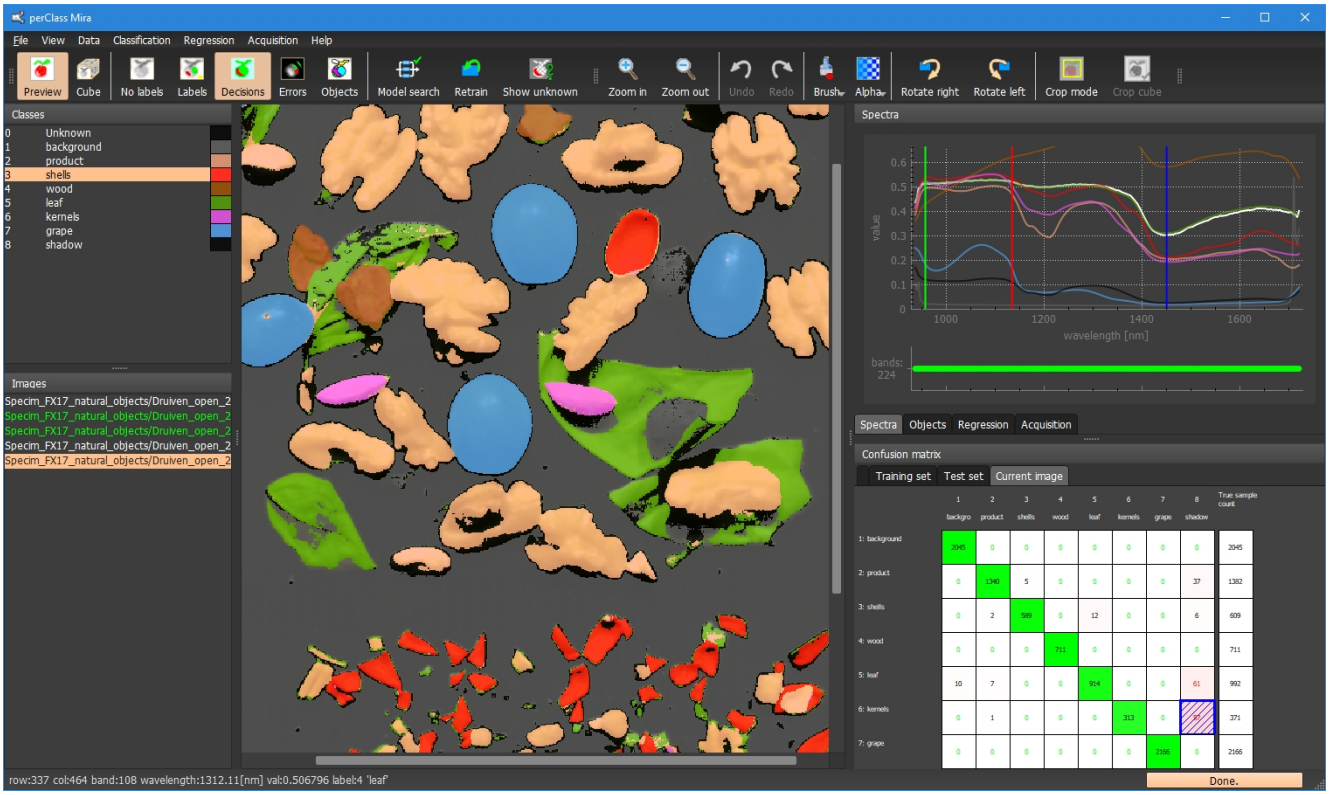
Copyright © 2018-2019 by perClass BV. All Rights Reserved.

Table of contents

Introduction

perClass Mira

perClass Mira is a user interface for interpretation of spectral images. It allows users to define classification and regression solutions and deploy them in custom applications. It automatically select machine learning models, and enable user to understand the problems and interactively improve the solutions.



Release Notes

2.0 10-oct-2019

- adding support for double-precision ENVI data cubes
- supporting model deployment for execution on live data from Cubert Ultris light-field hyperspectral camera
- enabling Cubert plugin export for ENVI-based projects.
- fixes in live acquisition using Specim FX cameras when device loading fails or opening FileReader gets cancelled
- fixing a crash due to very large training set
- fixing a bug in error visualization mode where switching to images without labels did not show proper image

2.0 20-sep-2019

- Fix: Installation directories with non-ASCI characters are now supported
- Live acquisition executables for Specim cameras included (perClass_Mira_live.exe and perClass_Mira_gpu_live.exe)
- Senop project: Images are automatically processed with per-band gain

2.0 6-sep-2019

- Estimate **object quality using regression** (examples: sugar content estimation per tomato)
    - annotate quality per object

    - automatic model selection reporting performance (R^2 and Q^2 statistics)
    - user-defined pre-processing (smoothing and derivatives)
    - apply regression to new images (show a bounding box + regression output per object)
    - allow localized information extraction by a radius around annotation points
- Images can be flagged for testing only (not used for building the model)
    - **Test confusion matrix** provides a detailed view of the performance on test images
- **Error visualization mode** brings insight in model performance.
    - visualize where the current model fails
    - this helps to identify incorrect labels or (together with test image flagging) whether the data is well represented in the training set
    - **Image confusion matrix** shows only labeled examples on the current image
    - interactive error visualization by moving mouse over the image confusion matrix
- **Object segmentation mode** with multiple options
    - one object / one class mode for object detection (e.g. detect plastic pieces in a food product stream for automatic removal)
    - one object / multiple classes for object classification (e.g. detect potato pieces, classify entire piece as defective if it contains more than 5% of greening or rot inside)
    - visualizing object labels or object decisions
    - object decisions by majority vote or rules (size of or fraction of a specific class)
- Usability improvements
    - **assign label stroke to the current class**. This allows one to exclude a specific label stroke from training and see the impact on model performance (define an additional class and exclude it, assign strokes to it and retrain)
    - the data validation mechanism excluding invalid spectra is now off by default. It can be enabled using context menu in the spectral plot.
    - all modes (labels, decisions, errors, objects) accessible by direct keystrokes
    - confusion matrix size can be decreases/increased (useful for large number of classes)
    - auto-check for software updates + direct link to download latest version from the GUI (Help / Check for updates)
- *experimental* **Live data acquisition** **from Specim FX cameras** using Specsensor SDK (needs to be installed separately)
    - apply a classifier and object segmentation to a live data stream
    - live visualization of processing speed and drop frame indication to assess production performance
    - user-control of exposure and camera frame-rate
    - supports practical situations where production light conditions are diffrerent from the training situation
        - the white and dark references used for live data processing can be specified without model retraining
    - automatic handling of spectral and spatial binning based on specific scan meta-data
    - support for **outdoor operation**: Define white reference by specifying an image region where a reference tile was placed
    - **recording data** from a live acquisition in the standard LUMO format

1.4 22-may-2019

- **perClass Mira Runtime** is now included in the distribution
    - high throughput (1.5ms/frame on NVIDIA GPU in an example foreign object detection project, Specim FX17, 640 spatial pixels, 224 bands, 6 materials)
    - the runtime directly reports object positions, sizes and classes
    - support for **NVIDIA Jetson** platform (both ARM CPU and NVIDIA GPU backend)
    - support for line-scan use-case on Specim projects (specific white/dark correction format)
- **Linux build** for both perClass Mira GUI and perClass Mira Runtime
    - accelerated CPU and GPU support on Linux
- new high-throughput segmentation engine
    - automatically discarding objects smaller than user-defined minimal size
    - supporting multiple foreground classes
    - high-speed line-scan segmentation with constant per-frame speed

- export visualization as PNG images (band or RGB, with labels, pixel decisions or segmented objects)\
- for Cubert projects, proper wavelength ranges are shown

1.3 8-feb-2019

- zoom using mouse wheel now follows cursor
- image rotation using toolbar buttons (and > < keyboard shortcuts)
- adding images using drag and drop from Windows explorer
- support for ENVI files with high-endian byte order uint16 (byte order=1)
- saved projects now preserve settings of the current band, R,G,B lines and allow direct execution of the trained model when project is loaded
- exported decision images (PNGs) contain meta-data such as class count and class names accessible by standard tools such as *tweakpng* or Matlab `imfinfo` command
- multiple directory selection for Specim FX and Tiff stack project types can be enabled in mira.ini file (using `useNativeDirSelection=false`). It is not enabled by default because it uses a non-native file dialog.
- new project type for Senop cameras (formerly Rikola)

1.2 5-dec-2018

- Added band-selection widget. It is now possible to manually select the wavelengths used for building models
  - Band brushing allows quick selection or clearing of wavelength ranges
  - Exported models start from the full set of wavelengths but use only the selected subset for the model. This allows quick deployment of different models to custom applications assuming full spectrum (single binding with perClass Runtime is needed)
- Added export of labeled data to perClass Toolbox sddata format
- Added export of entire data cube in Matlab format as 3D matrix
- Models results are now repeatable with a new random seed dialog controlling the internal data partitioning process.
- Separate CPU-only and CPU+GPU builds are available. The CPU-only build is always available by default to avoid issue related to GPU drivers or CUDA versions installed. The CPU+GPU executable is called perClass_Mira_gpu.exe
- Band index and the wavelength number are now updated on the status bar when dragging the band line in spectral plot
- Added support for logging of status messages when starting up the application. This is useful to understand some issues with GPU installations and CUDA versions. Logging is off by default, can be switched on in the mira.ini file.
- Licensing improvements:
  - For activated licenses, there is now an auto-update mechanism that pulls updated license from the activation server when the application starts. The application may be used without on-line connection - it is needed only once in two weeks.
  - Adding support for floating licenses obtained over network from a license server. Floating licenses are now checked out one per session.
- Fixed wrong file name of previous project used for saving new project with File/Save command
- Fixed a crash when preview image could not be loaded

1.1 10-sep-2018

- **confusion matrix view** showing detailed error information
  - interactive performance optimization in a confusion matrix (slider in right-click context menu or a mouse wheel on confmat entries)
  - confusion matrix shows normalized errors and precisions, absolute sample counts available as well
  - quickly switch to confmat with 'c' key and to spectral plot with 's' key
  - define **performance constraints** via double click on a confusion matrix field (create/remove constrain)
  - constraints may be adjusted live by Ctrl+mouse wheel
  - constraints may be enabled/disabled to understand available performance options
  - move between available solutions fulfilling all constraints with [and] shortcuts

- preview image from user-adjustable R,G and B bands when spectral cube is loaded
  - this view improves labeling experience for many material types that look similar in a single band but their differences may be highlighted in R,G,B view
- **undo/redo** for label painting speeds up labeling
- **image crop** providing significant memory use reduction and processing speedups
  - when a project with a cropped image is loaded, the original cube is loaded and cropped
  - original cube may be loaded as a new image and multiple crops from the same cube are supported
- including perClass Runtime DLL and example of spectral cube processing in C
  - support for both **single precision** and double precision pipelines (with a new perClass 5.4 Runtime)
  - significant speedup of exported classifiers
  - legacy export option supporting older deployed runtimes <= 5.2
- a **preview rotation** command allows one to fix the rotation between preview and spectral cube (e.g. on Specim IQ projects)
- adding an option to **exclude a class from training** (right-click in class list or press 'x')
  - this allows one to quickly check the impact of specific classes on the overall solution
- option to purchase a license online and directly turn the demo into a commercial product
- dialog to request Skype/Teamviewer session on start up
- fix for a wrong class index after removing a class
- fix for clear labels of an image

1.0 13-jul-2018

- fix for a dock shift bug (when resizing a docked window and clicking on the image, the docked pannel resized back)
- adding band line dragging by mouse
- adding max valid line which is automatically set on image load
- when user is on preview and tries painting, a dialog is shown to load the entire cube (allows quick image changing without load)

1.0 29-may-2018

- first public release

Installation

System requirements

perClass Mira system requirements:

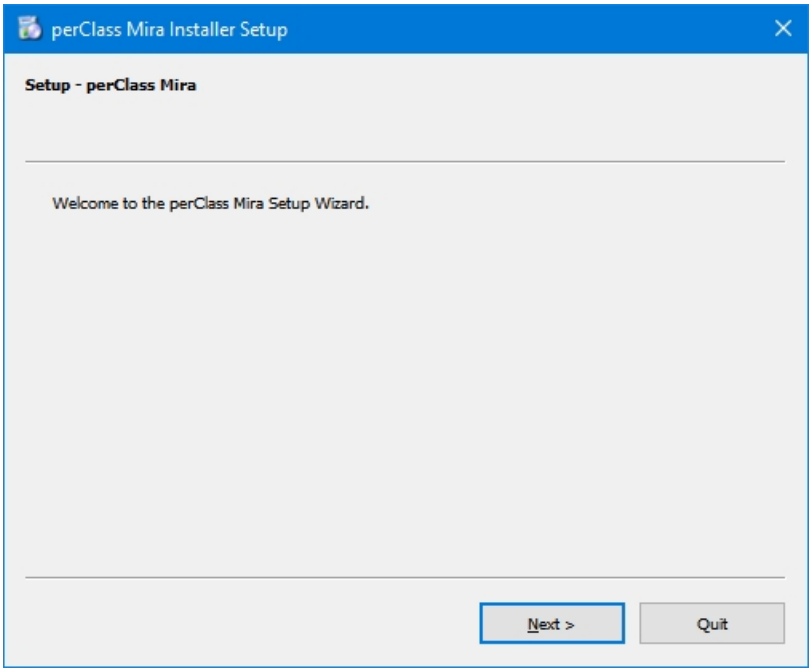
- PC with MS Windows (version 7 or higher) 64bit
  - memory of at least 8GB
- (optional) NVIDIA GPU supporting CUDA 9 or higher library (needs to be installed separately)
  - GPU memory at least 4GB
- (optional) mouse with a scroll-wheel is useful for many operations
- (optional) for live data acquisition using Specim FX cameras, Specsensor SDK needs to be installed

Installer

perClass Mira installer will guide you through the installation process.

Welcome screen

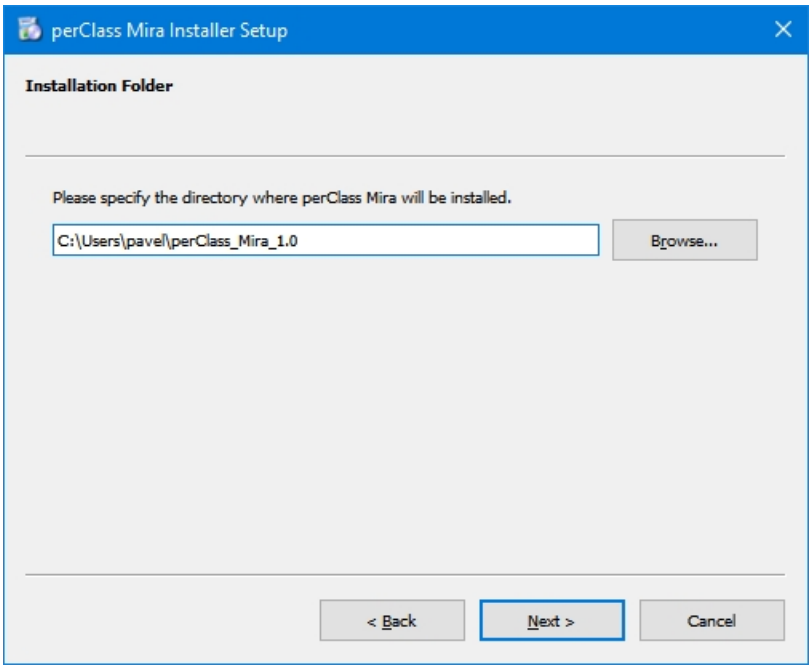
When the installer is launched, the following welcome streen appears:



TIP: In some situations, buttons at the bottom may not be directly visible. It is sufficient to resize the installer window to make buttons appear.

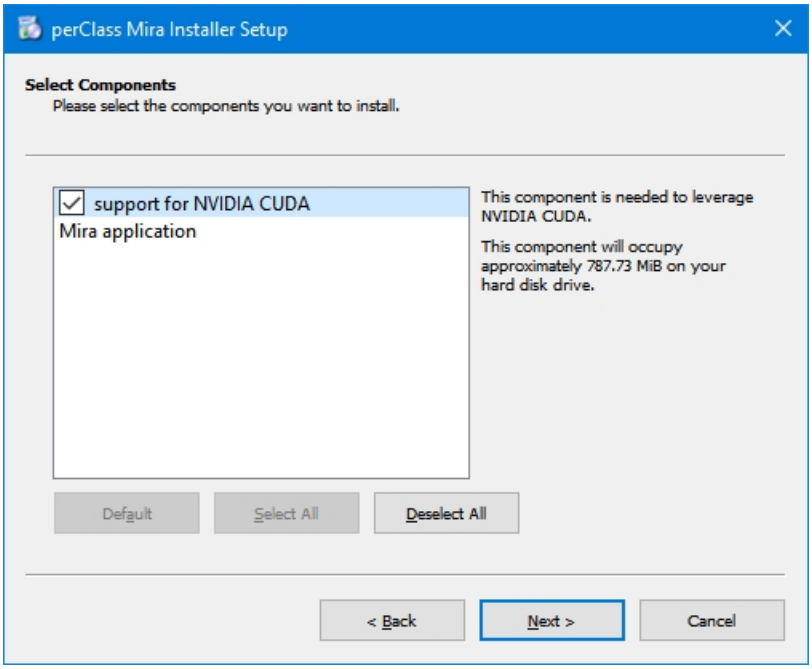
Installation directory

Next, installation directory can be defined. By default, perClass Mira is installed in a user directory



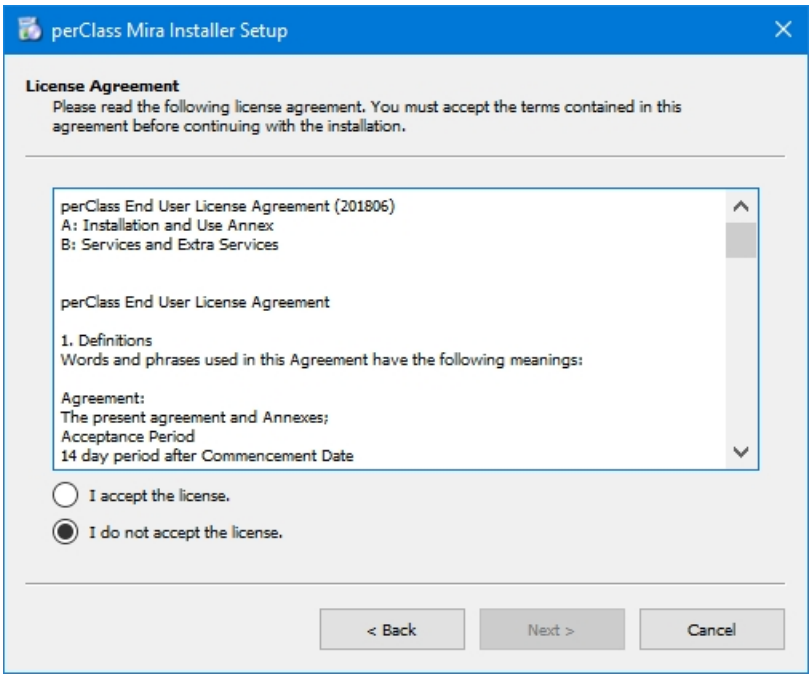
Component selection

In the next window, software components may be selected. Although it is possible to deselect NVIDIA GPU support, we recommend keeping the default setup if possible.



License agreement

In the next step, license agreement is provided and needs to be agreed in order to continue the installation.

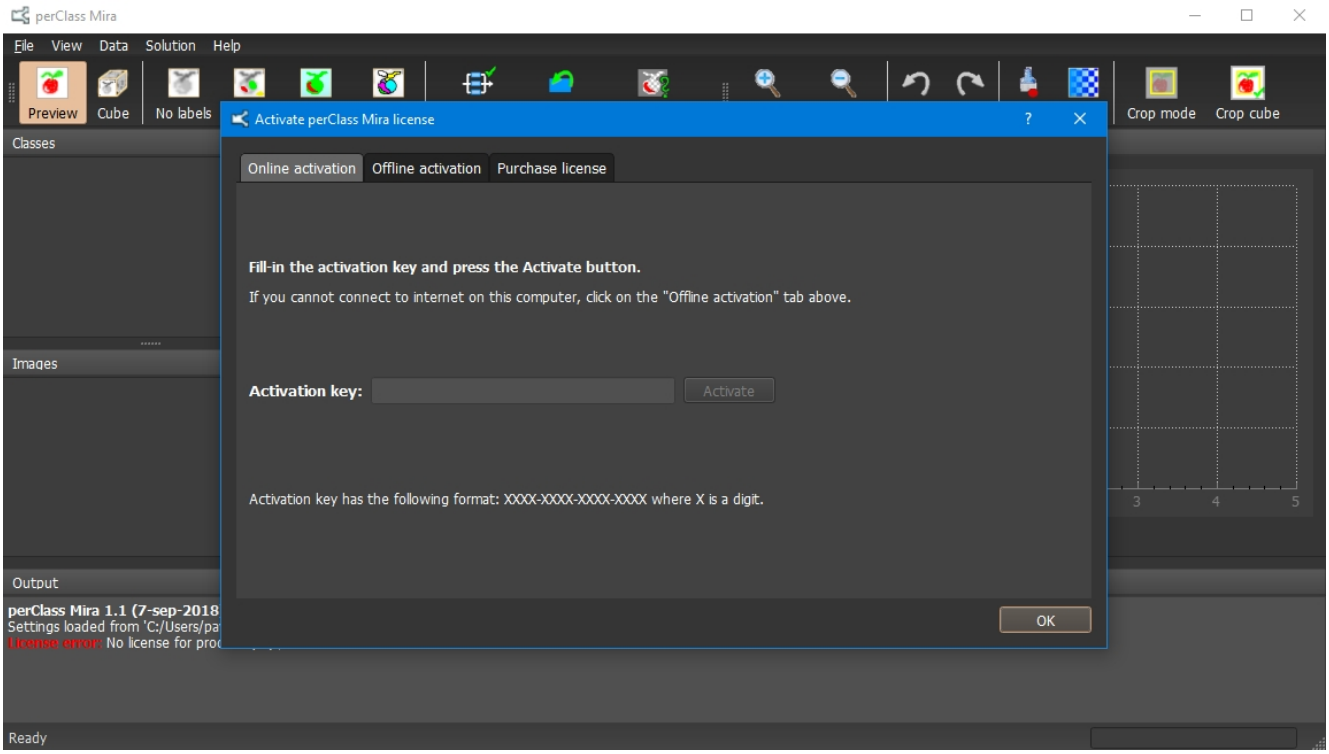


Start Menu Shortcut

Finally, start menu shortcut may be dfined.

First start

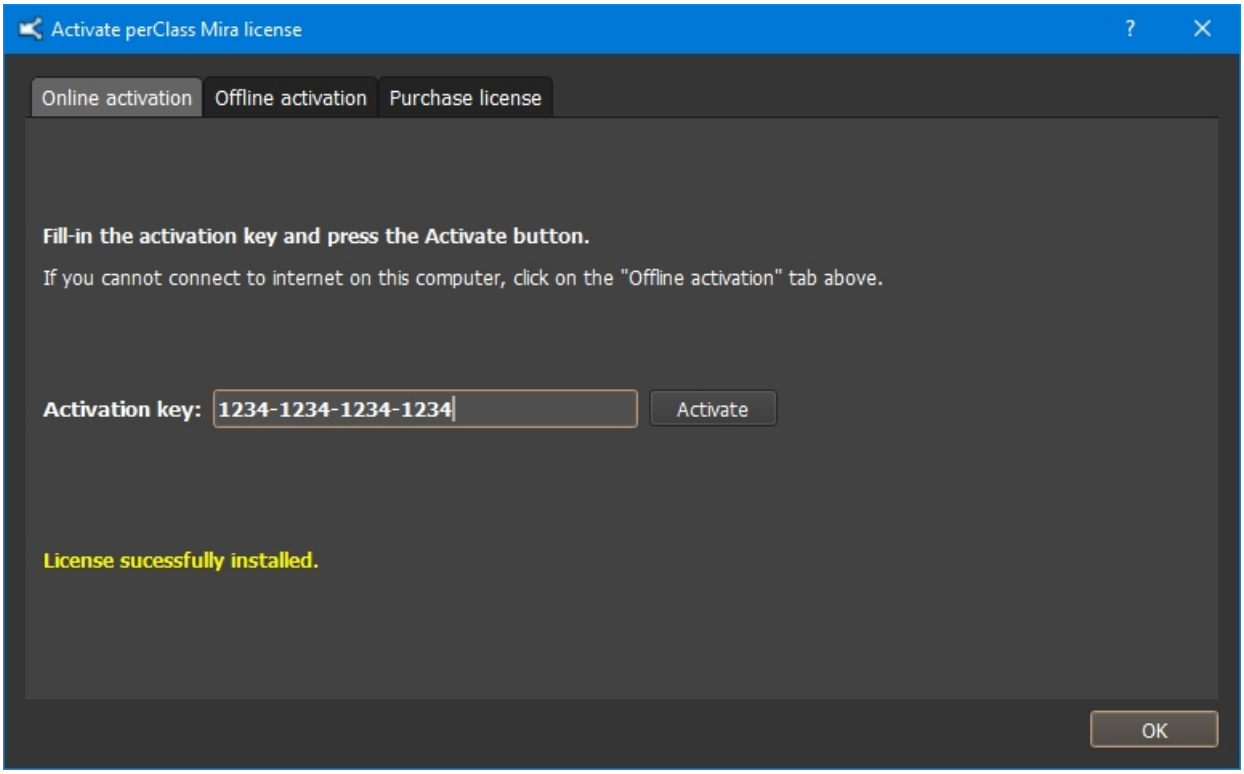
When starting perClass Mira for the first time, there is no license file present. Therefore, an activation dialog appears:



You need to provide an activation key that will pull a license, specific to your machine.

The key has twelve digits in a format such as: 1234-1234-1234-1234. Naturally, you need to use the key you obtained for a demo or for the full version, not this example.

By clicking *Activate* button, the license is installed:



TIP: The license file is stored in **`c:\users\USERNAME\AppData\Roaming\perClassBV`** directory together with perClass Mira configuration file.

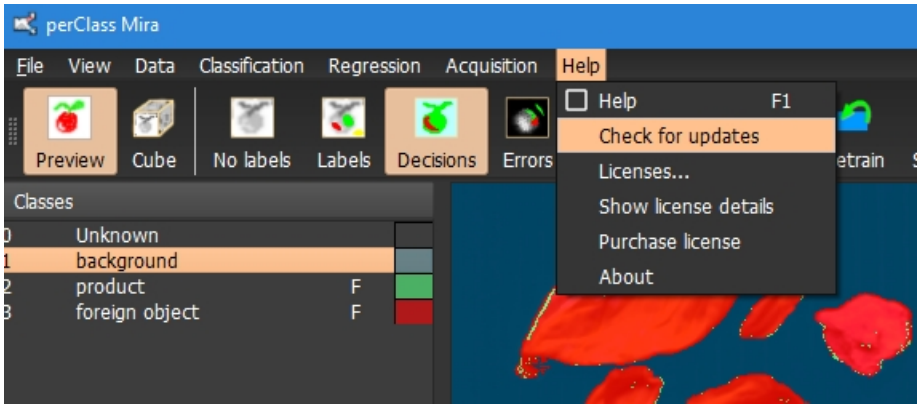
Uninstall

In order to uninstall perClass Mira, run the **maintenancetool.exe** application in its installation directory (by default **`c:\users\USERNAME\perClass_Mira_2.0`**)

Checking for updates

perClass Mira automatically checks for software updates. The default behaviour is that once in 10 days perClass BV server is contacted requesting the latest published software release. If a newer version exists a *Check for updates* dialog is shown with the details.

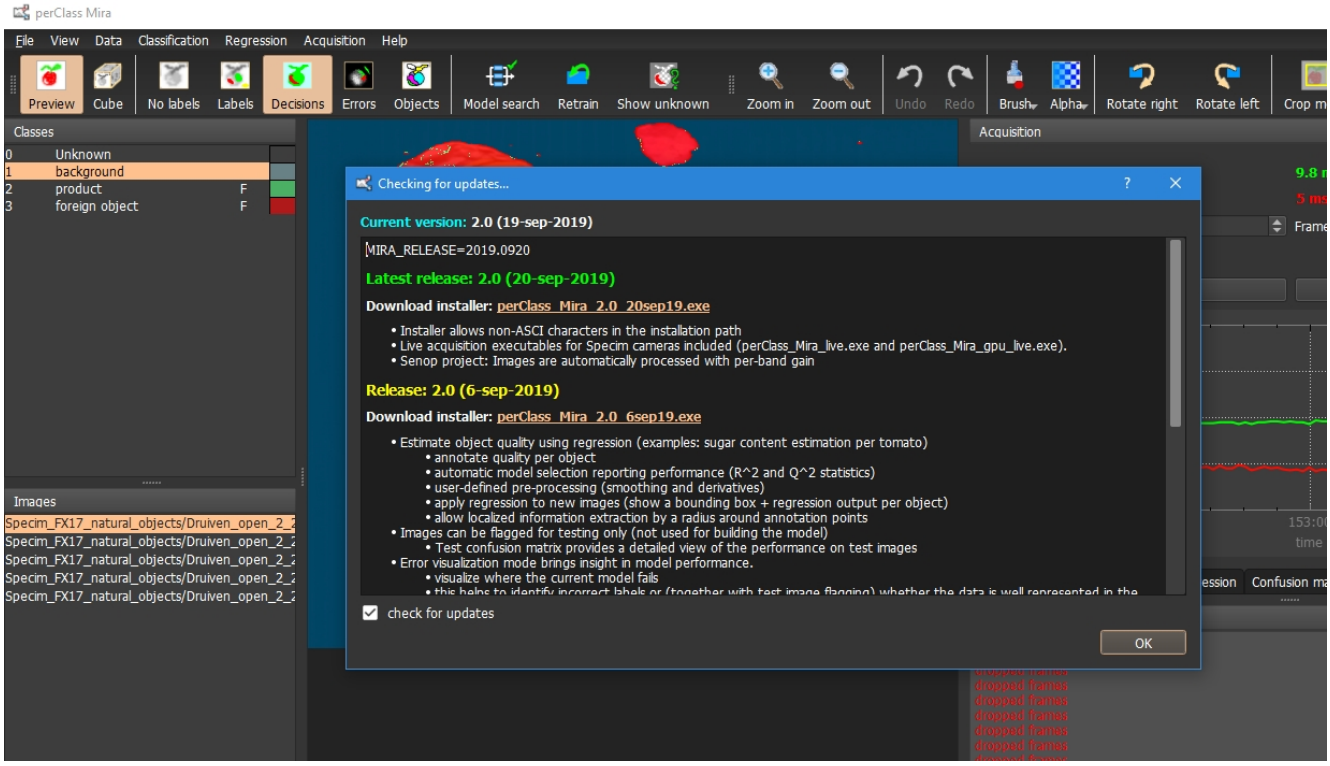
The check fot updates can be also initiated manually from the *Help* menu.



The Update dialog shows latest release notes and download links.

The user needs to manually click on the download link and install the software.

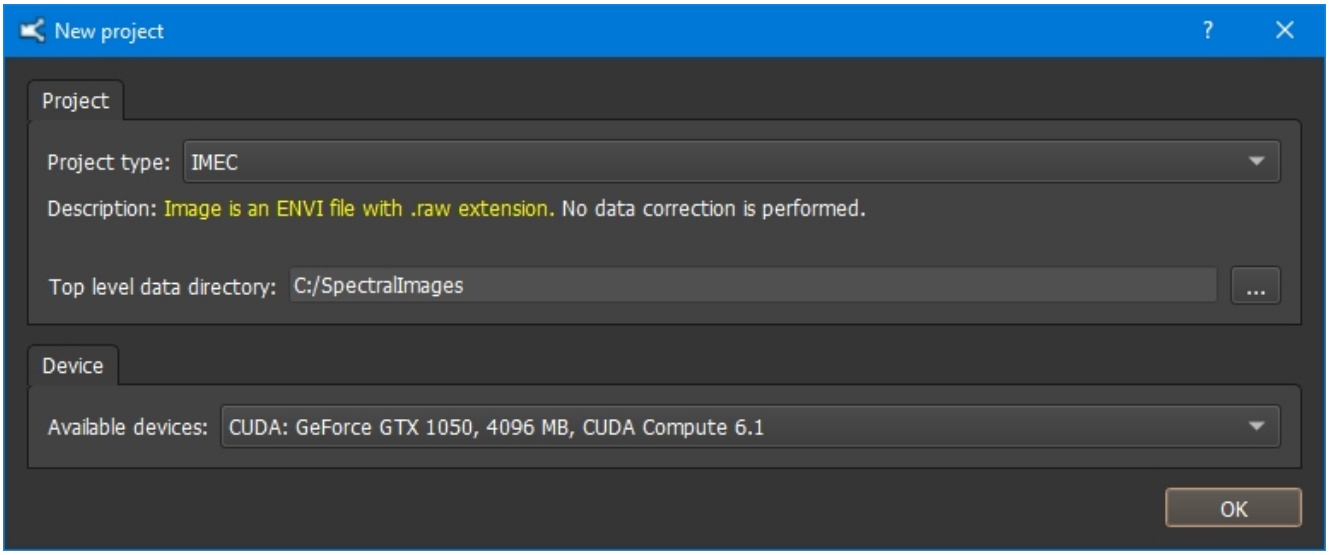
The default update checking behaviour can be switched off by the checkbox in the updates dialog or in the mira.ini file.



Walkthrough example

Selecting project type

Select *New Project* from *File* menu. A dialog will appear:



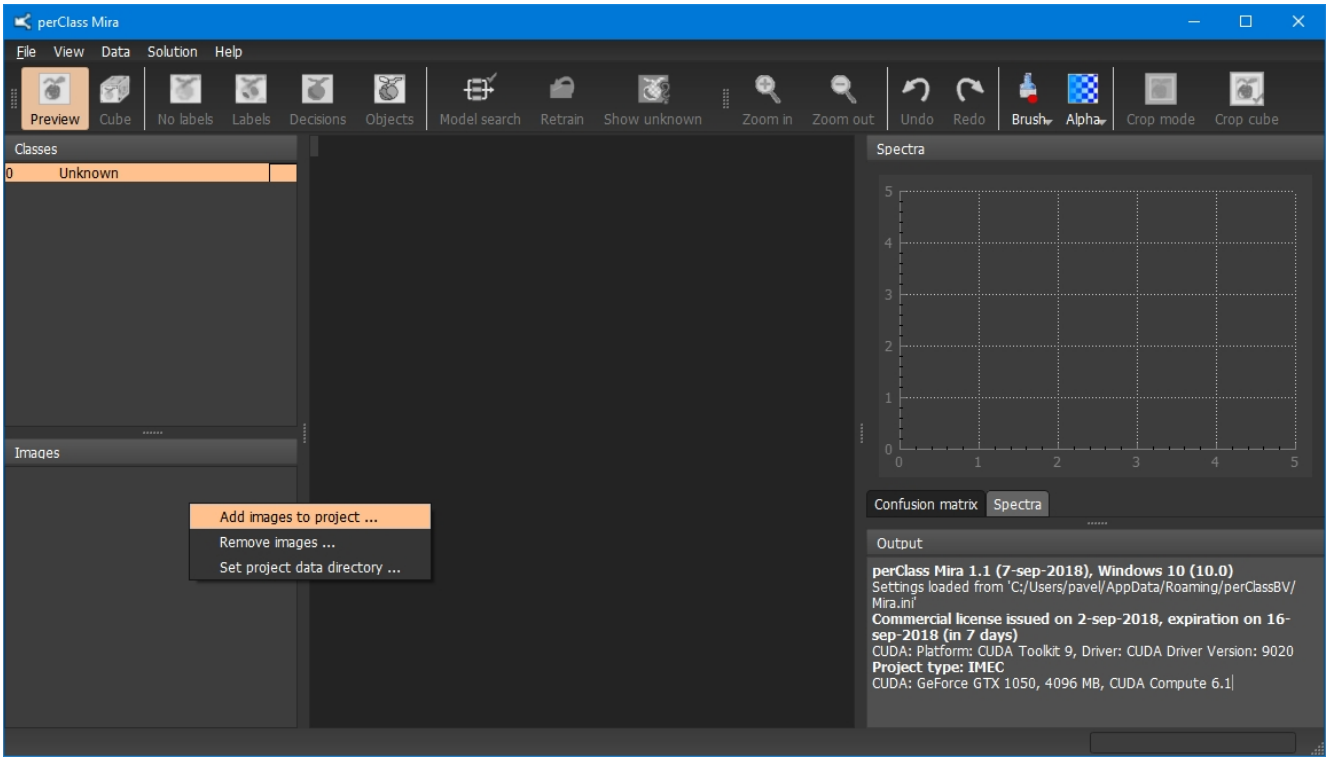
In the New project dialog, we need to specify project type, top level data directory and computational device

- *Project type* specifies the type of images. For example, IMEC project type expects ENVI files with .raw extension.
- *Top level data directory* specifies a directory under which all files are located (possibly in sub-directories). The idea behind is that data is stored in a directory hierarchy e.g. data drive or a network drive. perClass Mira does not write into data directly (the only exception being Specim FX project when white correction is included in a scan using *Create white correction* command)
- *Device* combo box allows us to select computational device such as a graphic card.
    - **TIP:** To use NVIDIA graphics card, CUDA 9.0 or later needs to be installed. When installing CUDA, graphical card drivers are often bundled in the same installer. Make sure you are not selecting by default older graphics card drivers than already present on your system. We recommend always performed custom installation making sure that newest possible NVIDIA drivers are used.
    - Device selection is not saved in the project and may be changed anytime based on the current situation of the used computer system. To do that simply select *New Project*, select graphical card of interest and then *Open project* you saved earlier.

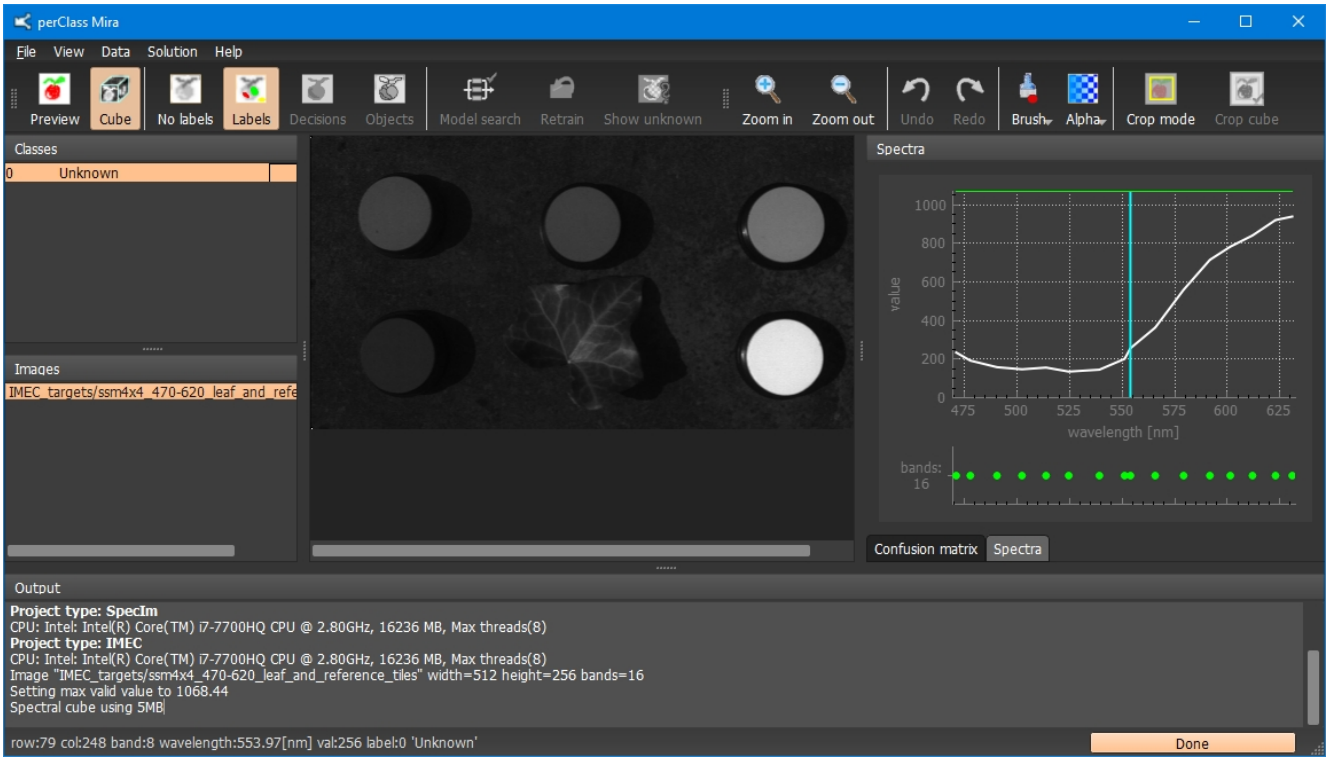
Adding images

Add image selecting *Add images to project...* command in:

- Context menu in the image list pannel (right-click) to open
- File menu



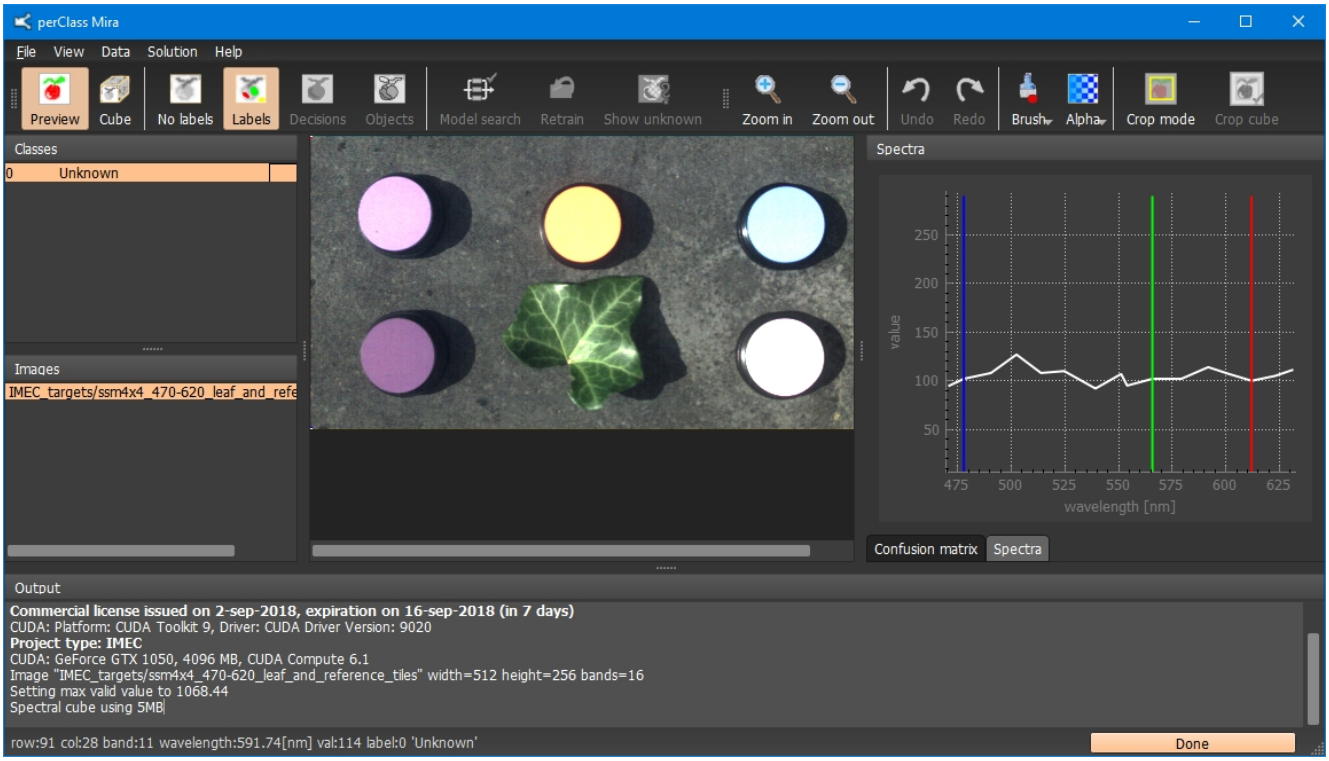
A file dialog will open where you can select one or more images.



RGB preview from selected bands

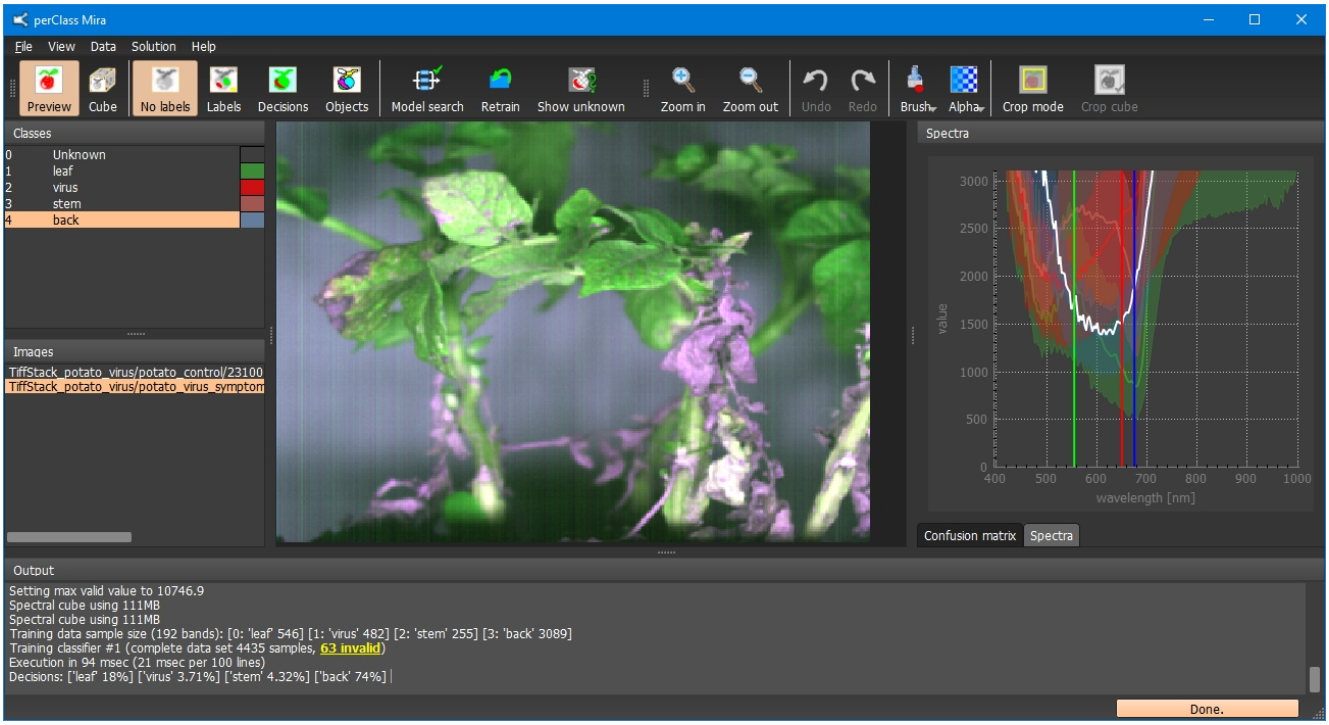
If a spectral cube is loaded, we may visualize RGB preview image from specific hand selected bands. When pressing the *Preview* button, the spectral plot will show R,G, and B band lines.

We may drag them to change the way preview image is generated:



TIP: To brighten or darken the image, use the mouse scroll wheel in the spectral plot

RGB preview is especially useful with natural objects and remote sensing scenes where certain types of materials may be visually distinguished simplifying the labeling process. For example, th eiage below shows a potato plant with some leaves infected with a virus (courtesy WUR University). Note that the virus-infected leaves exhibit distinct color in RGB preview. We would not be able to distinguish them on any single band image of the same cube without prior knowledge or without considering specific macroscopic folding of the leaves.



Selecting spectral band

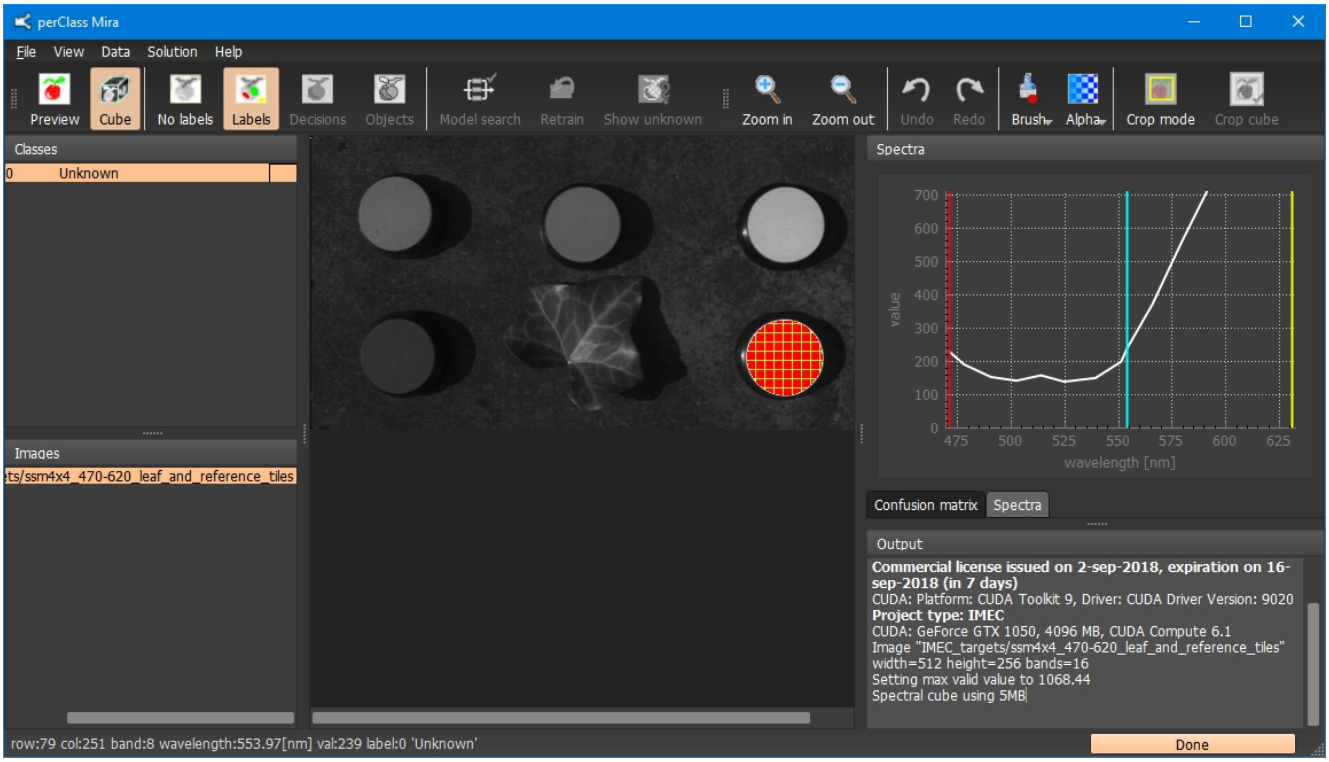
You may select a spectral band in several ways:

- By clicking in the spectral plot pannel on a desired band
- Via Page Up/Page Down keys
- By a mouse scroll-wheel when holding Shift key in the image view

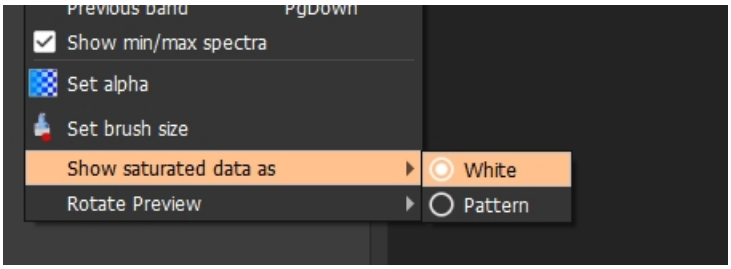
Adjusting display range

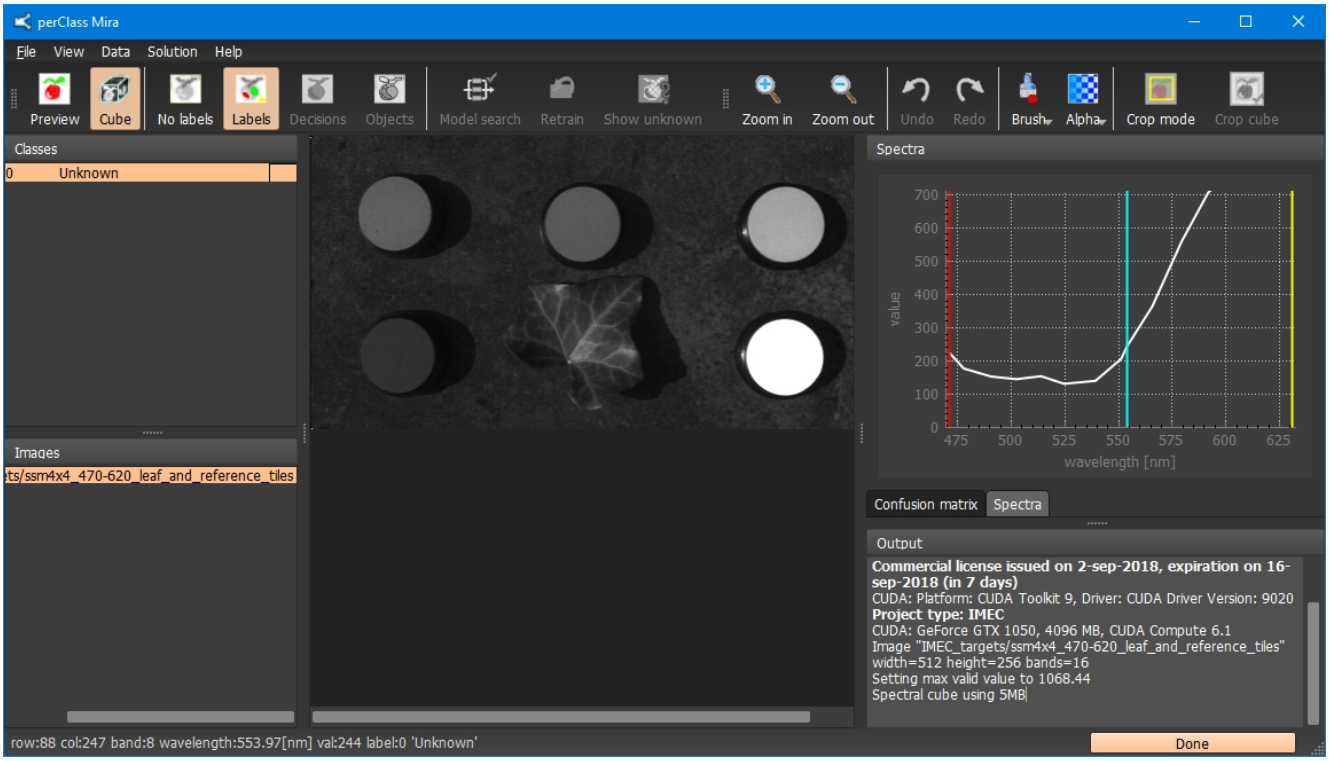
We may wish to adjust visualization range to see more details in dark image areas. This is possible using mouse scroll-wheel in the spectral pannel
Alternatively, if spectral pannel is selected (e.g. by clicking) you may use cursor Up and Down keys.

By default, image pixels with a value out of the current display range is rendered in red/yellow cross pattern. This is to highlight out-of-range areas.



It is possible to change the out-of-range rendering style in by *Show saturated data as* command in the *View* menu. Alternative style is to paint out-of-range values as white:



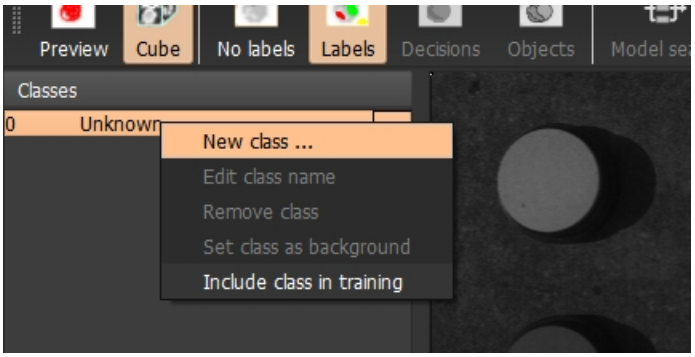


Defining classes

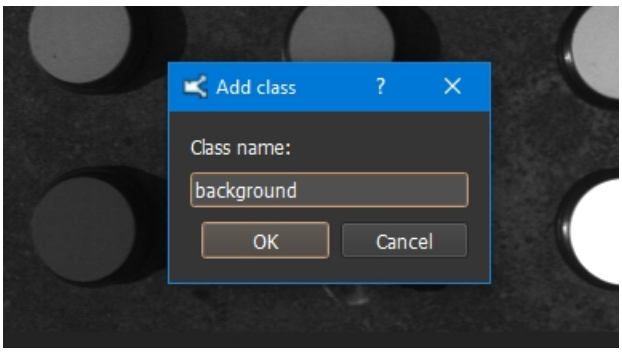
To build classification models, we first need to define classes of interest. We use the term "class" as a synonym of category - a group defining a material or function of a set of pixels. For example, in a potato segmentation task, we may want to define *leaf*, *stem*, and *background* as separate classes. Each pixel belongs only to one class.

perClass Mira interface presents a list of classes as a docked window, by default on the left side.

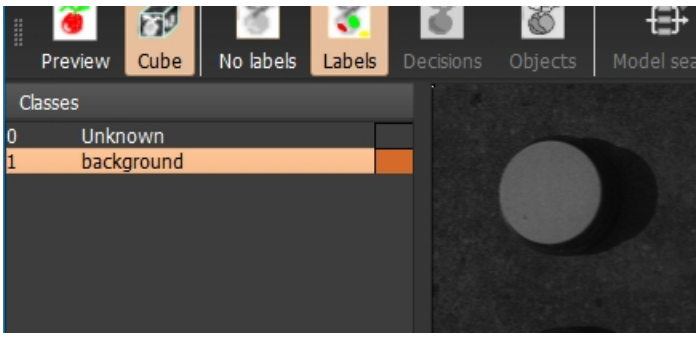
We may create a new class by right-clicking in the class list and selecting *New class* command:



A dialog window will open asking for the name of the new class:



The class is then added to the class list:

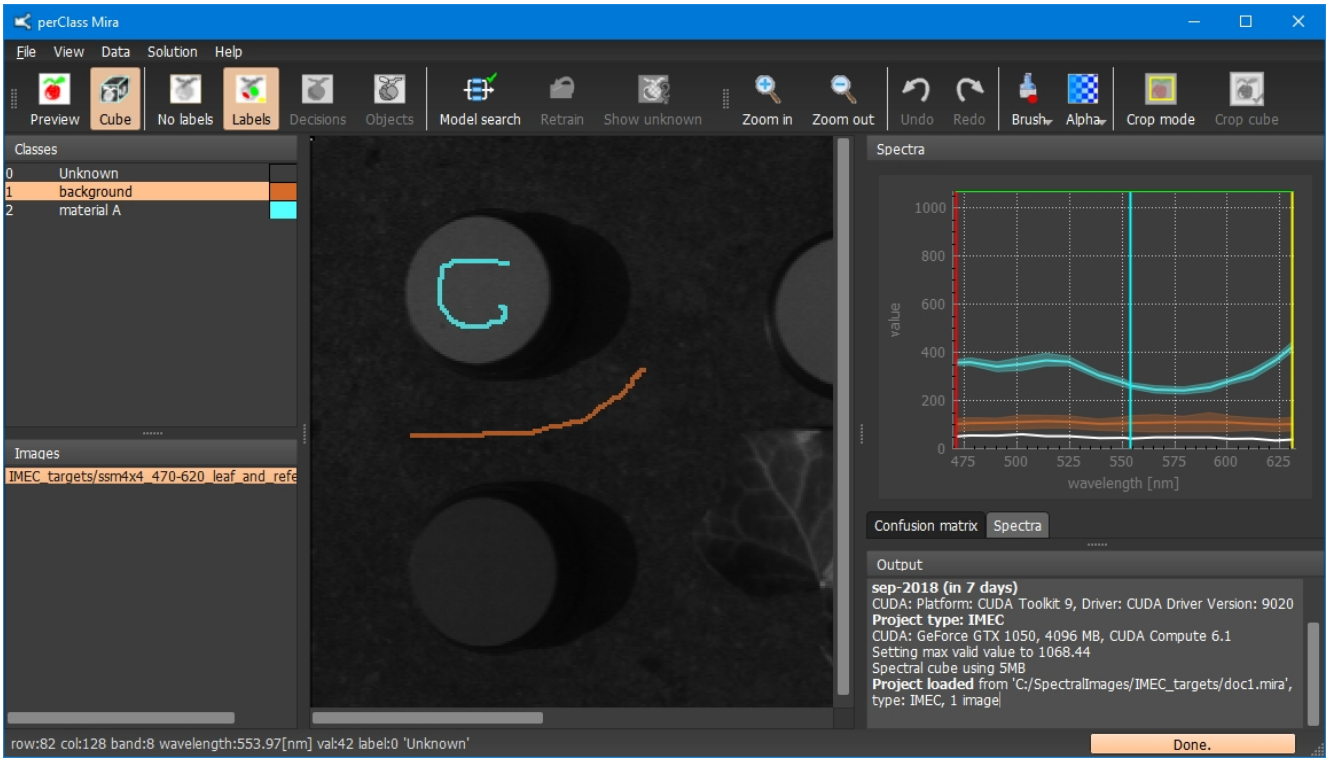


We may change class color by clicking on the color swatch widget in the right part of the list item.

TIP: To add a class quickly with a generated name and a random color, press *n* key

Labeling examples

Paint labels by selecting a class in the class list and moving mouse over image while holding the left mouse button.



You can see that spectral plot is updated after each stroke with information on class distribution. The thick central line represents class mean and the colored area min/max range for each wavelength.

TIP: To remove labels paint while holding *Shift* key or select the "Unknown" class.

TIP: To quickly switch between classes, press the corresponding digit key (e.g. press '1' to paint beckground in our example)

We may switch off label layer using *No labels* command on the toolbar.

TIP: We may switch to the last image layer by pressing *Space* key. In our example, this would flip between class labels and no labels.

Training a model

Once at least two classes with sufficient number of examples are labeled, we may build a statstical classification

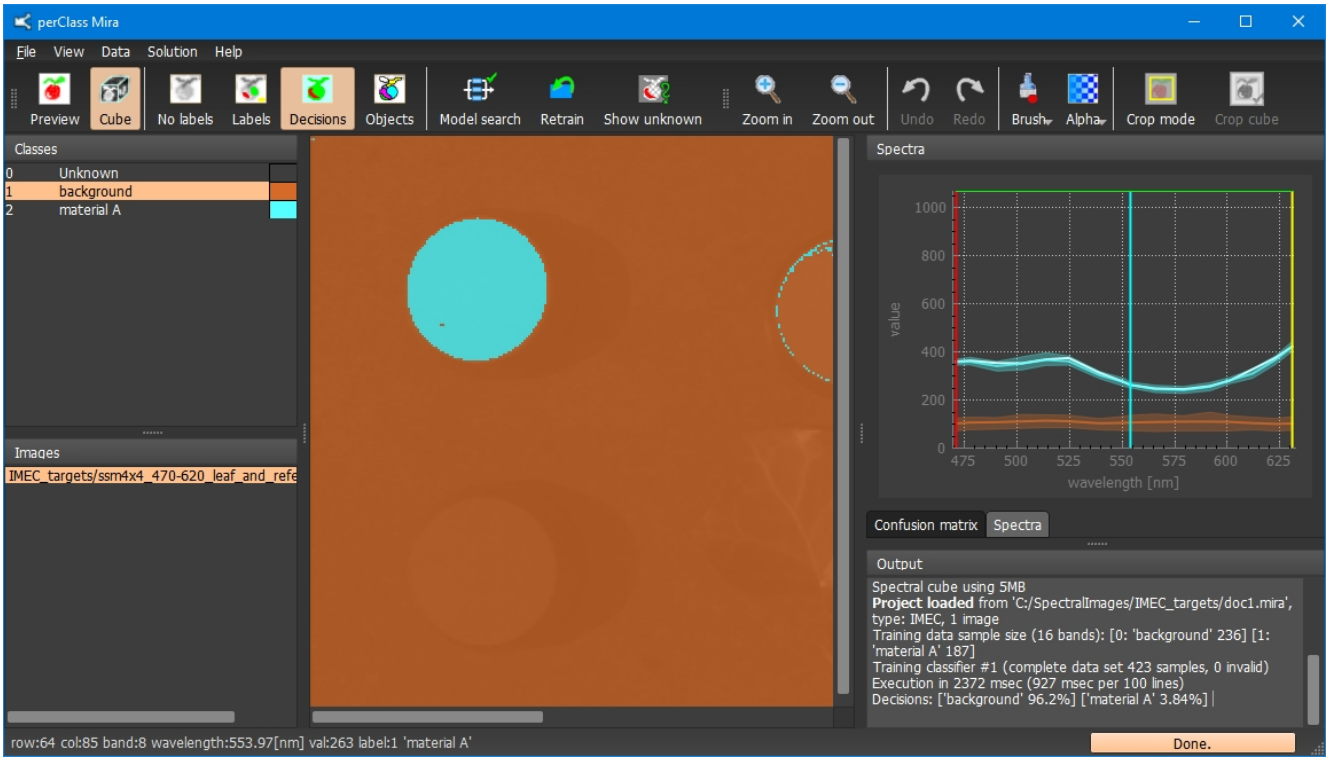
model.

perClass Mira uses a simple scheme for data validation that allows us to easily discard outliers. It is specified by the green line at the top of the plot representing the maximum valid value. The spectra with values higher than the green line will be discarded from training.

To train a model, we need to make sure that our training data is under this green line. We will discuss this in detail in the next section. To adjust the max validation value, simply move the green line by mouse.

The first time, we need to select a good model using *Model search* command (or 'm' key shortcut). The stage of the search is indicated by the progress bar in the right-bottom corner.

After the search, the best data representation and model is used to label the entire image. Note that this automatically the interface to the *Classifier decisions* layer.



Each pixel in our image is labeled to one of the defined classes.

TIP: We may switch back to the label layer by pressing *Space bar* key

When we paint more examples, we do not need to re-run the search. Instead, may only retrain the best selected model (*Retrain model* command or 'r' key shortcut)

TIP: If we include many more examples adding new sub-types of our classes (e.g. dark bacground), it may be beneficial to re-run the model search. The reason is that different model may be releavant in this new situation.

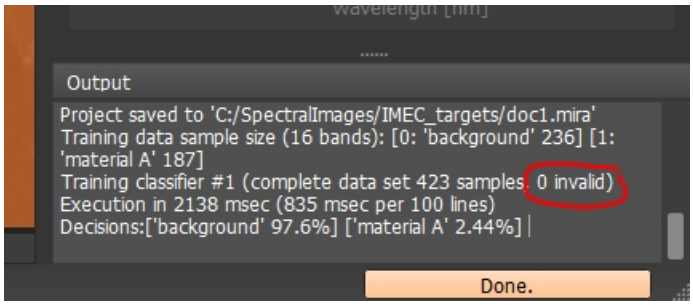
Sample validation

By default, perClass Mira 2.0 and higher leverages all labeled examples from training images to build a classification model. There are situations where some labeled examples should be excluded from training. For example, there may be artifacts and spikes on spectra, uninformative areas due to sensor saturation or defective pixels. In such cases a simple data validation mechanism can be applied exluding bad spectra from training.

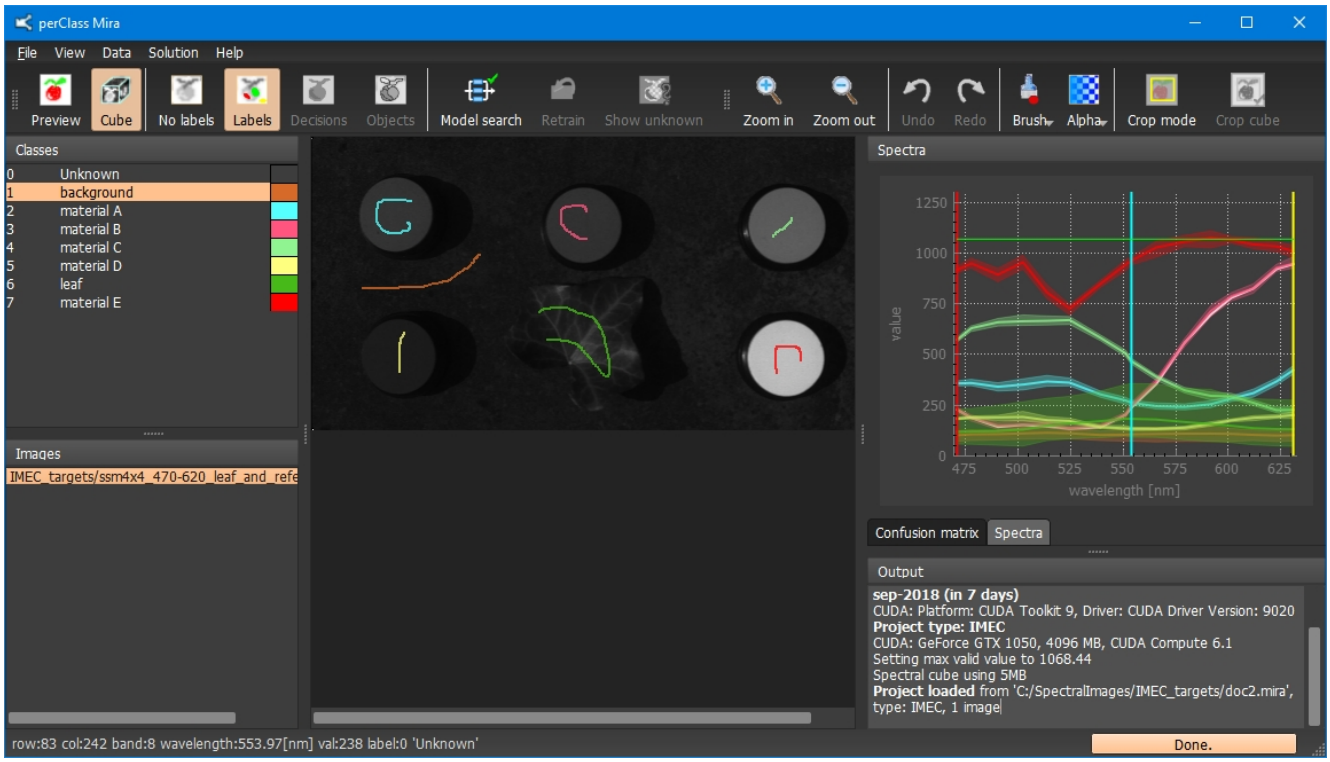
In the spectral plot, one may set a "maximum validation value" using the context menu. A green horizontal line will appear. By adjusting the green line in the spectral plot, we set a bound on spectra included in the training set. This allows us to remove badly scanned areas, outliers with spectral artifacts and other pixels if out-of-range values. Note, that this removal only concerns the training set, during model execution all image pixels are processed. For a method to highlight outliers in execution, see the section Active learning

You can remove the the validation line by double-clicking on it.

Upon training, the number of invalid examples removed from the training set, is displayed in the output window:



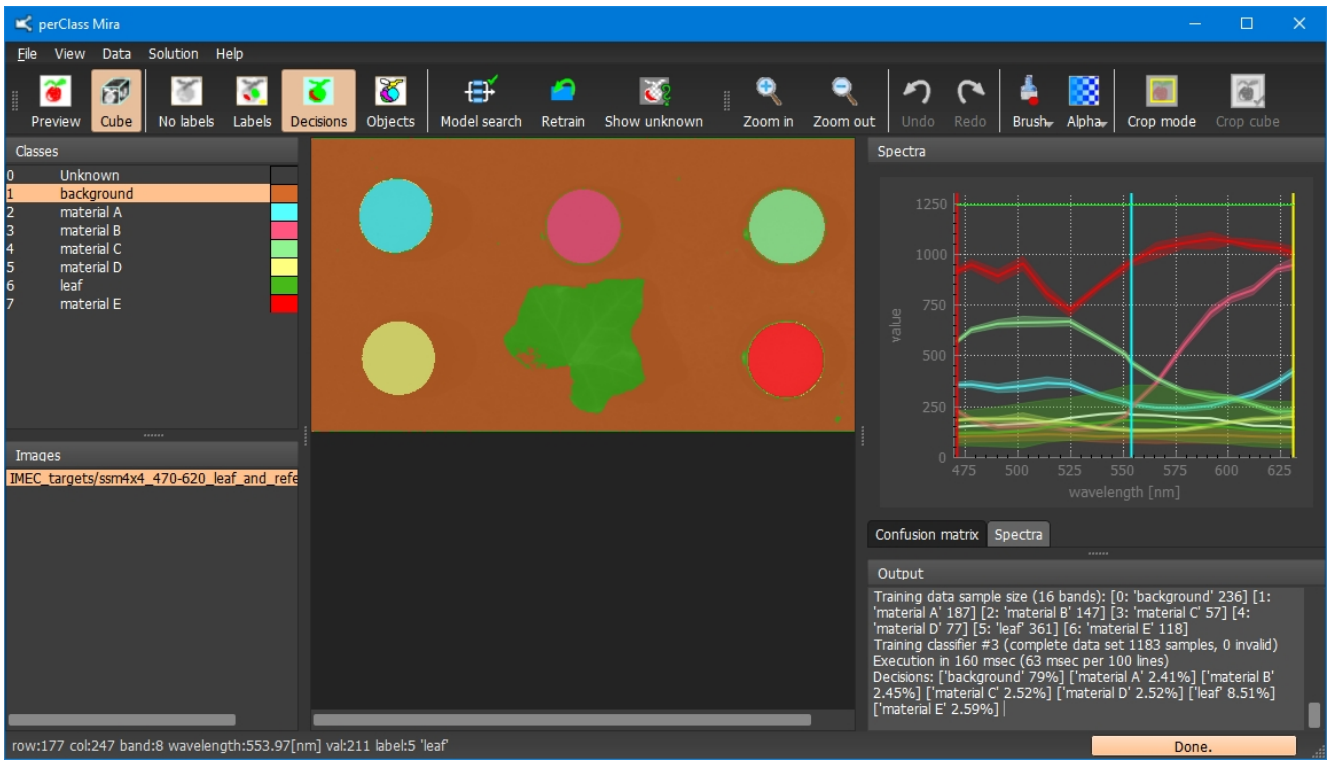
If we create more classes in our example we can see situation as follows:



Some of the spectra of the red class "material E" are above the green validation line between 575 and 605 nm.

If we try to run to train a model, we will see the number of samples for "material E" highlighted in red due to insufficient number of examples (less than 50):

The solution is simple: We will move the green line in the spectral plot above all relevant data. Now, the classifier can be built just fine using all labeled examples:

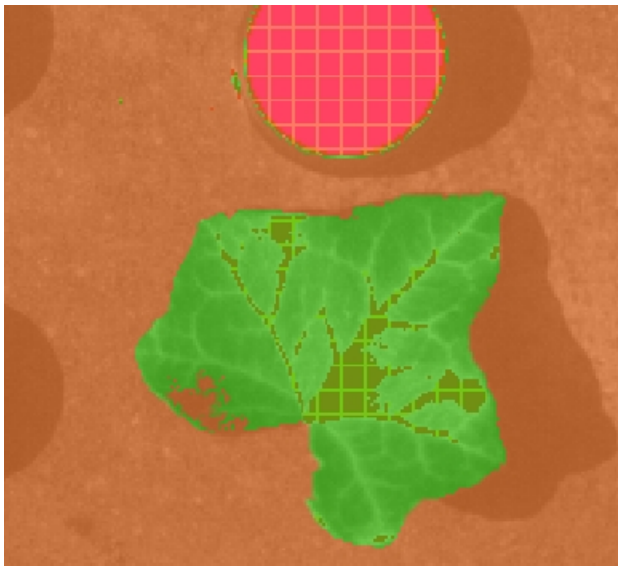


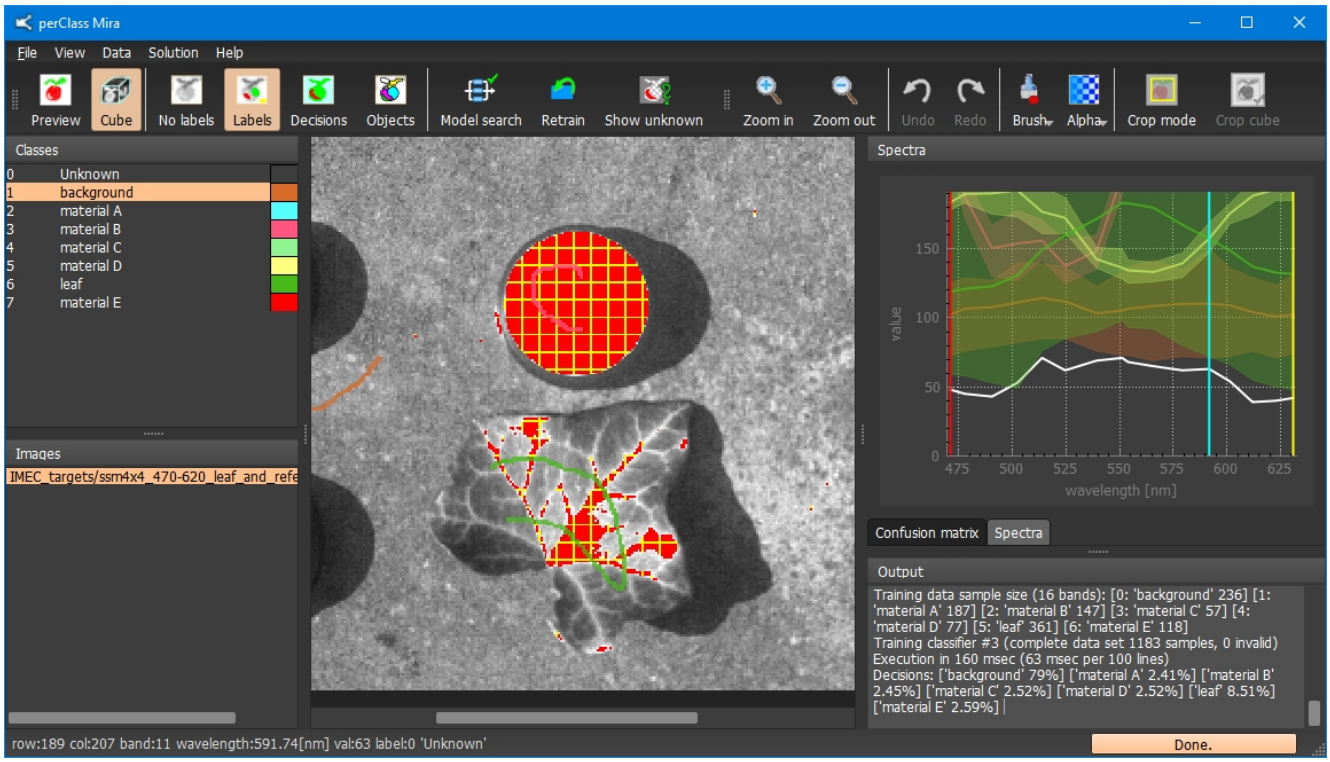
Improving the model

Labeling more examples

We may improve the model by labeling more examples in areas where classification does not work as expected.

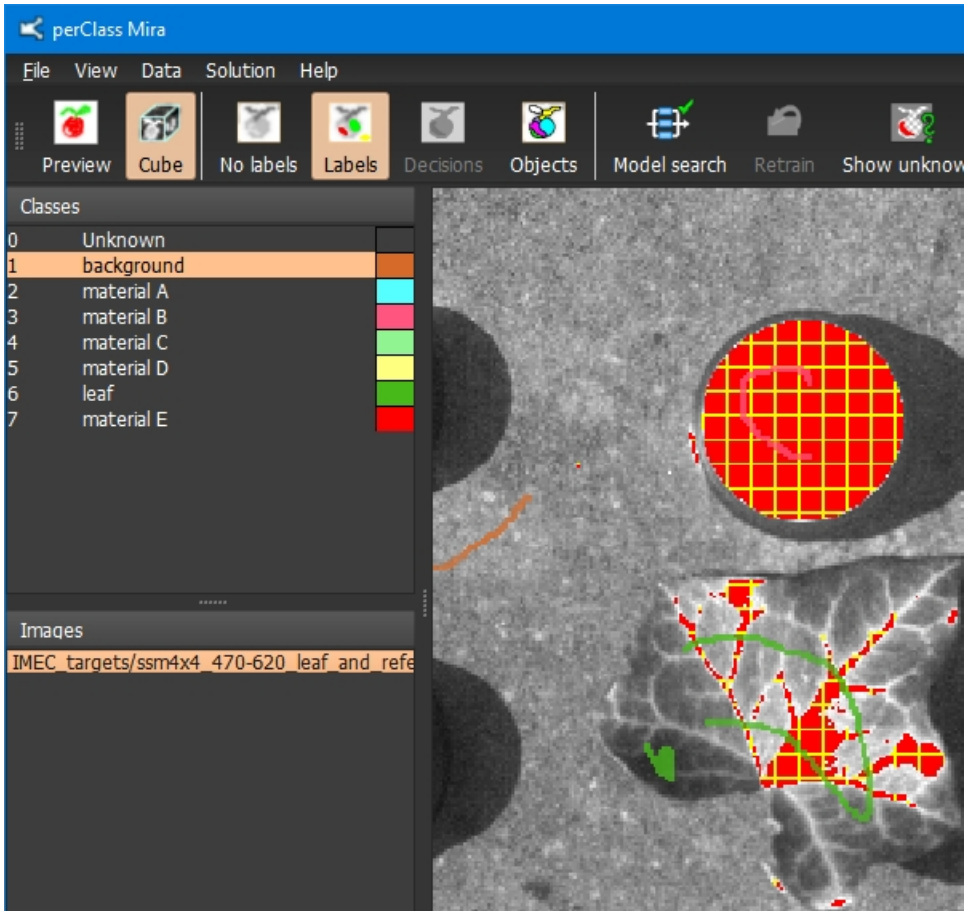
For example, the leaf in the center of the image has a hole labeled as background. We may first inspect, how the area looks when zooming. We also adjust the spectral display range to see more details in dark areas:



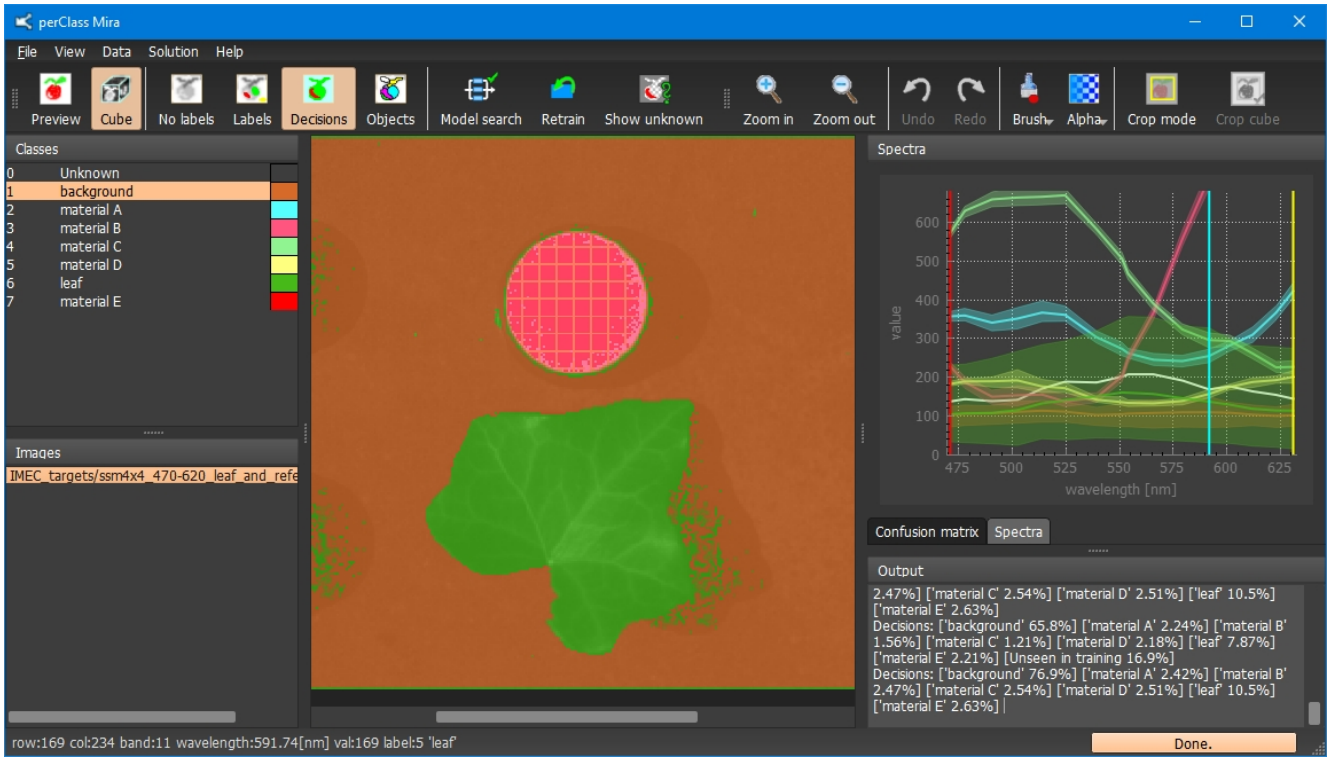


We can see that indeed the leaf is without any hole and provide more examples of the leaf class in this area.

TIP: To select a class based on image we may right-click on a desired pixel (labeled as 'leaf') and select *Set 'leaf' as current* in the context menu. Alternative is to press 't' keystroke while hovering mouse over any pixel of the desired class.



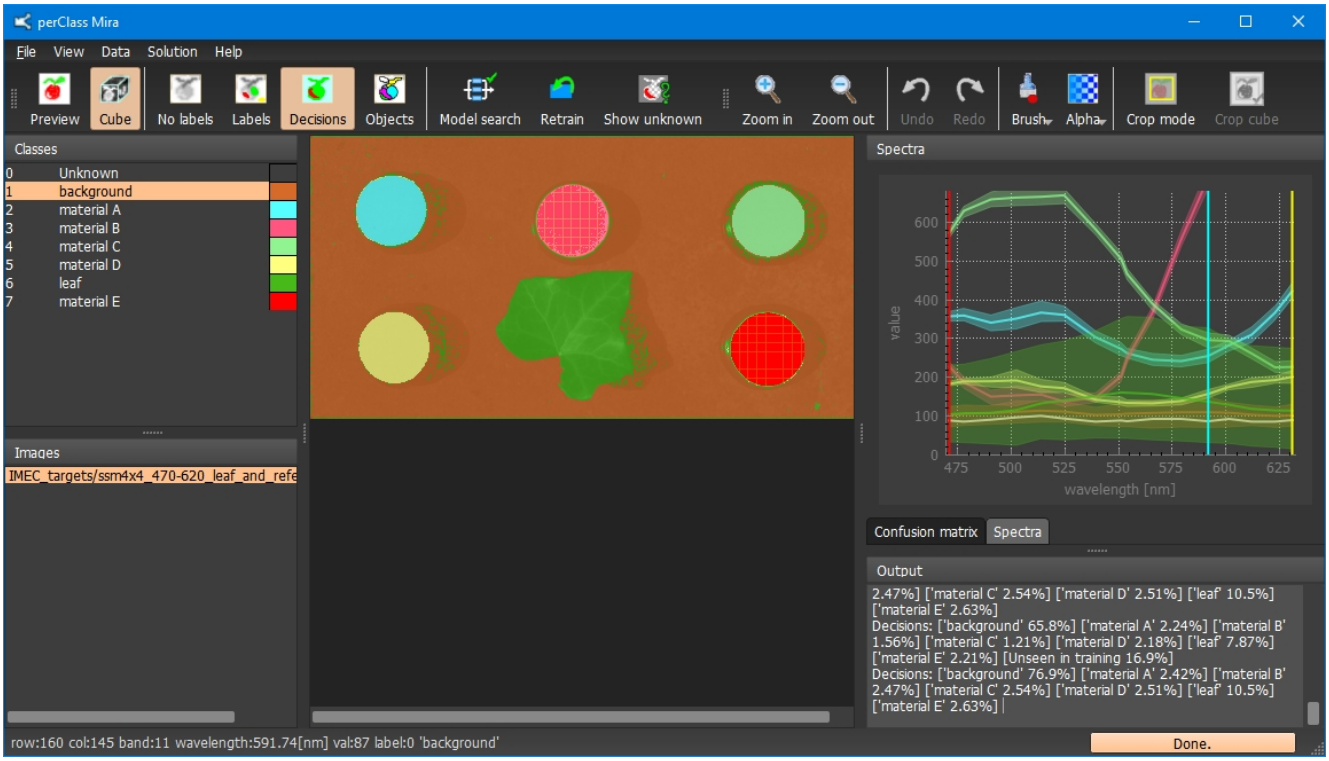
Now we may retrain the classifier. The hole is closed. Note the problematic classification in the right part of the leaf. We will discuss it in the next section.



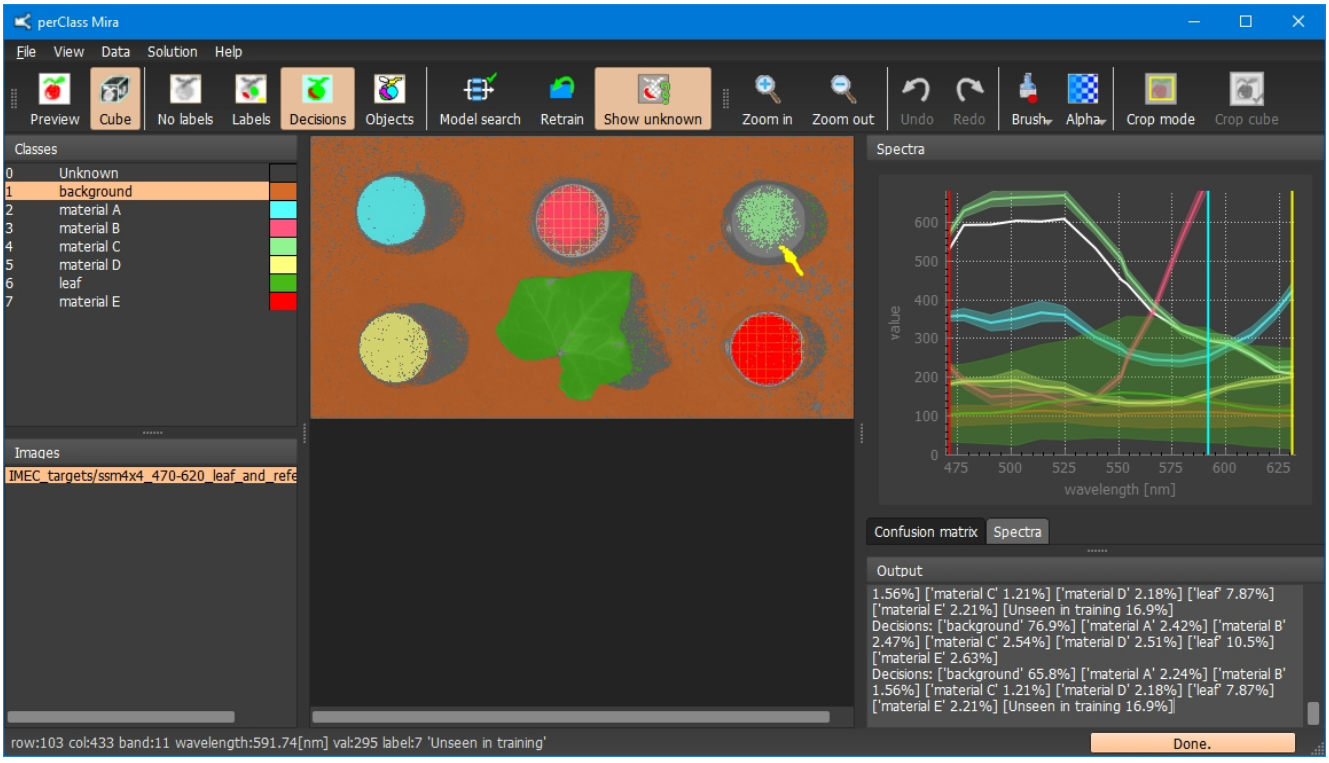
Active learning

perClass Mira provides easy to use active learning functionality offering easy-to-use guidance in labeling the moset relevant examples.

Let us consider the classifier we've built so far:



To enable active learning tool, click *Show unknown data* button on the toolbar (or 'u' keystroke). Portitions of the image will be now highlighted in transparency:

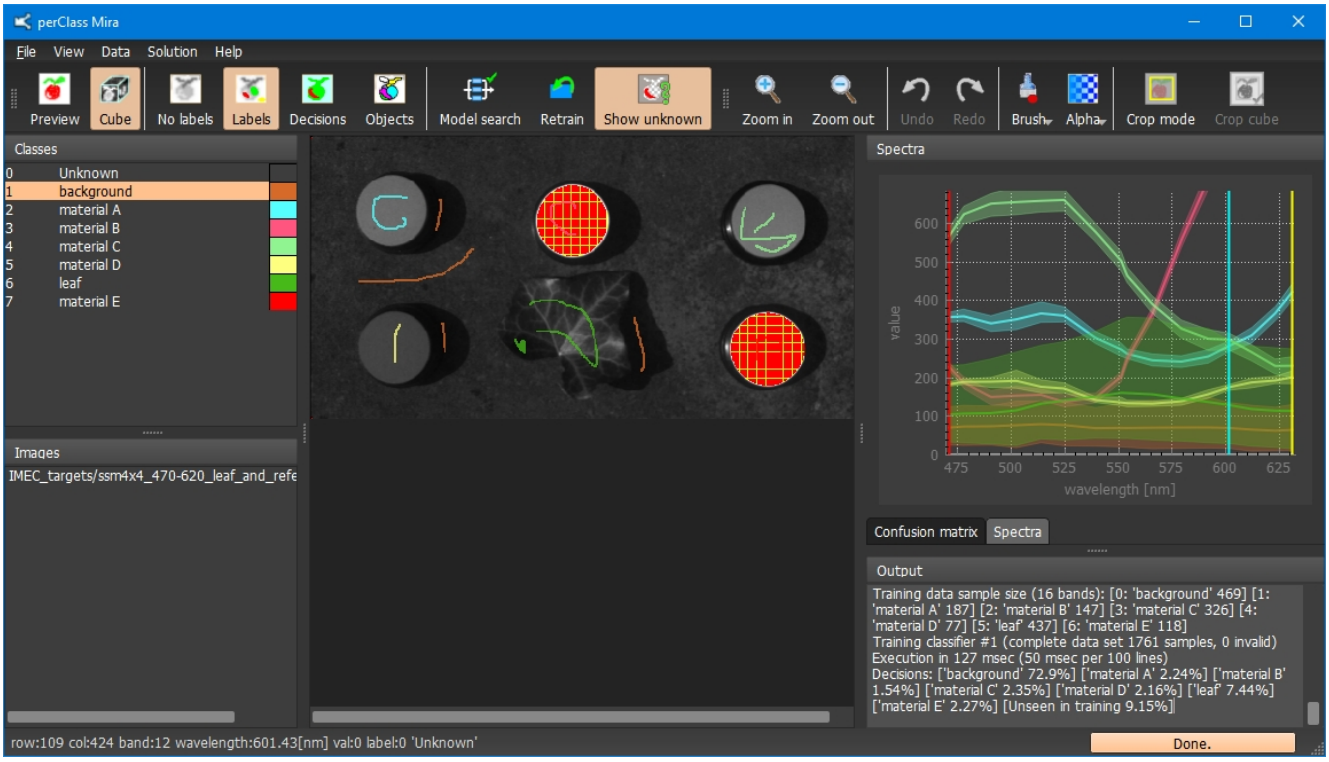


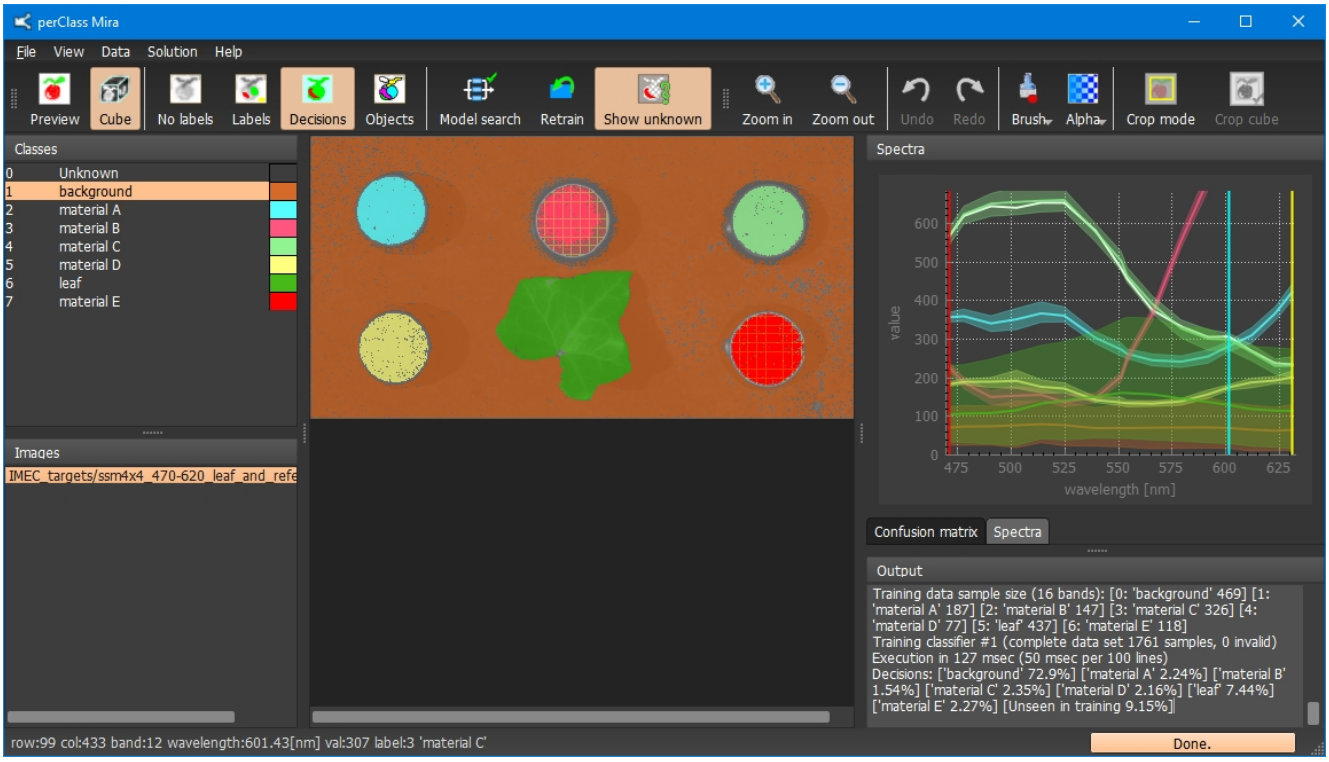
The yellow arrow above the image indicates the position of mouse pixel and the respective spectrum is given as white solid line in the spectral pane.

The *Show unknown data* tool highlights areas that were not seen in model training. Although, we may get good quality results for these examples, these are only extrapolations for situations not known to the model.

One insight from this result is that we need to label more background examples in dark areas. Also the "material C" target highlighted by the yellow error needs better labeling.

Here is example of improved labeling:





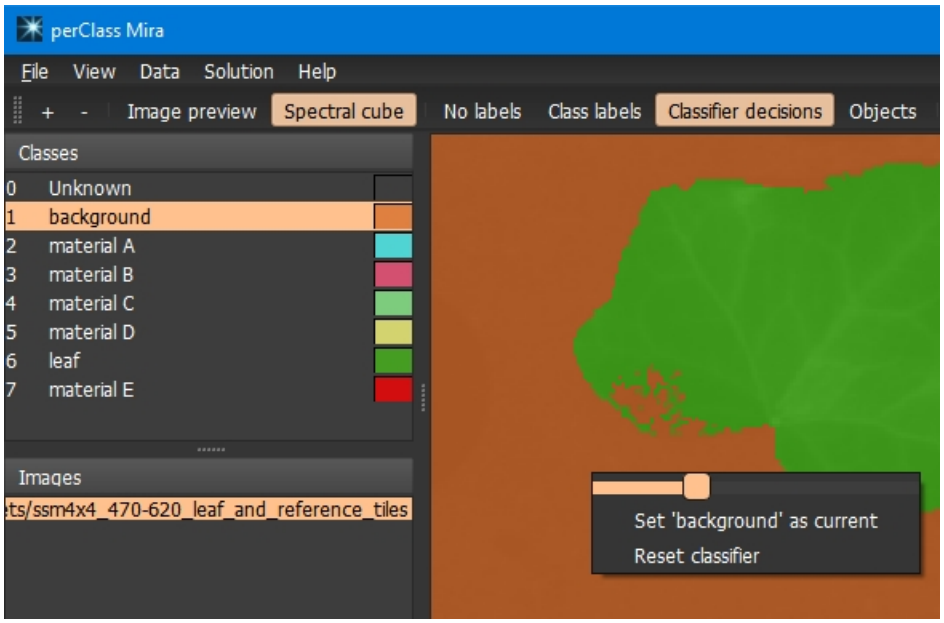
We may now wish to improve also the "material B" definition and maybe label target rims as background or introduce another class for this material.

TIP: Although perClass Mira can accomodate multi-modal classes such as the background, it is beneficial to define as many clarly distinct material categories as possible. This helps the machine learning engine to find better overall solutions.

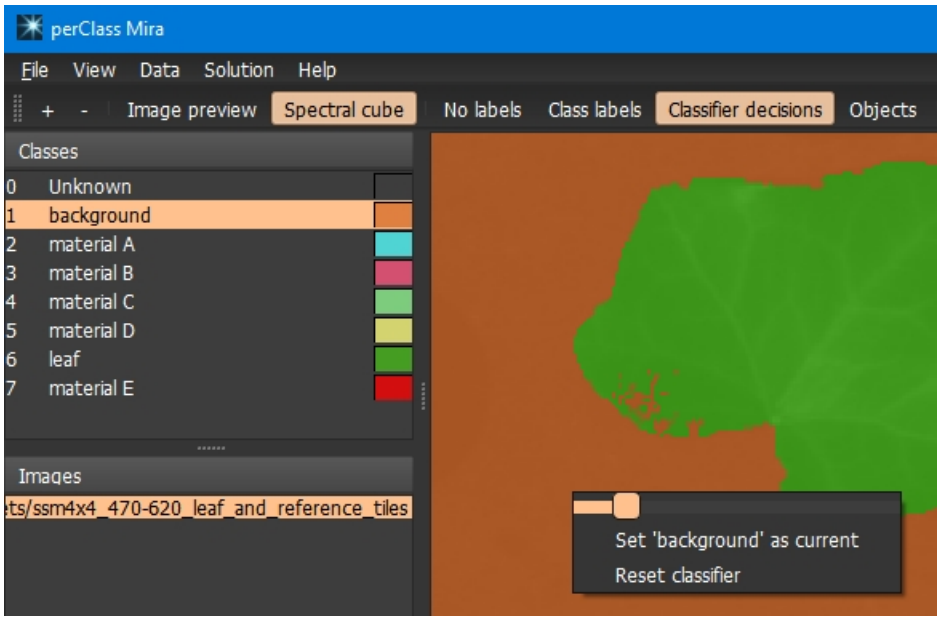
Tuninig model performance

Alternative way to improve classifier decisions is to fine-time model to desired performance.

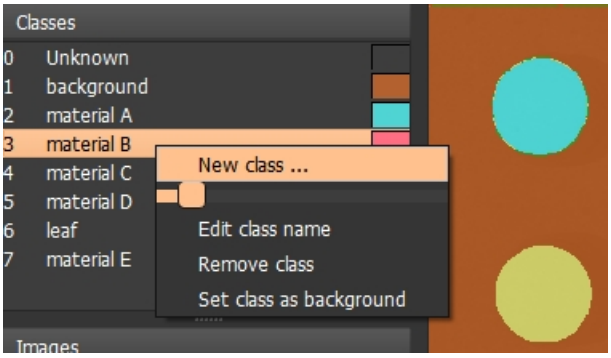
This can be achived by right-clicking on any pixek of a class of interest in the decision image. The context menu will show a slider for the class under cursor.



Moving this slider, we can adjust different performance trade-off for the given class. For example, by reducing slider value on background, we express that it is 'less important' and, as a consequence will see the whole on the leaf diminishing in extent. This happens live as we move the slider.



TIP: Sometimes, clesses of interest may not be easy to pin-point by mouse or entirely missing from the decision image. In these situations, it is easier to right-click on the desired class in the class list (while the decision layer is active) and use the slider:



TIP: To remove any performance trade-off adjustments, use *Reset classifier* command in the right-click context menu on the image layed with decisions.

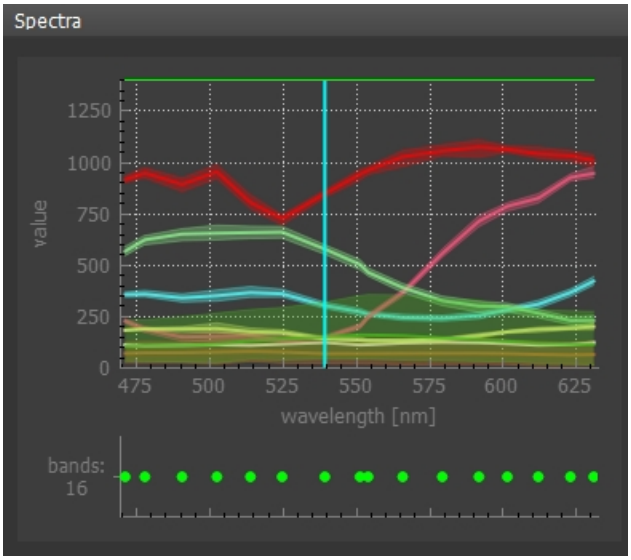
Limiting spectral range

By default, all available spectral wavelengths are used to build classifiers. We may want to adjust the used spectral range from several reasons:

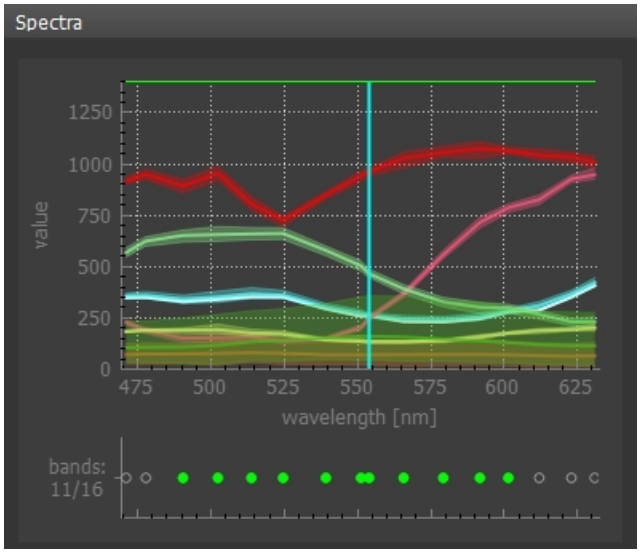
1. In this way, we may quickly validate whether specific range is sufcient to separate classes of interest

2. We may discard noisy spectral measurements at the start or end of the spectral range.

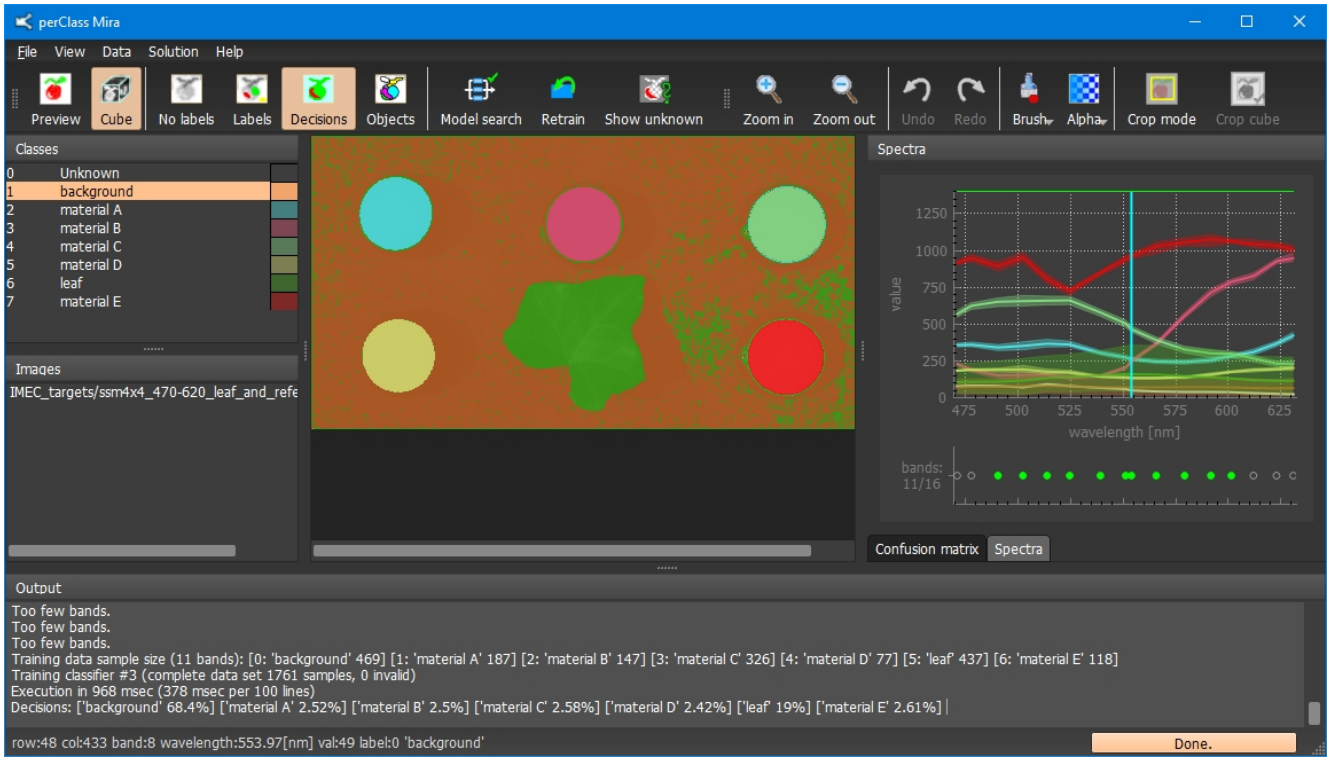
To adjust start and end of used spectral range, simply toggle spectral bands in the band selection bar under the spectral plot:



We disable bands at the start and end of the spectral range:



When retraining the classifier, we can observe that the "leaf" and "background" classes are more difficult to separate.



Executing on new images

To execute on new images, use *Add images* command in the image list. If decision layer is active, the current classifier will be directly applied to the selected image.

Note, that in one project, we can only work with images describing the same spectral range with identical number of spectral bands.

Exporting solutions for deployment

Once we are happy with the classification solution, we may export it for deployment outiside of perClass Mira interface, e.g. in custom application.

This is accomplied with *Export classifier...* commands in *Solution* menu.

The result of the export is a .ppl file that can be loaded by perClass Runtime library (perclass.dll) This library

needs to be integrated in custom application to process new images.

For the latest perClass Runtime 5.4, it is possible to choose single or double precision floating values. Older runtimes require double precision spectral data on input.

Existing integrations include:

- IMEC HSIViewer contains perClass Runtime component that allows to execute solutions from within the Viewer interface

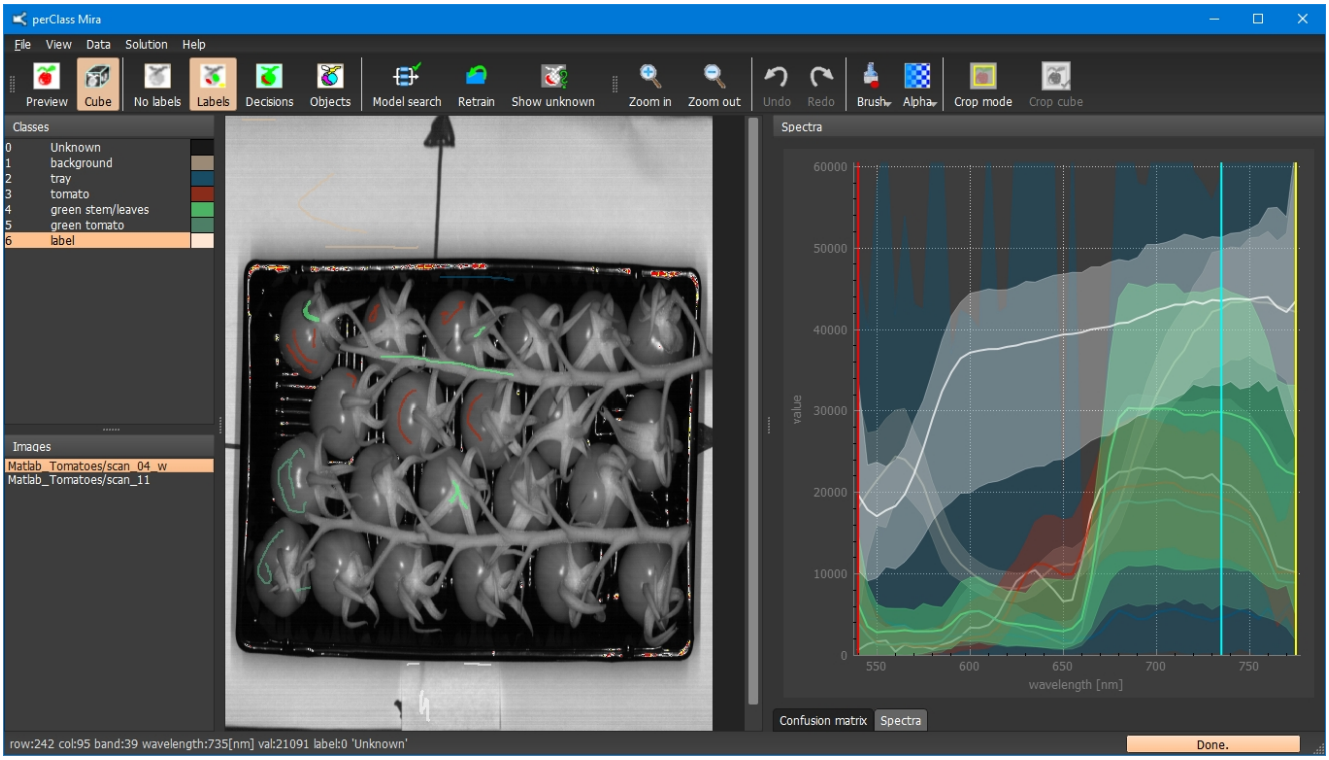
- Cubert Utilities application has perClass Runtime build in. Therefore, any solution exported from perClass Mira can be directly run on new images from Cubert Touch interface. Also, live spectral video can be processed by the perClass Mira classifier in real-time

For details on embedding perClass Runtime library to custom application, see online documentation

Performance optimization

When designing a classification system, one needs to understand its performance. perClass Mira provides a confusion matrix offering a detailed performance brake-down per class.
Confusion matrix highlights how labeled examples of different classes (ground truth) mapps to classifier output (decisions).

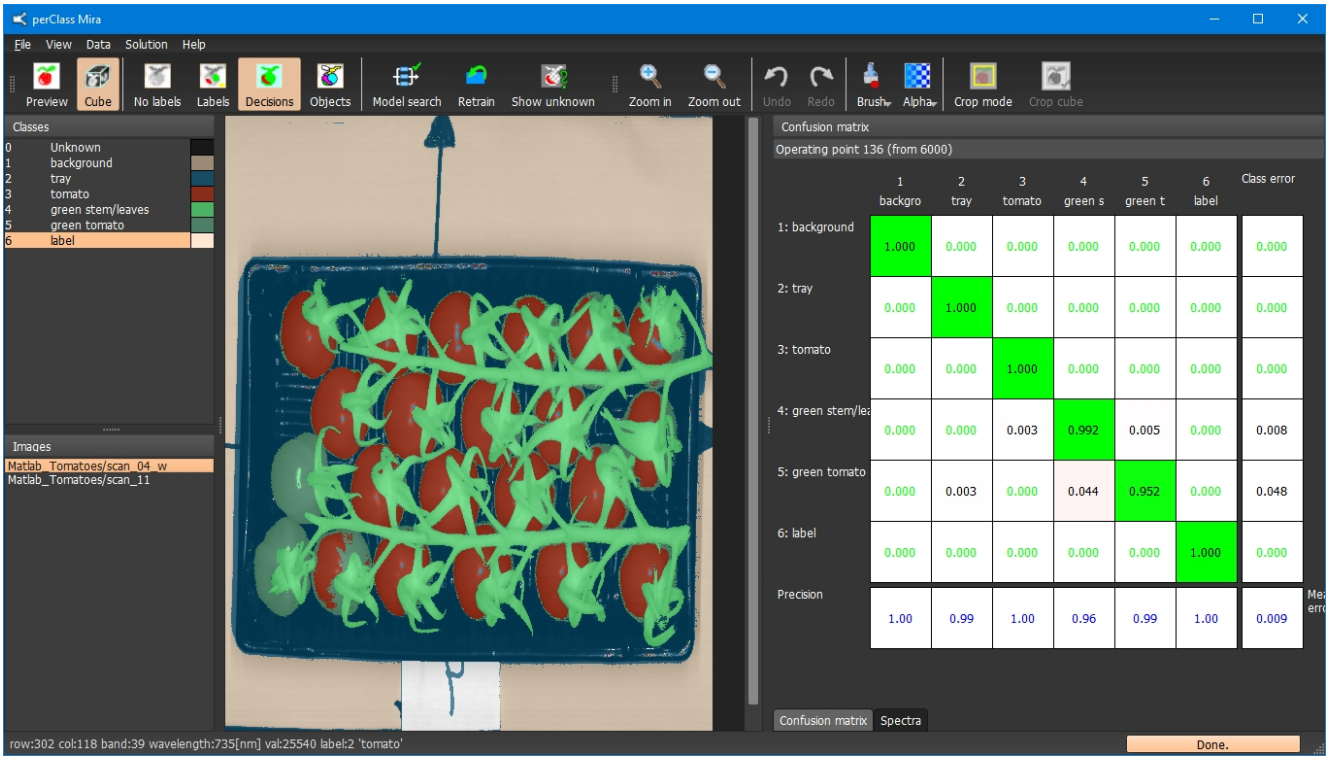
In this example, we have an image with a set of tomatos in a box.



We have labeled six distinct classes.

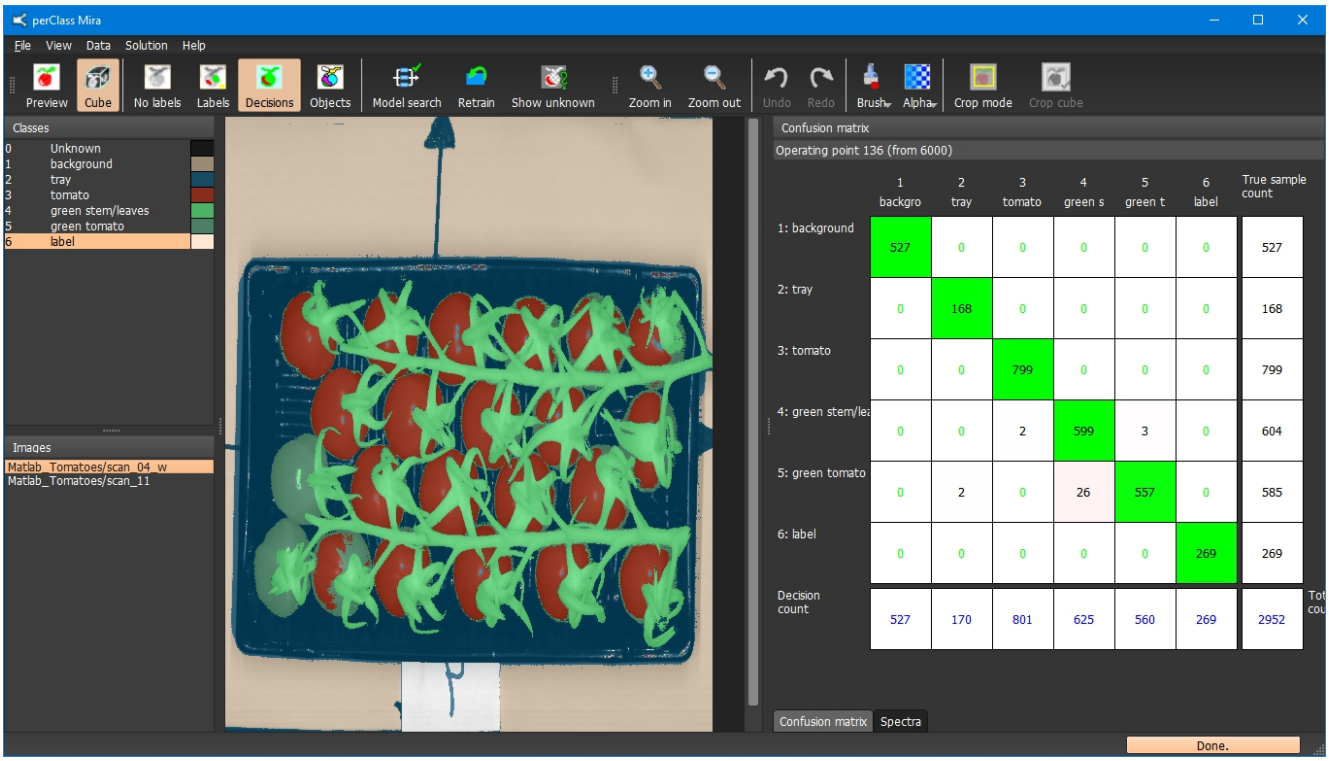
When we build a classification model, we may switch to the *Confusion matrix* docked pannel.

TIP: You may quickly switch to confusion matrix by pressing 'c' key and to spectral plot by pressing 's' key



The confusion matrix shows true class labels in rows and decisions in columns. Therefore, the diagonal represent correctly classifier examples and off-diagonal elements the errors.
By default, confusion matrix is normalized by sum of each row (total number of true class examples) to provide class errors.

To view the absolute, not normalized, values, toggle the *Show normalized matrix* command in the right-click context menu or press Shift+N key.



The right-most column shows total number of class samples, or (on normalized confusion matrix) the per class error rate.
The last row shows number of decisions per class, or (on the normalized confusion matrix), the precisions.
Precision is a total number of correctly classified samples dividied by the total decisions of this class. We wish to

have precision of 1.00 which means that all each decision for this class is correct (pure).

Observing the confusion matrix in our example, we can see that the *green tomato* and *green stem/leaves* classes are not well separable. This probably caused by overlap of these classes in our data.

Although we cannot fully separate them, we may fine-tune the respective error trade-off. **perClass Mira provides fully interactive confusion matrix.**

We may right-click on any field of interest, for example the true *green tomato* vs *green stem/leaves.* The context menu shows a slider which may be used to tune the respective trade-off.
By moving the slider, we may see that the error is lowered and iage decisions change to reflect this new setting. Note that our statistical model does not change, we only tune the importance of specific class in our application. Therefore, the error removed from one confusion matrix field will move to another one. In our case, there will be higher error between true *green stem/leaves* and *green tomato* decision.

When exporting the trained model for execution using perClass Runtime, the current operating point (performance setting) is used. The deployed classifier should, therefore, reflect the situation in the confusion matrix.

Performance constraints

To express application-specific rquirements perClass Mira allows definition of constraints in confusion matrix.

By double-clicking on any field of the matrix, we create respective constrain for the current value. For error (off-diagonal) elements the means that only solutions with error less or equal than current value are allowed.
For accuracies (diagonal) elements it is values higher or equal.

Each field with a constrain shows a small square in its left upper corner. Note, that due to equal sign in constrain definition our current solution does not change by creating a constrain. We only limit a subset of admissible solutions (see the text in the upper part of the confusion matrix widget showing that now 5178 solutions from the total 6000 are valid.)

We may install multiple constraints by double clicking. To remove a constrain, double click again.

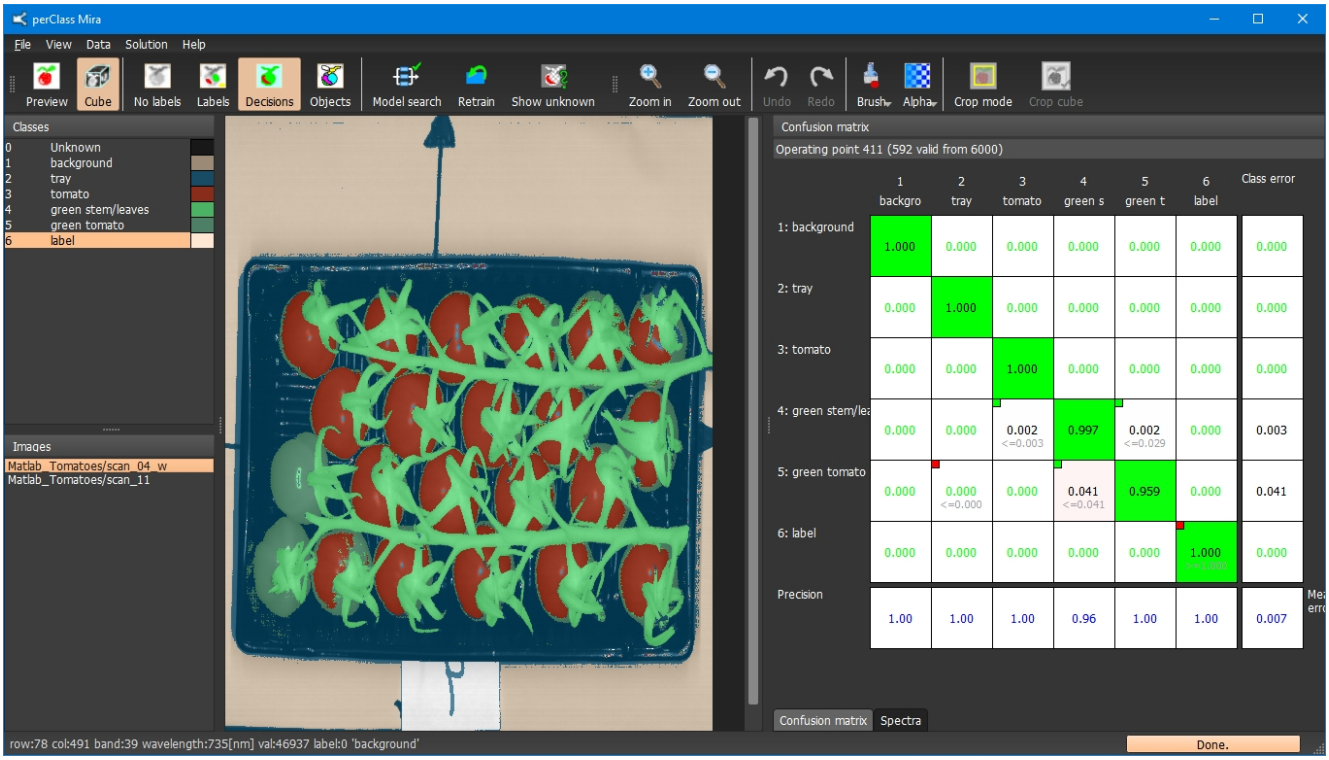
To change any constrain value, we may use Ctrl+mouse wheel on the specific field. This may lead to a new, better, operating point.

Note, that we may not minimze all errors simultaneously. Lowering the error between class 4 (*green stem/leaves*) and decision 5 (*green tomato*) the opposite error (class 5 vs decision 4) will increase. If we also install the constrain on class 5 vs decision 4 we may reach the situation where this constrain cannot be lowered further (by doing so, there would be no more solutions left). To explore fully these situations, constraints may be *disabled* by clicking on the small square in the left-upper corner. Only the constraints with green square are used in performance optimization, not the red square constraints.

Best practice procedure:

- Install constraints of interest by double-clicking
- Tune the constraints' values using Ctrol+mouse wheel
- If the errors cannot be lowered further as desired:
  - either increase value of other constraints (Ctrl+mouse wheel)
  - or disable other constraints and lower the error value of the more important ones

Example of multiple constraints:



Note, that re-training a model does not change the optimization options (operating poionts) only updates the model and selects one of existing points.
When selecting a new model via *Model search*, also new set of operating points (solutions) are created.

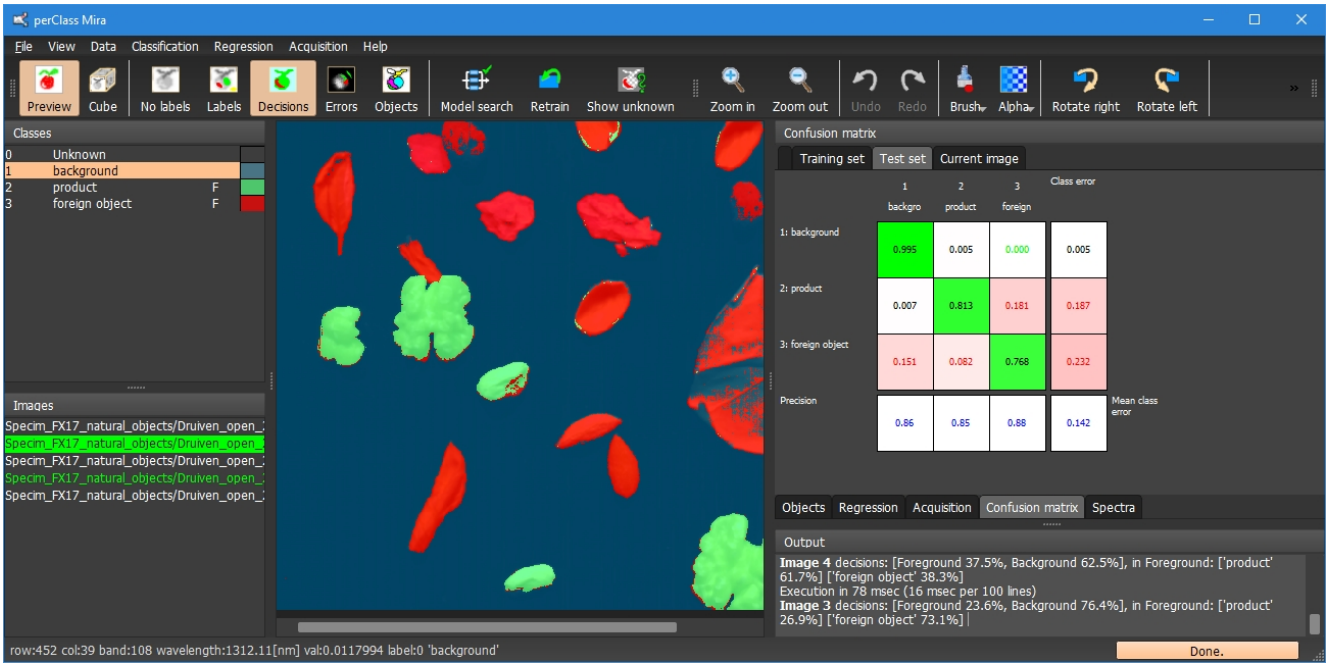
In both cases, there may be no solution that fulfills all enabled constraints. In this situation, all constraints are disabled. You may re-enable some of them and/or update value of others to find a desired solution.

Marking images for testing

Images can be marked for testing using context manu in the image list and *Set image for test* command or by pressing (Ctrl+T)

Images marked for testing are not included in the training set for building pixel classifiers.

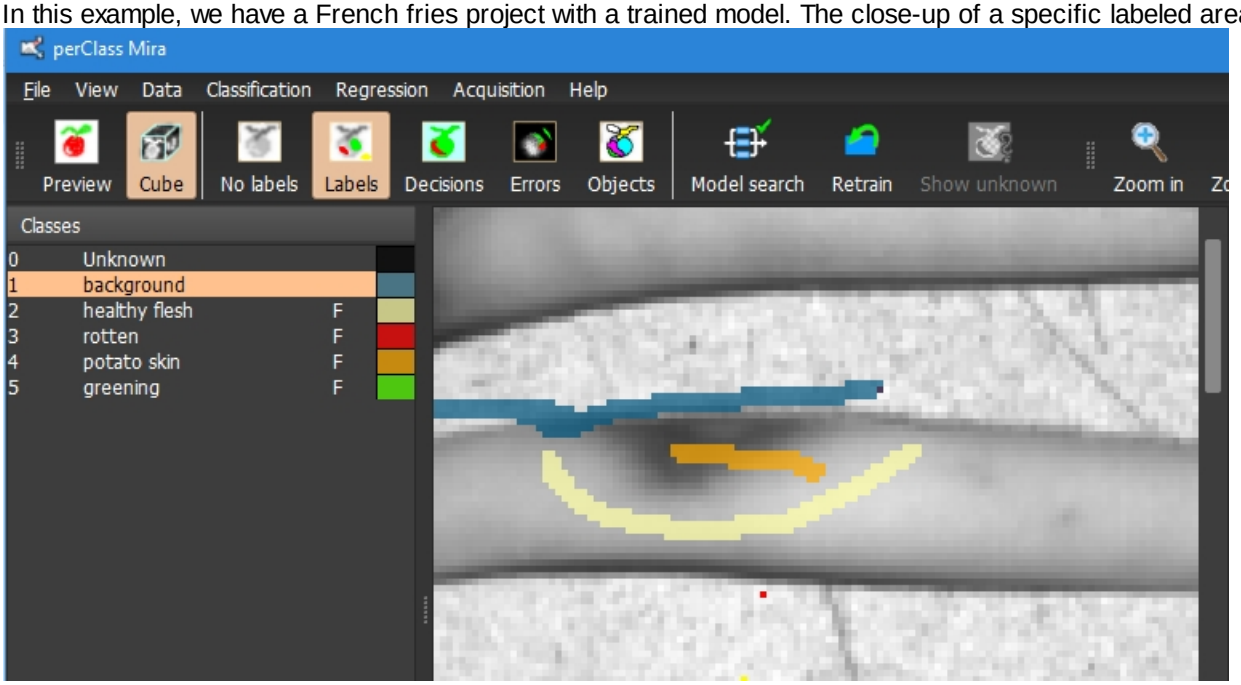
Separate test set confusion matrix is also available with a detailed brake-down of performance on labels in test images.
This enables us to understand how the model performs on unseen data.



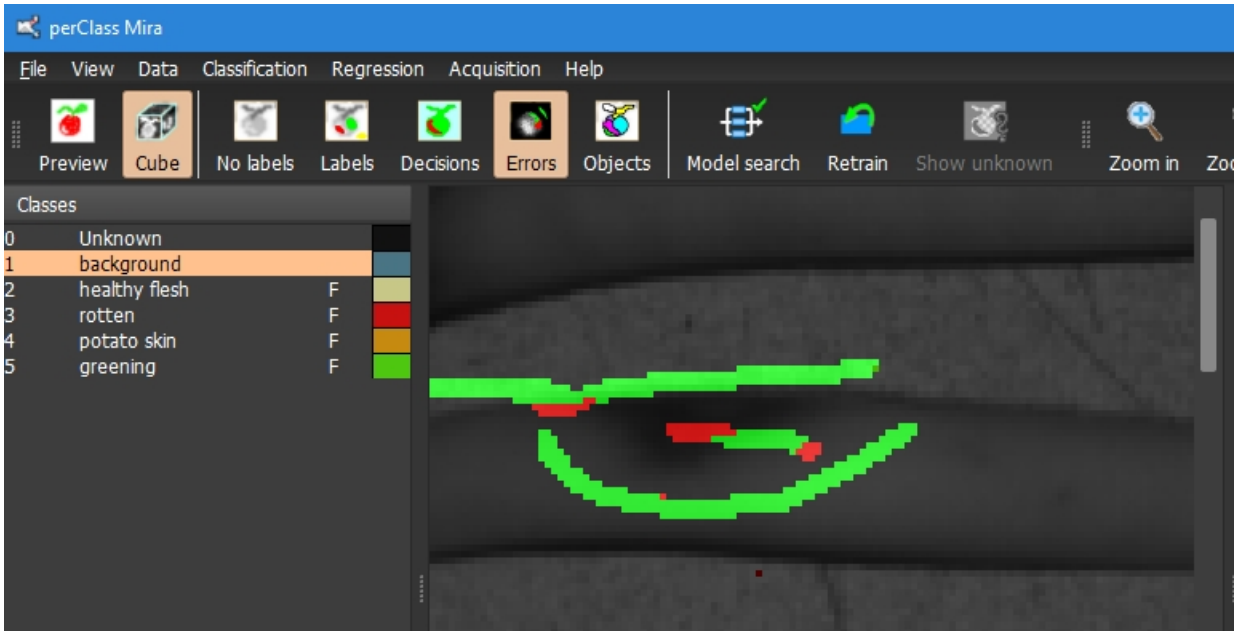
Error visualization

perClass Mira 2.0 brings error visualization. It enables the user to understand where the classiifer fails and facilitates the model improvements.

In this example, we have a French fries project with a trained model. The close-up of a specific labeled area:

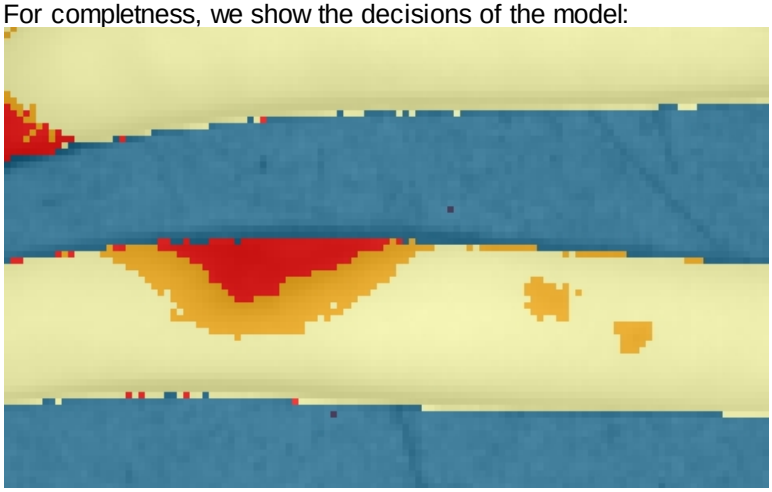


When we select the *Errors* mode in the toolbar (or press E key), the visualization will highlight correct model decisions in the labeled areas in green and incorrect decisions in red:



We can see, that there are three regions where the current model misclassifies the labeled information.

For completness, we show the decisions of the model:



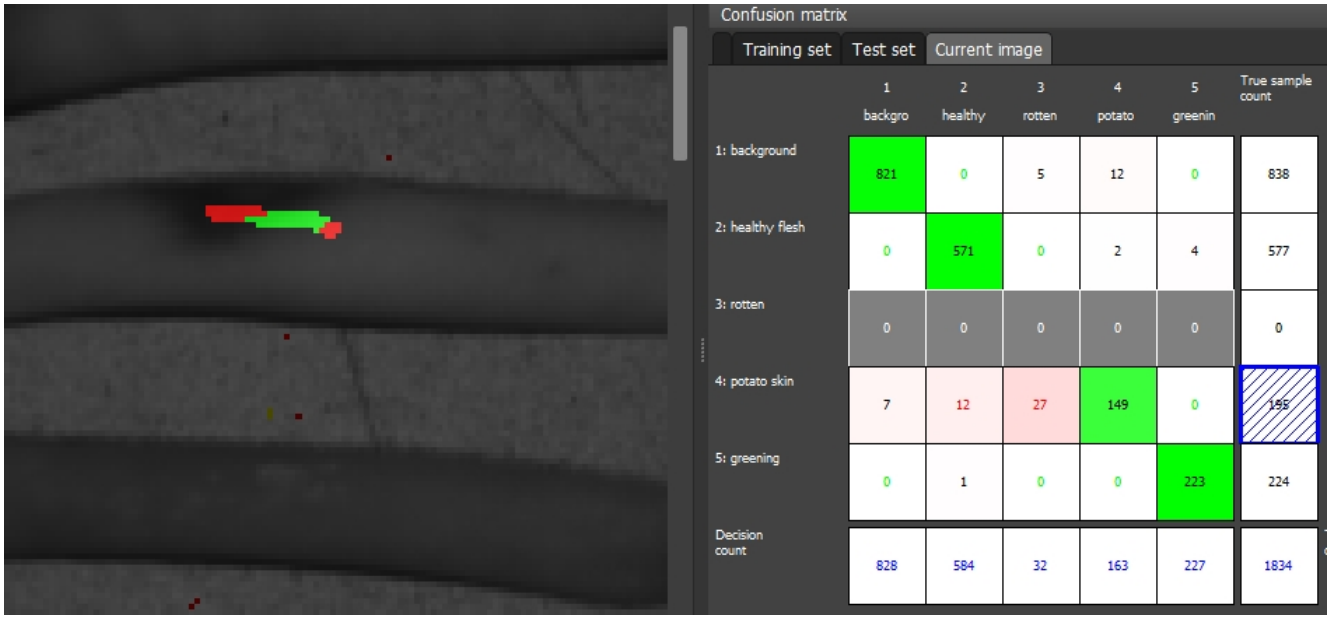
It is clear that the central area is classified as rotten while our middle label stroke assigns it to the *potato skin* class.
The background label stroke also reaches into the potato piece resulting into incorrect classification.

Interactive error visualization in image confusion matrix

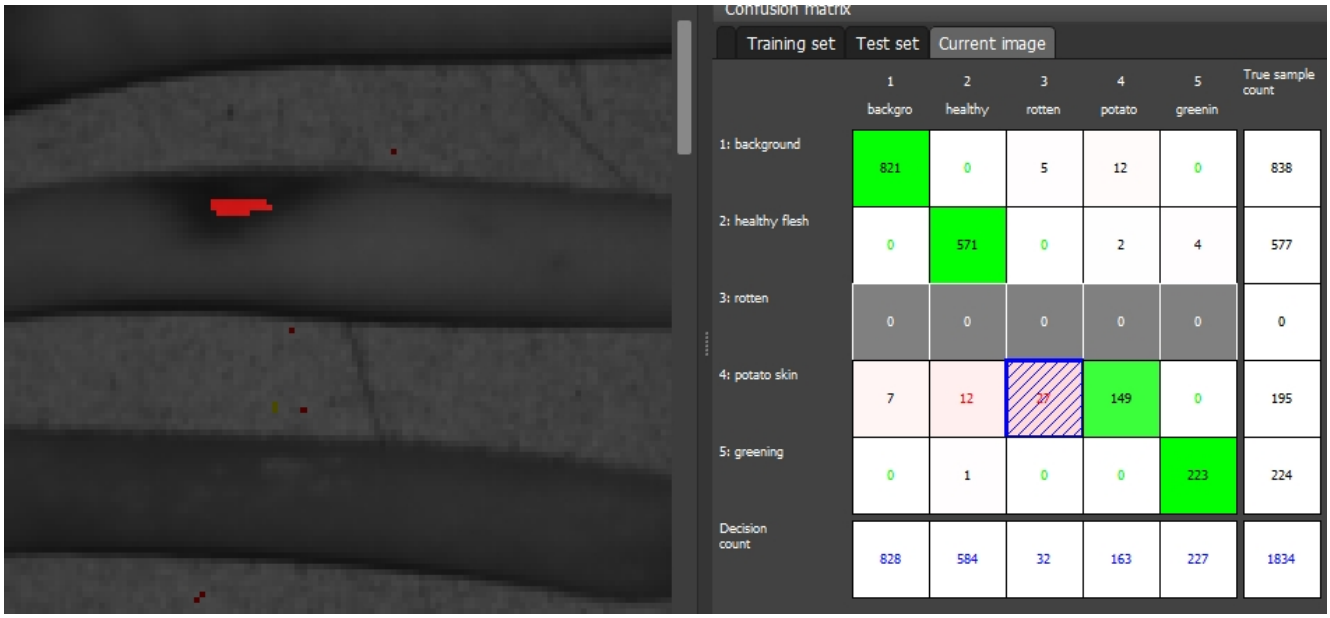
By default, the error mode shows all labeled pixels in an image. In order to gain deeper insight into model performance, it is possible to limit the visualization to specific type of errors in a confusion matrix.

Hovering the mouse over the *image confusion matrix*, we visualize only the corresponding subset of pixels.

Using the French fries example from previous page, we visualize only examples labeled a *potato skin:*



When we move the mouse to the entry *potato skin - rotten* (the value of 27), we can quickly identify the part of the stroke where the current models mislabels skin as rot defect.



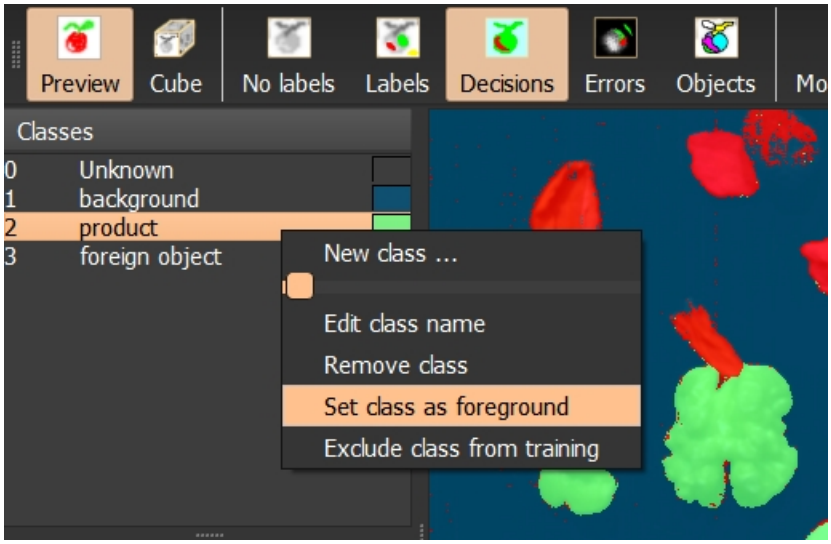
Important: Please note, that the green/red pixel visualization corresponds to the current model. There may be several reasons why we observe incorrect labeling in the left part of the stroke:

1. Our labeling may not be precise and the error suggests which part is incorrectly labeled. In this most common case, removing the *potato skin* on the left and the right end of the stroke would be advisable to improve the trainng labels. Leaving incorrect labels confuses machine learning models.
2. If we observe model errors when inspecting a test image (unused in model building), it may be that the areas highlighted as errors are, in fact, valid examples not represented in the training set. In such a case, it is advisable to include similar material exeamples to our training set.

Object segmentation

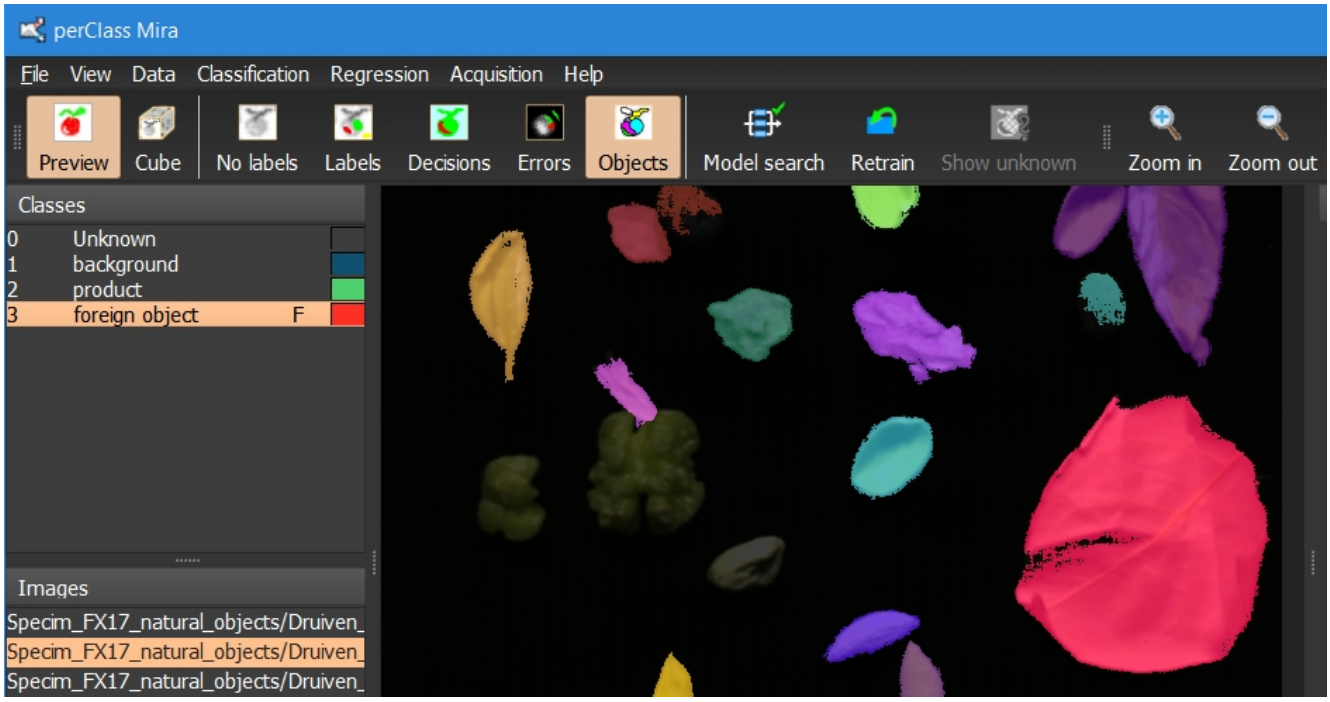
Apart of pixel classification perClass Mira provides object segmentation. In order to segment objects (connected components), one needs to set one or more classes as "foreground":

1. In the class list, use right click to open the context menu on the desired class and select *Set class as foreground* (or press F key)



2. On the toolbar, press the *Objects* button (or press O key)

Selected class/classes will be segmented out and each connected component will be visualized with a randomly selected color.

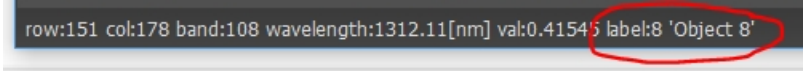


You may notice that small objects got removed from the segmentation result. The minimum size of accepted objects is adjustable in the Objects pannel or via *Classification* / *Set minimum objects size* menu.

Object labels and object decisions

By default, object segmentation shows object labels. This means, that each connected component is assigned into a unique category.

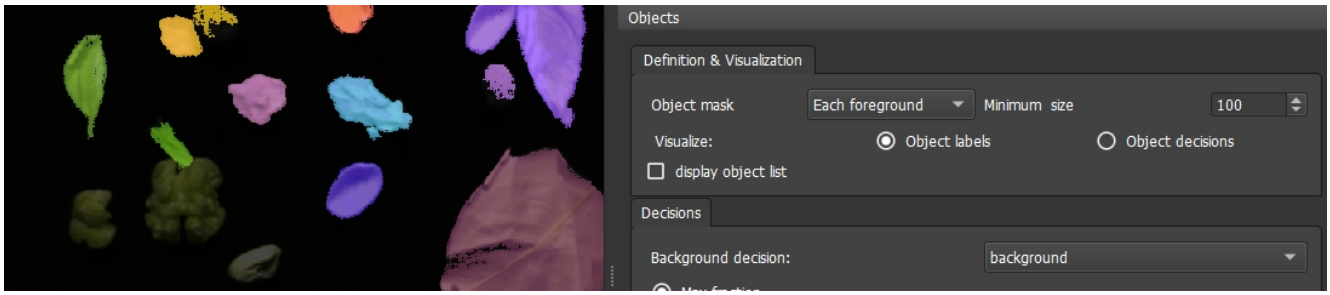
When hovering the mouse over the objects, the object label is displayed on the status bar



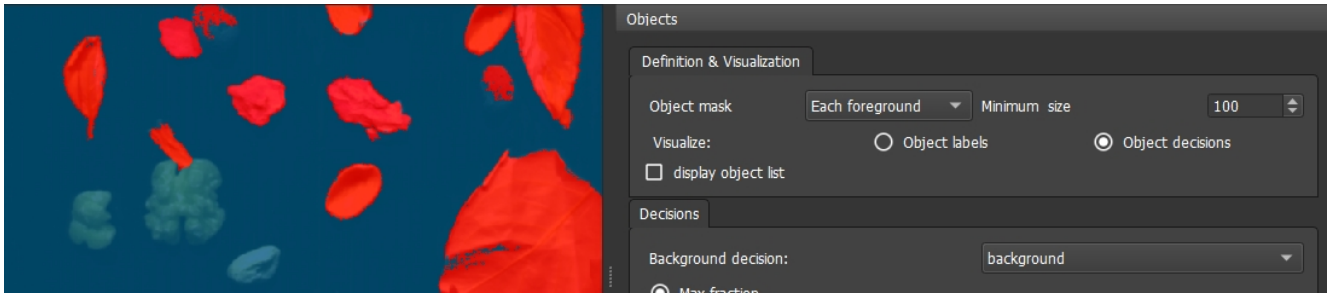
The object pannel provides more options on how to deal with detected objects.

Instead of object labels, we may visualize **object decisions**. It means, that each object is assigned into one of the user-defined classes.

Object labels output:



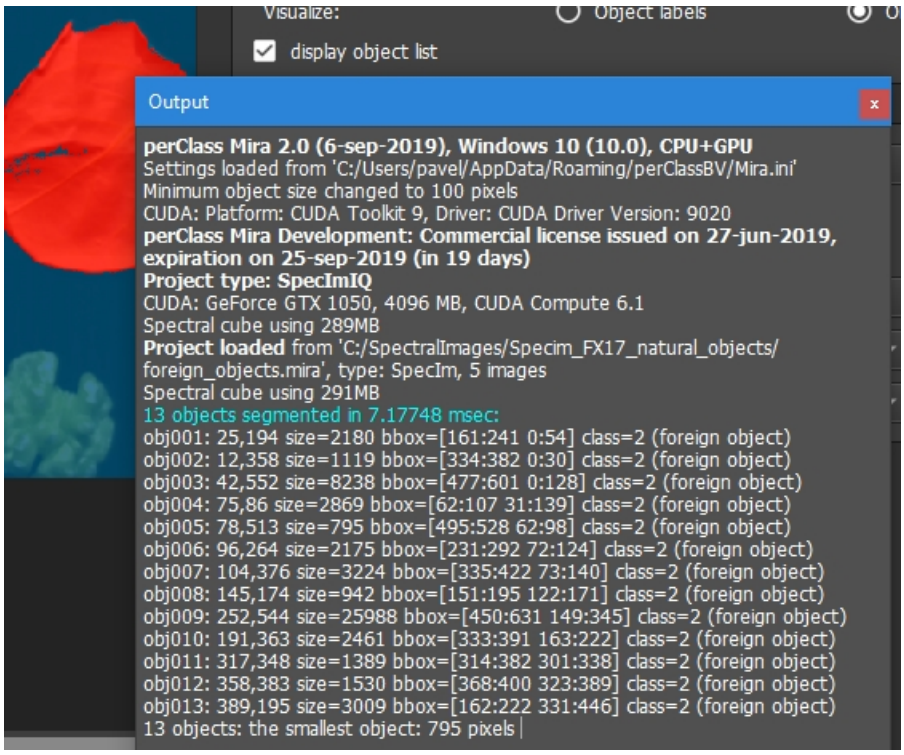
Object decisions output (when only the *foreign object* class is considered as foreground):



Note, that multiple classes may be considered as foreground. In such a case, perClass Mira can handle touching objects of different classes.

Displaying object list

Object list can be displayed in the output window by enabling the *Display object list* check box.



The object list summrizes:

- object label
- center of gravity (row, column)
- size in pixels

- bounding box (columns, rows)
- class

This information can be delivered by perClass Mira Runtime processing a live stream of spectral data.

Object modes

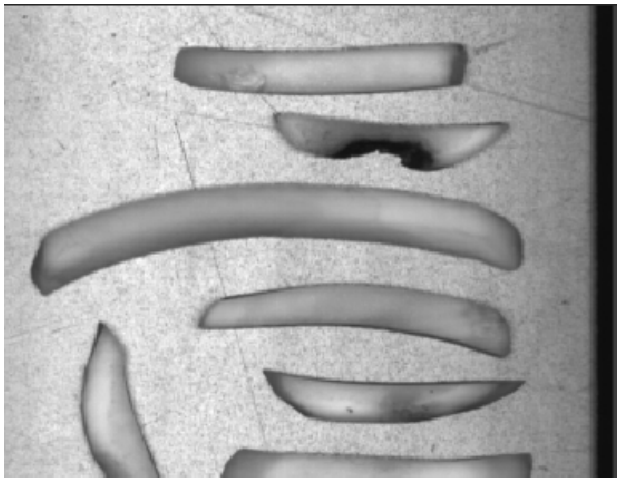
perClass Mira offers two object-definition modes, depending on the mask construction process:

1. **Object mask: Each foreground**. Each object is composed of a single class (or material). Example: Object is either a nut or a shell or a stone. This approach results in **object detection useful in sorting applications**.
2. **Object mask: All foregrounds**. An object is composed of multiple materials (classes). For example, potato chips sorting needs to remove potato pieces (objects) that contain any rotten or green part. In general, this approach is fitting to **object classification use-cases**

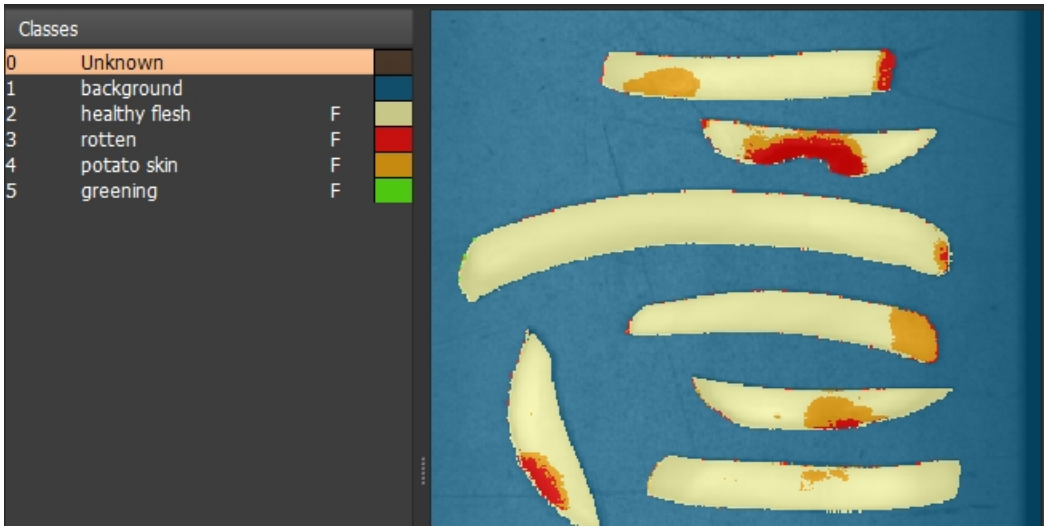
Example of both modes on a French fries data set

Pieces of French fries are to be sorted to remove objects containing defects. Within each piece of a potato, there can be healthy potato flesh or skin (both OK for the consumer) but also rot or greening defects than must be avoided.

Single band image



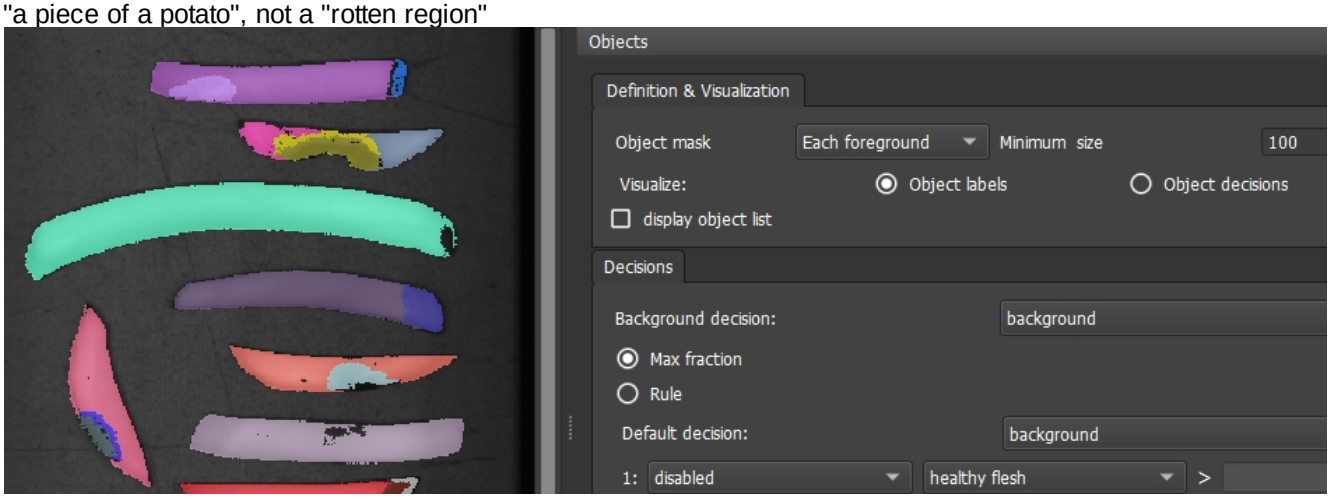
Decisions of a classifier



Object definition with "mask by each foreground" mode

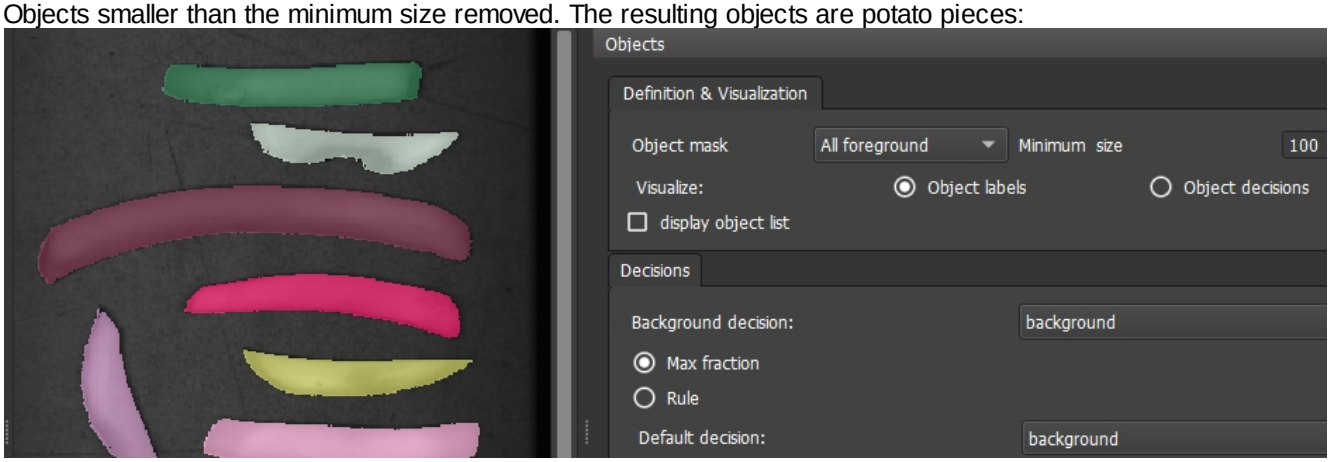
Using the "Each foreground" setting, each foreground class (healthy, skin, rot and green) is handled separately.

Small objects are removed. The resulting object labels will show a colorful patchwork of individual regions.
This approach does not help us to build a sorting problem as "an object" in the French fries sorting application is "a piece of a potato", not a "rotten region"



Object definition with "mask by all foreground" mode

Using the "All foreground" approach, a union of all foreground classes is considered as a mask for segmentation. Objects smaller than the minimum size removed. The resulting objects are potato pieces:



Object classification

Object classification use-case based on "All foreground" masking allows us to define specific way how to decide class on individual objects. The object pannel provides two approaches:

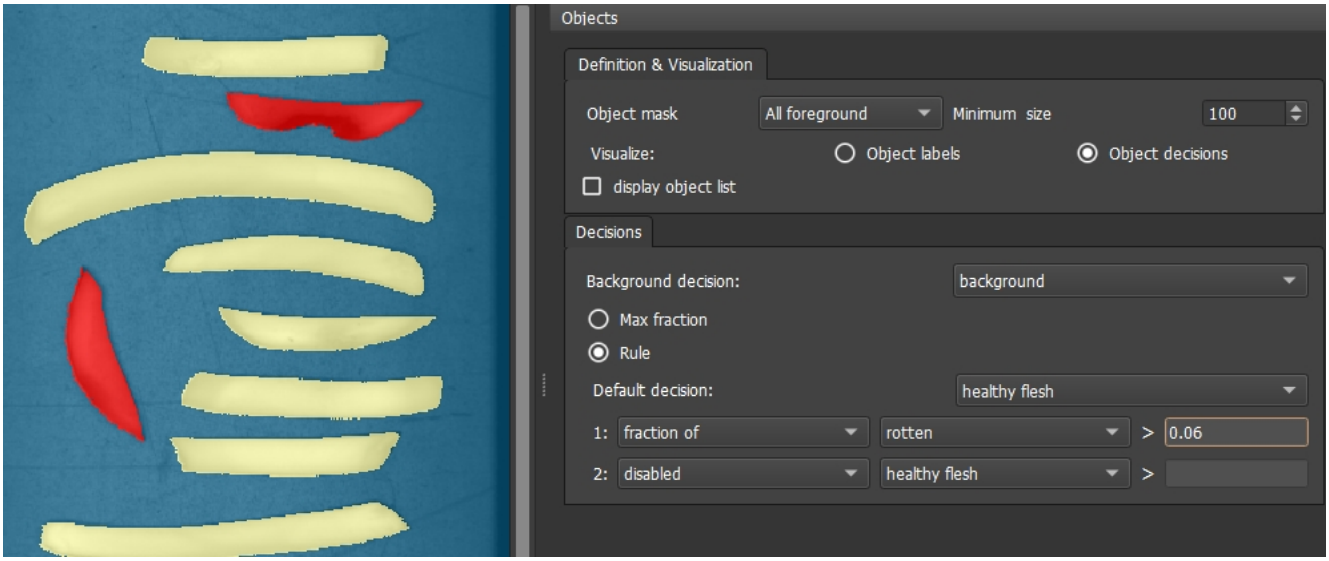
1. Maximum fraction (majority voting)
2. Rule-based classification

Maximum fraction object classification

Object is classified to a foreground class with the highest number of pixels within an object. Background is assigned to a class defined by the "*Background class"* combo box.
This option is useful in situations where we distinguish several spectrally different classes but the entire object is either one or the other. For example, in a fruit sorting application, we may train a "*firm fruit*" and *hard fruit* classes. The maximum fraction object classifier assigns the fruit piece to the most frequent class.

Rule-based object classification

In some sorting use-cases, the presence of even small amount or fraction of a specific class mandates rejection of an object. For example, more than 6% or a rotten potato may result in a piece rejection as defective. Object pannel allows us to define rules based on size (in pixels) or fraction (0..1) of an object.
The default decision is also available. It is applicable when no other rules trigger on a given object.



Regression

perClass Mira 2.0 brings per-object regression modelling. Individual objects can be annotated with numerical meta-data. A regression model is build that can be applied to a new object.

Example applications of the per-object regression:

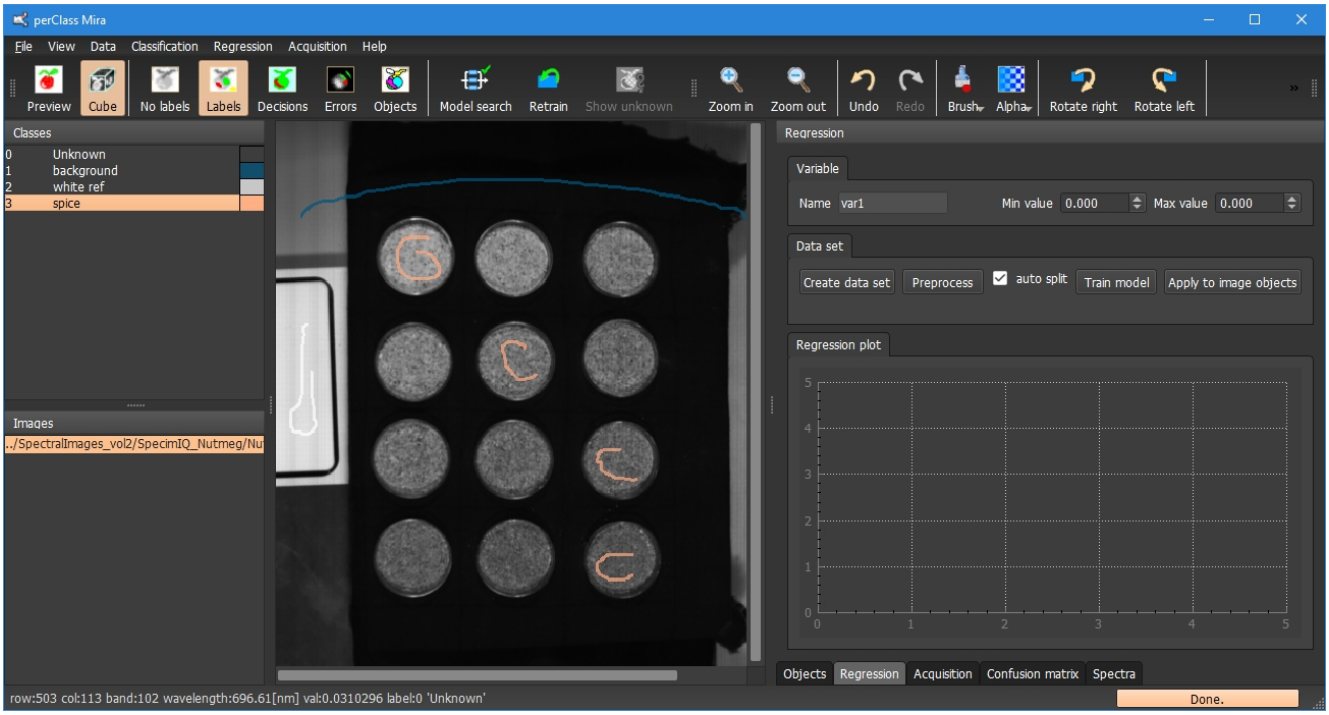
- estimate brix (sugar) content in a tomato
- estimate a dry-matter content of a plant leaf
- estimate fraction of true spice in a adulterated spice mixture

Regression example (spice adulteration)

1. We build a classification model that identifies area of interest for regression
2. We define the class of interest as foreground so that objects can be segmented out
3. We annotate number of objects with known ground-truth values for regression
4. We build a regression model
5. We can apply the solution to a new hyperspectral scan (objects get identified and for each a value is estimated)

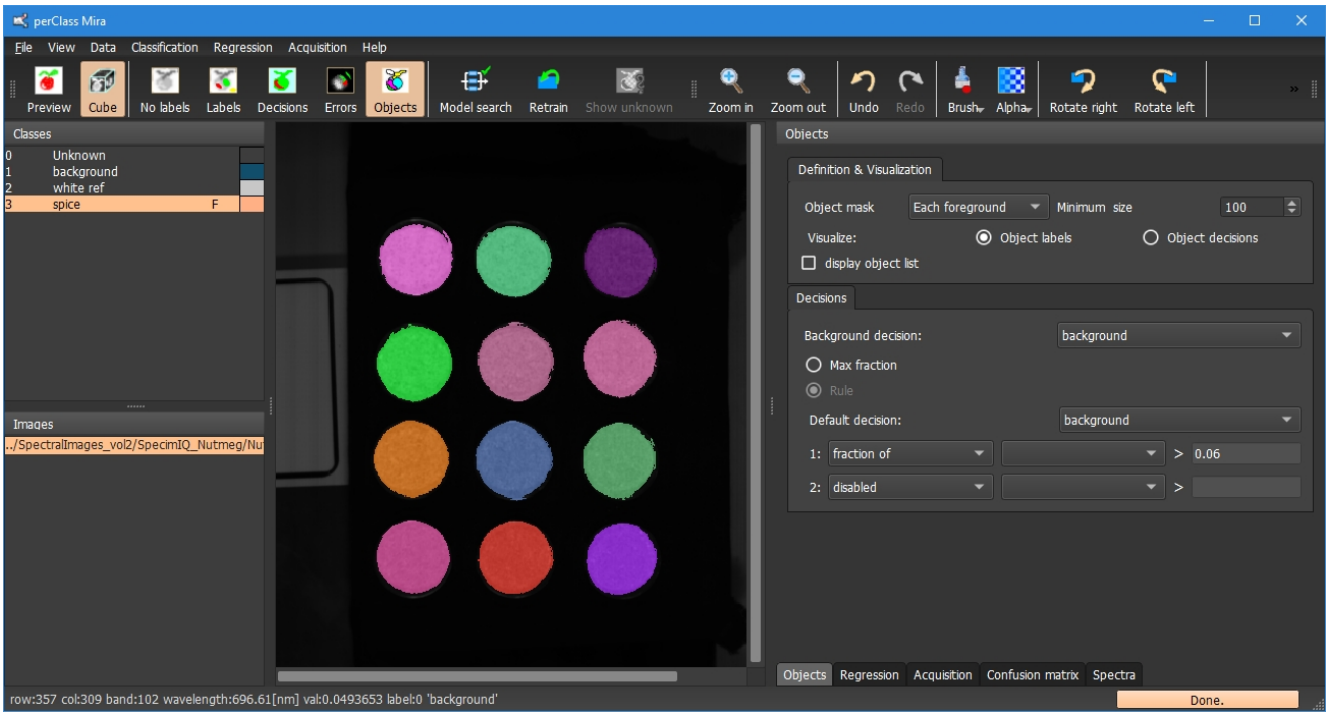
Step 1: Pixel classifier

Step 1: We build a pixel classifier separating objects (boxes with spice) from the background and white reference



Step 2: Object segmentation

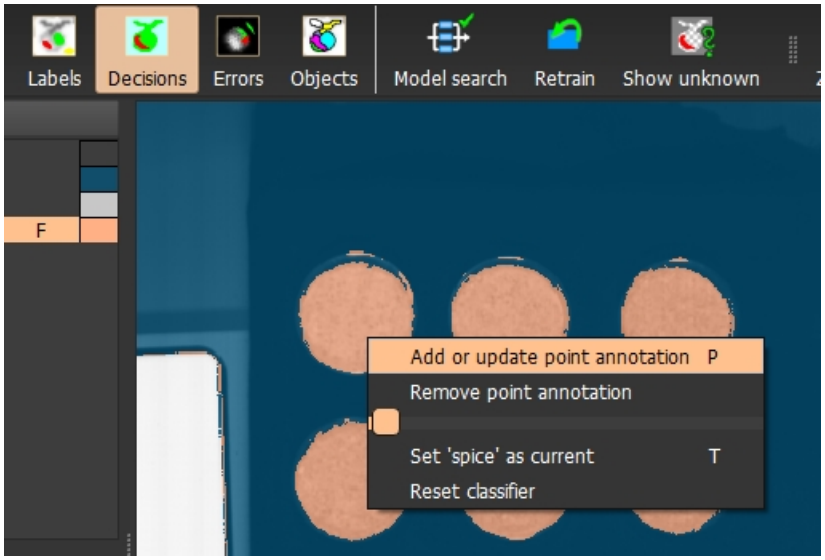
Step 2: We define the class of interest as foreground so that objects can be segmented out



Step 3: Object point annotation

Step 3: We annotate number of objects with known ground-truth values for regression

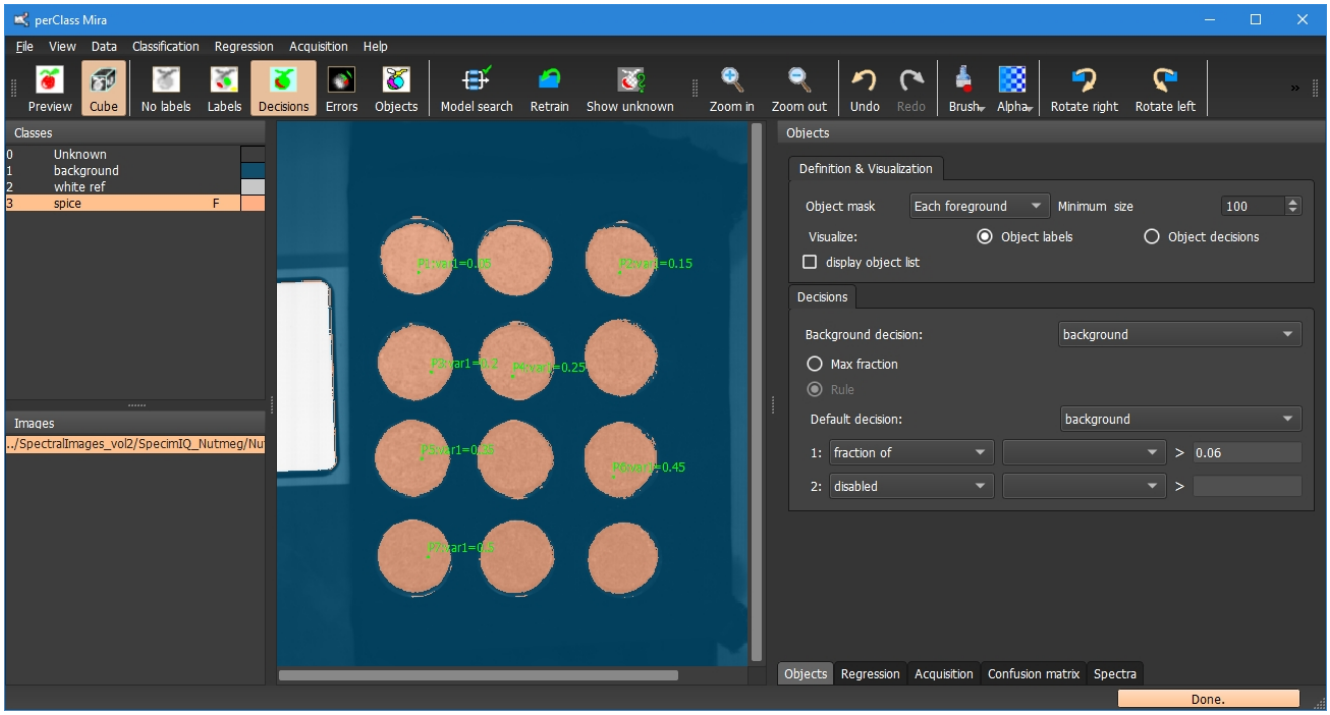
Now we need to place point annotations and assign numerical values to each point. This annotation captures the truth typically established by means of external or destructive testing.
Because the regression modelling is performed on the object level, we need sufficient number of annotated objects. This will depend on the project. The best guidance how many objects are needed is by building the regression model and comparing its performance on the training and test subset of objects. The training/test splitting is performed automatically by perClass Mira.



Each point will represent the entire object (conencted component of the single foreground class). Therefore, only one point should be placed per object.

When annotating objects, it is a good practice to leave some without annotation. We will be able to use them as indepdent blind test set (only for model application).

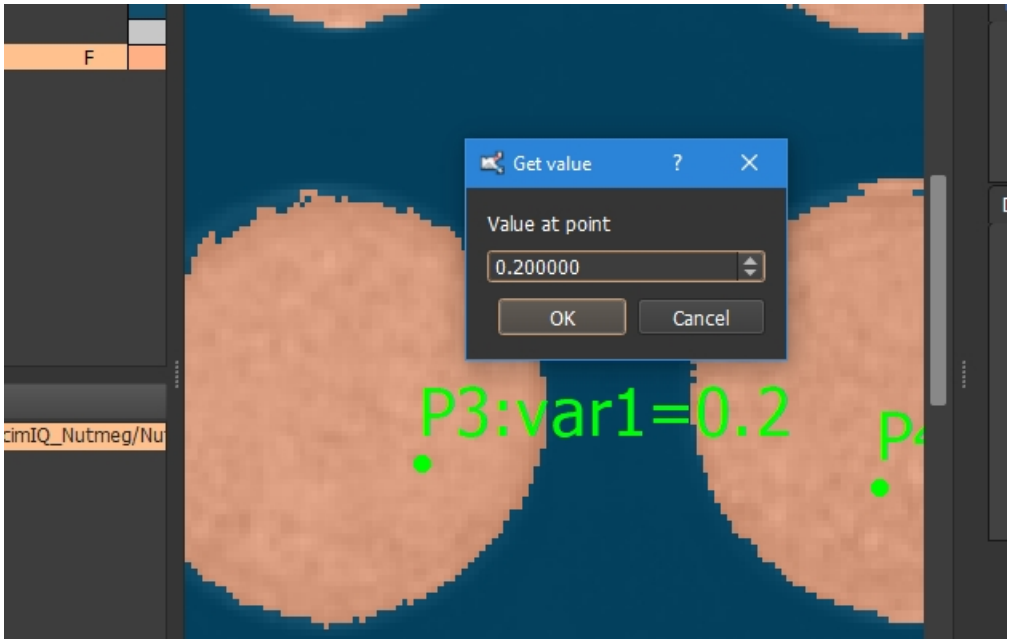
Example where we annotated multiple objects in our scan:



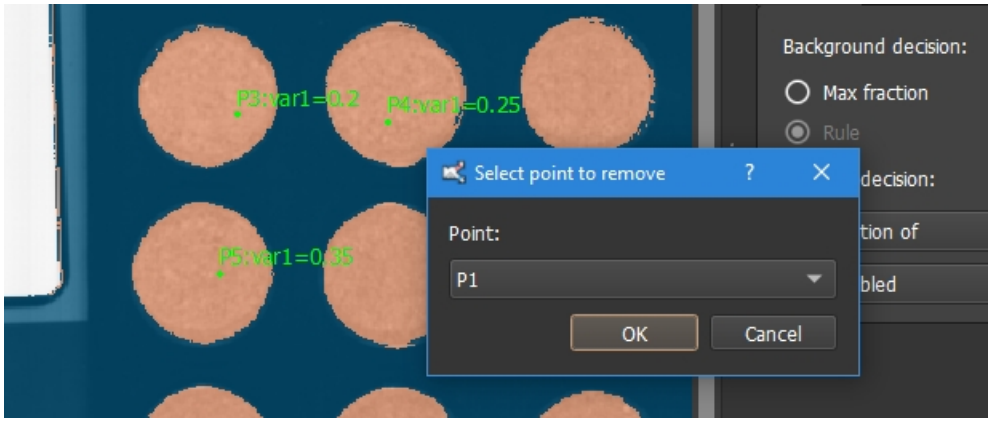
Naturally, we may annotate points in mutiple scans.

Step 3a: Editing and removing the points

Point annotation can be edited by placing the mouse pointer precisely on the point annotation and using the *Add or update point annotation* command.
The edit dialog will appear with the current value which can be changed and saved back.

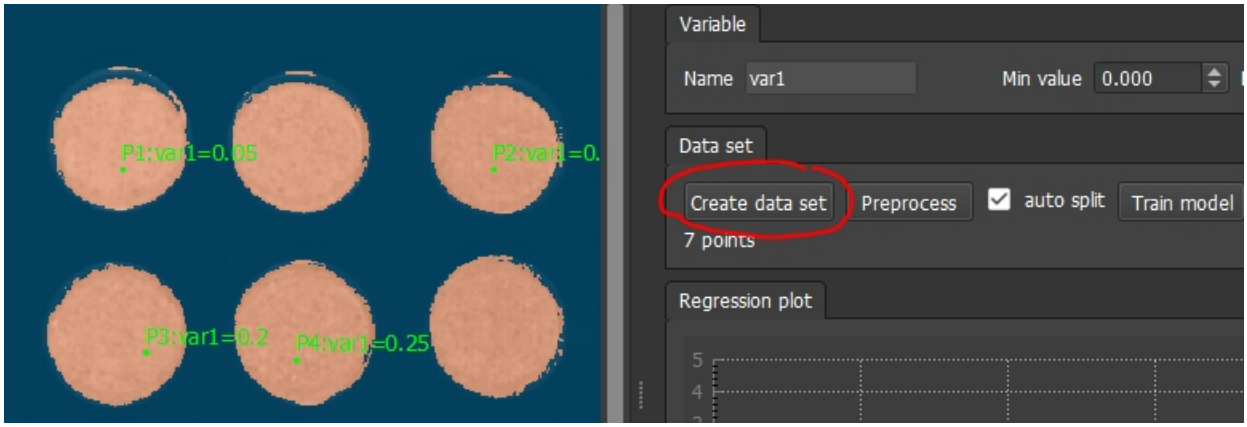


To delete points use *Rmove point annotation* command. A dialog will appear with a list of points in the current image.



Step 4: Build regression model

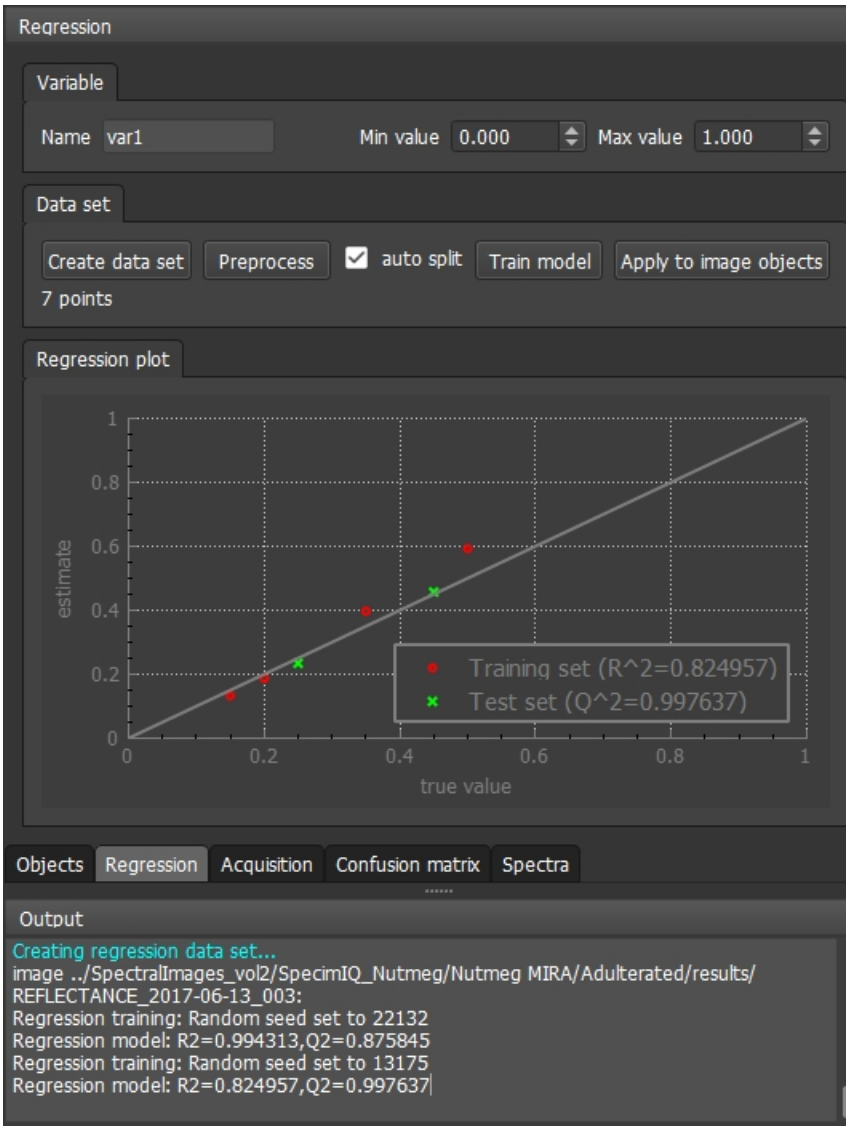
Before we can build the regression model, we need to construct the regression data set. This is achieved using the *Create dataset* button in Regression pannel:



Data set construction may take some time because all scans with point annotations need to be loaded and processed.

When the data set is constructed, we may click *Train model.* The annotated points are split into training and test subsets, ther trainng part is used to build a regression model. The model is applied on both subsets and the results are shown in the plot in the regression pannel:

The plot shows the true value on the horizontal and the estimated value on the vertical axis. The red points correspond to the training objects and the green to the test objects.
Note that in perClass Mira 2.0, all point annotations are used to build a regression data set and splitting into training/test subset is random. It is not affected by setting test flag on the image level.

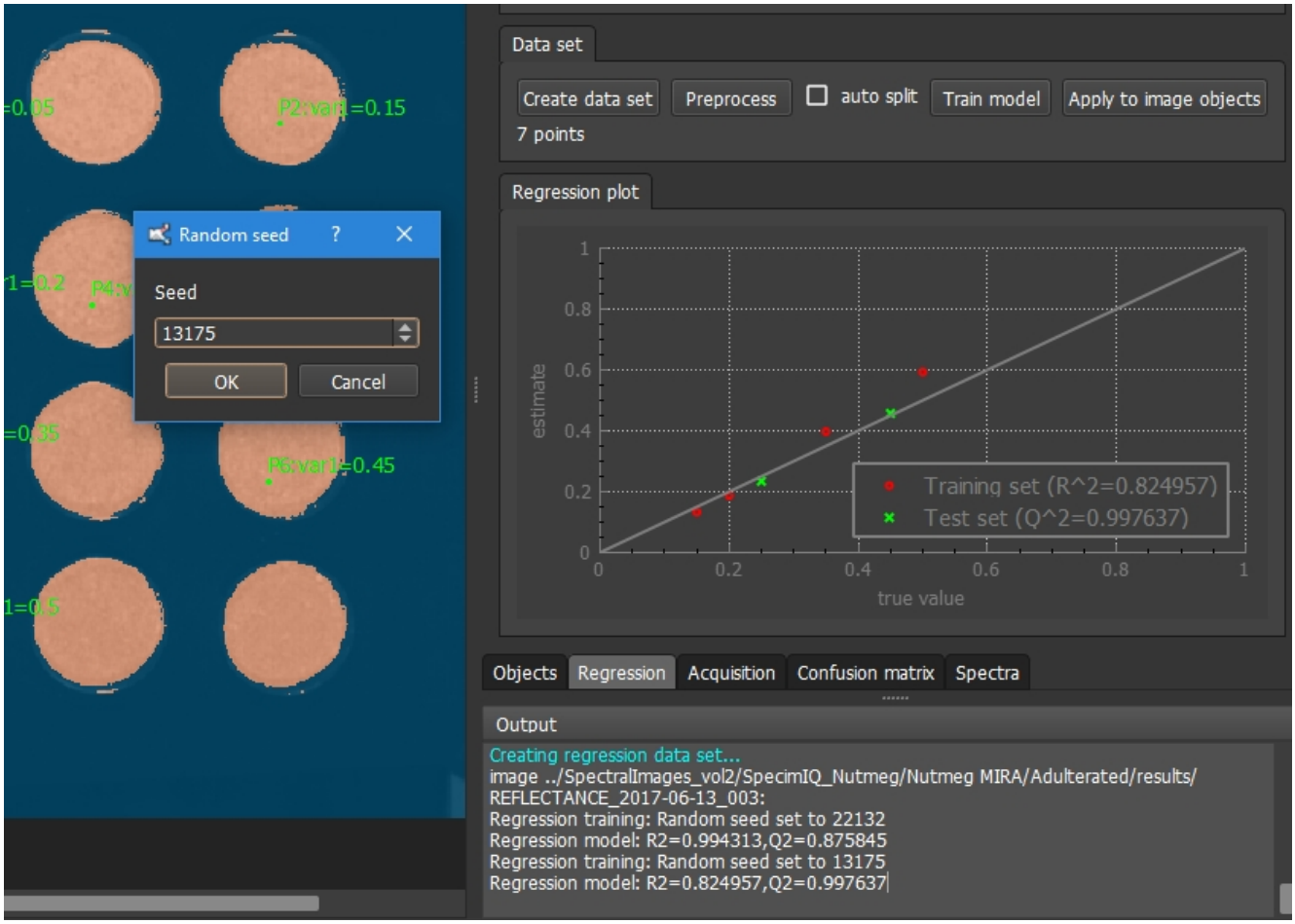


The regression plot lists also R^2 and Q^2 statistics that help us to understand regression performance. The R^2 statistics reflects performance on the training subset and the Q^2 on the test subset. The ideal value of both is 1.0. Well trained regresion model should shown both statistics close to 1.0 which means that model is able to explain training examples but also exhibits high prediction capability on unsee test examples.

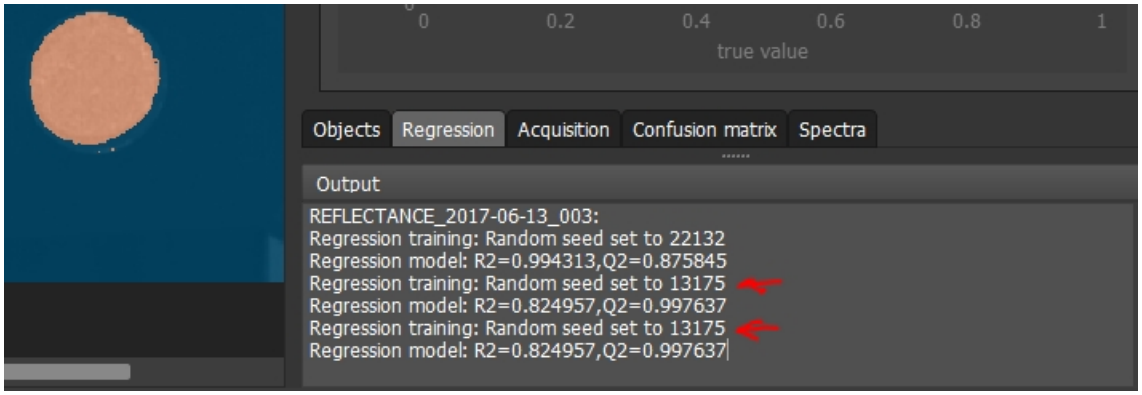
If we click *Train model* again, a new training/test split is performed and a new model is build. Each time a specific random seed is used to initialize the process. The random seed value is displayed in the output window. It allows us to recreate a given experiment.

For example, if we wish to repeat the experiment that yields 0.99 Q^2 (test performance), we can use the seed 13175 on our specific regression data set.

To specify a desired random seed value, we need to disable the *Auto split* check box in the Regression pannel. With the disabled *Auto split* mechanism, pressing *Train model* will present us with a dialog to fill the random seed value:



The random seed defines the data split. Therefore, we will end up with an identical regression model:



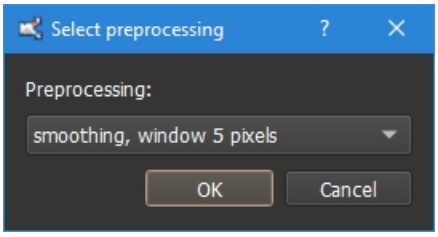
Step 4a: Adjust preprocessing

It is possible to control what pre-processing of the data is applied prior to building the regression model.

By default, no pre-processing is used.

Options include spectral smoothing with varying window sizes and also the first and the second derivatives of the spectra.

The selected pre-processing method is indicated in the Output pannel.



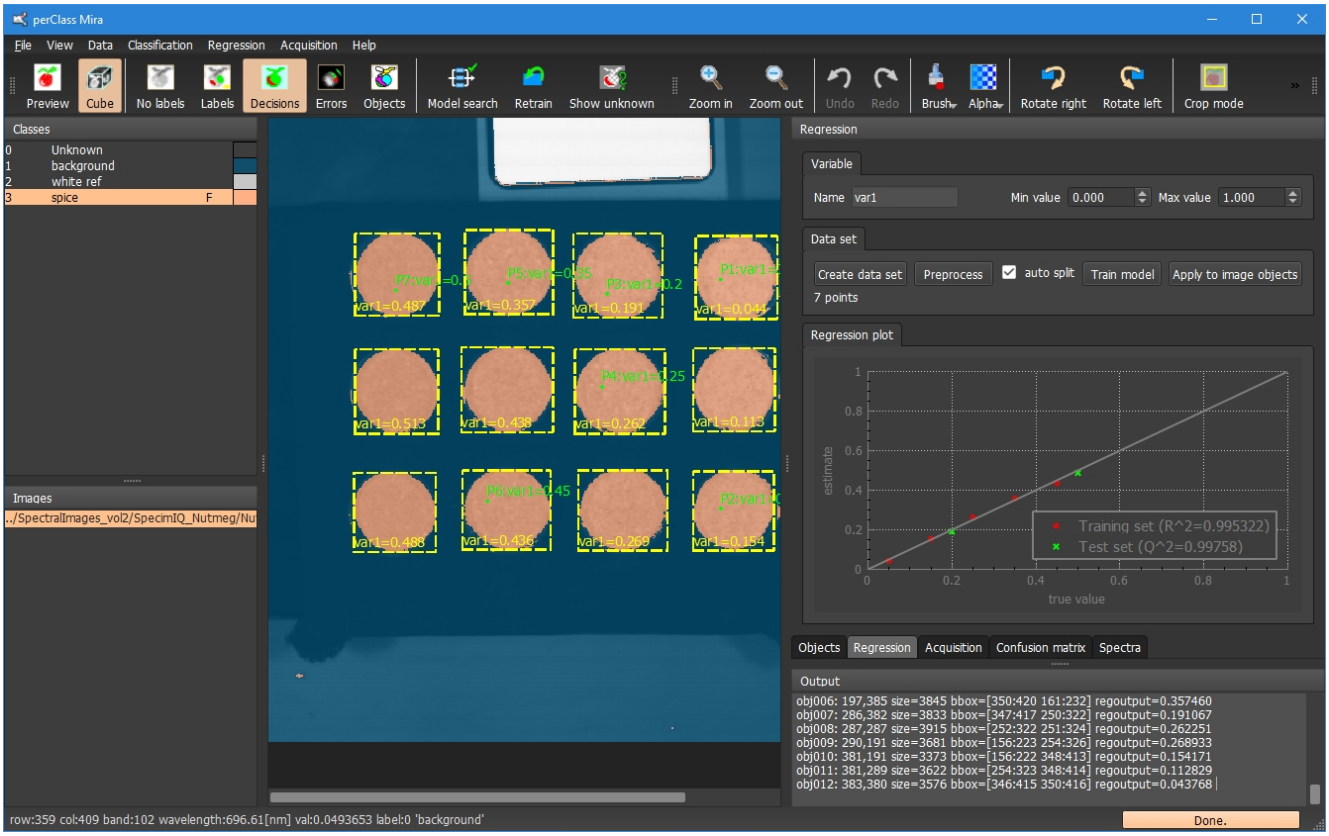
Step 5: Apply regression solution to a new image

Regression solution can be applied to any spectral image.

The pixel classifier is executed, objects segmented out. For each object the trained regression model is applied.

Each object is enclosed with a yellow bounding box and the estimated value is displayed.
The Output window also lists all objects found in an image together with their centers of gravity, sizes, bounding boxes and estimated values.

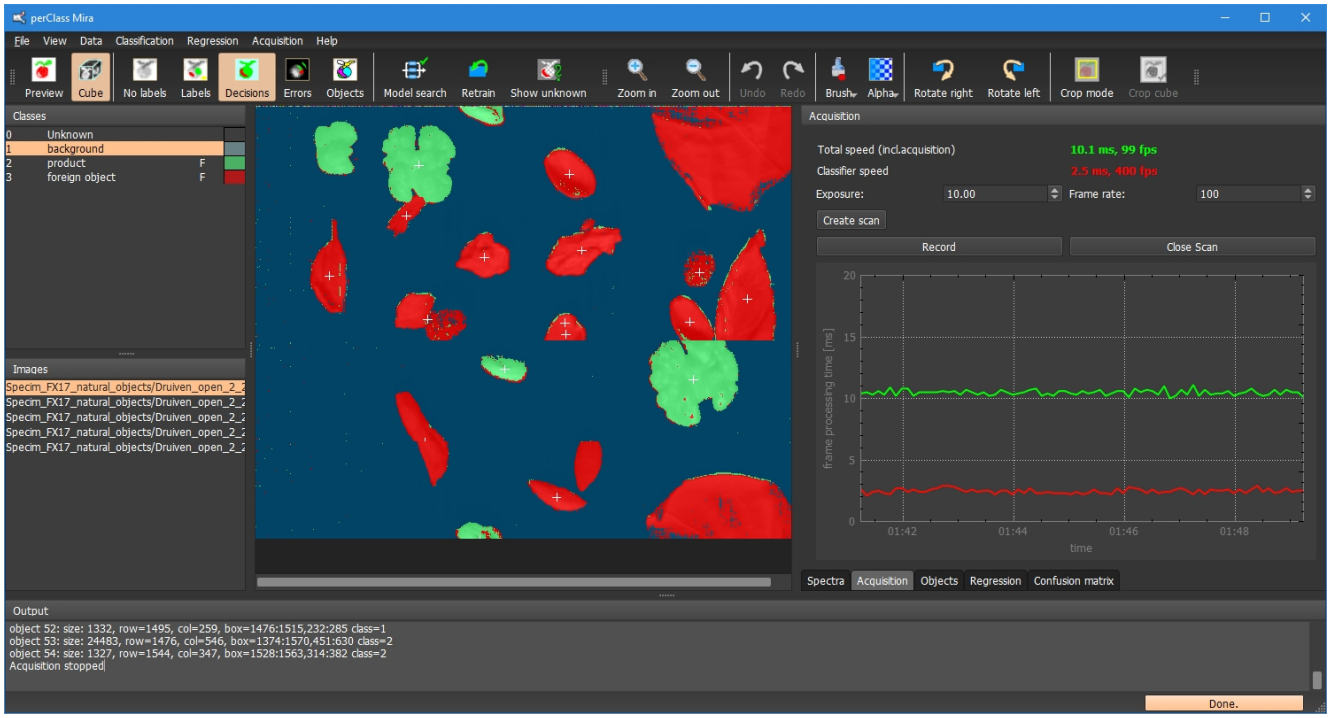
Note that as in perClass Mira 2.0, the regression model is not stored in the project file and needs to be re-trained when opening a project



Live data acquisition

Live acquisition functionality allows users with Specim FX cameras to apply models directly on data stream from a camera.

Please not the live acquisition feature is still experimental in perClass 2.0 release. Use for demo purposes, not production.



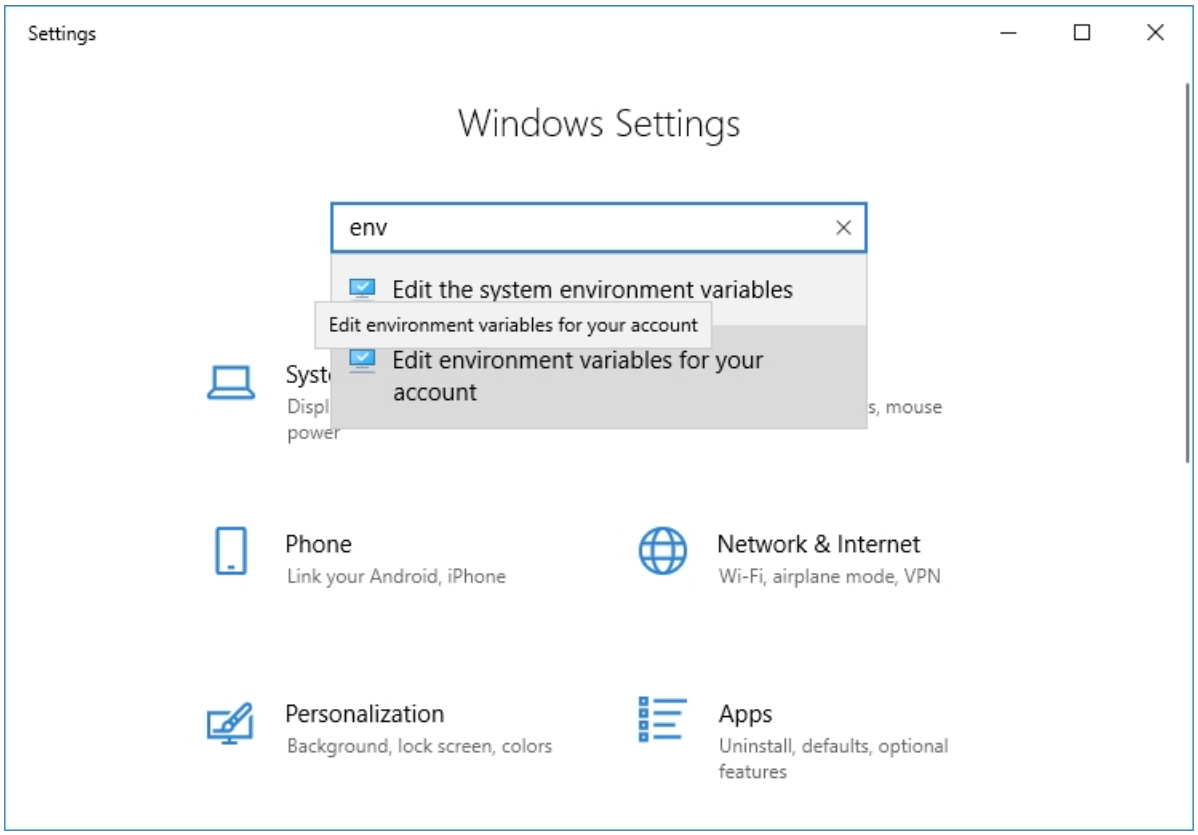
Installation of Specsensor SDK

Specim Specsensor SDK needs to be installed on the system running perClass Mira.

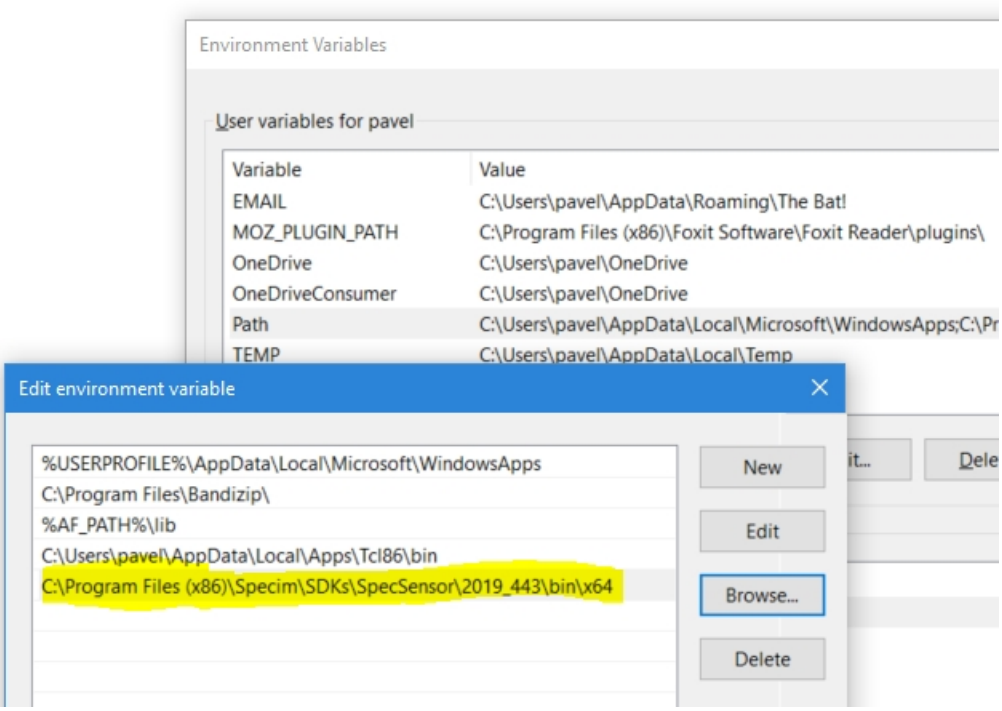
The Specsensor version needs to be at least 2018_415 (if you still use 2017 SDK release, please update). In general, it is adviceable to update to the latest Specsensor release.

After Specsensor installation, make sure that the system Path environment variable contains the path to Specsensor

1. Open System Settings dialog



2. Open *"Edit environmental variables for your account"* dialog

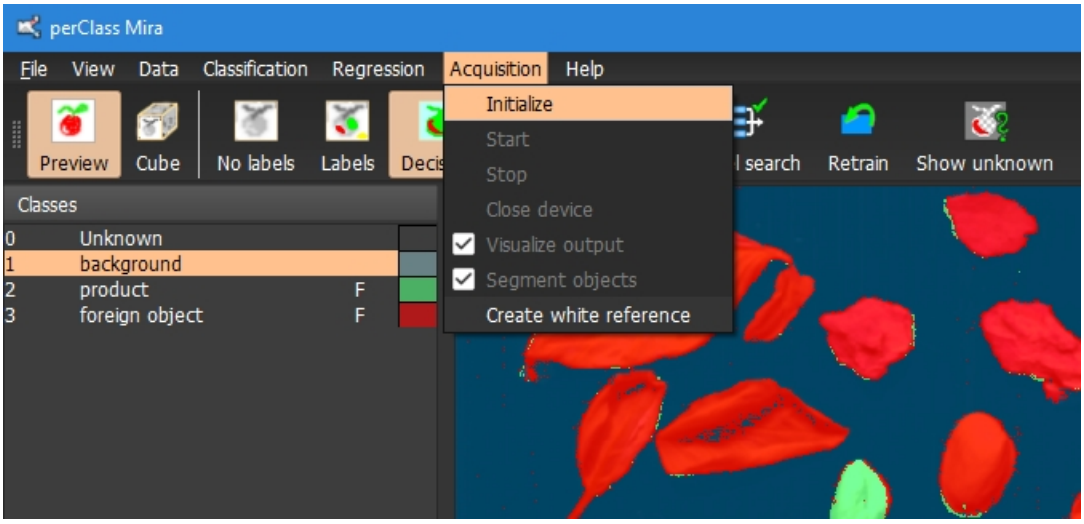


3. Make sure the path to points to the bin\x64 sub directory of the SDK

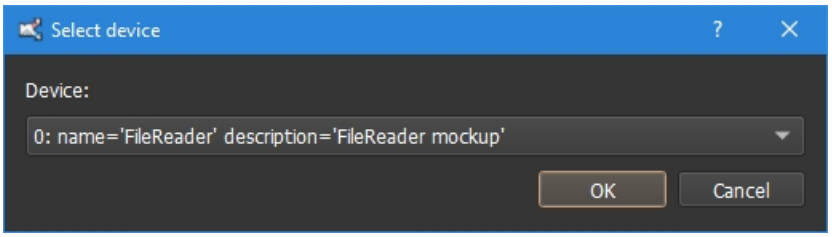
Restart of the machine is not needed, only restart of perClass Mira application.

Initializing acquisition

1. Create or open a Specim FX project
2. Train a classifier (a model must exist in order to process live data)
3. In the *Acquisition* menu, select *Initialize*

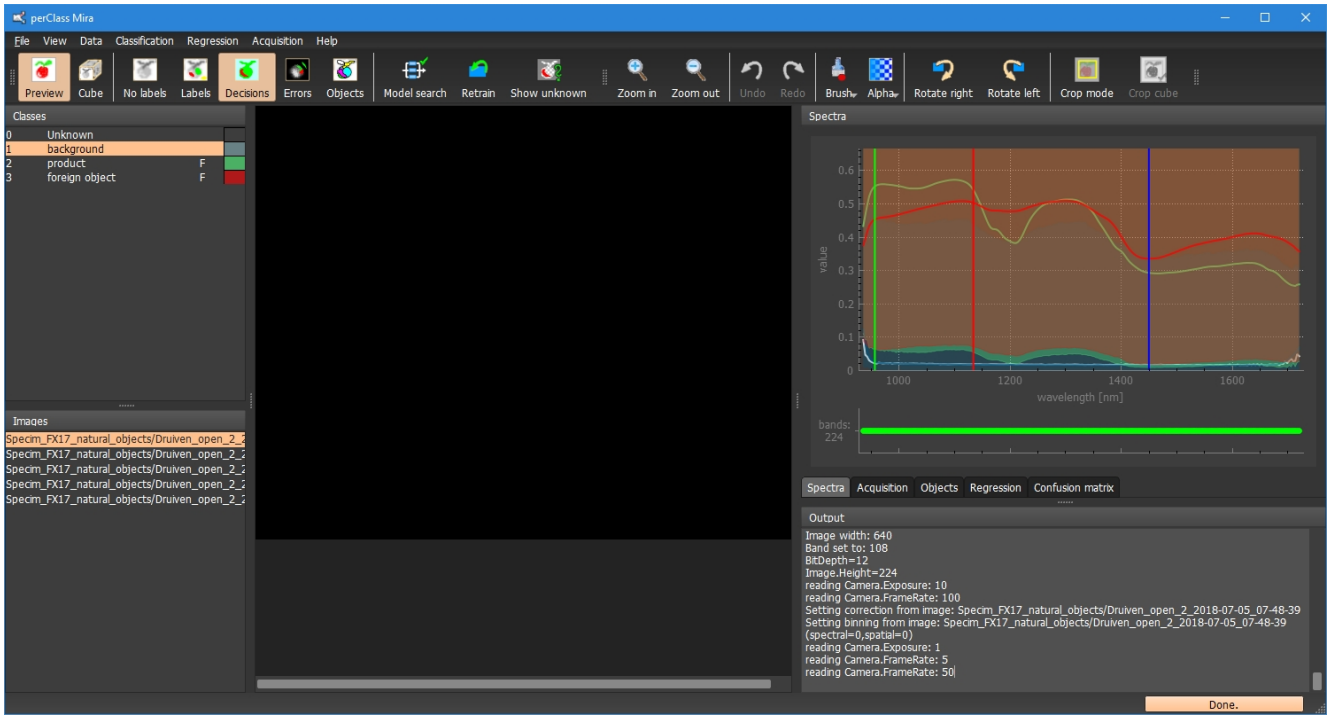


4. A dialog will appear listing all available devices accessible to Specsensor SDK



5. Select the camera or the FileReader interface and click OK
6. If using FileReader an Open File dialog will appear where you can point to the **raw spectral cube** simmulating sensor acquisition stream. Select an existing Specim FX scan directory and the raw cube header file under the capture sub-directory

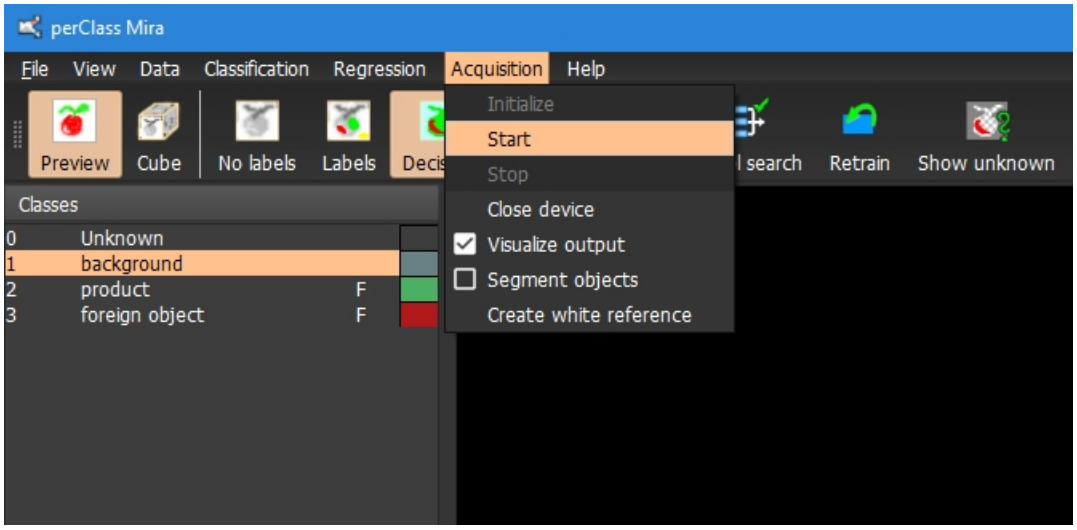
7. The window should show an empty buffer



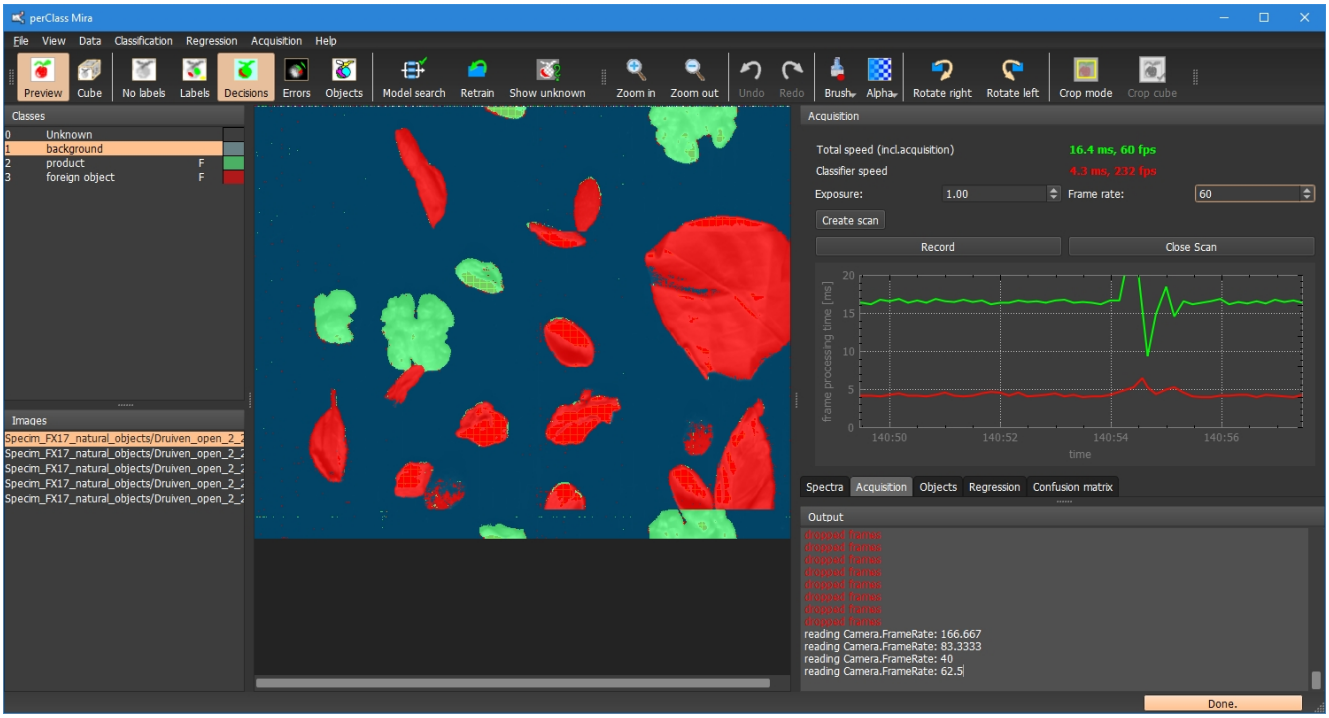
8. Process the data stream can now be started

live data processing

Select *Start* in *Acquisition* menu.



You should see new acqusition arriving in the window



On the right side of the screenshot you can see opened acquisition pannel. It shows the speed of the classifier in red and the total speed of the system in green.

You can change the exposure and frame rate of the sensor from the pannel. By increasing the frame rate, you can speed up acquisition. If the classifier cannot process data at a given speed, you will see red message in the output window pointing out that frames are dropping.

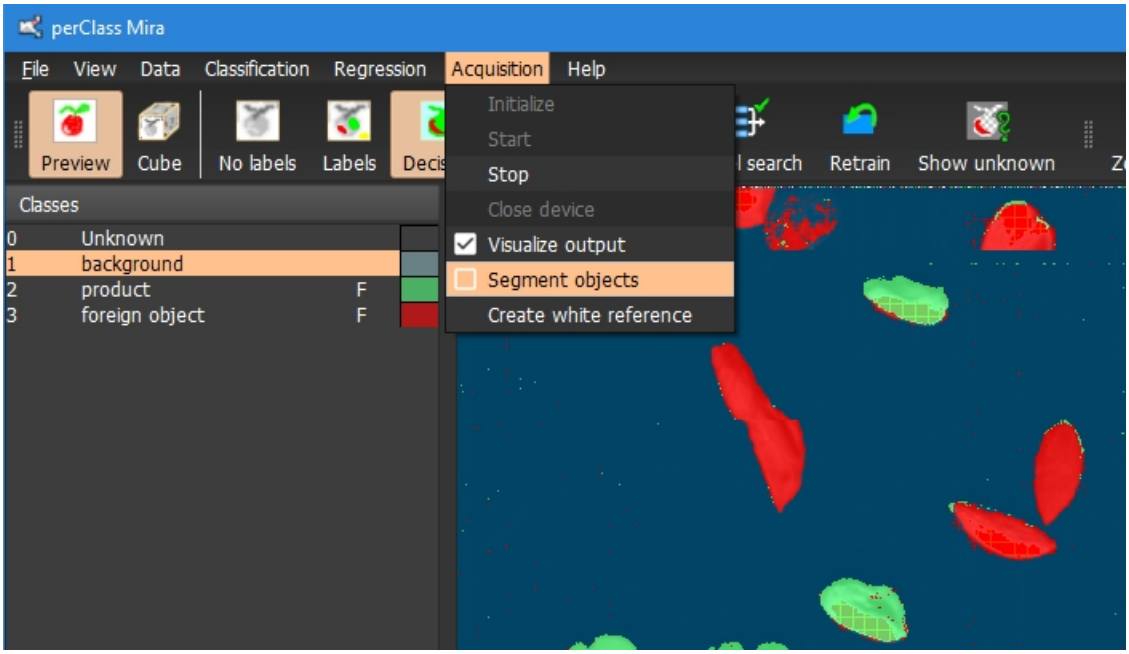
Please note that some combinations of exposure and frame rate are invalid i.e. you cannot have high frame rate and high exposure at the same time.

Segmenting out objects

To segment objects on the live data stream:

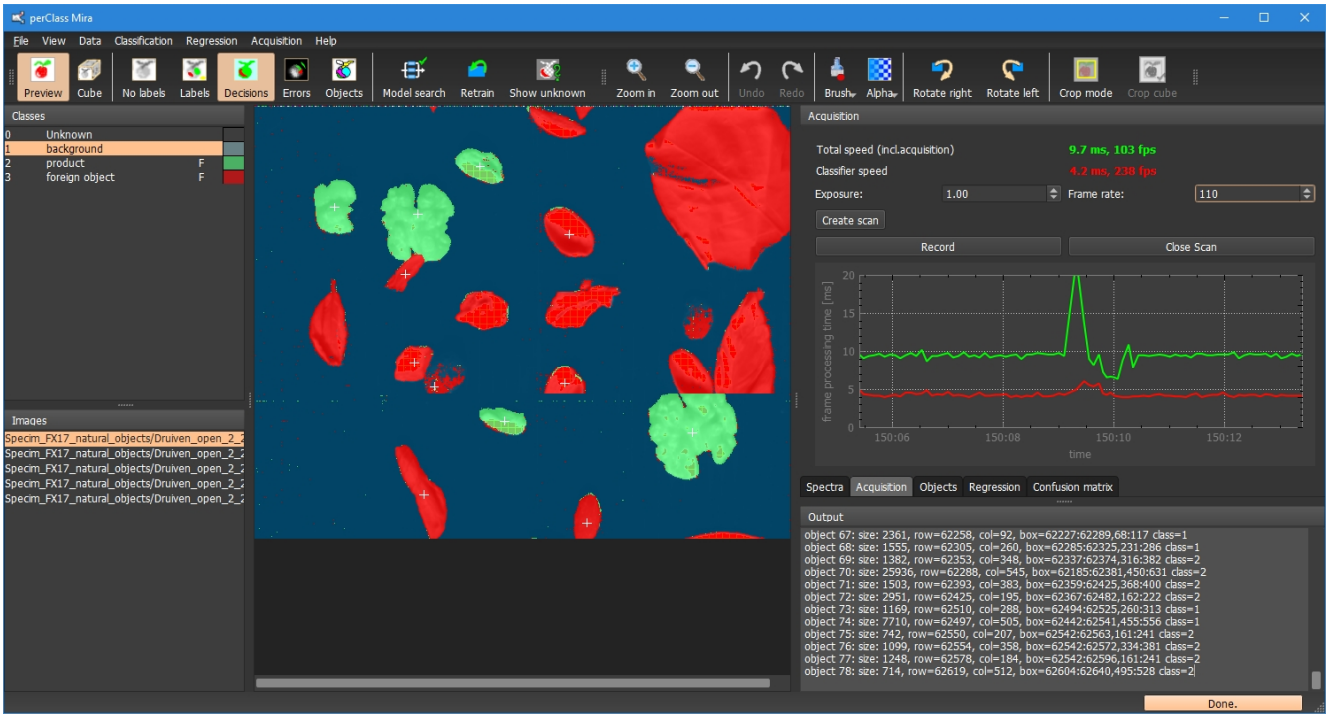
1. Make sure that one or more classes are flagged as foreground in the class list

2. Switch on *Segment objects* in the *Acquisition* menu



Segmented objects will be marked with crosses at their center of gravity.

The output windows also shows object sizes, bounding boxes and classes (if multiple classes are used fot segmentation)



The same per-object information is accessible from perClass Mira Runtime API from custom applications.

Closing live acquisition mode

Use *Stop* and *Close device* in the *Acquisition* menu to disable acquisition mode.

Reference information

Project types

perClass Mira offers several project types:

1. Cubert

- Images are represented by multi-layer tiff files
- User needs to point to .cu3 file
- Multiple files can be loaded at once

2. IMEC

- Images are ENVI cubes
- User needs to point to .hdr files
- It is assumed, that the data cube is stored in a file with the same name as .hdr file and with .raw extension
- There is no preview image expected

3. Hyspex

- Images are ENVI cubes
- User needs to point to .hdr files
- It is assumed, that the data cube is stored in a file with the same name as .hdr file and without any extension
- A preview in a jpeg file is assumed to be available

4. Matlab

- Images are stored in .mat files
- Data cube is a 3D matrix in uint8, uint16, single or double precision
- User may select which varaible to use upon load of the first image
- Optionally, a wavelength vector in single or double precisou may be provieded

5. SpecIm FX (LUMO scanner)

- Images are ENVI files
- User needs to point to the entire directory saved from LUMO scanner
- Preview image is assumed to be present
- Raw image cube is loaded and corrected by included white and dark standard images

6. SpecIm iQ (iQ Studio)

- Images are ENVI files
- User needs to point to the specific .hdr file representing already corrected cube (such as the reflectance cube in results directory)
- No preview is assumed
- Spectral cube is assumed to have .raw extension

7. Tiff stack

- A scan is a directory with individual band images in .tiff format
- Specific band image naming is expected (example: 231001_1_ch_0007_w_0419.tif - the scan name 231001_1, the 1-based channel index 7 and the wavelength in nm is 419)

8. Senop

- ENVI images with .dat extension for the spectral cube
- No reflectance correction is applied

-

Supported image formats

Supported image formats in perClass Mira are:

ENVI Files

- BIL, BIP or BSQ layout
- uint8, uint16, float data type
- Support for header offset
- Wavelengths are loaded from wavelength field

Matlab

- 3D cubes as uint8, uint16, float/single and double precision
- BSQ layout is assumed with two spatial dimensions followed by the spectral dimension
- Separate vector of wavelengths in single or double precision can be provided when loading the first image to use proper wavelength range in nanometers.

TiffStack

- Image is a directory with a stack of tiff images, each representing one band.
- Chennel index and wavelength is stored in a file-name
- Example: Band image for 7th channel (1-based indexing) representing wavelength 419nm. The first part of the filename is the image name (also the name of the directory)
    - `231001_1_ch_0007_w_0419.tif`

Image zoom

- To zoom, use + and - toolbar buttons
- Alternative is to hold *Ctrl* key and use the mouse scroll-wheel

Keyboard shortcuts

In image view:

Ctrl+n	new class, ask name
n	new class, fill automatic name
0 (zero)	set "Unknown" class as current (removing labels)
1..9	set specific class as current
+/-	zoom increase/decrease
Ctrl+1	zoom 100%
PgUp/PgDown	select next/previous spectral band
Ctrl+s	save project
Space	switch to the last label layer
Ctrl+Space	switch between preview and spectral cube
. (dot)	randomly change color of class under cursor
t	set class under cursor as current
m	run model search
r	retrain algorithm found in the last model search
shift-r	re-run the existing classifier on the image (to measure execution speed)
u	show data unseen in training (active learning)
Delete	switch to no labels mode
l	switch to label mode

d	switch to decisions mode
e	switch to errors mode
o	find connected components (objects)
F1	open help
F11	toggle full screen mode
hold mouse & move	paint using current class
Shift+hold mouse	paint removing labels under cursor
Ctrl+mouse wheel	zoom image
Shift+mouse wheel	select next/previous spectral band
c	switch to confusion matrix
s	switch to spectral plot
x	mark class as excluded from training
b	mark class as background (for object definition)

In spectral plot (after clicking on spectral plot)

up/down	increase/decrease max display value
Shift+mouse wheel	select next/previous spectral band

In confusion matrix

Double click	add or remove constrain
Ctrl+mouse wheel	adjust constrain value live
Shift+N	toggle between normalized confusion matrix and absolute values (sample counts)
] (square bracket)	next valid solution
[	previous valid solution