Functional Image Processing with ImageJ/FIJI

Kyle I. S. Harrington, Timothy S. Stiles*, Lakshmi Venkatraman, Claudia Prahst, Katie Bentley
Center for Vascular Biology Research
Beth Israel Deaconess Medical Center
Harvard Medical School
Boston, MA, USA
kharrin3@bidmc.harvard.edu

*School of Natural Science, Hampshire College, Amherst, MA, USA

Abstract—We present Funimage, a tool for developing image processing pipelines using the functional programming language Clojure built upon ImageJ and FIJI. Image processing pipelines developed with Clojure can be rapidly prototyped and automatically programmed with metaprogramming techniques. Three usage examples are presented on two types of biological images.

Index Terms—Clojure, Functional programming, ImageJ, FIJI

I. Introduction

ImageJ has become a standard tool for biological image processing [1]. While there have been multiple extensions that wrap around ImageJ, such as Icy [2] and CellProfiler [3]; FIJI [4] is one of the most well known and comprehensive of such ImageJ wrappers. Although there is an abundance of ImageJ plugins for handling various imaging modalities and analytical techniques, one of the most powerful features of ImageJ and FIJI is the ability to write macros and scripts. Macros and scripts allow researchers to quickly design custom image processing procedures, which can be reused across multiple images and data sets. This is particularly true in FIJI, where the incorporation of SciJava has vastly extended the set of programming languages in which scripts can be written. Here we focus on the functional programming language, Clojure [5]. Clojure is a Lisp language that facilitates meta-programming, allowing developers to easily write programs that write programs, and high-level programmatic manipulations commonly encountered in machine learning and artificial intelligence.

The Lisp (**LIS**t **P**rocessing) family of programming languages [6] has the second oldest history of all high-level languages, preceded only by Fortran. Development of the language was heavily motivated by the founding of the field of artificial intelligence (AI), and it has been used extensively within the field of AI. Lisp programs are easily recognized by their Polish prefix notation, which leads to a large number of parenthesis delineating program scope with closures. Lisp languages are often described as the most elegant and beautiful programming languages. This is because through the minimalistic language definition, a great deal of high-level power can be achieved. The most fundamental of these high-level features in Lisp is homoiconicity, whereby functions are first-class data types, allowing programs to create and manipulate functions during run-time.

Biological image processing is strongly rooted in the use of customized image pipelines. Images for certain experiments, equipment, and even scientists require unique processes for image analysis. Image pipelines are implemented in a variety of ways, including ImageJ macros, Matlab scripts, CellProfiler pipelines, shell scripts, and programs in C++, Java, Fortran, Python, etc.. All of these implementations are code of one kind or another. In this paper we present Funimage, a system for image processing built on ImageJ and FIJI that exploits the functional programming language, Clojure, for elegant high-level development.

II. Funimage

ImageJ and FIJI offer a broad set of features via core functions and plugins, making them an excellent option for building upon. As we have mentioned, lisp languages have a number of high-level features that not only allow developers to abstract away low level details but also write programs that write programs, meta-programming. The prospect of meta-programming is particularly appealing in the context of designing image processing pipelines, where customized procedures are often essential for small sets of images, but there is a high degree of code overlap between procedures. Clojure is currently one of the most actively developed lisp language, with a vibrant community of developers, and it has been incorporated into FIJI via the SciJava project. Clojure was originally designed to operate within the Java Virtual Machine (JVM), and was designed to embrace compatibility with Java through interop functions. In fact, any Java code can be used from Clojure with no modification required. This is particularly significant when building upon ImageJ/FIJI, which offer a wide-range of functionality already

implemented in Java. Funimage is open-source and publicly available[1].

In this paper we present the initial version of Funimage, which focuses on support of ImageJ data structures, such as ImagePlus, ImageProcessor, ImageStack, ROI, and more. We begin by introducing some example code:

```
(def imp (open-imp "RatBrain_ROI-1_DAPI.tif"))
(def mask (threshold (copy-imp imp) 25))
(show-imp mask)
```

These three lines open, threshold, and display an image using ImageJ data structures. Developers of FIJI scripts will be familiar with this type of code, and a programming guide for using Clojure within FIJI is already available [7]. It is important to highlight that Funimage extends beyond a recapitulation of the Clojure scripting guide, by providing type-hinted functions for faster processing, bindings to ImgLib2 [8] data structures, in addition to *de novo* functionality. Code written with Funimage can be run in a stand-alone manner. In other words, programs written in Funimage are complete pieces of software in and of themselves, distinct from ImageJ scripts written in Clojure which must be executed within ImageJ. However, Funimage programs can still be used within ImageJ, making it an appealing way to develop image processing code that needs to be distributed to users who are already familiar with ImageJ.

FIJI provides facilities for using high-performance computing clusters via the Archipelago plugin. This plugin allows users to launch clients on computing nodes and a master process on a central machine, and then distribute jobs over multiple machines. This approach to high performance computing (HPC) cluster usage offers some crucial benefits, such as the development of image processing pipelines that incorporate distributed computing. However, when not performing image processing operations in a distributed manner, this may not necessarily be the simplest or most desirable option for the average HPC user. Many image processing developers simply desire the ability to perform tasks in parallel, such as applying an image processing pipeline to a set of images.

To this end we exploit two Clojure tools: Leiningen, a package management tool for Clojure, and Brevis [9], a functional tool for scientific and artificial life research. Scripts developed in Funimage are, in all senses, normal stand-alone Clojure programs that do not require FIJI to be running. Using Funimage in conjunction with Brevis, researchers can simply execute a command on their local machine in order to perform a parallel task on an HPC as follows:

```
(start-run-array ["RatBrain_ROI-1_DAPI.tif"
"RatBrain_ROI-1_DAPI.tif" "RatBrain_ROI-
1_DAPI.tif"] funimage-bioimage.nucleus-counter
MyUsername hpc.myinstitution.org)
```

[1] https://github.com/funimage/funimage

This command, available through Brevis, will transfer the local Clojure project containing "funimage-bioimage.nucleus-counter" to the user's cluster, and launch a sequence of jobs parametrized by image filename. The prerequisites for this form of parallel computing are that the user has configured SSH keys on the cluster, and Leiningen is available on the cluster. This functionality has been successfully tested on clusters at multiple institutions running Sun Grid Engine and LSF cluster management systems.

III. Example Applications

Here we present a few examples of Funimage being used on real-world data. First, we begin with an example of nucleus detection using data from the 2015 Bioimage Nucleus Counting competition. Second, we detect and measure filopodia from an *in vitro* assay of HUVEC cells cultured on a 2D collagen matrix, and an *in vivo* sample of endothelium from mouse hindbrain. Finally, we demonstrate the visualization capabilities of Funimage on both the nucleus and filopodia detection data.

A. Nucleus Detection

Recursive convolution has proven to be useful for automatically generating image processing pipelines [10]. A single kernel recursively convolved with varying depth over an image can serve as a basis for naive search methods. Machine learning and search methods such as convolutional-

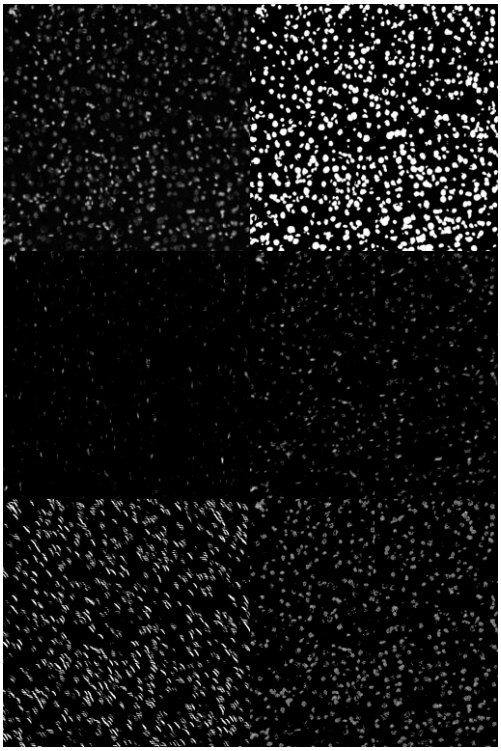

Fig. 1. An example of nuclei detection by searching through recursive convolution parameters. Top-left, original image; top-right, target mask; Remaining 4 images are approximations of the target mask increasing in accuracy from left to right, and top to bottom.

recursive neural networks can be used to generate, pool, and cluster features to produce novel image processing classifiers [10]. Given a target image, recursive convolution can be used within image processing pipelines to transform an input image into a resulting image that is close to the target image. Kernels used for such recursive convolution can even be discovered with naive search methods, such as Monte Carlo-type hill climbing. As a visual demonstration Fig. 1a is an original input image of nuclei from the 2015 Bioimage Nucleus counting competition. Fig. 1b is a hand-processed target used during hill climbing search. In Figs. 1c-f are images that were found to minimize the distance to the target image in order of increasing accuracy.

B. Filopodia Detection

In these examples we detect cellular protrusions, called filopodia, in a maximum Z-projection of a confocal image stack of cultured HUVEC cells plated on a 2D collagen matrix, as well as an isolectin-B4 stained wild-type mouse hindbrain (embryonic day 13.5). Here we use an algorithm inspired by [11]. Consider the *in vitro* HUVEC image as an example, the input image (Fig 2a) is thresholded and holes are filled to obtain a mask. Morphological operations are then performed to extract the central body of the cells. Upon obtaining the central body, image subtraction can be used to obtain an image containing only the filopodia. This is accomplished with the following code:

```
(def mask (fill-holes (threshold imp 7)))
(def body (extract-body mask num-iterations))
(def filopodia (imp-subtract mask body))
(def output (merge-imps filopodia body (create-
imp-like body)))
```

In Fig 2b we show the output image, which is a colored merging of the filopodia and body images. In Figs 2c-d we show an *in vivo* example, where additional preprocessing steps of despeckling are necessary due to additional background noise and greater morphological variability. Furthermore, the combination of noise and loss of information from using a 2D projection of a complex 3D structure lead to additional disruptions in extracted filopodia. For these reasons Funimage extensions are being made available to facilitate 3D analysis of morphological structures.

C. Visualization

Funimage can also be used to create images for visualization. In this section we present three examples, one where nuclei are color coded based upon density, and two where the filopodia extracted in section B are color coded based upon length. This functionality is accomplished in two steps, first the ImageJ particle analyzer is used to extract ROIs of segments and a mask is created with segments being assigned their respective measured values. Then, the lookup table (LUT) color coding functionality of ImageJ is utilized to assign colors to segments. This same procedure applies for both cases, where only the measuring method differs. nuclei color-coded by neighborhood density are shown in Fig 3a, filopodia color-coded by length are shown in Fig 3b and Fig 3c. The BAR plugin is used to create the corresponding legends in all figures.

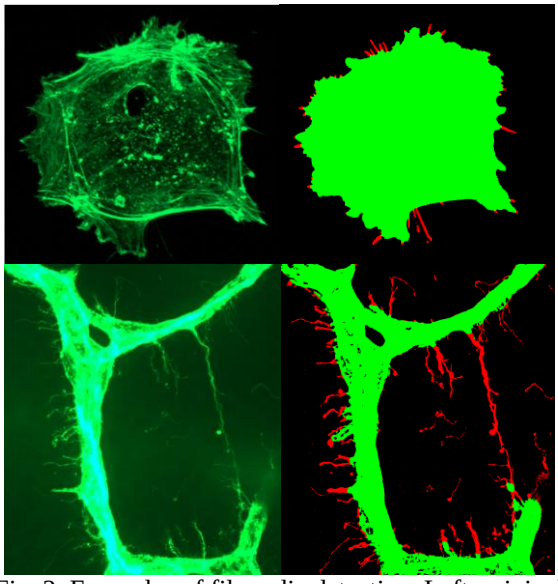

Fig. 2. Examples of filopodia detection. Left, original maximum Z-projection of confocal image; right, segmented cell mass (green) and filopodia (red). Top, *in vitro* cultured HUVEC cells; bottom, *in vivo* mouse hindbrain endothelium.

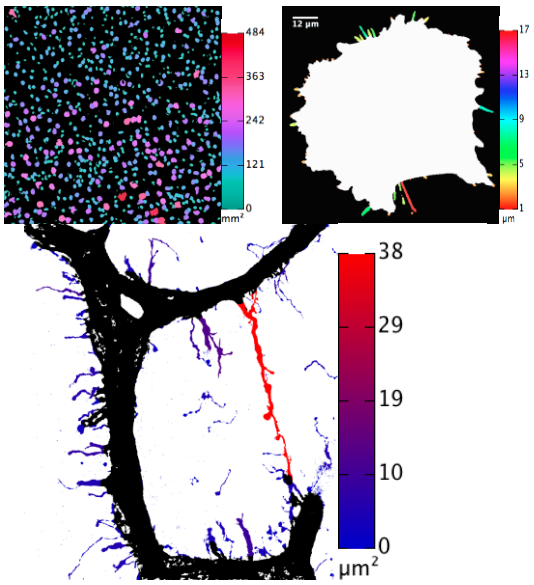

Fig. 3. Visualization of measured attributes with Funimage. Top, filopodia color coded by length; bottom, nuclei color coded by neighborhood density.

IV. Conclusion

In this paper we have introduced the Funimage platform for developing image processing tools. As we have noted, Funimage utilizes both ImageJ and FIJI, and can be used to create both ImageJ plugins and stand-alone image processing programs in the functional programming language Clojure.

This allows developers to rapidly prototype, test, and deploy customized image processing pipelines. While many existing pieces of software for biological image analysis are geared towards experimental biologists, Funimage is designed with the developer in mind, reducing development time and maintaining flexibility. We have used Funimage in analysis of multiple types of biological image data (confocal, immunohistochemistry stained serial sections, and lightsheet). Funimage is actively being used in projects across multiple institutions, and a wide range of scales, demonstrating the broad applicability of functional programming for image processing and analysis.

Acknowledgment

Kyle I. S. Harrington is supported by institutional training grant T32 HL07893 from the NHLBI of the NIH. Timothy Stiles is supported by the NSF-sponsored Four College Biomathematics Consortium, Ray and Lorna Coppinger Grant, FPR-Hampshire College Program in Culture, Brain, and Development, and the Dr. Lucy McFadden grant.

References

[1] Schneider, C. A., Rasband, W. S., & Eliceiri, K. W. (2012). NIH Image to ImageJ: 25 years of image analysis. *Nature methods*, *9*(7), 671-675.

[2] De Chaumont, F., Dallongeville, S., Chenouard, N., Hervé, N., Pop, S., Provoost, T., ... & Olivo-Marin, J. C. (2012). Icy: an open bioimage informatics platform for extended reproducible research. *Nature methods*, *9*(7), 690-696.

[3] Jones, T. R., Kang, I. H., Wheeler, D. B., Lindquist, R. A., Papallo, A., Sabatini, D. M., ... & Carpenter, A. E. (2008). CellProfiler Analyst: data exploration and analysis software for complex image-based screens. BMC bioinformatics, 9(1), 482.

[4] Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., ... & Cardona, A. (2012). Fiji: an open-source platform for biological-image analysis. *Nature methods*, *9*(7), 676-682.

[5] Hickey, R. (2008). The clojure programming language. In *Proceedings of the 2008 symposium on Dynamic languages* (p. 1).

[6] Steele, G. L. (1990). *Common LISP: the language*. Digital press.

[7] Cardona, A.. http://fiji.sc/Clojure_Scripting

[8] Pietzsch, T., Preibisch, S., Tomančák, P., & Saalfeld, S. (2012). ImgLib2—generic image processing in Java. *Bioinformatics*, *28*(22), 3009-3011.

[9] Harrington, K.. Brevis (Version 0.9.86), http://brevis.us/

[10] Socher, R., Huval, B., Bath, B., Manning, C. D., & Ng, A. Y. (2012). Convolutional-recursive deep learning for 3d object classification. In *NIPS* (pp. 665-673)

[11] Nilufar, S., Morrow, A. A., Lee, J. M., & Perkins, T. J. (2013). FiloDetect: automatic detection of filopodia from fluorescence microscopy images. *BMC systems biology*, *7*(1), 66.