



Utrecht University

Image Processing Assistant Notebook - IPAN -

Combining ImageJ and Python for microscopy
image analysis and large-scale data analysis

Applied Data Science profile
research project

Daily Supervisor: dr. **Y. (Eugene) Katrukha** - Dept. of Biology - Biodynamics and Biocomplexity

Supervisor: dr. **A.L. (Anna-Lena) Lamprecht** - Dept. of Information and Computing Science - Intelligent Software Systems

Second Supervisor: dr. **M.W. (Mel) Chekol** - Dept. of Information and Computing Science - Data Intensive Systems

Examiner: dr. ir. **Edwin Bennink** – Image Science Institute - UMC Utrecht - Dept. of Radiology

Student: **Nicolas Cristini** – MSc student Regenerative Medicine & Technology - Applied Data Science profile

Abstract

The field of biomedical imaging includes a wide range of techniques used to picture and extract the experimental results as images. The further acquisition of valuable data requires a process of image analysis. ImageJ is one of the most well-known software used in the scientific community to perform image processing analysis. This software has incredible potential because of its numerous features and plugins. However, when the analysis is performed on a large number of data, carrying out the analysis requires the adoption of automatic systems and a certain level of experience. The goal of the following project was to develop an online tutorial and assistant notebook by combining ImageJ functionalities and Python in the Jupyter Notebook platform. For this purpose, the python package PyImageJ has been implemented within python-based functions. Creating reproducible and ready-to-use analytical pipelines potentially reduced the time and effort required to perform the analysis. Moreover, providing this analytical tool help to standardized the processing steps and increased the reproducibility of the image analysis. The series of notebooks presented in this project both provided a learning tool to understand how to functionally operate with the proposed packages and an assistant tool to help the researchers to carry out their analysis. A series of interviews were performed on experts and potential users of this product. This step aimed to understand the user context and the possible applications of the final product of this project. Finally, an evaluation process was performed to assess the accuracy and the efficiency of the developed analytical method. This last evaluating step highlighted the necessity to further investigate the performance of the final product. Nevertheless, despite the complexity and the multiple limitations, the automatization of ImageJ 1.x functionalities throughout Python 3.x was demonstrated to be feasible and potentially improvable.

Table of Contents

Abstract	2
Introduction	4
The needs of the biomedical imaging field	4
Using ImageJ image analysis	4
The integration of ImageJ and Python	5
Goals of the project	6
Summary	6
Interviews	8
ImageJ expert - Nicholas Condon	8
ImageJ expert - Cameroon Nowell	9
PhD candidate – Ilaria Signoria	9
Design	11
ImageJ	11
PyimageJ	11
Jupyter Notebook	12
Implementation	13
01 - Working with ImageJ pt.1	14
02 - Working with ImageJ pt.2	15
03 - Working with the IPAN module	17
04 - Nuclei Count	20
05 - CTCF Measurements	21
Evaluation	23
Overhead of importing ImageJ	23
Methods comparison	23
Comment about CTCF macro	25
Discussion	26
Conclusion and Future Perspectives	27
Layman’s summary	28
Acknowledgements	29
Bibliography	30

Introduction

The needs of the biomedical imaging field

In the scientific research, the step of image analysis represents a fundamental stage. It consists in converting the result of the experiment into valuable data used to answer to the original research question. Most experiments require to be performed multiple times and on numerous samples and replicates to allow drawing a solid conclusion. Producing consistent and comparable requires that the execution of the experiment, the acquisition of the results and the final step of the analysis follow identical steps. If only one of these three points fails to be reproducible, all the following conclusions and discussions fail unquestionably. This project is centred on the last phase: the data analysis.

The biomedical imaging field is a flourishing area that includes a wide range of advanced imaging technologies used by researchers to visualize and extract the results of their experiments. The growing complexity of the acquisition techniques required a simultaneous development of sophisticated imaging processing and analysing softwares¹. In this regard, the amount of data produced for each experiment is getting bigger and bigger due to the increasing resolution of the equipment and the rising number of samples under analysis, especially when the experiment is performed in a high-throughput setting. The large amount of data generated with these methods are impossible to be manually processed for humans². For the before mentioned reasons, there is the need to incorporate tools that enables the automatization of the image analysis through algorithms and computing that can and assist the large-scale data analysis.

Using ImageJ image analysis

In the biomedical imaging scientific community, including scientists, technicians and students, one very popular software used to perform image analysis is ImageJ. ImageJ is an open-source Java software created by Wayne Rasband in 1997. Since then, the software has attracted the interest of the scientific community because of its practical features and easy applicability³. Java developers and bioimaging enthusiasts created a series of plug-ins⁴ that offer multiple tools to perform qualitative/quantitative analysis and image elaborations⁵ (histogram manipulation, background subtraction, image segmentation, 3D reconstruction, etc.) for other various applications that goes even beyond the mere biomedical field as for example agriculture⁶.

After 25 years from the launch of the software, 1000+ plugins and a new generation of the software called ImageJ2⁷ have been released. This version is available as part of Fiji (Fiji is just ImageJ) that incorporates a collection of plugins to simplify the scientific image analysis process. Fiji boasts hundreds of citations in the scientific fields and many books and user-made tutorials on how to use the software. Besides its open software feature, the large amount of available documentation and the endless collection of analytical tools, getting acquainted with this software requires a lot of time and energy. Despite the fact that most of the users that employ ImageJ manually accomplish their simple image processing tasks, some analysis may require to exploit the full potential of the software. For example, high-throughput repetitive analysis performed on hundreds to thousands of samples generate a large amount of

data. In this case, the image processing should be fully automatized and performed with batch processing features⁸. Through automatization, the analysis became faster and less tedious for the user. Moreover, in this way the analytical process is made to be repeatable, reproducible⁹ and transparent enabling the further comparison of the obtained data and the future review of the results.

Nevertheless, the automatization of the image analysis with Fiji is not an immediate task. On this topic, the ImageJ macro¹⁰ is a sequence of runnable ImageJ commands that can easily be recorded from the manual input and then relaunched in series. In this way, complex tasks can be fully automated and run with a single click. More structured macros can also be produced by combining the IJ commands built-in macro functions for a variety of demands and situations that allows to manage the results table, extract single values and numerous other options. The IJmacro language is quite intuitive but it needs time to be understood and fully mastered.

Regarding other established alternatives, the market already offers some options to assist the image analysis process. For example, Eclipse¹¹ is an ImageJ plugin that offers an interface and a macro editor that is used for software development, Bio7¹² is a Scientific Image Analysis platform that uses the Eclipse view. Cell Profiler¹³ is a software package that provides the interactive building of the processing workflow. It uses the ImageJ API and combines it with python software to control the workflow. However, none of these options offers the possibility to interact with the code and, at the same time, to produce tailor-made pipelines while being guided by comments and advices.

The integration of ImageJ and Python

Nowadays, Python is one of the most used programming languages because it is highly readable and maintainable and it is equipped with a comprehensive standard package library. It is one of the languages of choice in the field of Data Science because it offers multiple packages to perform data wrangling, elaboration and finally, data visualization¹⁴. Besides that, it can be employed for the automatization and the repetition of tasks due to the possibility to easily build iterative cycles. So, the incorporation of the ImageJ functionalities within a Python-based structure may result in a useful and powerful solution to carry out the complete image processing analysis: starting from raw images and finishing with descriptive graphs, all in one platform.

As shown from the ImageJ documentation, there are already some ways to combine ImageJ and Python. Jython¹⁵ represents one of the two alternatives. Jython scripting allows running ImageJ scripts inside ImageJ. One advantage of this option it allows exploits the tools supported by SciJava while one disadvantage is that most of the Python packages (e.g., NumPy¹⁶) cannot be run with this modality. Another big disadvantage is that Jython implements only Python 2.x and not Python 3.x. An interesting work that makes use of Jython is ImagePy¹⁷, an open-source platform-independent image processing framework equipped with a UI interface. This project offers a very interesting solution to carry out the image analysis and at the same time exploit some of the most used python packages.

The alternative to using Jython is to embed ImageJ inside the Python code, for example within a Jupyter Notebook¹⁸, by using the PyImageJ¹⁹ package to access the

ImageJ API from Python. In this context, we are going to make use of tool of Image 1.x. In this case, the main advantage is that it makes it possible to combine ImageJ with other image analysis libraries like scikit-image²⁰, ITK²¹ (C++ application), OpenCV²² (computer vision application) and various others, in a single Python program. The main disadvantage of this choice is that wrapping ImageJ in Python has some limitations and bugs, particularly surrounding the use of ImageJ 1.x. This is much visible when compared to the Java-based kernels such as BeakerX²³. The PyimageJ documentation and tutorials consists of a series of notebooks that explain how to install and initialize PyimageJ. It makes examples of how to open an image on the notebook, how to convert the java image into a NumPy array, how to deal with large images, how to process the image by applying different operators and finally, how to run macros, scripts and plugins. However, what is missing in this notebook is a representative example of an image analysis workflow that shows how to process an image and to get the results from it. The currently PyimageJ documentation lacks proper working examples and comments. A user experience guide about this package can be found on the homonymous channel on the image.sc forum and on the Gitter²⁴ channel.

Goals of the project

So, with this project I aim to answer at the following research question: “How can ImageJ users be enabled to combine their analysis with Python code?”. The product of the project is a number of Jupyter notebooks including a series of pre-coded chunks coupled with elaborate comment sections that will guide the users through the analysis while allowing for the personalization of the algorithm. The python functions employed in the Jupyter notebook are wrapped in a Python module that takes the name of the project. In this way, the processing and analysing functions can be easily imported in the notebook. With this project, I aim to simplify the image analysis procedures by encompassing common procedures required in the bioimaging field. For example, nuclei quantification will be one of the processes that will be broken down into singular steps. One other goal of the project is to deliver a robust combination of ImageJ 1.x and Python 3.x and to provide solutions to some of the problems that it is possible to encounter. I would like to answer the research question: “Is it possible to perform the entire process of image analysis by exploiting the ImageJ functions and plugins in a Python platform as the Jupyter notebook?”. Another considerable point of my project and what also distinguishes it from what is already available is that it provides a more comprehensive tutorial on how to use the PyImageJ package by giving different working examples. Finally, the combination ImageJ 1.x and Python 3.x has been achieved by developing a Python module called IPAN.py that embeds fully operative processing functions employing both Python structures and ImageJ macros. This module can be easily imported into the Jupyter notebook. The embedded functions allow to carry out the full image analysis, from the image processing to the data visualization with multiple plots.

Summary

In the following sections of the report, I will discuss the developmental steps of this project. As I started to work on this project, I felt the need to increase my knowledge about this software and how to use it. Firstly, I interviewed two experts of the image analysis field that were very helpful to me for better understanding how this software

is employed in a microscope facility. Secondly, I started a collaboration with a PhD candidate that demonstrated her interest in this project. She shared with me the data of her analysis giving to me the possibility to fulfil a real research task. The solution to her problem is represented by the last notebook of this project. The subsequent section will present the design of the work by illustrating the technologies that were used. It will follow an explanation of how I employed and implemented those technologies for the development of a new python module. Finally, I will provide an evaluation of the resulting product by comparing the time of execution and the accuracy of the composed analytical pipeline. In this section I will also explain the advantages and disadvantages of using this module to carry out the image analysis by comparing it with running the same analytical pipeline directly from ImageJ. Finally, I report a brief discussion on the topic and the section with the conclusions and future perspective of this study where I summaries which tasks of the project were successfully achieved and which ones failed to be accomplished.

Interviews

This section of the report summarizes the discussion that I had with two experts of the imaging field, Nicholas Condon and Cameroon Norwell, and one PhD candidate that proposed herself for a collaboration within this project. At the beginning of the project, I felt the need to acquire more knowledge about the FIJI and its functionalities. Therefore, I followed a workshop available on the free video platform YouTube. Among many similar videos, I found them particularly interesting and exhaustive workshop video tutorial of Nicholas Condon. After completing the online workshop²⁵, I contacted Nicholas to directly ask him some questions about the software. At the end of this interview, he proposed me to his colleague Cameroon Norwell with which I had the possibility to continue my investigation. Finally, in this section I will include discuss about the problems raised by the PhD candidate Ilaria Signoria that decided to collaborate with me in this project.

ImageJ expert - Nicholas Condon

Nicholas Condon is the senior Microscopist and CZI imaging Scientist at the Institute of Molecular Bioscience at Queensland University, Brisbane. By following the workshop, structured into three lectures: “beginners, intermediate, advanced”, I learned many functionalities and tools that I was not aware of. I then decided to contact and interview Nicholas Condon so that I could discuss with him the software, the user community and if he had thought about the implementation of Python in the process of image analysis.

At the beginning of the interview, I asked him how the microscope facility was structure and how it was possible to have access to the equipment. He replied me that the facility is open to users that goes from the students to the postdocs. All of them are required to perform an introduction course and test about how to use the equipment and also how to perform their analysis with ImageJ. At this point, I was already quite impressed because I could not avoid to think about my personal experience as a trainee in the UMC (RMCU) where there was almost no (or very poor) planned preparation and training to the use of such kind of software.

He then reported that out of the broad group of users only 10% performed their image analysis independently by using the information shared on the online platform of the university, 30% did not need to perform an analysis and that, a strikingly 60% of all users end up asking support and assistance to their analysis because there were not able to achieve it on their own. His testimony gave me evidence of the potential of developing and sharing with the community a series of assistant and tutorial notebook.

At the end of our conversation, he also made me reflect about the fact that many of the experiments performed nowadays generates a very large amount of data that sometimes exceeds hundreds of gigabytes. Therefore, the analysis needs to be run from a server that could handle this data size. For this reason, the facility where he works made multiple workstations where students, PhD and postdocs can perform their analysis.

Finally, I asked him an opinion about performing the image analysis with the use of Python. He replied to me by pointing out that there are already some products developed for that: Napari²⁶, a fully python-based image viewer that allows the visualization and manipulation of the image from a user-friendly GUI, and ZeroCost²⁷,

a project that consists in a series of self-explanatory Jupyter Notebooks for Google Collab that shows how to perform deep-learning tasks for microscopy assignments.

ImageJ expert - Cameroon Nowell

The second interviewed of this project Cameroon Nowell, manager of the imaging, flow cytometry and analysis facilities at the Monash Institute of Pharmaceutical Sciences. His work is more related to the development of analytical workflow using ImageJ. The question him an opinion about the level of accessibility and usability of the Jupyter Notebook to perform an image analysis. He replied to me saying that he was used to employ Python to manage and elaborate his data but that he was not used to incorporate his analysis on a Jupyter Notebook. However, he supported the idea that, as long as the code was self-explanatory and associated with comments, it should have been easy for not experienced users to run the analysis on such a platform.

After that, he spent some time showing be examples of analytical pipelines developed and share by him on Figshare²⁸, an online platform to store, share and discover analysis with a more simple use than Github²⁹, the platform of choice in the informatic field. While explaining to me his analytical workflow he raised my attention to an interesting point. He stressed the attention on the ratio between the time employed in coding a script and the time that will be saved by automatizing the task covered by the script. This is an essential concept that I had to keep in mind during the development of my project and that it is was worth a further exposition in the following discussion section.

On top of that, he wanted me to pay attention to the fact that there are already many scripts and plugins available that can be used and adapted for slightly different tasks. In this regard, he mentioned Cellpose³⁰, a generalist algorithm to perform cellular segmentation and Stardist³¹, an ImageJ plugin used for the automatic detection and segmentation of nuclei. The latter is widely used among the research community because he manages to count the nuclei with high sensitivity and accuracy. Later on in this project, this same plugin will be used as a reference to measure the accuracy of the pipeline developed in this project.

PhD candidate – Ilaria Signoria

To accomplish the goal of the project of offering to the research community a useful and valuable product, it was important to define a real research problem and to solve it. In order to do that, I decided to collaborate with Ilaria Signoria, a PhD candidate at the UMCU in the research group of Ludo van der Pol and supervised by Senior researcher Ewout Groen. This collaboration allowed me to work with real research data and to tackle one of the tasks that is frequently required while during research.

Ilaria is studying the cellular mechanism behind the spinal muscular atrophy (SMA) motor neuron disease. In her group, they are performing an experiment to assess the efficacy of different drugs on the production of a cytoplasmatic protein. This protein is marked with a fluorescent dye and so, based on the measurements of the cell fluorescent intensity they can define the concentration of the protein of interest. Ilaria need a standardized analytical pipeline to measure the fluorescence intensity across multiple samples and replicates. The final output of requested in this analysis is a plot

that shows the average level of fluoresce intensity for each treated samples and the controls. To perform this analysis, they require to use an imaging analysis software as ImageJ. Ilaria is not familiar with the software and she is not practical with neither programming nor coding. Building an ImageJ macro would be very complicated for her because she has no experience with the program. However, she felt the need to improve her knowledge and to learn how to use the software to fulfil her analysis. Later in the report, I will talk discuss how I decided to perform the measurement of the fluorescent intensity and I will show the results of the performed analysis.

In addition, this collaboration with a PhD daily involved in research activities helped me in the development of the final Jupyter Notebook. Thanks to her evaluation I understand where it was needed to include more comments and which steps required a better explanation. Her final assessment is reported in the following evaluation section.

Design

In this section, I will introduce the tools employed during this project. The main tool used in this project is clearly the imaging software used to perform the elaboration, processing and measurements and briefly that allows to extract the data from the image. The second most important tool consists of the scripting language Python and in particular of the Python package PyimageJ that represents the essential bridge between Python 3.x and the plugins and macros of ImageJ 1.x. Finally, the essential Jupyter Notebook represent that platform that hosted the analysis while allowing the inclusion of comments and the visualization of the results.

ImageJ

ImageJ is an open software for image analysis. The first version of ImageJ (ImageJ 1.x) includes a series of standards plugins and functions. The second version ImageJ 2.x, written by Curtis Reuden, includes additional functionalities and supports the analysis of multi-dimensional images data overcoming the limitation of the original version of the software. ImageJ2 is incorporated with FIJI that includes a collection of plugins to facilitate the scientific image analysis.

As introduced before, one important tool to automate and replicate processing tasks is the IJmacro scripting. The macro is essentially the sequence of commands demanded to accomplish the task of interest. The simplest way to produce a macro is to adopt the command recorder tool. In this way, every single manual action performed on ImageJ is noted. After recording the sequence of commands, by clicking on the “create” button, a text file is created and directly loaded in the macro interpreter. The macro can be run, manipulated and finally saved as .ijm file. Saving this sequence for the following performing again this task is a simple way to speed up the execution of a repetitive assignment. When a task requires a manual operation (e.g., manually drawing or moving a selection) it is possible to build multiple macros and run them in intervals.

PyimageJ

PyImageJ is a python package that consists of a series of wrapper functions for the integration of ImageJ and Python. The main advantage of using this approach is to combine ImageJ with other tools available from the Python ecosystem, including NumPy, scikit-image, and more. PyimageJ can be installed by using Conda³². Conda is an open-source packages and environments management system that runs on all main three operating systems (Windows, macOS and Linux). Conda is used to find and install packages. It is employed to set up separate environments that can include different versions of Python and multiple package libraries. In this project, a new environment (named “pyimagej”) was created to install the PyimageJ package along with all the other packages that will be used in the notebooks. The environment requires to be activated before calling the Jupyter Notebook. The steps required for the installation of the package and the setting of the environments are well explained on the PyimageJ documentation and they are further explained in the Read.me section of the project.

Jupyter Notebook

The Jupyter Notebook is a web application used to create and share documents containing code, narrative text, visualizations, equations and the code output. It is useful for many applications including data science, statistical modelling and, machine learning. The Notebook works by communicating with computational Kernels that are processes that run interactive code in a particular programming language, in this case, Python 3.x. The Jupyter Notebook is particularly suited for the field of data science. In this regard, the pandas Python package is the one used to import and work with the dataset and the matplotlib and seaborn package is used for the data visualization by plotting graphs and plots of many types.

Implementation

Here I explain the content of the project in words going through the multiple notebooks that constitute the project. In the first notebook “01 - Working with ImageJ pt.1”, I show how to make use of the PyimageJ package by performing tasks of increasing difficulty. This notebook represents the “tutorial” portion of the project that help the user to learn how to employ ImageJ functionalities within the Jupyter Notebook. In the second notebook “02 - Working with ImageJ pt.2”, I show how to perform more complex tasks by combining IJ macros and python structures. In the third notebook “03 - Working with the IPAN module”, I show how to employ the IPAN.py python module that wraps the composed functions (figure 1). The fourth notebook “04 - Nuclei Count” is the demonstrative example of an analyses, consisting the counts of the number of nuclei in multiple images, completely performed on Jupyter Notebooks with a pipeline that use the function from the IPAN.py module. The fifth and last notebook “05 - CTCF Measurements” shows the elaboration and the visualization of the data of the cell fluorescence assays. All the before mentioned Jupyter Notebooks are available at the following GitHub IPAN repository (<https://github.com/NicolasCristini/ImageJ-Processing-Assistant-Notebook>) along with all the input images, the output files and output files and the ImageJ macro.

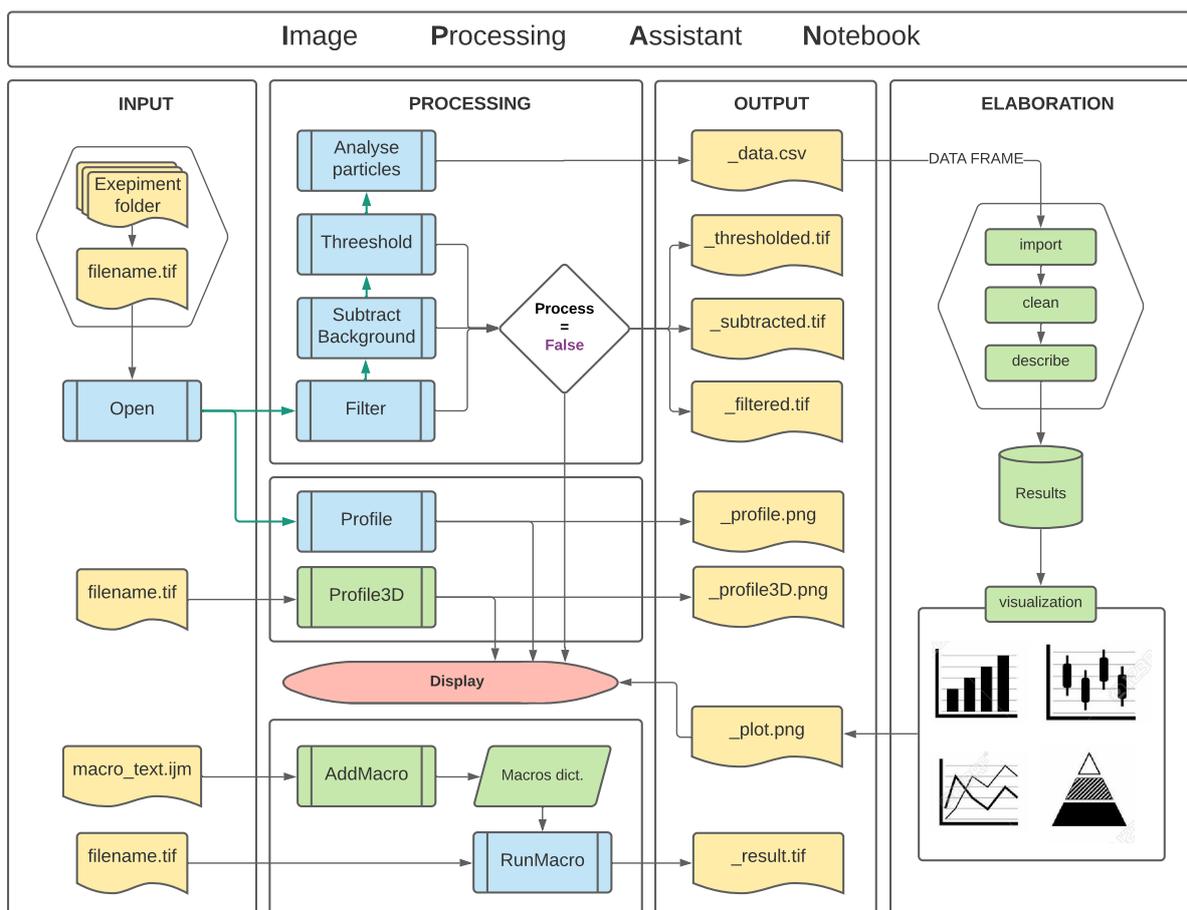


Figure 1. Image Processing Assistant Notebook workflow. The workflow can be divided in four main sections: Input, Processing, Output, Elaboration. Firstly, the raw images are opened, secondly, the image is processed and occasionally the intermediates are displayed to screen, thirdly the output of the processing steps are locally saved and finally, the data are re imported to the final elaboration steps that lead to the visualization of the outcome. Colour legend: in yellow the files, in blue the composed functions including ImageJ functionalities, in green python functions and elaboration steps, in red the visual display.

01 - Working with ImageJ pt.1

In the first Jupyter Notebook of this projects, I provide a tutorial to explain how to work with images within the Jupyter Notebook platform in combination with ImageJ. To work with PyimageJ the first step is to import the package and initialize a version of the software. To do that, there are multiple options available as reported in the table presented in the “PyImageJ-Tutorial.ipynb” notebook³³ (Table 1). The initialization³⁴ of image without arguments imports the latest version of ImageJ from the web. For most of the project, ImageJ has been initialized with the support for Image 1.x that supports the use of ImageJ 1.x plugins.

Requirement	Code ¹	Reproducible? ²
Newest available version of ImageJ	<code>ij = imagej.init()</code>	NO
Specific version of ImageJ	<code>ij = imagej.init('2.1.0')</code>	YES
With a GUI (newest version)	<code>ij = imagej.init(headless=False)</code>	NO
With a GUI (specific version)	<code>ij = imagej.init('net.imagej:imageJ:2.1.0', headless=False)</code>	YES
With support for ImageJ 1.x (newest versions)	<code>ij = imagej.init('net.imagej:imagej+net.imagej:imagej-legacy')</code>	NO
With Fiji plugins (newest version)	<code>ij = imagej.init('sc.fiji:fiji')</code>	NO
With Fiji plugins (specific version)	<code>ij = imagej.init('sc.fiji:fiji:2.1.1')</code>	YES
From a local installation	<code>ij = imagej.init('/Applications/Fiji.app')</code>	DEPENDS

Table 1. Faithfully reported legend:

¹ pyimagej uses jgo internally to call up ImageJ, so all of these initializations are tied to the usage of jgo. You can read up on the usage of jgo to find out more about this initialization.

² Reproducible means code is stable, executing the same today, tomorrow, and in years to come. While it is convenient and elegant to depend on the newest version of a program, behaviour may change when new versions are released—for the better if bugs are fixed; for the worse if bugs are introduced—and people executing your notebook at a later time may encounter broken cells, unexpected results, or other more subtle behavioural differences. You can help avoid this pitfall by pinning to a specific version of the software. The British Ecological Society published Guide to Better Science: Reproducible Code diving into the relevant challenges in more detail, including an R-centric illustration of best practices. A web search for reproducible python also yields several detailed articles.”

Since it represents the first process in any type of analysis, I started by showing how to open an image and then to display it. For this purpose, there are already many alternatives that do not need the use of ImageJ. The function *Image()* imported from the *IPython.display* package allows to directly display both local (providing the file path) and web (providing the URL) images to the screen with the downside that .tif files are not supported. There is then a list of functions that can be used to open an image. This list includes *io.imread()* from the ski-kit package and also *mpimg.imread()* from the matplotlib package. The latter needs to be coupled with the function *plt.imshow()* and *plt.show()* to further display the image. PyimageJ offers an opener function *ij.io().open()* to first open the image and then the command *ij.py.show()* to display the image to the screen. It is also possible to specify in the arguments the image colourmap (e.g., *cmap='gray'*). This one can be used in association with the previously cited function *io.imread()*. Both of them upload the image as an array that contains information about each pixel intensity value. However, by using these commands, the opened image will not be noticed as an opened window on ImageJ.

To open and to work with an image, it is essential to open the image within a macro with the command IJ command *open()*. To run a macro, PyimageJ provide a built-in function *ij.py.run_macro(macro_text)*. The macro allows specifying the input and the output of the macro by giving as one of the arguments a dictionary with the specified variables. The corresponding name of the variable must be explicit at the beginning of

the macro with a specific syntax. The image resulting from the macro will be the active image. After running the macro it is possible to associate the active image to a variable by using the command `ij.py.active_image_plus()` so that it will then be possible to show it to the screen. The results computed within the macro, including strings, images, plots and tables can be saved with the built-in macro functions `save()` and `saveAs()`. As said before, the last selected image will be the one left activated. So, if no other images are opened or selected, subsequently calling a macro affects the previously opened image.

In this notebook, I introduce also the possibility to compute the picture profile by using the command `run("Plot Profile")` within the macro text. This command requires drawing a line on the picture. It then shows the value of the pixel intensity along that line. This function can be useful to understand the pixel values of the image and it can help to better appreciate how the following processing steps affect the image at the pixel level. In the figure below I show the examples of plot profile and the sample image (named "AuPbSn 40") on which it has been calculated (Figure 2).

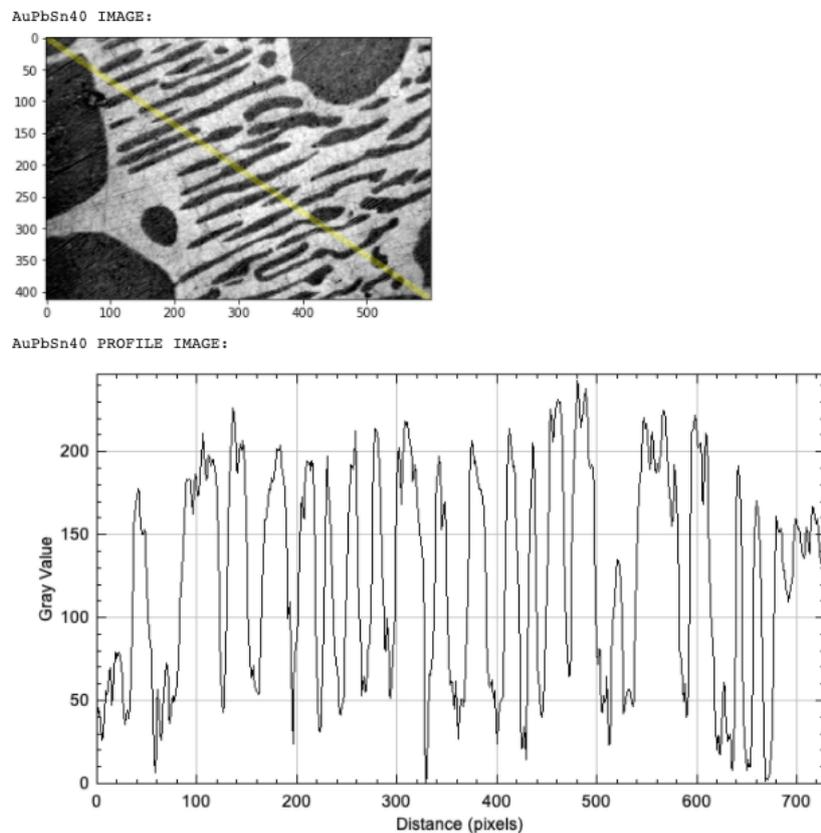


Figure 2. This figure shows the ImageJ sample Image "AuPbSn 40". The profile plot shows the intensity of the pixels computed along the yellow line. The line has been automatically drawn from the upper left corner to the bottom right corner.

02 - Working with ImageJ pt.2

In the second notebook, I work with ImageJ 1.x plugins. To properly use this ImageJ functionality, ImageJ demands to be properly initialized. In particular, it must initiate with the legacy supported that allows running original ImageJ 1.x plugins on the ImageJ 2 gateway for backwards compatibility. After the initialization, it is suggested to check if the legacy is supported by running the dedicated function `ij.legacy.isActive()`. Importantly, the ImageJ initialization can be performed only once

otherwise the legacy will become inactive and the plugins will not work anymore. Later in this notebook, I apply the mean filter plugin with the function `ij.py.run_plugin()`, this function requires to input the name of the plugin and the arguments requested by the plugin itself. The output of the plugin can be associated with a variable, as already done before, and then saved by converting it to a NumPy object and by using the tiff file package. The filtering step is shown in Figure 3.

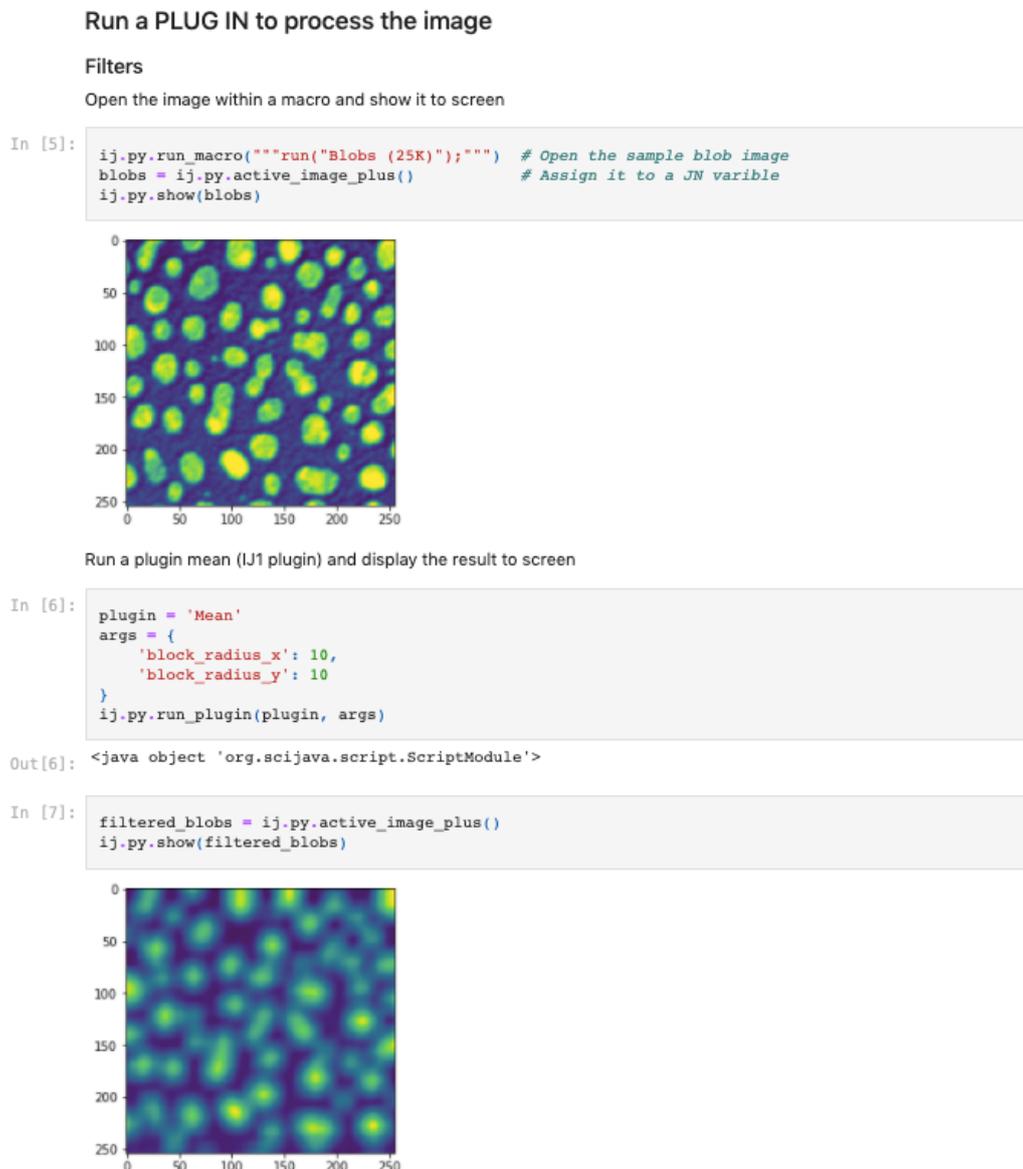


Figure 3. This figure is a screenshot of the Jupyter Notebook where I show how to run the plugin “Mean” on the previously opened ImageJ sample image “Blobs (25K)”.

The following step of this notebook gives detailed examples about how to build a python function by embedding IJ macro text, PyimageJ functions and, supporting python structures including the dictionary that collects the inputs and outputs. The function that shows this example is called *OpenAndProfile()* and does exactly what it stands for. By building this function I demonstrated how it is possible to implement the PyimageJ, ImageJ macro and Python objects in one unique function.

After achieving this milestone, I defined a series of functions that work in sequence.

The first function is the *Open()* function, it imports and shows to screen the image under analysis. Then, the function *Profile()* computes the pixel intensity along the main diagonal, builds a plot and shows it to the screen. The function *Filter()* applies the “mean” filter plugin shown at the beginning of the notebook. The function *SubtractBackground()* applies the “rolling ball” subtracting background algorithm. The function *Threshold()* applies the Default dark threshold, converts the selection to a mask and then uses the command *run(“Watershed”)* to separates overlapping objects in separate regions. Finally, the function *Count()* sets the interested Measurements and uses the command *run(“Analyse particles”)* to get those measurements from the image. The results are saved in the newly created result directory as a .csv file. In the last chunk of the notebook, I finally show how is possible to use pandas to import the .cvs file, create the data frame, clean and tide it and finally summarize it by calling the main descriptive statistics.

03 - Working with the IPAN module

The functions built in the previous notebook demonstrated the possibility to functionally combine the simple ImageJ macros with essential PyimageJ functions and other required Python objects (arguments dictionary for the input and output of the documents) to perform the steps of the image processing. However, as it is visible from the code, those functions are quite extensive.

The following step of the project consisted in building a python module named IPAN.py that works as a container of all these functions and allows to easily import and call them in the Jupyter Notebook. The functions provided in this notebook are ready to use and do not require a particular programming competence. Using these functions allows performing the imaging processing steps that sometimes requires the proper combination of multiple commands. So, the role of these functions is to simplify the ImageJ learning process and also to provide an analytical pipeline that can be used as an example that could assist to perform their analysis.

In this notebook, I show how to use these functions. Firstly, I define a new function called *Count_nuclei()* that embed the functions imported from the IPAN.py module and that are required in the analytical process, those are: *Open()*, *Filter()*, *SubtractBackground()*, *Threshold()* and *Count()*. These functions return the path file of the.csv file that has been saved in the last step of the pipeline. By just calling this function and selecting the file path of the image under analysis, I can generate the result dataset. The workflow is visualized in the map that outlines how to use the IPAN module of this analytical pipeline has been already illustrated in Figure 1 while the actual portion of the notebook is showed in Figure 4 and Figure 5.

One of the potentialities of the IPAN.py module is that it gives to slightly more expert users the possibility to automate their own analysis but just modify the already exciting functions. For example, in order to modify the algorithm used for the thresholding step, he/she would just need to: (1) record the commands from Fiji, (2) open the IPAN.py file, (3) select the *Threshold()* function and change deputed command line of the IJ1 with the one of his/her interest. In this way, any user with a minimum competence in python may produce its IPAN_personalized.py module with the commands suited the analysis and them, save it and import it in the Jupyter notebook and use it exactly as shown in this notebook.

```
In [4]: def Count_nuclei(file):
#Open the image
IPAN.Open(INPUT_filename = file, process = True)

#Process the image
IPAN.Filter(process = True)
IPAN.SubtractBackground(process = True)
IPAN.Threshold(process = True)

#Analyse the particles and save results
data = IPAN.Count(process = True)
IPAN.CloseAll()

return data
```

```
In [5]: %%capture
results = Count_nuclei("image5.tif");
```

Figure 4. The figure shows the construction of the function that embeds the processing and analysing functions imported from the IPAN.py module. The *Count_Nuclei()* represents the analytical pipeline to count the nuclei.

```
In [6]: results
```

```
Out[6]: '/Users/nicolascrismini/IPAN-Project/IPAN/RESULTS/image5_data.csv'
```

```
In [7]: import pandas as pd
Data = pd.read_csv(results, header = 0, sep=',', encoding='latin-1', index_col=0).drop("Label", axis = 1)
Data
```

```
Out[7]:
```

	Area	Circ.	AR	Round	Solidity
1	192.660	0.834	1.279	0.782	0.925
2	214.630	0.753	1.526	0.655	0.920
3	326.593	0.870	1.373	0.728	0.948
4	190.970	0.843	1.471	0.680	0.928
5	215.475	0.846	1.498	0.667	0.943
...
88	379.406	0.870	1.406	0.711	0.948
89	198.575	0.842	1.572	0.636	0.933
90	376.448	0.734	1.487	0.672	0.914
91	193.505	0.826	1.742	0.574	0.930
92	97.598	0.815	1.493	0.670	0.911

92 rows × 5 columns

Figure 5. The figure shows the pipeline returning the results filepathfile that is subsequently given in input to the pandas python package function to import the dataset.

Additionally, the IPAN module includes another important set of functions that would allow running any type of macro text from the notebook. This method represents an alternative function to the already existing *ij.py.run_macro()* proposed by the PyimageJ package. The first function of this set is called *AddMacro()*. This function is a full python function that enables the creation of a library of macros (as a dictionary) by defining the text and the title of the macro. The second is called *MacroRun()* and it represents a functional skeleton that manages the input and the output and exploits the standard *ij.py.run_macro()* function to run a previously added macro by just calling its title and selecting the path of the filename in input. This set of functions is a quite

powerful method because it would allow creating personalized analytical pipelines by combining the text of an IJ macro (with the commands of interest) and the filename in input.

The last of the major functions that have been included in the IPAN.py module is the *Profile3D()*. This one is a full python function that creates a 3D profile plot out of an image. The plot shows the intensity value of each pixel in the matrix. This function makes use of the os package, to deal with the path directories, the NumPy package to work with the matrix image and matplotlib package to finally plot the profile and to show it to the screen. This function can be applied to any 8-bit and squared image. It takes as an argument the file path of the image and automatically save the plot in output. It may be useful to visualize how the computer sees the image, always with the intention to better understand how the processing steps affects the pixels values. (Figure 6).

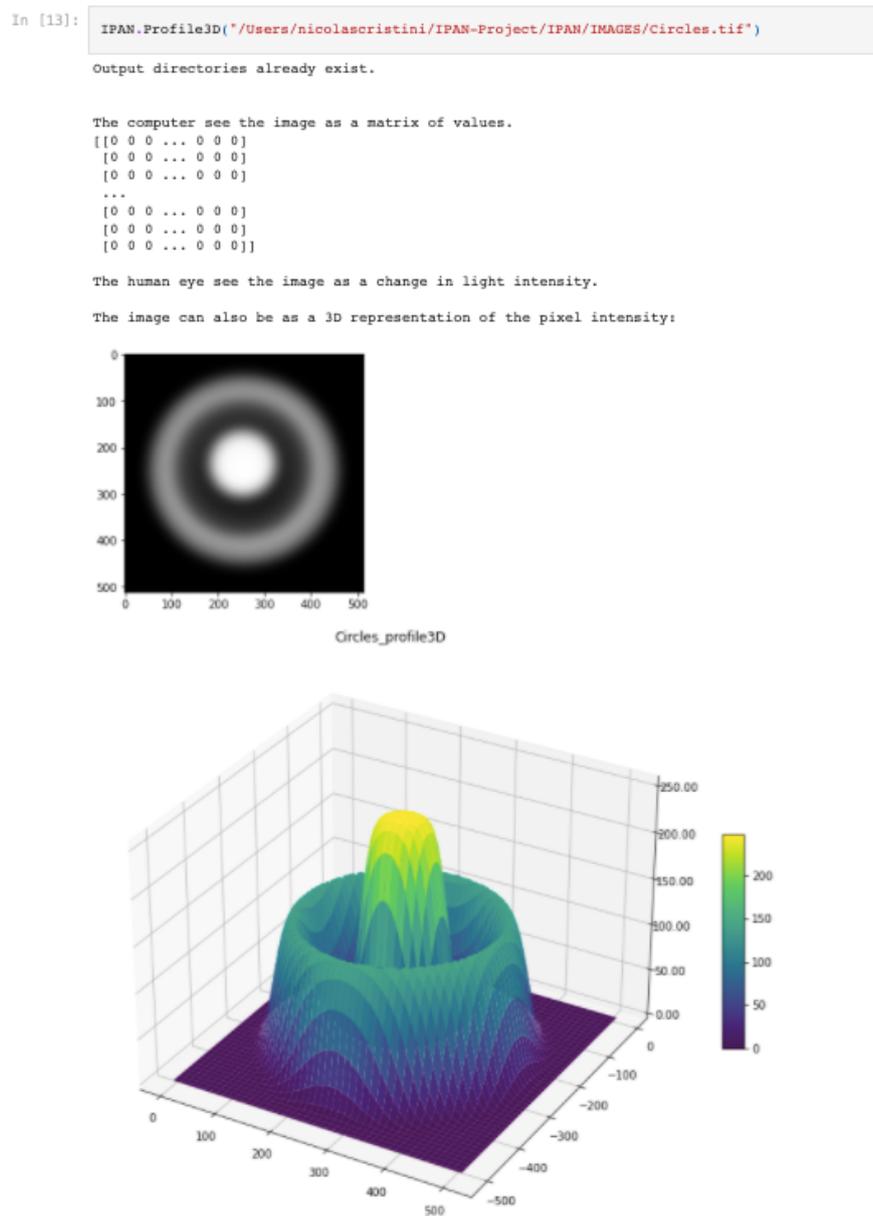


Figure 6. This figure shows the result of *Profile3D()* applied on a simple image that represents concentric circles with different grey intensity. In the image is illustrated the numeric matrix with the intensity value of the pixels, the image as we can see it and finally, the 3D plot of the pixels value.

04 - Nuclei Count

The fourth notebook of this project offers a real example of image analysis. In this notebook I make use of the previously developed analytical pipeline to perform the nuclei counting previously on one image and then on images from multiple samples. In the first case, the analytical pipeline is called only once. The data are managed by using the pandas package and then visualized by using matplotlib to create boxplots and bar plots. In the second case, the analysis is performed on images from multiple samples by calling the analytical function *Count_Nuclei()* within a *for* cycle that iterates on the total number of images present in the experiment folder. The result is directly imported and finally collected in a dictionary that contains multiple data frames containing all the acquired data.

The following extraction of the information is performed by the function named *Describe_data()*. It takes as arguments the dictionary of the results and the name of the feature of interest. The features in the acquired data frames correspond with the measurements previously selected with the *Set Measurements* macro command in the *Count()* function. This step is essential to compute the descriptive statistics that will be plotted in the following phase. The resulting plot shows the comparison of different samples by showing the number of nuclei counted for each sample (Figure 7).

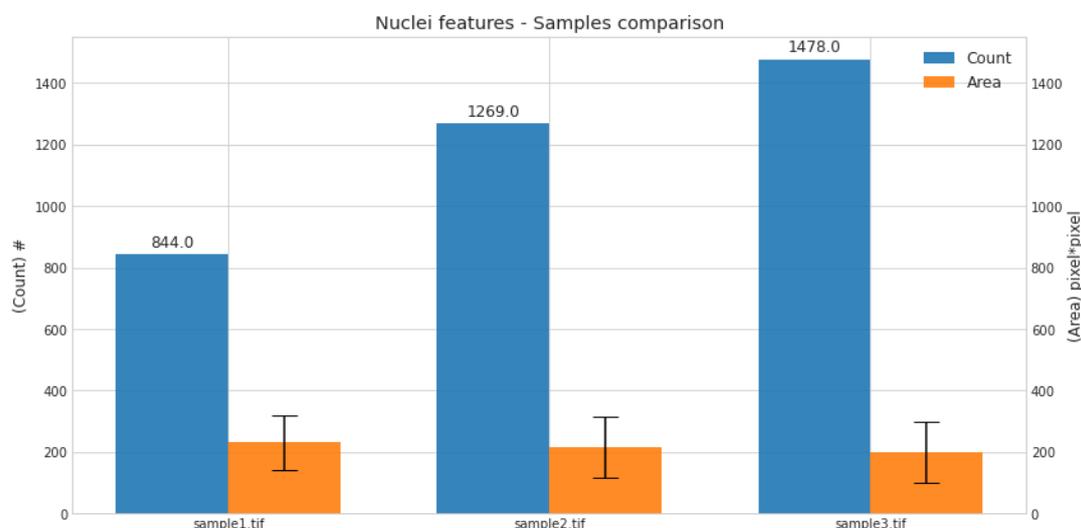


Figure 7. Comparison of the results of the *Count_Nuclei()* performed on three samples. The number of nuclei and the average area of the nuclei is shown as a dodged bar plot.

The power of this notebook is that it serves both as a tutorial that helps during the learning process and also as an assistant because it provides a real example of analysis. In this context, the analysis can be easily applied on different experimental folders. For this last purpose, it is essential that the input data are organized and structured in the same way as the proposed cases.

05 - CTCF Measurements

Previously in the project, I highlighted the importance of developing a product that could be useful for real research problems and that could find a functional application. For this reason, I decided to focus on a quite common task consisting of the measurement of the cell fluorescent intensity. The images used for this analysis have been acquired by the PhD candidate Ilaria Signoria with whom I started this specific collaboration. In this experiment she adopted an immunofluorescence staining to visualize her protein of interest by conjugating it with GFP (green channel), in this experiment the cells' nuclei were stained with DABI (blue channel) (Figure 8). Basic concept of this experiment is that the fluorescence intensity is correlated with the amount of protein produced by the cells. The objective of the experiment is to assess the effect of two different drugs (drug_A and drug_B) on the protein production.

The fluorescent intensity of a cell can be measured by computing the Corrected Total Cell Fluoresce (CTCF). This measurement can be easily computed as shown below.

$$CTCF = \text{Integrated Area} - (\text{area of selection} * \text{Mean fluoresce of background})$$

This procedure is quite simple to perform on ImageJ by manually drawing the region of interest (ROI) around the cell and then acquiring the measurements required in the equation. However, this was not the case for this experiment. The reasons behind this are that firstly, in the resulting raw images the cells were overlapped making it impossible to distinguish the edge of each cell and that secondly, each sample under analysis had multiple acquisitions and so, performing this analysis would have been extremely tedious and time-consuming. These two problems drove me to build an ImageJ macro that could overcome the problem of overlapping cells and allow performing the analysis in a batch processing mode.

The resulted macro is called *CTCF-function.ijm*. It computes the average CTCF for each sample by (1) selecting the total area covered by the cells, (2) measuring the fluorescence over that area on the GFP channel, (3) counts the number of cells by counting the number of nuclei on the DABI channel and finally, (4) computing the average CTCF by dividing the previous total CTCF over the total number of cells.

$$\text{average CTCF} = \frac{\text{Integrated Area ROI} - (\text{area of selected ROI} * \text{Mean fluoresce of background})}{\text{number of cells}}$$

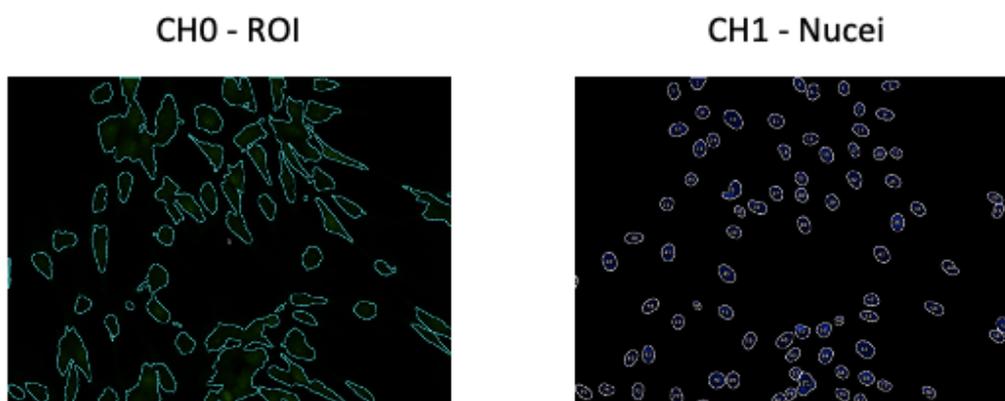


Figure 8. Resulted images from the CTCF-function. The image on the left shows the identified Region of Interest ROI. The image on the right reports the edges and the number of the counted nuclei.

This fifth and final notebook shows how to perform the data analysis consisting in the import, cleaning and visualization steps. The data under analysis were obtained by running the IJ macro “CTCF-function.ijm” that was tailor-made for this analysis. Accordingly to the concept of this project, I intended to automate the acquisition of these measurements by incorporating this macro in the Jupyter Notebook and running in a cycle by using the newly developed functions of the IPAN.py module: *AddMacro()* and *MacroRun()*. However, this macro resulted to be too complicated to be automated through PyimageJ. The reasons behind it are that this macro works by performing multiple measurements and acquiring at different steps specific values from the *Result table* of ImageJ. To complete this analysis, the data have been acquired by simply running the macro from the FIJI macro interpreter. Subsequently, the generated data have been imported and elaborated in the Jupyter Notebook showing how to compute the CTCF values from the raw data.csv files. The result of this notebook is a bar plot that shows the mean and the standard deviation of the CTCF for each of the samples in the experiment (Figure 9).

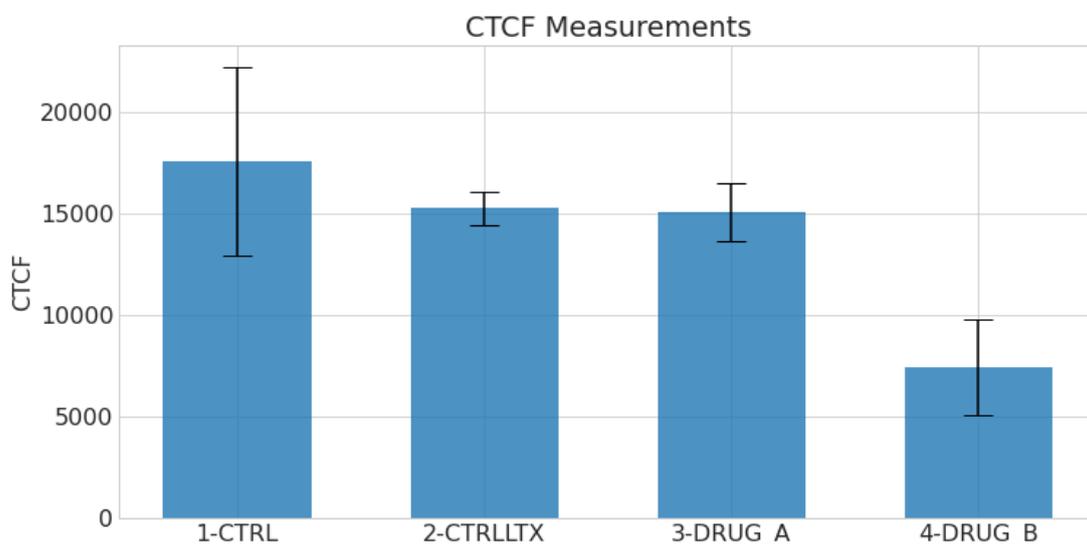


Figure 9. CTCF measurements bar plot shows the different fluorescence intensity in the control and in the samples treated with drug_A or drug_B.

Evaluation

An evaluation of the product of the project is essential to understand if the settled goal has been achieved. In this section, I include an assessment of the final product of the project. Initially, I wanted to measure the time required for ImageJ to be imported and initialized when called from the IPAN.py module. I compared this initialization time with the time required to launch the FIJI application. This gave me an idea of the overhead of importing ImageJ and confirmed that the method employed in this project is not exceptionally slow. Secondly and most importantly, I wanted to assess the accuracy of the constructed analytical pipeline used in the fourth notebook. To do that, I compared the number of particles measured by (1) using the analytical pipeline represented by the *Count_nuclei()* function, by (2) running the same sequence of commands from the macro interpreter and by (3) using the Stardist plugin. The result of this evaluation step is reported below. Finally, to evaluate the usability and functionality of this method, I will report the evaluation comment of the PhD candidate that collaborated in this project.

Overhead of importing ImageJ

For a computer science project that sees the development of a new working method, I retain important to compare the developed method with the techniques of reference. In this context, I wanted to have an idea of the overhead of using ImageJ from Python by measuring the time required to import and initialize the software. The time shown below represents the average time calculated over 7 measurements without considering the highest and lowest measurements. Importing the IPAN.py module from the Jupyter notebook includes importing the required packages and, importing and initializing ImageJ. This step required on average 19.6 seconds. Similarly, launching the latest FIJI version on my personal computer required on average 16.2 seconds. From this data, I can conclude that importing ImageJ from the notebook requires 20% more time than using FIJI from the local machine but that it should not be seen as a limitation for the fulfilment of large-scale data analysis that usually requires much more time.

Methods comparison

Besides the time required for accessing the application, I intended to gather some evidence of the performance and accuracy of the newly developed analytical pipeline: *Count_Nuclei()*. I compared it with (1) an IJ macro including the identical processing steps performed in my analytical pipeline launched from the FIJI interpreter and (2) the plugin Stardist that I choose as a standard tool because of its extensive use in performing nuclei segmentation. The parameters under analysis for this comparison were the execution time required to complete the analysis and, most importantly, the number of nuclei counted in the image. For this evaluation, I used the Image5.tif saved in the IMAGES folder of the project repository. The numerical results of this evaluation are reported in Table 2 while the images that exhibit the results of the different method are showed in Figure 10.

Method	Execution time (seconds)	Number of nuclei (-)
Count_Nuclei()	6.3	99
Stardist plugin	5.4	106
ImageJ macro	0.9	113

Table 2. Results of the evaluation for the comparison of three analytical methods

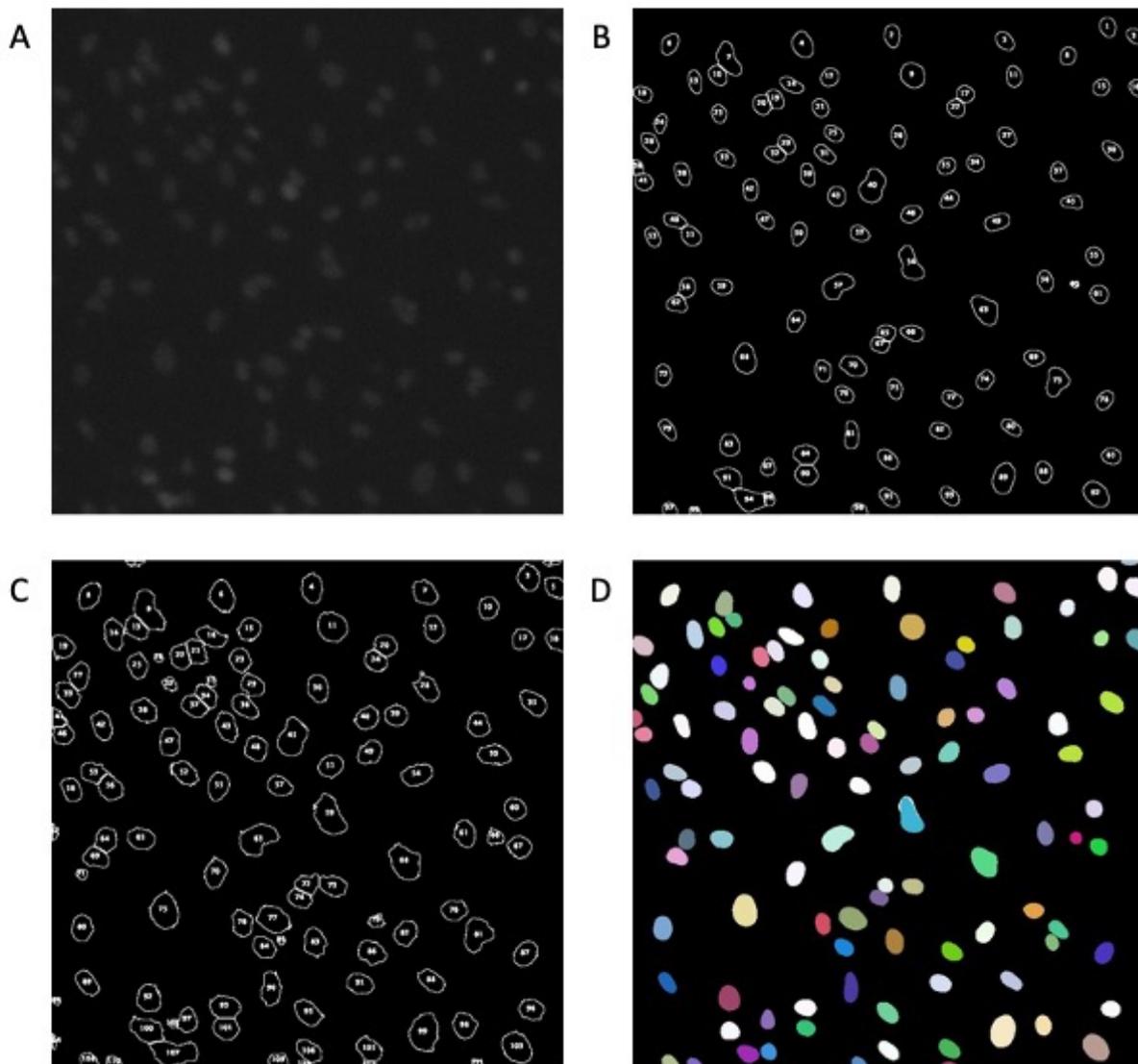


Figure 10. Comparison of the counting methods. (A) Original 8-bit image of cell nuclei. (B) Result of the *Count_Nuclei()* function employing the processing and analysing functions from the IPAN.py module. (C) Result of the IJ macro with the exact same commands launched from the FIJI macro interpreter. (D) Result of the Stardist plugin.

The performed evaluation shows some interesting results. Firstly, we can note that the execution time required from the plugin and the analytical pipeline are similar. However, running the same sequence of commands directly from the FIJI macro interpreter is faster. In this regard, it must be specified that the *Count_Nuclei()* is a complete pipeline that includes saving the resulted .csv file. Moreover, this pipeline can save all the intermediate images of processing steps. On the contrary, Stardist

performs the mere segmentation of the objects and to further acquire the wanted measurements it is necessary to additionally use the *measure* command on the ROI manager.

The result that raises most of the interest is the number of nuclei counted with the three different methods. The analytical pipeline is the one that measures the fewer nuclei but by looking at the results I can say that it is comparable to the result obtained when performing the segmentation with the Stardist plugin. In this case, the plugin is more sensitive to overlapping objects and it is a factor that increases the number of counted nuclei.

The data that concern me the most is the one acquired when running the IJ macro with the same commands. If a different execution time was somehow acceptable, I would not expect to obtain a different result in the count. These results need to be further investigated and so I compared each of the intermediated resulted images to understand which was the step causing this divergence. At the end of the investigation, it was possible to conclude that the variability of the results was due to a difference in how the “mean” filtering plugin processed the images. By looking at the intermediate resulting images I discovered that some smaller objects were present in the thresholded image of the IJ macro while they are absent in the thresholded image resulting from the *Count_Nuclei()* pipeline.

Taking into consideration this difference in the results, it is important to highlight the necessity to perform the analysis always in the same way even if the difference may be little. In this context, sharing an open tool that allows for the registration and storage of its analysis is a good tool for the further standardization of the results of the analysis. All in one, the coefficient of variation computed over these measurements is around 0.05.

[Comment about CTCF macro](#)

Finally, I wanted to focus the attention on the last notebook of the project “05 - CTCF measurement” and the newly developed *CTCF-function.ijm*. To evaluate the utility and functionality of these tools, I asked the PhD candidate Ilaria Signoria to adopt the ImageJ macro to measure the cell fluorescence in another experiment. Moreover, I asked her to download the project repository and to perform the data elaboration of her analysis by using the “05-CTCF measurement” notebook. Below there is her comment:

“I had to start doing image analysis with ImageJ for a side project of my PhD. Having never used it before, I encountered several problems inefficiently performing analysis on a large number of images and conditions. Since I’ve never used Phyton I had some issues during the installation of all the packages. However, once understood how to proceed, I was able to use the notebook for the elaboration of my results. In this way, I managed to decrease the time and effort needed to perform the analysis. Moreover, I do have not a standardized procedure that will help me to get fully comparable results. I think that both the IJ macro and the notebook for the elaboration of the results can be particularly useful for all the researchers that need to automatically measure the fluorescence of their cells but have no particular knowledge about the ImageJ software.”

Discussion

Before in the text, I mentioned the importance to consider the ratio between the amount of time required to code a function and the amount of time that will be saved by using that function instead of manually calling the single commands also based on how often that specific task is going to be performed. In the context of this project, I retain that this ratio is highly favourable because all the written scripts have been shared online and they are available to others to use. In this way, the time required by me to compose these functions will be, hopefully, compensated by the time saved by the number of people that will make use of this product. In this regard, I intend to propose this software within my network so that I could already reach a small, but a consistent number of researchers that can use or take inspiration from my job.

Another discussion point that can have a positive weight on the importance of this project consists in the impelling need to increase the reproducibility of the analysis across the research community. In the last years, data analysis has become more complex and flexible because of the growing possibility to include analytical steps. However, the increasing flexibility results in variability in the outcomes affecting the scientific conclusion that can be drawn³⁵. This aspect is a big downside that could be faced by allowing the open peer-review of the analytical step involved in the research. Increasing communication and comparison within the researchers can only boost the production of more reliable results. Moreover, the open confrontation of both methods and results is at the base of scientific advancement.

In the context of the image analysis, there is a great ongoing communication going between developers of PyimageJ that are trying to resolve issues. This Gitter chat and platform was useful to me to solve some of the problems encountered during the programming of the module. Moreover, the Image.Forum revealed to be an extensive source of solutions. On both these platforms, I had the opportunity to answer and comment on some threads. Noteworthy is a thread started as a GitHub issue³⁶ about running an ImageJ plugin in headless mode from ImageJ. I also received a comment by Nicholas Schaub, neuro material scientist at the National Center for the Advancement of Translational Science (Rockville, MD). He wrote that in his team, they come to the same solution represented by opening image files by using a macro. Nevertheless, this solution represented to them also an issue because they were required to open their files using an image reader able to capture the metadata of the analysis. This topic has been brought to the attention of the ImageJ maintainancers who are working on the publication of an improved version of the software. In conclusion, the work done in this project managed to cover a topic of interest in the image analysis field. Additional work will be carried out to improve the interoperability of image and data analysis platforms.

Conclusion and Future Perspectives

In this project, parts of the proposed goals were achieved. The first two tutorials effectively provide a working example that can be used to learn how to use PyimageJ in a more complete and complex way. The creation of the IPAN.py module is a good but still quite simplistic example of the functional combination of ImageJ1.x and Python 3.x. However, the module has much space for improvements, in particular in its architecture and accessibility. Nevertheless, it accomplished to provide a method to construct and run ImageJ-based analytical pipelines in the Jupyter Notebook. Finally, the last two notebooks contain a real example of data manipulation and visualization that can be adapted for further analysis.

From the evaluation of the project, it resulted that running the same analytical pipeline by using the IPAN.py module and from the FIJI application gives different results. It is not completely clear the reasons behind this inconsistency and furthermore, it cannot be stated which of the three methods is the most correct one. To actually define the real and true number of nuclei in the image, the only alternative is to perform a manual count. However, that would include the possibility of a human error including also the subjectivity of the counter. In this regard, a further investigation of this problem is required.

For what concerns the usability of the product, the comment reported in the evaluation section highlights the need to give additional instructions to the users. To accomplish that, it could be very helpful to provide a more comprehensive text-based tutorial and alternatively also a video tutorial. This kind of resources are particularly necessary if we want to reach a target of users that are not familiar with ImageJ and/or Python. Moreover, sharing additional tutorials and guide sheets may also represent an opportunity to share this project with more people and build interest in the topic.

In conclusion, this project represented for me a great opportunity to enhance my knowledge in the field of biomedical imaging and also to prove to myself that I was able to functionally exploit what I learned in the field of the data science. I conclude by saying that I accomplished my intention to provide the research community with a product that could be seen as a checkpoint for both students, researchers and developers. Moreover, I demonstrated the possibility to realize and build something from an idea originated by a necessity. The willingness to develop a new tool was converted into a real project and finally end up with an exciting product.

Layman's summary

In the biomedical field, many experiments make use of microscopic techniques for studying cellular characteristics. The images acquired with this type of equipment undergo a series of processing steps to obtain the results needed to draw conclusions and to answer to the original research question. This step requires to use a software for processing and analysing scientific images to extract data from the image. For this purpose, there is one software broadly employed by the scientific community, ImageJ, that currently comes within a software package called FIJI. This software is widespread across the community because of its broad utility and vast functionalities. ImageJ is an open-source software, many developers contribute to the improvement of the software capabilities by the integration of plugins for the most various assignments. One downside of this software is its low accessibility for not experienced users including both students and researchers. In the context of biomedical research field, the type of image analysis may vary from the simplest task to a quite complex and articulated sequence of processing steps that constitute one specific analytical pipeline. In the latter case, and in particular, when the amount of data under analysis is considerably large, acquiring considerable results while maintaining the fairness and the reproducibility of the analysis may result complicate and expensive in terms of time and energy. FIJI already provides some tools to assist the automatization of the analysis on multiple files. However, this is an advanced feature that requires a certain level of expertise. In this regard, getting acquainted with the software functionalities is quite consuming and demands some effort. The goal of this project was to provide a series of assistant and tutorial notebooks that could be used to guide the user through some important processing steps of the image analysis. In addition to that, an objective of the project was also to show the feasibility to use the ImageJ functionalities within Python-based functions by employing the PyImageJ package. The purpose of this last goal was to provide an example of automatization of the image analysis throughout a scripting language that has already been demonstrated to be particularly useful in the field of the data science. Combining these two tools represented a great example that showed the possibility to wrap the multiple analytical stages including processing, measurements, data import, data wrangling and visualization all in a notebook by completely embodying the concepts of reproducibly and accessibility. This project wanted to face some common assignments that can be encountered while doing research. For this reason, multiple interviews were performed to firstly, gain insights on aspects of the software that demanded attention and secondly, to develop a product while addressing a real research problem. Finally, an evaluation procedure was performed to understand the level of efficiency and usability of the product. In conclusion, the combination of ImageJ and Python was demonstrated to be feasible although multiple complications. A further investigation about the accuracy of the product is still demanded to ensure the reliable employment of this method.

Acknowledgements

I desire to acknowledge my supervisors for the opportunity given to me. I want to thank in particular my daily supervisor dr. Y. (Eugene) Katrukha for giving me advice and tips on how to proceed with my work. I want to express my thanks to my main supervisor dr. A.L. (Anna-Lena) Lamprecht for helping me out in building this project from scratch and guiding me during the development process. I would also like to acknowledge dr. M.W. (Mel) Chekol for being available for my supervision during the last part of the project. Finally, I want to thank dr. ir. Edwin Bennink for being willing to cover the figure of the second examiner for the review of the project. Last but not least, I am thankful to all the people who contributed to this project by sharing their experience and knowledge.

Bibliography

1. Castleman K. Digital Image Processing. *Prentice Hall, Up Saddle River*. Published online 1995.
2. Scott DH, John FC, Rivero R, Baumes L. High Throughput Experimentation Drives Better Outcomes. *Technology*. 2010;lab Manage:36-37. <https://www.labmanager.com/big-picture/lab-automation-benefits/high-throughput-experimentation-drives-better-outcomes-24503>
3. Rueden CT, Eliceiri KW. The imageJ ecosystem: An open and extensible platform for biomedical image analysis. *Opt InfoBase Conf Pap*. 2018;Part F89-M(608):518-529. doi:10.1364/MICROSCOPY.2018.MTh2A.3
4. Burger W, Burge M. Principles of Digital Image Processing. *Springer, New York*. Published online 2010:1-221.
5. Abràmoff MD, Magalhães PJ, Ram SJ. Image processing with imageJ. *Biophotonics Int*. 2004;11(7):36-41. doi:10.1201/9781420005615.ax4
6. Sunoj S, Igathinathane C, Saliendra N, Hendrickson J, Archer D. Color calibration of digital images for agriculture and other applications. *ISPRS J Photogramm Remote Sens*. 2018;146:221-234. doi:10.1016/J.ISPRSJPRS.2018.09.015
7. Rueden CT, Schindelin J, Hiner MC, et al. ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics*. 2017;18(1):1-26. doi:10.1186/s12859-017-1934-z
8. Collins TJ. ImageJ for microscopy. *Biotechniques*. 2007;43(1S):S25-S30. doi:10.2144/000112517
9. Cutler Z, Kiran G. R i a w. Published online 2021.
10. Mutterer J, Rasband W. ImageJ Macro Language Programmer ' s Reference Guide v1.46d. *RSB Homepage*. Published online 2012:1-45.
11. Eclipse plugin. <https://marketplace.eclipse.org/content/imagej-plugin>
12. Bio7. <http://bio7.org>
13. Kametsky L, Jones TR, Fraser A, et al. Improved structure, function and compatibility for cellprofiler: Modular high-throughput image analysis software. *Bioinformatics*. 2011;27(8):1179-1180. doi:10.1093/bioinformatics/btr095
14. VanderPlas J. *Python Data Science Handbook*.; 2016.
15. Jython. <https://www.jython.org/jython-old-sites/archive/221/archive/22/>
16. Travis E. Oliphant. Guide to Numpy. Published online 2006. <https://archive.org/details/NumPyBook/mode/2up>
17. Wang A, Yan X, Wei Z. ImagePy: An open-source, Python-based and platform-independent software package for bioimage analysis. *Bioinformatics*. 2018;34(18):3238-3240. doi:10.1093/bioinformatics/bty313
18. Jupyter Notebook. <https://jupyter.org/>
19. Pyimagej - GitHub. <https://github.com/imagej/pyimagej>
20. scikit-image. <https://scikit-image.org/docs/stable/api/api.html>
21. McCormick M, Liu X, Jomier J, Marion C, Ibanez L. ITK: enabling reproducible research and open science. *Front Neuroinform*. 2014;8. doi:10.3389/fninf.2014.00013
22. Bradski G. The OpenCV Library. *Dr Dobb's J Softw Tools*. Published online 2000.
23. BeakerX. <http://beakerx.com/>
24. Gitter. <https://gitter.im/imagej/pyimagej>
25. IMB Microscopy. ImageJ video workshop. <https://www.youtube.com/playlist?list=PLkGKyUslzfzobHBGYnwVCtWPFx9QxZ57f>
26. Napari. doi:10.5281/zenodo.5587893
27. von Chamier L, Laine RF, Jukkala J, et al. Democratising deep learning for microscopy with ZeroCostDL4Mic. *Nat Commun*. 2021;12(1):1-18. doi:10.1038/s41467-021-22518-0
28. Figshare. <https://figshare.com/>
29. GitHub. <https://github.com/>

30. Stringer C, Wang T, Michaelos M, Pachitariu M. Cellpose: a generalist algorithm for cellular segmentation. *Nat Methods*. 2021;18(1):100-106. doi:10.1038/s41592-020-01018-x
31. Schmidt U, Weigert M, Broaddus C, Myers G. Cell Detection with Star-Convex Polygons. In: ; 2018:265-273. doi:10.1007/978-3-030-00934-2_30
32. Conda. <https://docs.conda.io/en/latest/>
33. PyImageJ Tutorial. <https://github.com/imagej/pyimagej/blob/master/doc/PyImageJ-Tutorial.ipynb>
34. PyImageJ Initialization. <https://github.com/imagej/pyimagej/blob/master/doc/Initialization.md>
35. Botvinik-Nezer R, Holzmeister F, Camerer CF, et al. Variability in the analysis of a single neuroimaging dataset by many teams. *Nature*. 2020;582(7810):84-88. doi:10.1038/s41586-020-2314-9
36. Github issue. <https://github.com/imagej/pyimagej/issues/126>