Parameter Selection using Evolutionary Strategies in ImageJ

Roland Bärtschi
Advisor: Janick Cardinale, Prof. Dr. Ivo Sbalzarini

04.07.2011

Abstract

ImageJ is an open source image processing tool written in Java which is extendible through macros as well as through plug-ins. There exists a large collection of both kinds of extensions for various tasks such as image segmentation. Plug-ins often have various arguments and complex macros a lot of parameters. These need usually be selected by hand which is a task that often depends on prior knowledge for what values produce the expected results. In the absence of this knowledge the parameters are usually found by trying a few values until the results approximately match the expectations.

The Plug-in developed in this thesis aims at automating this process with an intuitive user interface using Evolution Strategies such that the work on the user side mostly consists of ranking a few results in comparison to one another until a satisfying result has been reached.

Contents

1 Introduction

Images of all sorts have become an important data source for various fields of academia as well as for industrial applications. With the ever increasing amount of images which are being produced the task of processing them becomes more and more important. ImageJ is an open source image processing tool which is written in Java and is therefore very portable. ImageJ offers the essential image processing tools as its basic features and allows for extensions through plug-ins as well as through macros. Macros can even be produced by recording the users interaction with ImageJ which makes it rather easy to automate complex processing steps. Typical image processing tasks such as image segmentation often involve many parameters which have to be chosen by hand. In some cases one might have prior knowledge about what might be good values for a specific parameter and in other cases one has to try one value after an other in order to find one which is suitable for the purpose of the task at hand.

The aim of this thesis is to create a Plug-in for ImageJ which automates this process in such a way that it guides the user through an optimization procedure where they are only required to select the parameters which need to be optimized and to rank results based on their mutual quality. The idea is to use Evolution Strategies such that the resulting images become better in each iteration based on the user's ranking until they are satisfied with one of the results and choose to apply the selected parameter values on the original image.

2 ImageJ

2.1 ImageJ 1.4x and Fiji

ImageJ is an open source image processing toolkit which is implemented in Java. It supports many file formats such as PNG, JPEG, BMP, GIF etc. and images in the following data formats: 8-bit grayscale, 8-bit indexed color, 16-bit unsigned integer, 32-bit floating-point and 32-bit RGB color. It is extendibe through macros and plug-ins. Macros can be used for automation of processing steps or complex processing pipelines. It is possible to record the actions performed by the user on ImageJ in the form of a macro. Macros can be edited, saved and run at any time. Plug-ins are an even more powerfull extension because they can contain arbitrary Java code which can interact with ImageJ through a specified interface. There are plug-ins available for download on various websites and ImageJ even contains a text editor and java compiler for writing plug-ins. ImageJ can be used as an application or as a toolkit. ImageJ is one of the fastest Java image processing tools available. Basic editing and processing features are already implemented as tools. There are for example various selection tools for rectangular, elliptical etc. shapes which can be combined and make it possible to apply other tools and operations just to the selected area.

2.1.1 Core classes of ImageJ

As a first step towards this thesis the current implementation of ImageJ has been analyzed in order to make use of the existing code as much as possible. Unfortunately it turned out to be much more difficult to do so than expected because of the architecture and the specific implementation of the core classes of ImageJ. Many methods and fields in the core classes have default visibility which makes them visible to other classes within the same package which unfortunately is of no use when developing a plug-in which is being loaded by a different class loader which makes it impossible to access any of those fields or methods no matter what package the accessing class is in.

Lets never the less take a look at some of the core classes of ImageJ which matter for the implementation of this plug-in.

IJ

This class is the core or ImageJ and all its fields and methods are static. It contains a lot of utility methods which are used by other classes and is the center of interaction with ImageJ.

ImagePlus

The ImagePlus class is basically a wrapper for the image. It can either contain a single image or a whole stack of images. Besides the image/stack data it also contains some meta data on the image such as width and height. It also contains the ImageProcessor which is used to manipulate the image and for images which are more than 2 dimensional it contains an ImageStack which basically consists of an array of images.

ImageStack

An ImageStack is an expandable array of slices which are arrays of pixels. The pixels of the current slice can then be used to create a new ImageProcessor. Slices can be added or removed and some meta data can be retrieved and set using methods of this class.

ImageProcessor

ImageProcessor is used to operate on images and is the basic container holding the pixels of an image. ImageProcessor is an abstract class which is extended by the classes ByteProcessor, ColorProcessor, FloatProcessor and ShortProcessor which implement the basic data manipulation operations based on the type of the pixels.

ImageWindow

This class extends the AWT Frame class and is the basic window which is used to display images in ImageJ. It contains an instance of ImageCanvas which is where the image is actually painted on the UI.

ImageCanvas

This is an extension of the AWT Canvas class which is used to draw an image in ImageJ. It is associated with an instance of ImagePlus from which it retrieves the AWT Image which it draws. This class implements mouse and keyboard listener interfaces which are being used for interacting with the image. Based on the tool which is selected in the tool bar of the main window of ImageJ a click or turn of the mouse wheel can have different effects on how the image is being displayed. It is also creating the region of interest selections on the ImagePlus instance that is being displayed.

These core classes are strongly interlinked and it is difficult to extend one without having to extend others due that strong dependency and the visibility of some fields and methods.

One more class is worth being mentioned here and that is WindowManager. This class is as its name suggests a manager for the windows in ImageJ but it has also a role in ImageJ which is important for the processing of the images in the plug-in developed in this thesis. ImageJ uses the WindowManager to determine on which image a macro is going to be run. This is usually the image contained in the instance of ImageWindow which was last in focus but it can also temporarily be set to a different ImagePlus instance with a call to setTempCurrentImage. This is important because it makes it possible to run a macro on an image which is not displayed in an instance of ImageWindow as it will be the case in the plug-in.

2.1.2 Plug-ins

Plug-ins in ImageJ can be anything from an analysis tool to an image generator and it need not even have to do anything with ImageJ. All that's required for a plug-in to be used in ImageJ is that its main class implements the PlugIn interface or if the plug-in is supposed to manipulate the currently focused image then it can also implement the PlugInFilter interface. Just one more detail needs to be taken into account and that is that the name needs to contain an underscore in order for ImageJ to recognize it as a plug-in. But besides that it can contain arbitrary libraries and can have a user interface implemented using a framework other than AWT or Swing.

2.1.3 Macros

As mentioned in the introduction ImageJ also offers the possibility to automate processes with macros which can even be recorded from actions performed in ImageJ. This makes automation quite easy but macros can also be written by hand. Macros have a Java like syntax and can basically run all features which are also available in the user interface.

2.1.4 Fiji is just ImageJ

As the maintainers of Fiji put it:

> "Fiji is an image processing package. It can be described as a distribution of ImageJ together with Java, Java 3D and a lot of plug-ins organized into a coherent menu structure. Fiji compares to ImageJ as Ubuntu compares to Linux."[1]

As stated in this quote Fiji is based on ImageJ and extends it with useful features such as an updating mechanism and a large collection of plug-ins. Fiji also introduced ImgLib which will be used for image representation in ImageJ2. The plug-in accordingly works both in a basic build of ImageJ as well as in Fiji.

2.2 ImageJ 2.0

The release of the next major version of ImageJ is due in October[2] and a first beta version should be released in June. ImageJ 2.0 is being redesigned from ground up using modern Software Engineering principles to make it more modular and extendible. One of the many changes is about the representation of the data. In previous versions data was represented as an image or as a stack of Images organized in slices. Although this basically allows for arbitrary dimensional data by arranging the slices in a particular order it can become very inefficient for high dimensional data in terms of memory access.

This issue has been addressed by Preibisch, Tomančák and Saalfel who developed ImgLib[3], a generic Java Image Processing library which is already in use in Fiji. ImgLib allows to implement image processing algorithms independent of the type of data that is being processed. This is achieved through the Cursor which defines an access strategy for the data, Type which defines a data type and the operations on it and Container which addresses the storage of the data. Due to the generic structure of ImgLib some performance issues have appeared for various plug-ins using it but many of them have been addressed and resolved with ImgLib2 which is now the basic data library in ImageJ2.

Through the extensive use of interfaces in ImageJ2 many of the before mentioned problems of extending an existing class of ImageJ have basically been eliminated. There is for example now an interface ImageCanvas which can be implemented to draw images and that implementation could then be used to replace an existing implementation such as AWTImageCanvas which is the basic AWT implementation of said interface without any problems.

3 Evolution Strategies

Evolution Strategies are one subclass of Evolutionary Algorithms which are often used for optimization problems. Evolutionary Algorithms mimic Darwin's natural

[1] Official website: http://fiji.sc/wiki/index.php/Fiji

[2] See roadmap of the development project on http://imagejdev.org

selection principle in that they consist of a mutation, recombination and selection operators which are applied to individuals of a generation which leads to a next generation.

One kind of Evolution Strategies are so called (μ, λ)-ES which consider a parent generation of μ individuals from which an offspring generation of λ individuals is created. The selection operation is based on the fitness which is associated with each individual and selects those individuals which have the largest fitness. An individual of the population consists of an object parameter value y, a set of strategy parameters s and its fitness $F(y)$.

4 The Parameter Optimization Plug-in

Before the plug-in can be used the user needs to open the image on which the optimization is supposed to be performed. The macro which is to be applied on the image needs to be written to a file in order to load it into the plug-in. Once these conditions are met the user can use the plug-in.

The macro has been implemented with a focus on the MVC pattern so in the following sub sections the model, view and controller are being discussed separately.

4.1 View

One of the main goals of this thesis is to make the process of finding good parameters for a macro an easy task. Therefore the user interface needs to be intuitive and should only require a minimal amount of work by the user. To guide the user through the process each step has been labeled accordingly. You can see a walk through with each step of the procedure in the user manual in Appendix A.

4.1.1 The Selection UI

The selection UI is where all necessary configurations for the optimization process are made. It contains

- a button which opens a file dialog to select the macro,
- a drop down box to select the strategy,
- a check box to select whether or not to generate a history of the optimization process which can then be imported into MATLAB,
- a table for the selection and configuration of the parameters which are found in the macro,
- a text plane where the macro is displayed and the line containing the currently selected macro is highlighted,
- a button to save the configuration
- and a button to start the optimization process once all necessary settings have been made.

The UI has been implemented using Swing and uses only standard components and containers.

4.1.2 The Ranking UI

This UI is used to select and rank the images which have been produced by applying the macro with the parameters set to the values of the individuals of the evolution strategy. But it is not restricted to displaying images from the evolution process as it is possible to set arbitrary ImagePlus resources for the images.

The constructor of the UI takes three arguments. The first one is the number of images which will be displayed in this window. The second is the minimal number of images which need to be selected to make a step of the evolution process and the third is the maximal number of images which are to be selected. By selecting an image it is being assigned the next rank starting from 1. This rank is then passed back to the controller which uses it to compute the fitness of each individual.

The UI consists of 4 main areas:

- On the top there is a small information area which displays the instruction of how many images need to be selected and it also contains a small info icon which expands a hidden label if the user hovers the mouse over it. In this hidden label are further instructions on how to use the ui and which tools of ImageJ work on the displayed images.
- In the center is the main panel which contains the images. Using the scroll wheel of the mouse one can scroll through the slices of the image if it is a stack. If the hand tool is selected in the ImageJ toolbar then one can pan the images if they are zoomed in. If the zoom tool is enabled then a left mouse click zooms all images in and a right mouse click zooms all images out. It is also possible to pan the images if the hand tool is not selected but if the space bar is held down. The last manipulation of the display is possible by holding the shift key and scrolling the mouse wheel. This changes the opacity of the overlay image which is the original image before the macro has been applied to it.
- On the bottom left there are controls for manipulating the image display with scroll bars instead of the scroll wheel. This is also where the current slice or zoom factor is displayed along with the opacity of the overlay image.
- On the bottom right there is an area containing 4 buttons along with a short description of what their purpose is. There is
  - a 'clear selection' button which clears the selection that has been made,
  - a 'Next' botton which is enabled as soon as the required number of images has been selected/ranked and which calls the controller to make a next optimization step,

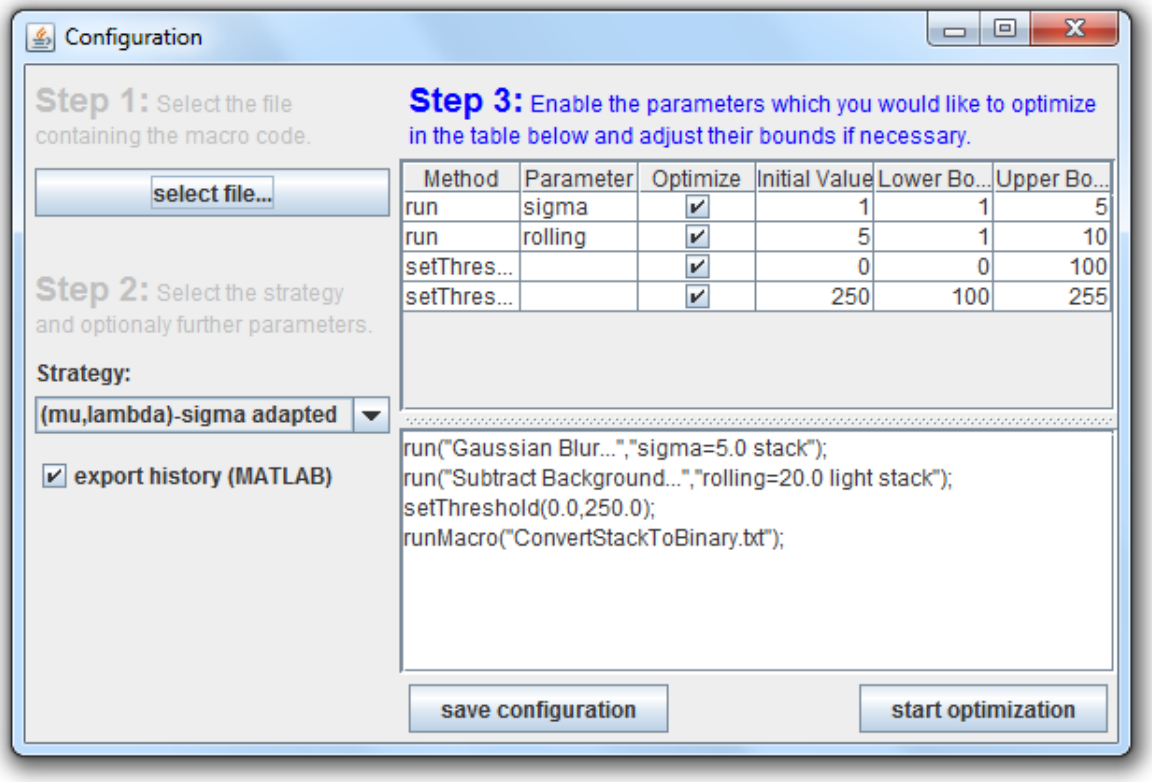


Figure 1: The Selection UI with a simple segmentation macro.

- a 'Repeat' button which can be used to re-sample the current population without selecting/ranking any images and it is only enabled as long as no images are currently selected
- and a 'Finish' button which completes the optimization and calls the controller to apply the macro with the value of the selected image on the input image. If more than one image is selected then the controller will first compute the mean of the selected values and then apply the macro with the mean to the input image.

The controls in the bottom left are synchronized with the displayed images such that a click zoom also changes the value of the zoom scroll bar.

The slice selection scroll bar is only displayed if the image is a stack such that the user only sees controls which he can actually use.

In order to display the images a customized Canvas subclass has been implemented which contains a few lines of code from the ImageCanvas class which ImageJ uses in the standard image displaying windows.

4.2 Controller

The main class of the plug-in is the Parameter_Optimization_Plugin class which implements the PluginFilter interface and the two internally used controller interfaces. Those internal interfaces are used to separate the controller from the user interface such that implementations of either the controller or the user interface can be changed without changes to the other. It also allows to hide methods of the controller intended to be used by the selection UI from the ranking UI and vice versa.

The configuration of the plug-in which includes

- the selected file containing the macro,
- the flag for whether or not the history of the values and assigned fitnesses is being exported at the end of the optimization process,
- the selected Evolution Strategy,
- and the settings for the parameters in the parameterized macro

is being stored in an instance of class Configuration which can be written to a file and later again loaded from a file to avoid having to configure the same parameters again and again for each image on which the optimization plug-in is being run.

The controller creates both an instance of ParameterRankingUI and of ParameterSelectionUI which are being used for interaction with the user. These instances are being created at the point where they are first required. For the ParameterSelectionUI this is already upon start of the plug-in and the ParameterRankingUI is being created when the configuration is complete and the optimization has been started.

Besides the use of ImagePlus in the Ranking UI the class Parameter_Optimization_Plugin is the only class which directly interacts with ImageJ so the classes for

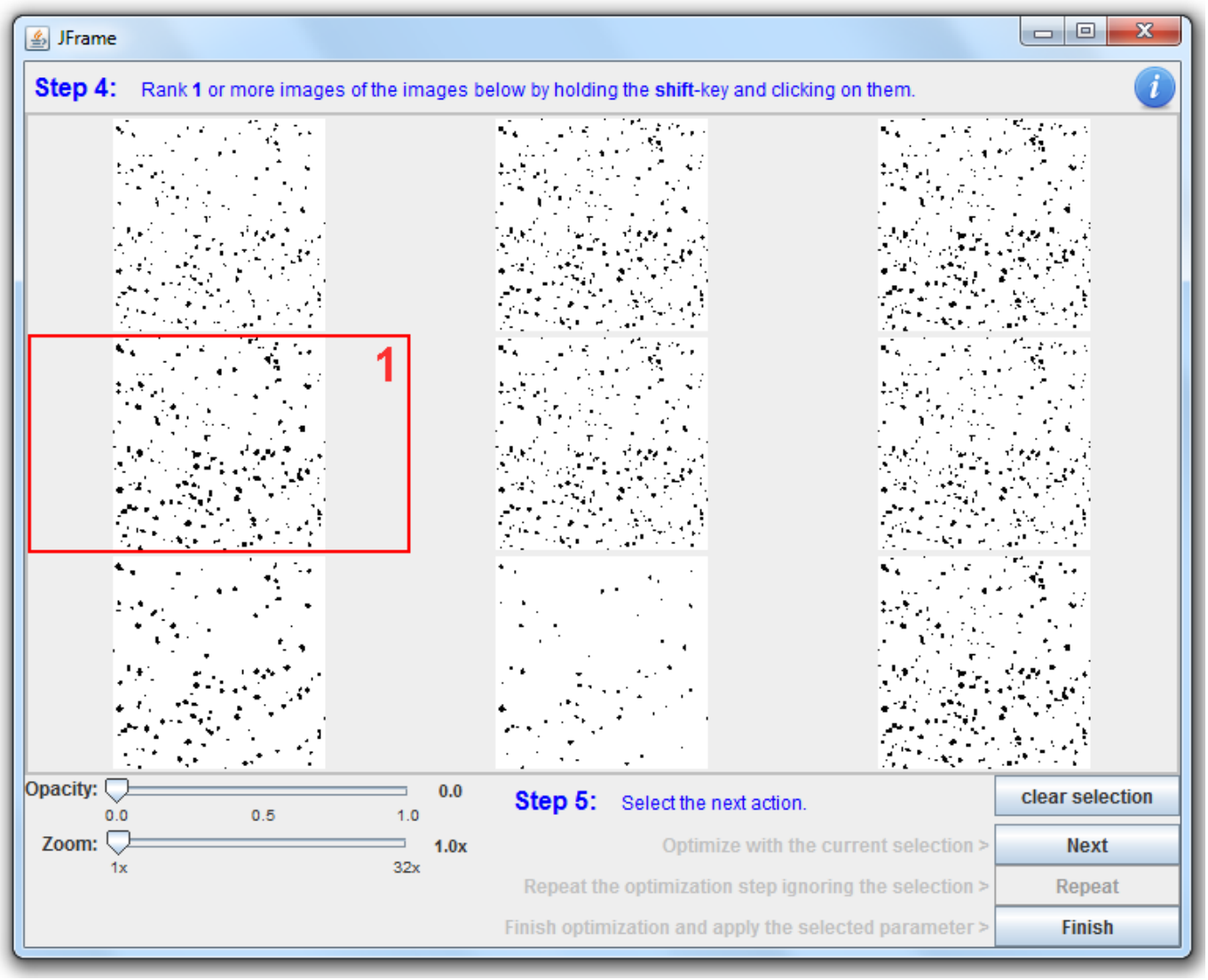


Figure 2: Ranking UI with one ranked image. (Slice scroll bar hidden because image is not a stack)

the user interface are independent of ImageJ and could be used with other libraries through a different controller class.

4.3 Model

The model in this application consists of the parameterized macro and the Evolution Strategies.

4.3.1 Parameterized Macro

In order to detect parameters in a macro script a simple macro parser has been implemented which recognizes possible parameters in function calls. The parameters which can be detected need either be numbers or be contained in a constant string argument having the form "name=value". Numbers in other places such as loops and if-statements are ignored and treated as static code.

The macro parsing is implemented in the ParameterizedMacro class which needs to be instantiated with a string containing the macro code. This code is then split into tokens by

```
Tokenizer tk = new Tokenizer();
Program pgm = tk.tokenize(macroString);
```

where the Tokenizer and Program classes of ImageJ are used. This is basically the same step that the macro interpreter of ImageJ makes before actually interpreting the code. The instance of Program which is being created here contains a list of of tokens such as an "if"-token or a opening bracket token. Unfortunately the current version of ImageJ does not actually parse these tokens into an AST[3] but rather sequentially process the tokens and evaluate the statements which are being recognized and run the plug-ins which are being called. Given an AST it would have been much easier to find possible parameters. Given the lack of said AST the tokens are being processed sequentially and when a token is reached which stands for a function name then an attempt is made to parse a parameterized method. This attempt succeeds if either a number is found as an argument or a constant string which contains at least one substring matching the before mentioned "name=value" pattern

[3] Abstract Syntax Tree which is a tree like representation of the code.

where name consists of letters and numbers and value can be parsed as an instance of type double. If no potential parameter is found in the function call or if an unexpected token appears in the argument list of the function call then the whole sequence of tokens making up the function call is treated as static code and written into a string. But if at least one potential parameter is found then an instance of Parameter is being created and instantiated with the value of the parameter in the macro.

This way at the end of the parsing step we get a list of Strings and Doubles which are being referred to by instances of Parameter with corresponding ParameterSettings which make up the whole macro. Whenever a new instance of Parameter is being created a reference to this instance is added to the list of parameters such that it is possible to directly manipulate the values of the parameters within the macro.

In order to run the macro the code list containing the Strings and instances of Parameter are simply being concatenated and then this new macro string is passed to the runMacro method of ImageJ. In the concatenation process the toString method is called on each instance of Parameter which inserts the current values into the right positions within the macro string and thus runs the macro with the current values for the parameters.

4.3.2 Evolution Strategies

The plug-in uses the interface EvolutionStrategy to interact with strategy implementations during the optimization process.

Following are the methods which a strategy needs to implement with a description of what the purpose of the method is.

```
public void setDimension(int dim);
```

Set the dimension of the optimization space which equals the number of parameters which have been enabled for optimization.

```
public void setDimensionBounds(
        Double[] lower, Double[] upper);
```

Set the bounds for the dimensions. These are the bounds which have been defined by the user for the parameters.

```
public void setDimensionBounds(int dim,
        Double lower, Double upper);
```

Set the bounds for one individual dimension.

```
public void setOffspringSize(
        int offsprings);
```

Set the number of offsprings of each generation. This must be set to the number of images which will be displayed in the ranking UI.

```
public int getMinSelectionSize();
```

Retrieve the minimal number of individuals which need to have positive fitness. This value is then set in the ranking UI as the minimal number of images which need to be ranked before a next step can be made.

```
public int getMaxSelectionSize();
```

Retrieve the maximal number of individuals whose assigned fitness will be used in the evolution step. This value is then set in the ranking UI as the maximal number of images which can be ranked in each step.

```
public void setInitialX(double[] x);
```

Set the initial value from which the first population is created.

```
public void initialize();
```

Initialize the strategy. Once the strategy has been initialized there will always be a current population available.

```
public void setFitness(double[] fitness);
```

Set the fitness of the individuals of the population. The fitness is determined by the ranks which have been assigned to the images and set to be the inverse of the rank for all non-zero ranks.

```
public void makeEvolutionStep();
```

Makes one evolution step based on the fitness which has been set and creates a new population.

```
public void repeatEvolutionStep();
```

Repeats an evolution step which usually corresponds to resampling from the same distribution with a slightly smaller step size. This method is used if the user decides that none of the images is worth ranking it.

```
public double[][] getCurrentPopulation();
```

Retrieves the current population from the strategy. The values of the population are then used to run the macro with the different values to produce the resulting images.

The plug-in currently contains two strategy implementations. One is a simple (μ, λ)-Strategy and the other is a CMA implementation by Hansen and Alumni.

The simple (μ, λ)-Strategy just takes the μ individuals with highest fitness and computes the new mean of their values and their strategy parameters σ. The λ new individuals are then sampled from a normal distribution with the computed mean and step size σ. For each individual a strategy parameter σ_i is sampled from a different distribution around σ.

5 Future Work

ImageJ 2.0

As a major release of ImageJ - namely ImageJ 2.0 - is due in October 2011 it is suggested that the plug-in will be adapted to follow the new design principles of the completely refactored version. Note that due to the backwards compatibility of the new version which will basically include the old version as a library it should be possible to run the plug-in with only minor changes or maybe even without any changes at all. But the new version of ImageJ offers quite a few opportunities for a better integration especially on the side of the user interface.

More Strategies

The current implementation of the plug-in includes just 2 strategies. It would be nice to have a larger collection of strategies such that if one strategy works slow on a particular optimization problem then an other might be chosen which could work better.

Adding a new strategy currently requires a small change in the code of the main plug-in class where the class of the new strategy needs to be added to the list of available strategies. One possible improvement would be to implement a discovery procedure which checks the plug-in directory and/or some subdirectories for implementations of the EvolutionStrategy interface and then dynamically loads them into the plug-in.

Macro Parser

The current macro parser has quite a few limitations. It is only able to detect parameters in function calls on the deepest level. If a function call is nested in an other function call then only parameters in the nested function call can be detected. This is due to the scanning procedure which was chosen for this simple parser as the focus was on finding parameters in function calls.

One improvement could be to just slightly modify the existing parser to make it consider the nesting functions as well but a more suitable attempt to resolve the issue would be to completely rewrite the parser such that its output would be a parameterized AST instead of a sequence of static code in the form of a String interleaved with Parameter objects.

Method awareness

The comfort of the configuration for the user could be taken much further by explicitly analyzing in which method the parameter appears. In this case one could restrict the range of parameters for example for the 'makeRectangle' method automatically because the we know the dimension of the image in which the selection is to be made.

Right now the plug-in always uses doubles for the values when they are replaced in the macro. But some methods only work for integers which can be circumvented by wrapping the values into a floor or ceil function. In the current version of the plug-in it is necessary to do this by hand before the macro is loaded. A simple type selection in the selection table could avoid this and as mentioned before one could even automatically wrap the arguments of ImageJ methods for which the type is known.

Visualization of evolution path

The values of the individuals of each generation are being recorded along with their assigned fitness and this data can be exported as a simple MATLAB file. The standard user probably hasn't got any interests in this data but it could be quite useful for the analysis of the evolution strategy if an advanced user could see a visualization of the evolution path during the optimization process.

History

The values and fitnesses of individuals are kept in a history data structure which can then be used for the data export as mentioned in the previous paragraph. One idea to improve the usability of the plug-in could be to make this history available such that the user can take a few steps back and take an other direction at some point if he realizes that he has taken some sort of a dead end.

Appendix A. User Manual

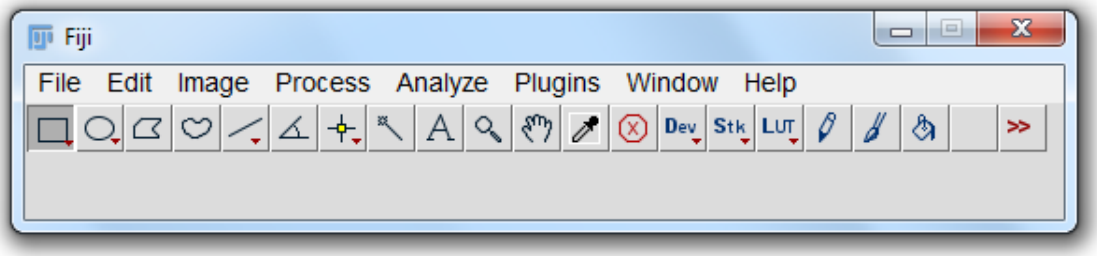
Parameter Optimization Plug-in

What's the purpose of this plug-in?

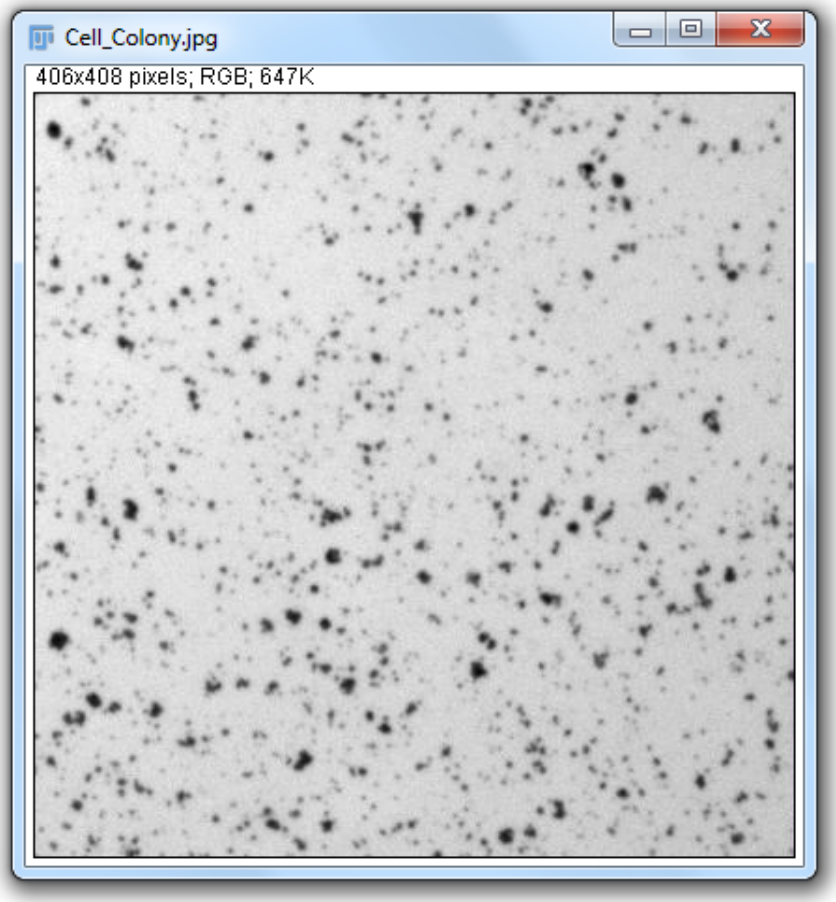
If you're working with complex macros that are calling plug-ins and have various parameters for which you need to find good values then you often had to just try over and over again with different values until you got a result which was approximately what you wanted it to be.

This is where this plug-in enters the stage. Instead of setting different values in the macro by hand and applying the different macros on the same image/stack you can now simply select the image and start the Parameter Optimization Plug-in.

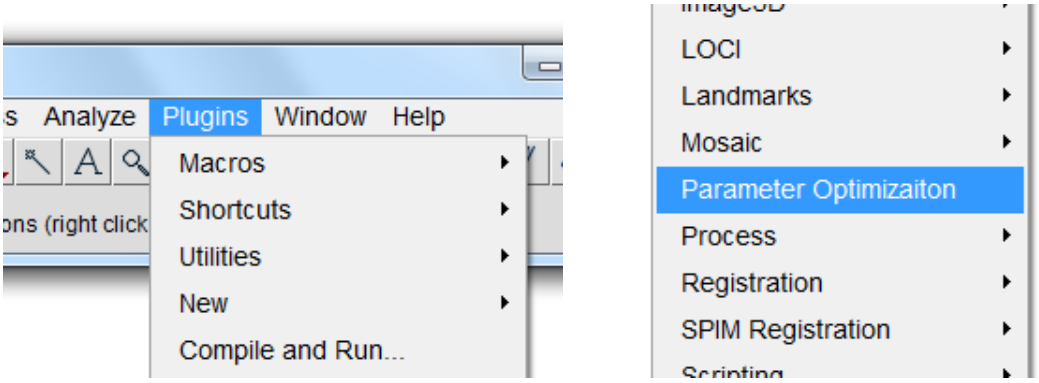
How does the plug-in work?



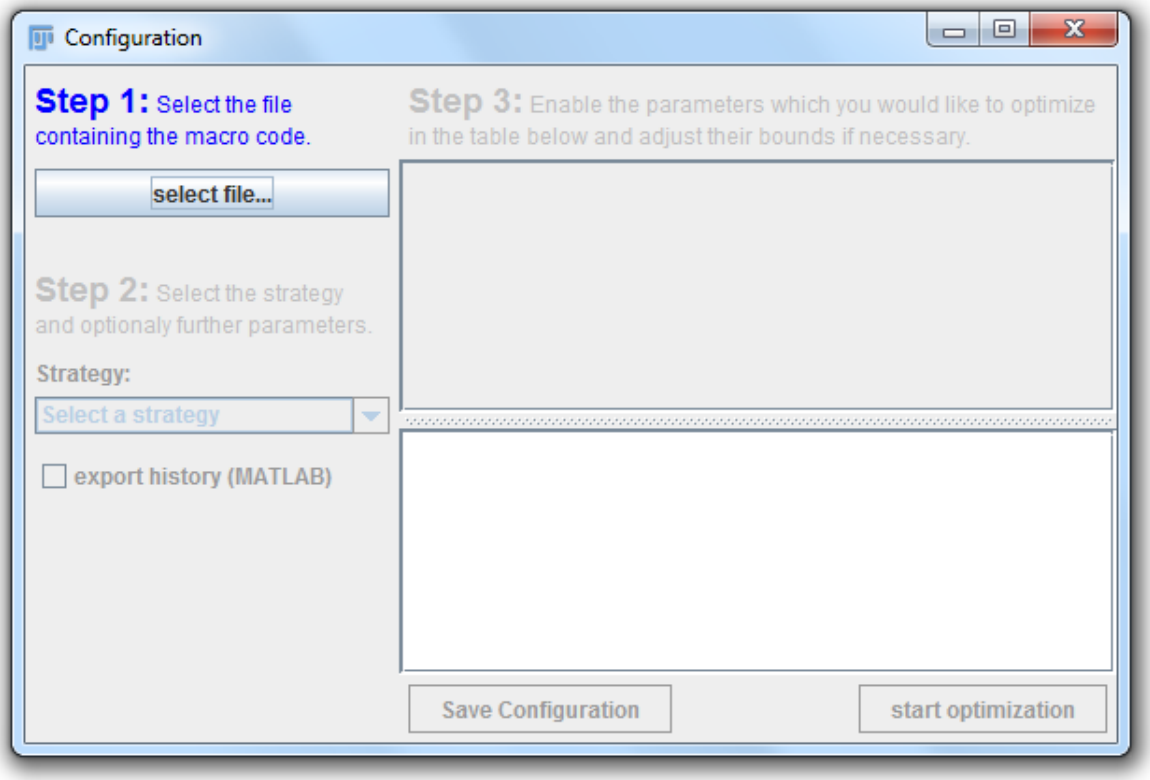
First you need to make sure that the macro you want to run is written to a file and that you know its location. Then you just open the image or stack you want to run the macro on (like the cell colony image in the samples).



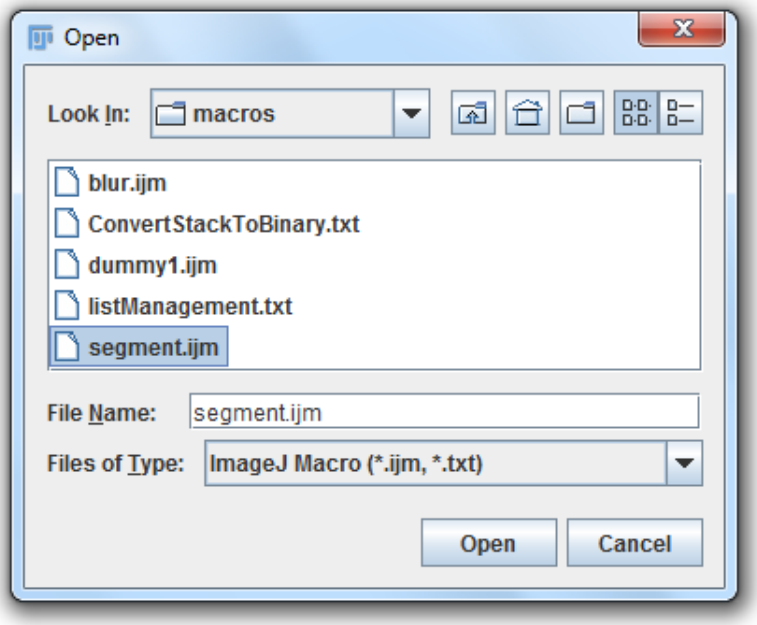
Then navigate to the Plug-ins menu and choose the Parameter Optimization plug-in



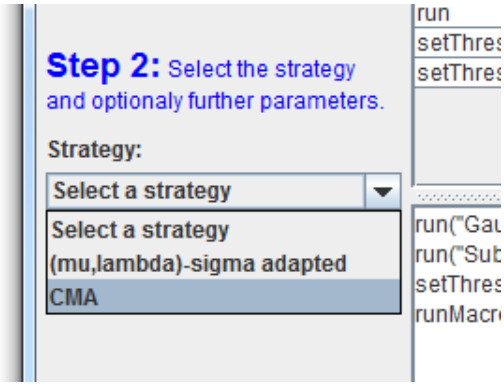
Now you should see the following window. This is the configuration window where you set everything up for the optimization. The first step is now to select the file which contains the macro code which is to be run on the image.



When you click on the 'select file...' button a standard file dialog should pop up. By default it opens the macros folder of ImageJ for the macro selection.

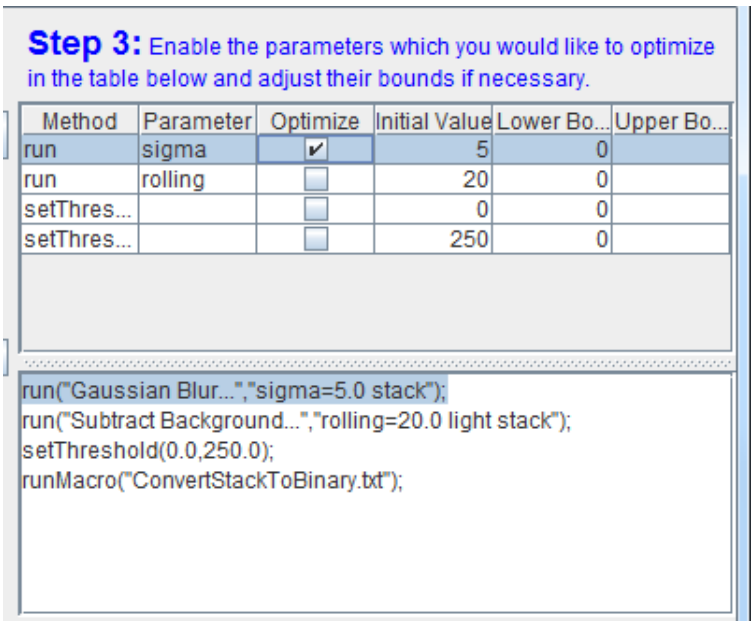


In the next step you need to select the strategy which you want to use for the optimization. In this example you have the choice between a simple (μ,λ) strategy with step size adaption and a CMA strategy.

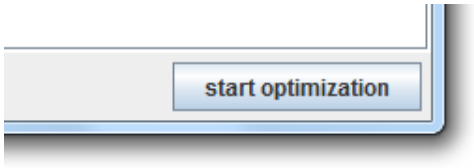


In the above step you can also select the checkbox for the export of a history file if you are interested in analyzing the optimization process.

Once you have selected a strategy you can configure the parameters which have been detected in the plug-in. You can enable or disable a parameter, set its initial value and the bounds which it has to satisfy. It is highly recommended that you insert the correct bounds for the parameter if they are known because this can have an impact on the quality of the results and the performance of the optimization.

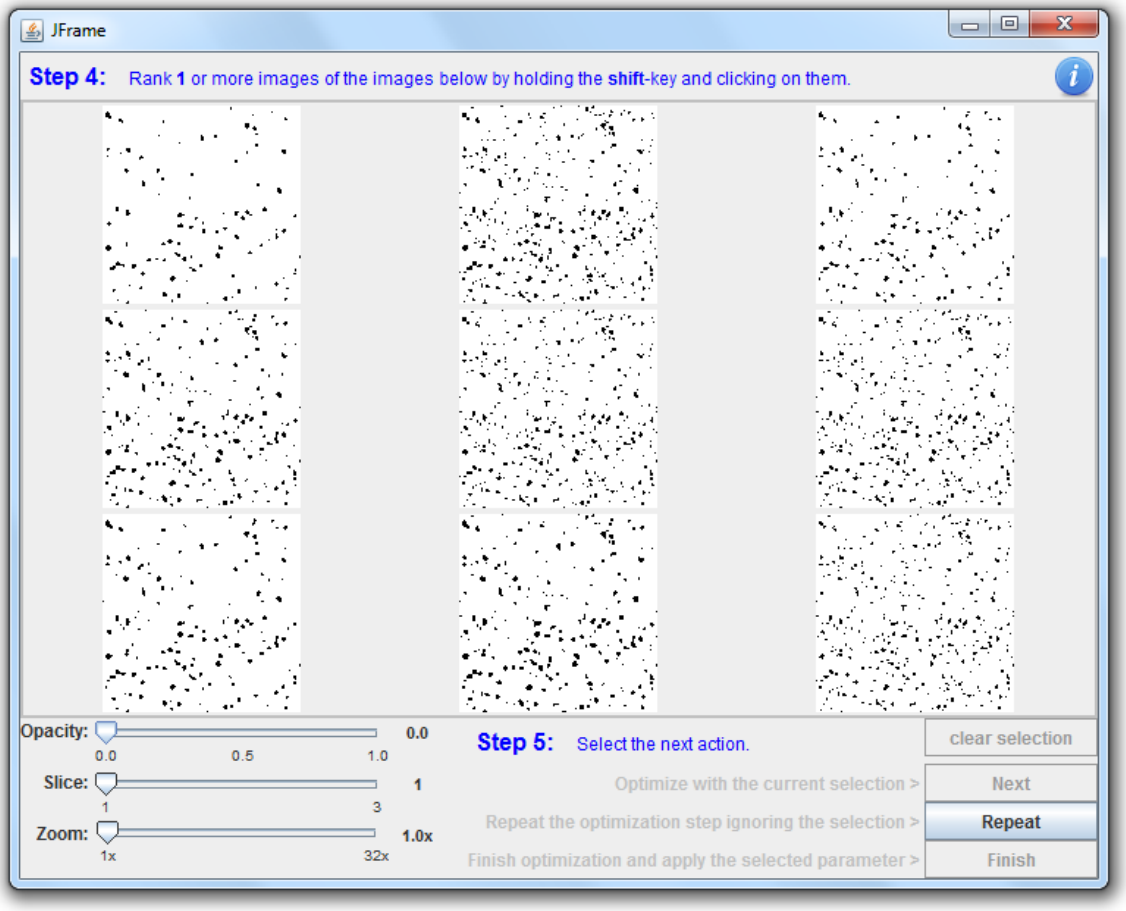


Once you have selected at least one parameter for the optimization you can click on the 'start optimization' button to start the optimization process.

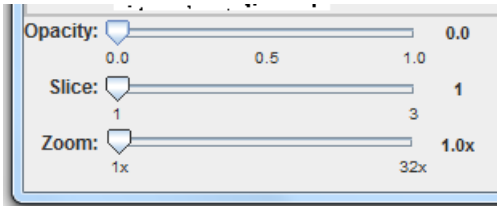


Note that there is also a possibility to save the current configuration to a configuration file by clicking on the 'save configuration' button. This configuration can later be opened in the first step instead of a macro file.

After a click on the 'start optimization' button you should see a window similar to the following.

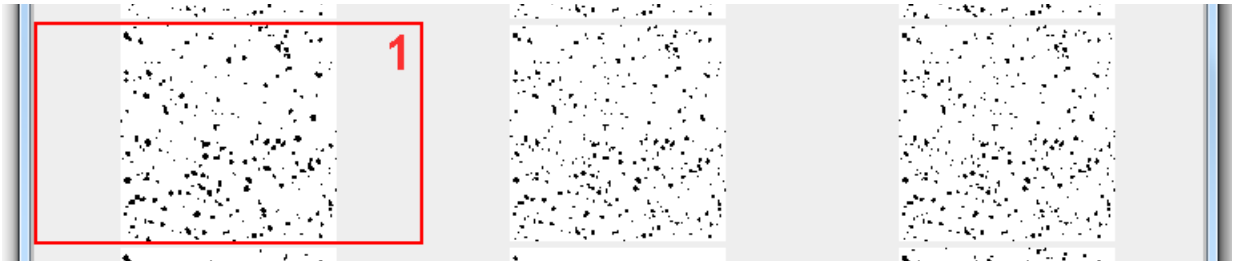


On the bottom left you have an area with 3 sliders which can be used to zoom in, change the currently displayed slice and change the opacity of the overlay image which is the original image.



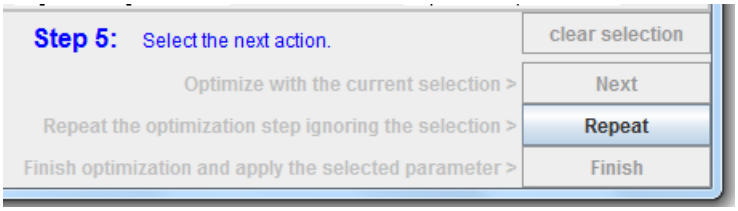
Use these controls to examine the different images.

Next you have to rank the images by holding the shift key and clicking on them starting with the one you rate best. A number should appear in the top right corner of each ranked image depicting the rank which has been assigned to the image.



As you rank images you will notice that the controls on the bottom right either become enabled or disabled based on the number of images you have ranked. If you made a mistake while ranking you can clean the current selection and try again. If the images are so different from what you expect them to be that you are not able to rank them then you can try to repeat the evolution step and hope that better results are being produces in that trial.

Once you have ranked the number of images which is being displayed in the info field on top then you can click on the 'Next' button to generate the next set of images based on your ranking.



When you made a few optimization steps and arrived at a point where one or several images fulfill you expectations then you can finish the optimization process by clicking on the 'Finish' button. This applies the macro with the value of the selected image on the original image.

If you checked the box for the export of the evolution data then a file dialog will pop up asking you for the file name to which the values will be written.

You should now see your image on which the macro has been applied with the optimized values.

References

[1] J. Cardinale. *Bachelor thesis description - Parameter Selection using Evolutionary Strategies in ImageJ* (unpublished)

[2] *ImageJ Features.* http://rsbweb.nih.gov/ij/features.html

[3] S. Preibisch, P. Tomančák, and S. Saalfel. *Into ImgLib - Generic Image Processing in Java.* http://fly.mpi-cbg.de/~preibisch/pubs/imagejpaper2010.pdf

[4] M. D. Abramoff, P. J. Magelhaes, and S.J. Ram. Image processing with ImageJ. *Biophotonics Int*, 11(7):36-42, 2004.

[5] Dirk V. Arnold and Hans-Gregor Beyer. *Noisy Local Optimization with Evolution Strategies.* Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[6] Hans-Georg Beyer. *Theory of Evolution Strategies.* Springer, 2001.

[7] Stephen B. Chrisholm, Dirk V. Arnold, and Stephen Brooks. Tone mapping by interactive evolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 515-522, New York, NY, USA, 2009. ACM.

[8] Hans-Georg Beyer (2007) Evolution strategies. Scholarpedia, 2(8):1965