JIPipe: Visual batch processing for ImageJ

Marc Figge (Thilo.Figge@leibniz-hki.de)
Leibniz Institute for Natural Product Research and Infection Biology https://orcid.org/0000-0002-4044-9166

Ruman Gerst
Leibniz Institute for Natural Product Research and Infection Biology https://orcid.org/0000-0002-0723-6038

Zoltan Cseresnyes
Leibniz Institute for Natural Product Research and Infection Biology

Article

Keywords:

Posted Date: May 11th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1641739/v1

JIPipe: Visual batch processing for *ImageJ*

Ruman Gerst[1,2,#], Zoltán Cseresnyés[1,#], Marc Thilo Figge[1,3,*]

[1] Applied Systems Biology, Leibniz Institute for Natural Product Research and Infection Biology – Hans Knöll Institute (HKI)

[2] Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany

[3] Institute of Microbiology, Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany

[#] These authors contributed equally

[*] Correspondence should be addressed to thilo.figge@leibniz-hki.de

Abstract

The continuous development of new microscopy techniques requires the parallel evolution of image analysis workflows. *ImageJ* provides a high level of accessibility to bioimage processing, which is still impeded by the necessity of developing scripts to achieve reproducibility, and to comply to the FAIR principles. We provide a visual language termed *JIPipe* that allows the construction of an *ImageJ* workflow purely by designing a flowchart. We already included over 1000 functions from *ImageJ* and its plugins. In return, *ImageJ* is extended with custom-designed algorithms, thus forming a symbiotic relationship with *JIPipe*. Our software includes a fully reproduceable and standardized project format, zero-cost scalability of pipelines, as well as automated data saving into an open format. *JIPipe* was already utilized to solve numerous demanding image analysis tasks, showcasing its wide applicability and adaptability. *JIPipe* contributes towards making bioimage analysis more accessible, thereby fostering collaborations between experimentalists and computer scientists.

Introduction

ImageJ[1] is a widely used tool for bioimage analysis[2] due to its interactive analysis workflow and high extensibility via plugins. Its intuitive graphical user interface (GUI) makes functions easily accessible, even to unexperienced users, while still providing more advanced operations for experts. However, the interactive approach of *ImageJ* has currently two major shortcomings: (1) *ImageJ* does not provide means to ensure analysis in accordance with the FAIR (Findability – Accessibility – Interoperability – Reusability) principles[3], where all processing steps and parameters would be tracked automatically rather than relying on researchers' manual notes on all executed steps and their parameters. (2) *ImageJ* does not provide batch processing via the GUI making the analysis of larger projects with hundreds or thousands of datasets time-consuming and error-prone, as every step needs to be executed manually via the GUI. To solve these issues, *ImageJ* includes a macro language that can be used to write fully automated and reproducible pipelines of all functions available inside the GUI. A major disadvantage of such a scripting language is that the development of custom pipelines requires programming experience. Additionally, to comply with the FAIR principles, there is a significant amount of code required to be implemented that reads and manages images and metadata, keeps track of these during the analysis, and exports the results in the desired

format. Especially in comparison with the *ImageJ* GUI, writing macros is excessively complex and non-accessible for researchers without programming experience.

In recent years, visual programming languages have been developed in various scientific fields to conveniently solve the aforementioned shortcomings of scripting languages. A well-known example is *KNIME*[4] that is an open source, visual data analysis environment focusing on general data processing, while community-developed functions handle images. Another example of a visual programming language is available in *Icy*[5], which provides functionality similar to *KNIME*, whilst focusing exclusively on image analysis and providing methods that allow the application of *ImageJ* commands. Furthermore, *CLIJ*[6] is a visual language that enables interactive utilization of GPU-based pipelines. Visual languages are as well implemented in commercial languages, for example *Apeer*[7], applying the concept of customizing machine learning algorithms for image processing. Outside image analysis applications, *Galaxy*[8] provides a cloud-based visual language for genome sequencing analysis, where web-based image analysis functionality is currently being implemented. In the area of education, *Scratch*[9] became popular as a graphical software for children to learn programming in an accessible way.

All visual programming languages have in common that they provide a GUI for creating a pipeline of nodes that have one or multiple inputs and outputs. These allow for the integration of a node into the pipeline that eventually realizes the automated analysis by the execution of functions in the right order. Such graphical platform is not yet available in *ImageJ*. Therefore, we developed the Java Image Processing Pipeline (*JIPipe*) as a visual programming language that is specifically designed for the *ImageJ* software ecosystem. All functionality that is familiar to *ImageJ* users is as well available in *JIPipe* in form of nodes, including operations on images, regions of interest and tables. *JIPipe* also extends to adapted functionalities that would not be easily achievable in native *ImageJ* methods, including batch visualization, advanced plotting, table processing and automated data export. In other words, *JIPipe* integrates visual programming into the existing *ImageJ* software ecosystem providing a user-friendly way to create reproducible and fully automated pipelines based on *ImageJ*. In addition, *ImageJ* can utilize algorithms and pipelines made for *JIPipe* via the GUI or via macros in analogy to any other plugin. The simplicity of scaling, pipeline building, compartmentalization and automated saving features make *JIPipe* a well-suited environment for *ImageJ* users to create reproducible and scalable workflows. *JIPipe* fills a niche by bridging the gap between researchers[10] with limited programming experience and the advancing development of modern methods of automated image processing. In particular, popular *ImageJ* plugins including *MorphoLibJ*[11], *FeatureJ*[12], *CLIJ*[6], *Multi-Template-Matching*[13], *OMERO*[14], and *Bio-Formats*[15] can be accessed via *JIPipe*, as well as *Cellpose*[16] to perform deep learning by visual programming. Furthermore, *JIPipe* comes with nodes for *Python*[17], *Jython*[18], *R*[19], and *ImageJ* macro scripts, thus extending the flexibility of workflow development for experienced users with programming skills. JIPipe has already been successfully applied to solve numerous image analysis tasks, including recently published works on nematode activity analysis[20] and the quantification of multispectral optoacoustic tomography (MSOT) images[21], thus underlining its flexibility and wide applicability.

Results

Symbiosis of *ImageJ* and *JIPipe*. The relationship between *ImageJ* and *JIPipe* is symbiotic in that the two platforms share their functionality (see **Fig. 1**). *JIPipe* incorporates a growing set of already more than 1000 functions from *ImageJ* and encapsulates them into features that are suitable for the visual batch processing environment. GUI elements, including manager windows for images, tables, and regions of interest (ROI), are transformed into data objects to lift limitations on the number of instances and to

improve the processing speed. Operations on these data, available as menu commands in *ImageJ* are encapsulated into nodes with standardized interfaces for data transfer and parameters. This comprises common image processing utilities, including thresholding, extraction of measurements and ROI, as well as functions provided by popular plugins, including *MorphoLibJ*[11], *FeatureJ*[12], *Multi-Template-Matching*[13], *OMERO*[14], and *Bio-Formats*[15]. We also added support for *CLIJ*[6] to allow processing of images on graphics cards with the benefit of significantly increasing the performance. Our visual programming approach comes with an easy-to-learn interface, and automated and standardized (meta-)data management, allowing even non-programmers to connect all these familiar functions in batch processing pipelines. *JIPipe* introduces functionality currently not present in *ImageJ*, including extended visualization methods designed for batch processing, table processing algorithms, nodes for data and metadata management, plotting with the widely used *JFreeChart*[22] library, and support for deep learning via *Cellpose*. To extend the flexibility of workflow development, our software also allows users to embed *Python*, *Jython*, *R*, and *ImageJ* macro scripts. Any functionality designed for *JIPipe* is not exclusive to our software, as any compatible node is automatically made available to be utilized as an *ImageJ* command. Consequently, *ImageJ* benefits from *JIPipe*-exclusive operations that include improved visualization functions, table processing algorithms, plotting, access to *Python* and *R*, as well as Cellpose-based segmentation. This symbiotic relationship additionally simplifies the development of *ImageJ* plugins, as developers can target the *JIPipe* API to develop a library usable in both *ImageJ* and *JIPipe*.

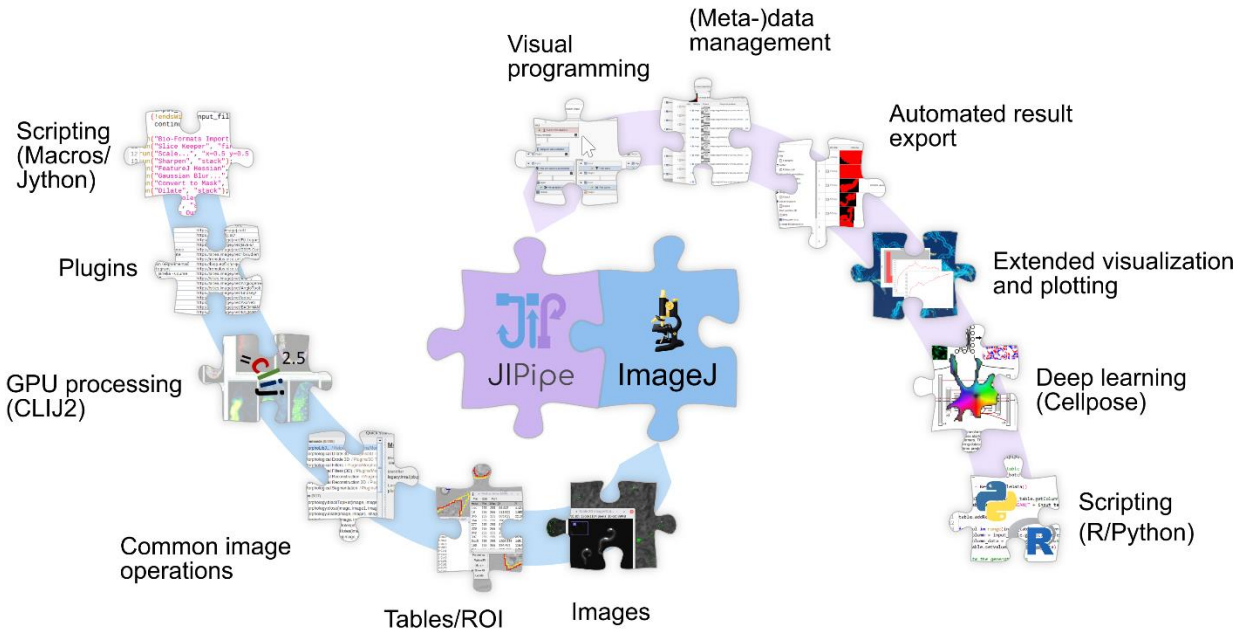

Figure 1 | *JIPipe* and *ImageJ* have a symbiotic relationship where both tools gain access features from each other. *JIPipe* encapsulates *ImageJ* features, including the manual management of images, tables, and regions of interest into a visual programming language with automated data management and export. Operations on these data types are available as nodes and include basic image processing algorithms, and methods from popular plugins, e.g., *CLIJ* for utilizing GPU. *ImageJ* can utilize *JIPipe* functionalities and has access to extended algorithms for visualization and plotting, and integration of *Cellpose*. *JIPipe* can execute *ImageJ* macros and *Jython* scripts, while *JIPipe* provides support for *R* and *Python* scripts.

JIPipe is an easy-to-use alternative to programming *ImageJ* macros, allowing to develop analysis pipelines within a GUI that are the visual counterpart of macros (see **Fig. 2**). *JIPipe* features include automated batch processing, parallelization, and result export. The current set of over 1000 available operations can be extended via plugins that are either provided as Java library or can be created by simply exporting a

pipeline as a new node type (see **Supplementary information 1.1** for an overview). As our software already comes with nodes for scripting languages, a background in Java programming is not needed to integrate plugins and advanced operations not yet encapsulated into nodes. *JIPipe* stores user-created plugins and pipelines in an open and standardized format (see **Supplementary information 1.2** and https://www.JIPipe.org/ for specifications), making it easy to share workflows with the community on a level comparable to *ImageJ* macros.

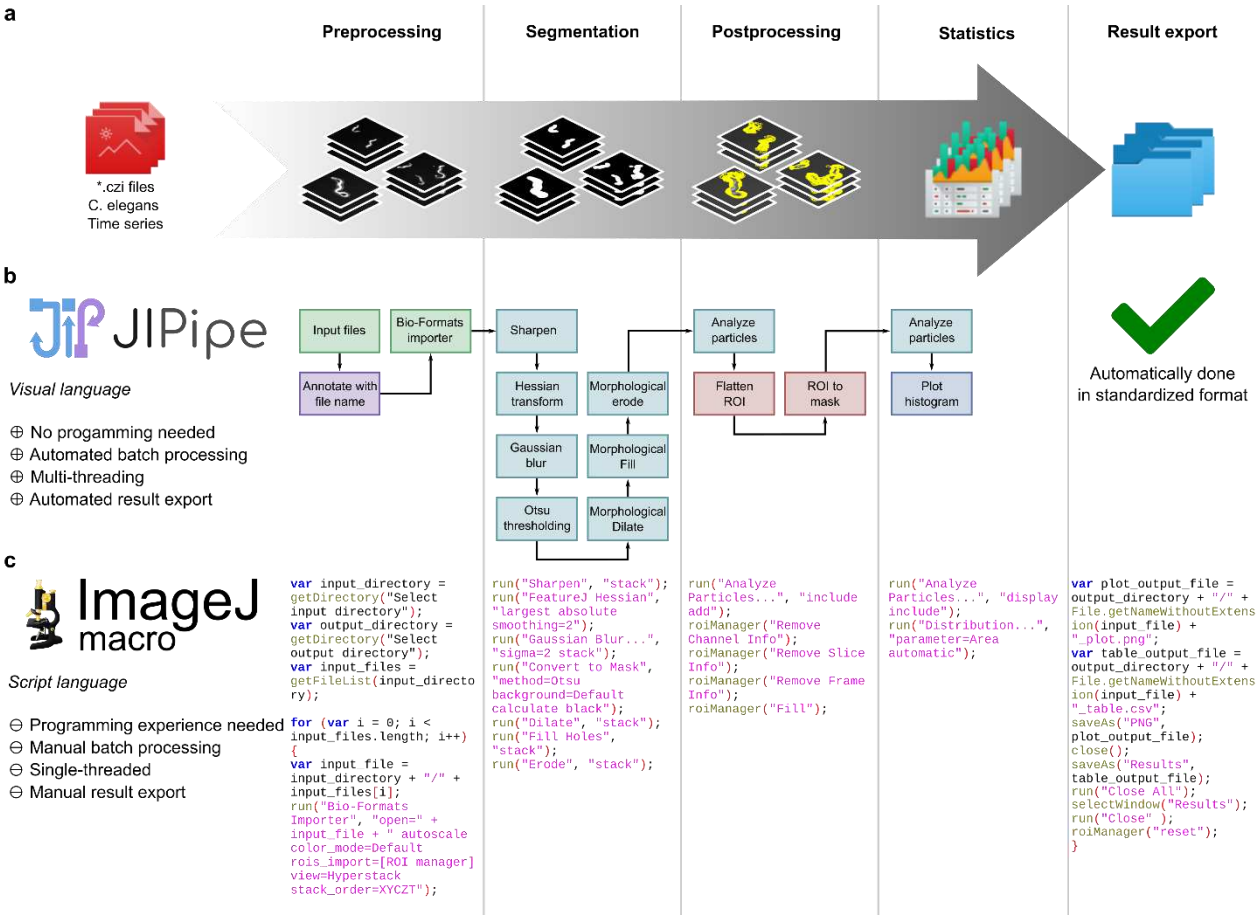

Figure 2 | Comparison of a batch analysis workflow designed with *JIPipe* versus an equivalent *ImageJ* macro. (**a**) The analyzed image data are time series of 2D+time images showing the moving nematode *C. elegans*[20]. This particular workflow extracts the total area covered by *C. elegans* over time from the image files. (**b**) *JIPipe* flowchart that contains the same steps and edges as a *JIPipe* project that applies the analysis (see **Supplementary information 5d**). *JIPipe* automatically saves all results in a standardized format (green check mark). (**c**) *ImageJ* macro code that applies functionality equivalent to the *JIPipe* flowchart.

Hallmarks of *JIPipe* by representative applications. *JIPipe* applications share a set of common features, including file operations, data management and annotations, table processing, visualization, plotting, and saving. The graphical workflow editor allows the arrangement of parameterized nodes into compartmentalized pipelines (**Supplementary Figures 3.1-3.7**) that implement zero-cost scalability due to a table-based "one node -- multiple data" (meta-)data management (**Supplementary Figure 3.7**). Compartments are logical units that can be fully flexibly arranged within a dedicated interface, although they are often arranged according to the preprocessing–analysis–postprocessing–visualization workflow

(**Supplementary Figures 2.1–2.5, Supplementary information 3, Supplementary Figures 3.1-3.6**). Typically, the preprocessing compartment is responsible for file loading and image quality enhancement, as well as channel separation of multichannel images. The analysis compartment carries out the segmentation, masking, and ROI correlation tasks in a channel-specific or conditional way. The outcome is passed on to the postprocessing and visualization compartments (**Supplementary information 2.1-2.5**).

We exemplified the hallmarks of developing image analysis workflows on a set of representative and diverse biological systems that are summarized in **Figure 3**. For instance, the analysis of microfluidic droplets succeeded at characterizing the extent of bacterial growth under various environmental conditions set up inside these picoliter bioreactors (**Fig. 3 row 1**)[23]. Determining the organ-wide distribution of nanocarrier-delivered drugs into the liver of the mouse was the goal in the next example (**Fig. 3 row 2**)[24]. The motility ratio of nematodes characterized the survival of earth worms that consumed soil-beneficial fungi either with or without endosymbiotic bacteria (**Fig. 3 row 3**)[20]. 3D analysis of big volume data was applied to characterize the morphology and spatial distribution of glomeruli in mouse kidneys (see **Fig. 3 row 4**)[25]. Confrontation assays between immune cells and pathogens were quantified by calculating phagocytic measures (**Fig. 3 row 5**)[26]. The various types of image data generated for all these representative biological systems were analyzed in *JIPipe* by exploiting the following features:

Deep learning–based image segmentation. The identification of cell-like objects in both transmitted light (TL) and fluorescence-based images was simplified with the advent of the *Cellpose* framework[16]. *JIPipe* adopted this technique by encapsulating both the training and application of *Cellpose* into nodes, as well as Python environment management tools that allow its fully automated setup. The node extracts the gradient flows (**Fig. 3 A-1 top left, Fig. 3 A5**), the probability map (**Fig. 3 A-1 bottom right**) and the segmented ROIs (**Fig. 3 D-1, D-5**). These outcomes of the analysis indicate that the *JIPipe* integration worked well for TL images of microfluidic droplets (see **Fig. 3 row 1**; **Supplementary information 2.1**) as well as for images of confrontation assays (see **Fig. 3 row 5**; **Supplementary information 2.3**).

Multidimensional image analysis and object tracking. Nodes dedicated to handling multidimensional datasets were utilized when analyzing nanocarrier-based drug delivery kinetics in intravital microscopy of the liver (see **Fig. 3 B-2**; **Supplementary information 2.2**), time series TL data of live nematodes (see **Fig. 3 B-3**; **Supplementary information 2.4**), and the analysis of big volume 3D image stacks as shown in the example of analyzing light-sheet microscopy data of the kidney (**Figure 3 B-4; Supplementary information 2.5**). To this end, these include operations for automated splitting and merging of multi-dimensional data, as well as functionality provided by existing *ImageJ* plugins.

Label-free and fluorescence-based image analysis. JIPipe supplies a complete set of tools to analyze TL and fluorescence-based microscopy data. Hessian filtering was utilized to identify the outline of microfluidic droplets (**Figure 3 C-1, Supplementary information 2.1**), nematodes (**Figure 3 C-3, Supplementary information 2.4**), and unlabeled macrophages in confrontation assays (**Figure 3 C-5, Supplementary information 2.3**). *JIPipe* provides all basic analysis nodes, including blurring, global and local thresholding, morphological operations, watershed algorithms, image normalization and illumination correction, as well as ROI-handling. These tools were applied during the analysis of bacterial growth in microfluidic droplets (**Figure 3 C-1, Supplementary information 2.1**), liver-targeted drug delivery by nanoparticles (**Figure 3 C-2, Supplementary information 2.2**), glomeruli identification and quantification in kidney light-sheet microscopy images (**Figure 3 C-4, Supplementary information 2.5**), as well as labelled fungal spore segmentation (**Figure 3 C-5, Supplementary information 2.3**).

ROI tracking and manipulation. More complex segmentation tasks are also supported by nodes designed for ROI operations including functions provided by *ImageJ* and additional nodes that make use of *JIPipe's* exclusive functionality. These features are utilized in examining i) the extent of bacterial growth inside microfluidic droplets (see **Figs. 3 D-1**, **Supplementary information 2.1**), where the growth area search had to be limited to within the droplet region; ii) the number of dye-loaded liver sinusoidal endothelial cells (LSECs; see **Fig. 3 D-2**, **Supplementary information 2.2**), where masking operations limited the search to be carried out only in the immediate vicinity of the sinusoids; iii) the total area touched by migrating nematodes (see **Fig. 3 D-3**, **Supplementary information 2.4**) where the individual nematode ROIs were merged into the footprint that characterized the motility of the worms; iv) the kidney light-sheet images where ROI-merging via a 3D-connected-components algorithm allowed the volumetric characterization of kidney status under physiological and pathological conditions (see **Fig. 3 D-4**, **Supplementary information 2.5**); v) the distribution of phagocytosed fungal conidia where the ROIs describing the fungal spores were examined for their overlap with those of the macrophages to identify phagocytosed and adherent fungi (see **Fig. 3 D-5**, **Supplementary information 2.3**).

Advanced statistical operations. JIPipe extends the functionality of *ImageJ* with operations for filtering, merging, and manipulating tabular data via user-friendly nodes as well as *Python* and *R* scripts. These were used to calculate percentage growth area of bacteria in microfluidic droplets to identify growth and no-growth classes of the samples (see **Fig. 3 E-1**, **Supplementary information 2.1**). The total footprint covered by a worm divided by the area of the worm at each time point characterized the viability of the nematodes, here showing an example of a low-motility and a high-motility nematode (ratio = 1.3 and 9.0, respectively) (see **Fig. 3 E-3**, **Supplementary information 2.4**). The volumes of individual glomeruli were calculated from the axially merged ROIs in the kidney and plotted in the vicinity of the object (see **Fig. 3 E-4**, **Supplementary information 2.5**). Finally, the calculation of various phagocytic measures from the number of overlapping ROIs was made easy by per-row table manipulations, as demonstrated for the phagocytosis ratio (see **Fig. 3 D-5**, **Supplementary information 2.3**).

Automated plotting. Analysis results of a fully or partially executed workflow can be plotted automatically as supported by a set of dedicated nodes. These provide many graph formats that can be further manipulated in a GUI or via the node parameters. The delivery of nanoparticle-linked drugs to the hepatocytes was described by the kinetic curve of the cargo-linked average fluorescence value (see **Fig. 3 F-2**, **Supplementary information 2.2).** The instantaneous motility of individual worms was calculated and plotted for the duration of the observation (see **Fig. 3 F-3**, **Supplementary information 2.4**), whereas a histogram plot of the glomeruli volume distribution characterized the health status of the kidney sample based on its glomeruli number and size (see **Fig. 3 F-4**, **Supplementary information 2.5**).

These examples demonstrate that *JIPipe* greatly complements *ImageJ* by simplifying the management of a multitude of complex image analysis and management tasks.

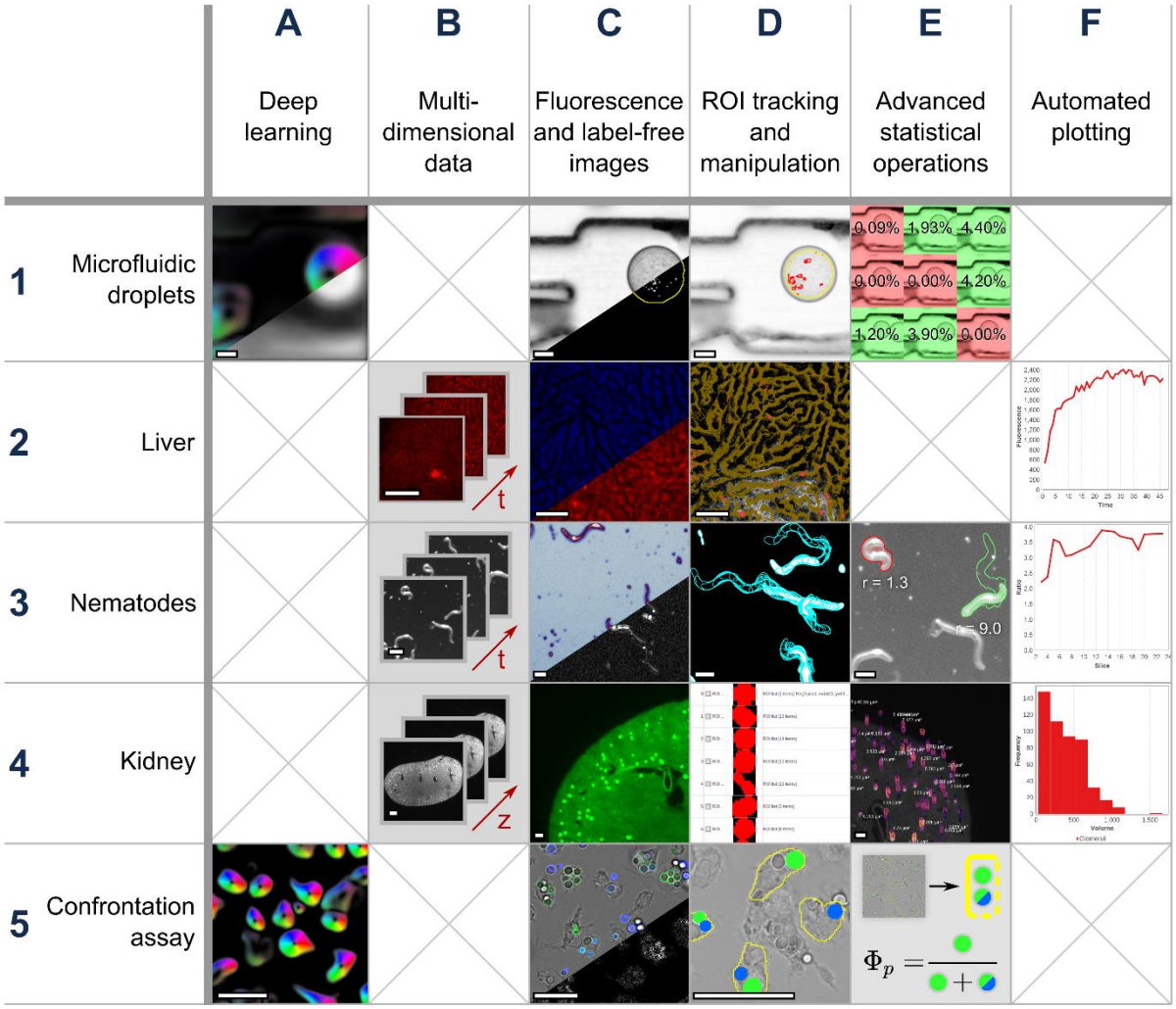

Figure 3 | Examples of biological applications and hallmarks of *JIPipe*. The rows are arranged according to the biological samples, whereas the columns correspond to sets of main features of the *JIPipe* platform. The highlighted techniques include deep learning (**column A**), the handling of multidimensional images (**column B**), the analysis of label-free and fluorescence microscopy data (**column C**), the flexible tools of region of interest (ROI) tracking and manipulations (**column D**), mathematical and statistical operations (**column E**), and data plotting via automatable nodes (**column F**). The rows show examples from the analysis of microfluidic droplets (**row 1**), intravital microscopy of the liver (**row 2**), time-tracking of live nematodes (**row 3**), the segmentation and quantification of kidney glomeruli (**row 4**), and the analysis of confrontation assays between macrophages and fungal spores (**row 5**). (**A-1**) The gradient flow (top left) and probability map (bottom right) of microfluidic droplets, provided by the *Cellpose* algorithm. (**A-5**) The gradient flow of unlabeled macrophages provided by the *Cellpose* node output. (**B-2**) Representative images from a time series of fluorescence images recorded from live liver. (**B3**) Earth worms in transmitted light microscopy images during a time series microscopy experiment. (**B-4**) Selected z slices of a 3D light-sheet microscopy image of mouse kidney. (**C-1**) Microfluidic droplets in a TL image (top left), as well as after segmentation of the droplet inner area (bottom right, yellow line) and of the bacteria (bottom right, white speckles). (**C-2**) Images of autofluorescence (top left, blue) and cargo-linked fluorescence labeling (bottom right, red) of a liver sample during intravital microscopy. (**C-3**) Nematodes in a live time-series TL microscopy experiment before processing (top left, "BuPu" look-up table[27]) and after Hessian filtering (bottom right). (**C-4**) One z slice of a 3D fluorescence microscopy image of mouse kidney (light green), overlaid with a maximum intensity z projection of the segmented glomeruli ROIs (bright green speckles). (**C-5**) TL images of a confrontation assay (top left) with the macrophages shown in grey, the phagocytosed spores shown in green, and the external conidia presented in blue. The macrophages were identified without fluorescence labeling by applying a Hessian filter (bottom right). (**D-1**) A microfluidic droplet imaged via TL microscopy, with the segmented inner line of the droplet shown in yellow, and the bacterial growth indicated by red. (**D-2**) Mouse liver as revealed by

intravital microscopy, showing the sinusoids as a time series maximum projection (dark yellow) superimposed with the time-series projection of LSECs (red speckles). (**D-3**) The time superposition of nematode outlines during a TL microscopy experiment (blue: nematode outlines at consecutive time points, white: binary image of segmented worms). (**D-4**) A selection of detected 3D ROIs representing individual glomeruli from a light-sheet microscopy experiment displayed in the JIPipe cache GUI. (**D-5**) The outline of segmented macrophages (yellow) superimposed with phagocytosed conidia (green) and adherent spores (blue). (**E-1**) Bacterial growth levels about 0.5% area/area were classified as positive growth (green hue), whereas lower values than 0.5% were considered as negative growth (red hue). The superimposed numbers indicate the per-droplet growth percentage values. (**E-3**) Nematodes imaged with TL microscopy, recorded as part of a time series. The red and green outlines are the footprints of two individual worms, one each with low and high motility values (red, motility ratio at 1.3; green, motility ratio at 9.0). (**E-4**) The volume values of individual glomeruli are calculated from light-sheet microscopy images of the mouse kidney. The original image appears as a greyscale background, the ROIs of the segmented glomeruli are shown in perspective view as generated by a special *JIPipe* node, where the individual 2D ROIs are colored according to their z position, and the volume values are printed in white. (**E-5**) Representation of the principle behind calculating the phagocytosis ratio from confrontation assay images. The segmented macrophages (yellow outline) were matched against the green and blue-labelled conidia in order to find the phagocytosed (green) and adherent (blue) spores. Using the *JIPipe* table calculator node, the ratio of the phagocytosed conidia over the total number of spores (phagocytosed plus adherent) was determined (Φ_p). (**F-2**) Using *JIPipe*'s automated plotting nodes, the mean fluorescence value of the cargo-linked dye measured in the hepatocytes was presented for a time series experiment (red curve). (**F-3**) The motility ratio of an individual worm was plotted over the course of a time series experiment (red curve). The ratio was defined as the total footprint of the worm covered during the video divided by the area of the worm at each time point. (**F-4**) Histogram representation of the glomeruli volume for a light-sheet microscopy image set measured in a mouse kidney. Scalebars represent 200 µm in column B, 100 µm elsewhere.

***JIPipe* user interface and data model.** Here we will provide a brief overview of some of the technical details of *JIPipe*, whereas the full organization of our software can be retrieved from the Supplementary Material (3: "JIPipe user interface and data model"), as well as from the *JIPipe* website (http://www.JIPipe.org/). Whilst *JIPipe* builds upon *ImageJ* functionality, it also contributes its own solution for handling (meta-)data and encapsulating it into a GUI. *JIPipe*'s backbone is a directed acyclic graph (DAG) that organizes all functional units, the so-called nodes. A pipeline is formed by connecting nodes through their input and output slots, thus modelling the flow of data. We simplified this concept by the introduction of compartmentalization, which clearly separates projects with a large number of nodes into functional units. *JIPipe* provides over 1000 nodes that cover every core aspect of image analysis and data management, and can be easily searched by name, functionality, and compatibility to the preceding node. To simplify the creation of workflows, many nodes were manually provided with integrated documentation and customization options, for example to test multiple parameter sets, or to apply calculations via user-defined mathematical expressions. These features are made possible by the table-based data management that enforces constraints, for example on data types, presence of a primary data column, or type of annotations. The consequence of a simplified table is zero-cost scaling, meaning that *JIPipe* workflows can be adapted to batch processing without any changes to the pipeline structure. At the same time, a table allows for great flexibility and can be utilized to track biological conditions, dataset identifiers, or image properties, thus providing an easy way to find and reproduce data and analysis details according to the FAIR principles[3]. According to the symbiotic relationship detailed earlier, all nodes and *JIPipe* algorithms can also be directly accessed from *ImageJ*. Pipelines can be executed in their entirety or up to a user-specified node, with the option of saving the results into a memory cache or on the hard-drive in a standardized format, including all parameters and the complete project file. This allows *JIPipe* to open existing results, making further processing and plotting at a later timepoint possible.

In conclusion, *JIPipe* is a user-friendly and powerful tool that introduces visual programming into *ImageJ*. Users who are already familiar with the *ImageJ* software find equivalent functionalities and can continue to use their existing scripts, in both *JIPipe* and *ImageJ*. Batch analyses can be designed efficiently by laying out the steps for a single analysis and then increasing the set of input files – all required functionality to

track datasets during the analysis are handled by the powerful table-based data architecture. We showed by the examples presented in **Fig. 3** that *JIPipe*, like *ImageJ*, can be applied to virtually all kind of biological data. *JIPipe* is fully open-source and licensed under BSD-2. Detailed user guides, tutorials, videos, examples, and instructions for developers are available on https://www.*JIPipe*.org/. We are working with the image analysis community (https://forum.image.sc/) to further improve *JIPipe* and establish the software as meaningful contribution to collaborative environments.

Discussion

With the advent of visual programming in many application fields, the need for a visual language for *ImageJ* programming has become apparent. This motivated us to develop *JIPipe* and by that fill a niche that bridges the gap between researchers with limited programming experience and the advancing development of modern methods of automated image processing, including deep learning approaches, parallel- and cloud computing. *ImageJ*[1] gained immense popularity[2] amongst biologists and bioimage informaticians due to its very wide applicability, ease of use and of expansion, and open-source nature. When it comes to analyzing large amounts of images, especially in a complex structure due, e.g., to a wide set of biological conditions, pipelines in *ImageJ* become complex, making it difficult to record the individual steps and their parameters according to the FAIR principles[3]. The necessity to have certain programming knowledge to turn single-image analysis workflows into complex batch analysis via macro or plugin programming discourages many less experienced users from developing automated pipelines in *ImageJ*. Here we identified the niche, where our contribution to visual programming can be meaningful for the image analysis community. With *JIPipe*, we established such a visual programming environment, where current *ImageJ* users can utilize their existing knowledge of *ImageJ* to convert existing pipelines into fully automated batch-processing workflows that generate exactly the same results as the *ImageJ* equivalents.

At this stage, *JIPipe* focuses on implementing the first version of the *ImageJ* API (*ImageJ1*)[28], because the majority of functions in the *ImageJ* GUI still rely on *ImageJ1*, where only one non-standardized string is available for transferring algorithm parameters. In addition, there is a concept of a single “active image”, a single table of results and ROI manager — restrictions that are not compatible with visual programming. In addition, custom GUI components are not suitable for GUI-free and parallel-execution environments. For this reason, we manually curated these functionalities into *JIPipe* nodes, instead of applying an automated algorithm for the conversion. Functions were added, merged, or split into multiple nodes, whereas the concepts of image windows, result tables and ROI manager were adapted into data types compatible with visual programming. As a result, *JIPipe* functions cover the same or very similar tasks as the corresponding commands in *ImageJ*. Due to this conversion process, it was not always sensible to keep the same function names as in *ImageJ*, adding a small learning curve to *JIPipe*. To ease this transition phase, we provide integrated search options, numerous application examples with complete code and data, as well as a thorough online training material (see **Supplementary Information 4**, https://www.JIPipe.org/tutorials/).

ImageJ incorporates a second, more modern API, *ImageJ2*, which is richer in features. For example, there are standardized interfaces for data and parameters, flexible service-based architecture, as well as individually addressable parameters. We adapted our approach of curating the integration via *ImageJ2* by incorporating the Multi-Template-Matching plugin[13]. An automated integration of all compatible operations was achieved by interfacing with the *ImageJ* Ops API[2] and applying on-demand conversion between *JIPipe* and *Image2* data.

In terms of documentations, *JIPipe* adopts the principle of immediate access to all functionality descriptions, greatly simplifying visual programming. The automated documentation tool was built from manually recovered online *ImageJ* documentation and source code, and it is available directly from the nodes.

Two of the most established and best-known visual applications in image analysis are *Icy* and *KNIME*. These tools integrate certain aspects of *ImageJ* into their own framework, while *ImageJ* cannot use functions designed for these tools, thus lacking the symbiotic relationship with *ImageJ*. This means that *ImageJ* itself is not the main focus of operation for these visual applications. Consequently, a niche was open for an application that would provide (i) a GUI for *ImageJ* operations with easy batch processing, automated saving, maintained full compatibility with *ImageJ*, (ii) a learning curve that makes this new language accessible for non-programmers, and (iii) a standardized project format that saves all information about the progress and implementation of the project according to the FAIR principles. This niche is filled by *JIPipe*.

Methods

Software development. *JIPipe* was developed in Java version 8 and utilizes software libraries provided by *ImageJ* and other developers. A full list of all dependency libraries is given in **Supplementary Table 5.1**, whereas the system requirements are listed in **Supplementary Information 5.2**.

Software availability. The version of *JIPipe*, and its source code used in this paper is available as supplement. The newest version of *JIPipe* and the current version of the source code are available on our website https://www.*JIPipe*.org/ and on https://www.github.com/applied-systems-biology/JIPipe/ (https://doi.org/10.5281/zenodo.6532719). *JIPipe* is also available from within the *ImageJ* update service. Instructions are available on our website.

Data availability. The data that supports our findings are available as supplements. This is also the case for the project files and example data used to demonstrate the usage of *JIPipe*. The example pipelines are available at https://doi.org/10.6084/m9.figshare.19733320.v1.

Acknowledgements

This work was financially supported by the International Leibniz Research School for Microbial and Biomolecular Interactions Jena – ILRS Jena. Furthermore, the German Research Foundation (DFG) funded this project through the Collaborative Research Center PolyTarget 1278 – project number 316213987, subproject Z01. This work was also supported by the Collaborative Research Center Funginet 124 – project number 210879364, subproject B4 – and by the Cluster of Excellence "Balance of the Microverse" under Germany´s Excellence Strategy – EXC 2051 – Project-ID 390713860, as well as by the Leibniz ScienceCampus *InfectoOptics* Jena, which is financed by the funding line Strategic Networking of the Leibniz Association. Furthermore, we received support from the Federal Ministry of Education and Research, Germany (grant number 13GW0456B) in the context of the *InfectoGnostics* Research Campus Jena. We are particularly thankful to Dr. Martin Roth (microfluidic droplets data), Prof. Christian Hertweck (nematode imaging), Drs. Kerstin Voigt and Mohamed Hassan (confrontation assay images), Dr. Adrian

Press (intravital liver microscopy data), and Prof. Matthias Gunzer (kidney light-sheet images) for kindly providing image data.

Author contributions

R.G. developed the software. Z.C. designed the bioimage analysis pipelines and tested the software. M.T.F. and Z.C. conceived the idea. M.T.F. directed and supervised the project. All authors wrote the initial draft, read and contributed to the paper and approved the content.

References

1. Rueden, C. T. *et al.* ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics* **18**, 529 (2017).
2. Schroeder, A. B. *et al.* The ImageJ ecosystem: Open-source software for image visualization, processing, and analysis. *Protein Sci.* **30**, 234–249 (2021).
3. Wilkinson, M. D. *et al.* The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data 2016 31* **3**, 1–9 (2016).
4. Berthold, M. R. *et al.* KNIME: The Konstanz Information Miner. in *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)* (Springer, 2007).
5. de Chaumont, F. *et al.* Icy: an open bioimage informatics platform for extended reproducible research. *Nat. Methods* **9**, 690–696 (2012).
6. Haase, R. *et al.* CLIJ: GPU-accelerated image processing for everyone. *Nat. Methods* **17**, 5–6 (2020).
7. APEER. https://www.apeer.com/.
8. Afgan, E. *et al.* The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Res.* **46**, W537–W544 (2018).
9. Maloney, J., Resnick, M., Rusk, N., Silverman, B. & Eastmond, E. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* **10**, 16:1-16:15 (2010).
10. Martins, G. G. *et al.* Highlights from the 2016-2020 NEUBIAS training schools for Bioimage Analysts: a success story and key asset for analysts and life scientists. *F1000Research* **10**, (2021).
11. Legland, D., Arganda-Carreras, I. & Andrey, P. MorphoLibJ: Integrated library and plugins for mathematical morphology with ImageJ. *Bioinformatics* **32**, 3532–3534 (2016).

12. Meijering, E. FeatureJ. https://imagescience.org/meijering/software/featurej/.

13. Thomas, L. S. V. & Gehrig, J. Multi-template matching: a versatile tool for object-localization in microscopy images. *BMC Bioinformatics* **21**, 44 (2020).

14. Allan, C. *et al.* OMERO: flexible, model-driven data management for experimental biology. *Nat. Methods* **9**, 245–253 (2012).

15. Linkert, M. *et al.* Metadata matters: access to image data in the real world. *J. Cell Biol.* **189**, 777–782 (2010).

16. Stringer, C., Wang, T., Michaelos, M. & Pachitariu, M. Cellpose: a generalist algorithm for cellular segmentation. *Nat. Methods* **18**, 100–106 (2021).

17. Python. https://www.python.org/.

18. Jython. https://www.jython.org/.

19. R Core Team & others. R: A language and environment for statistical computing. (2013).

20. Büttner, H. *et al.* Bacterial endosymbionts protect beneficial soil fungus from nematode attack. *Proc. Natl. Acad. Sci. U. S. A.* **118**, 2110669118 (2021).

21. Hoffmann, B. *et al.* Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data. *Photoacoustics* **26**, 100361 (2022).

22. JFreeChart. https://jfree.org/jfreechart/.

23. Svensson, C. M. *et al.* Coding of Experimental Conditions in Microfluidic Droplet Assays Using Colored Beads and Machine Learning Supported Image Analysis. *Small* **15**, 1802384 (2019).

24. Muljajew, I. *et al.* Stealth Effect of Short Polyoxazolines in Graft Copolymers: Minor Changes of Backbone End Group Determine Liver Cell-Type Specificity. *ACS Nano* **15**, 12298–12313 (2021).

25. Klingberg, A. *et al.* Fully Automated Evaluation of Total Glomerular Number and Capillary Tuft Size in Nephritic Kidneys Using Lightsheet Microscopy. *J. Am. Soc. Nephrol. JASN* **28**, 452–459 (2017).

26. Cseresnyes, Z., Kraibooj, K. & Figge, M. T. Hessian-based quantitative image analysis of host-pathogen confrontation assays. *Cytometry A* **93**, 346–356 (2018).

27. Hunter, J. D. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* **9**, 90–95 (2007).

28. Schneider, C. A., Rasband, W. S. & Eliceiri, K. W. NIH Image to ImageJ: 25 years of image analysis. *Nat. Methods* **9**, 671–675 (2012).

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- IPipeNatureMethodsSupplementsfinal.pdf