BMC Bioinformatics

SOFTWARE **Open Access**

CellProfiler 4: improvements in speed, utility and usability

David R. Stirling[1], Madison J. Swain-Bowden[2], Alice M. Lucas[1], Anne E. Carpenter[1], Beth A. Cimini[1*†] and Allen Goodman[1†]

*Correspondence: bcimini@broadinstitute.org
†Beth A. Cimini and Allen Goodman have contributed equally to this work
[1] Imaging Platform, Broad Institute of MIT and Harvard, Cambridge, MA, USA
Full list of author information is available at the end of the article

Abstract

Background: Imaging data contains a substantial amount of information which can be difficult to evaluate by eye. With the expansion of high throughput microscopy methodologies producing increasingly large datasets, automated and objective analysis of the resulting images is essential to effectively extract biological information from this data. CellProfiler is a free, open source image analysis program which enables researchers to generate modular pipelines with which to process microscopy images into interpretable measurements.

Results: Herein we describe CellProfiler 4, a new version of this software with expanded functionality. Based on user feedback, we have made several user interface refinements to improve the usability of the software. We introduced new modules to expand the capabilities of the software. We also evaluated performance and made targeted optimizations to reduce the time and cost associated with running common large-scale analysis pipelines.

Conclusions: CellProfiler 4 provides significantly improved performance in complex workflows compared to previous versions. This release will ensure that researchers will have continued access to CellProfiler's powerful computational tools in the coming years.

Keywords: Image analysis, Microscopy, Image segmentation, Image quantitation, Bioimaging

Background

Microscopy can be used to capture images which contain a wealth of information that can inform biomedical research. Image analysis software can allow scientists to obtain quantitative measurements from images that are otherwise difficult to capture via subjective observation. The increasing use of automated microscopy now allows researchers to capture images of samples treated with many thousands of individual compounds or genetic perturbations. Scientists increasingly image cells in 3D or across time series; this expanding bulk of raw data necessitates automated processing and analysis. Such analysis is best achieved through using software to perform automated detection of cells or organisms and extract quantitative metrics which objectively describe the specimens.

Many microscopes are now sold with accompanying proprietary analysis packages, such as MetaMorph (Molecular Devices), Elements (Nikon), Zen (Zeiss) and Harmony (Perkin Elmer). These ecosystems are powerful but can lack the flexibility to work with data from other manufacturers' equipment. Cost of these proprietary solutions can also limit accessibility, and their closed-source nature can obscure exactly how scientists' data is being analyzed. Free, open-source software packages such as ImageJ, CellProfiler, QuPath, Ilastik and many others have therefore become popular analysis tools used by researchers [1]. ImageJ is the most widely-used package and excels in performing analysis of single images, assisted by a vast array of community-developed plugins [1]. Numerous smaller packages are tooled towards specific types of data: for example, QuPath is a popular program geared specifically towards pathology applications [2], while Ilastik delivers an interactive machine learning framework to assist users in segmenting images [3].

In 2005 we introduced CellProfiler, an open-source image analysis program which allows users without specific training to automate their image analysis by using modular processing pipelines [4]. CellProfiler has been widely adopted by the community, and is currently referenced more than 2000 times per year. Built-in modules provide a diverse array of algorithms for analyzing images, which can be further extended through the use of community-developed plugins. In an independent analysis of 15 free image analysis tools CellProfiler scored highly in both usability and functionality [5]. Our previous release, CellProfiler 3, introduced support for analysis of 3D images to further expand the tool's applications [6]. However, some popular features from CellProfiler 2 could not be brought forward into that release and certain modules struggled to operate efficiently in 3D pipelines.

Implementation

CellProfiler was originally written in MATLAB, but in 2010 was rewritten in Python 2, which reached its official end-of-life in 2020. In order to ensure ongoing compatibility with future operating systems we ported the software to the Python 3 language to create CellProfiler 4. This provided the opportunity for a broader restructuring of the software's code to improve performance, reliability and utility. CellProfiler 4 is available for download at cellprofiler.org.

As part of the migration to Python 3, we split the CellProfiler source code into two packages: cellprofiler and cellprofiler-core. The new cellprofiler-core package contains all the critical functionality needed to execute CellProfiler pipelines, whereas the cellprofiler repository now primarily contains the user interface code and built-in modules. The core package has been developed to introduce a stable API which will allow users to access CellProfiler's functionality as a Python package within popular environments such as Jupyter [7] and for future integration with other packages and software suites.

User interface refinements

Guided by feedback from biologists, we have made several improvements to the CellProfiler user interface with the goal of making the software more accessible and easier to use. The basic 3D viewer introduced in CellProfiler 3.0 has now been replaced with a more fully-featured viewer which allows users to inspect any plane in a volume

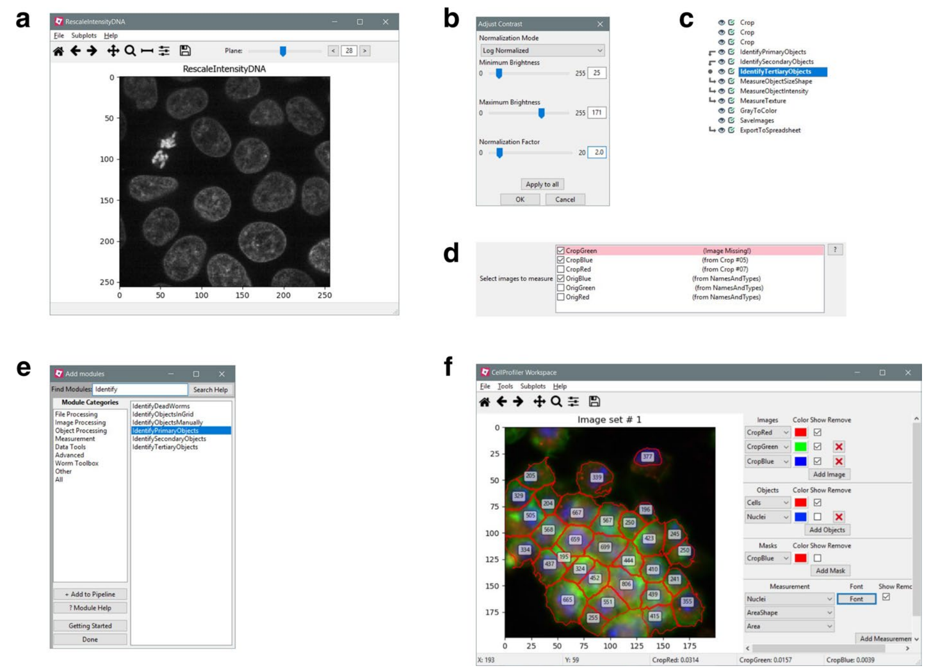

Fig. 1 User interface refinements in CellProfiler 4. **a** The new 3D viewer window with plane controls in the top right. **b** Contrast and normalization adjustment popup available with any image window. **c** Interface displayed when the "trace" command is called on a module. Arrow icons on the left represent modules which provide data to or use data from the selected module (dot icon). **d** Selection widget for choosing multiple images for analysis. Images sourced from disabled or missing modules are highlighted. **e** "Add modules" pane in search mode, the modules list is filtered based on the text entered into the search box. **f** The Workspace Viewer module displaying a custom overlay of data from the example pipeline

(Fig. 1a). We have also expanded the figure contrast dialogs to give users more granular control over how images are displayed in both 2D and 3D mode (Fig. 1b). These changes will help users to better visualize and understand their data.

Other changes make it easier to develop and configure pipelines. We added an interface to visualize which modules produce inputs needed by, or use outputs from, a module of interest, which will aid in modifying complex pipelines (Fig. 1c). We also revised the interface for selecting multiple images for analysis within a module, replacing dropdown menus with a checklist in which multiple images can be selected quickly and efficiently (Fig. 1d). Furthermore, a new search filter in the "Add module" popup allows users to more easily find desired modules by module name rather than by category (Fig. 1e).

We also restored some features which were previously lost in the migration from CellProfiler 2 to CellProfiler 3. Most notably, we rebuilt the Workspace Viewer, where users construct a customized view of their data and can stay focused on a specific region of interest as the pipeline is modified (Fig. 1f), making it much simpler to monitor and refine segmentation of problematic regions of an image. In addition, new icons in the Test Mode pipeline interface provide a stronger visual indication of which module is currently about to be executed, and provide the ability to return to and execute earlier modules in the pipeline. This replicates and replaces the functionality of the slider widget from CellProfiler 2, which could not be carried forward into CellProfiler 3 but was popular with users.

New and restored features

In CellProfiler 4 we introduced several new analysis features and settings. A common workflow issue we identified was that analysts often segment highly variable objects in multiple stages (such as segmenting and masking out bright objects to aid segmentation of similar-but-dimmer objects), but previous versions could not simply treat resulting segmentations as a single object set when performing and exporting measurements. To resolve this we added the CombineObjects module to allow users to merge sets of objects which have been defined separately. A key issue when designing this module was how to handle objects that would overlap if the sets were merged, therefore we built several strategies detailed in Fig. 2. The resulting merged set can then be carried forward throughout the pipeline without the need to merge measurement tables outside of CellProfiler.

Many users were disappointed with the loss of the RunImageJ module [8] in CellProfiler 2.2; we have now replaced it with the new RunImageJMacro module. The new module allows a user to export images from CellProfiler into a temporary directory, execute a custom ImageJ macro on that directory and then automatically import resulting processed images back into CellProfiler. In practice this will allow users to access ImageJ functions and plugins within a CellProfiler pipeline, greatly expanding its interoperability. Unlike its predecessor, the RunImageJMacro module relies on the user's copy of ImageJ rather than a built-in copy. This allows users to take advantage of any new ImageJ upgrades and simultaneously poses less danger to CellProfiler's stability because releases between the two softwares need not be kept in sync.

We also upgraded several existing modules. We rewrote the Threshold module to allow all pre-existing threshold strategies to be used in 'adaptive' mode, giving users

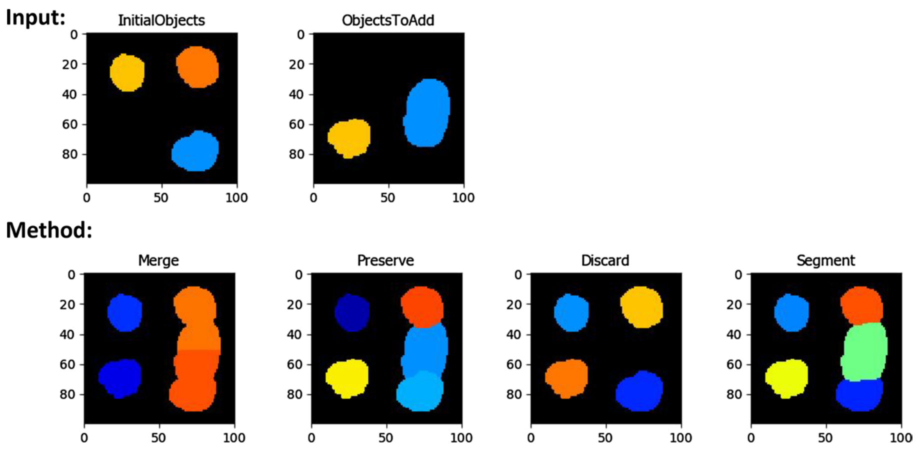

Fig. 2 Approaches for combining object sets within the CombineObjects module. Results represent the output from different methods available within the module. "Merge" will join touching objects and distribute conflicting regions to the nearest object from the initial set. "Preserve" will add only regions of objects from the second set which did not overlap with the initial set. "Discard" will only add objects with no overlap. "Segment" will add both object sets and re-segment disputed regions

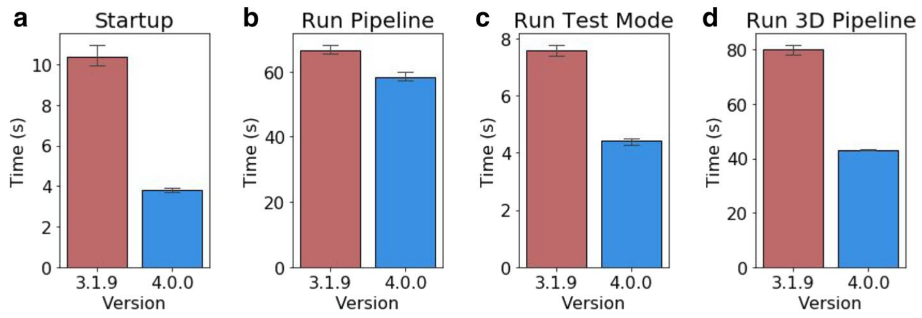

Fig. 3 General performance in CellProfiler 3 versus CellProfiler 4. Results represent independent runs on a machine running Windows 10, using 1 worker process. **a** Time from launching the CellProfiler executable to display of the full GUI (n = 5). **b** Time taken to run the ExampleFly pipeline in Analysis Mode (n = 3). **c** Time to run the ExampleFly pipeline in Test Mode (n = 5). **d** Time to run the 3D monolayer tutorial pipeline in Analysis Mode (n = 3)

more options in images with highly-variable background. We have also added the Sauvola local thresholding method as an alternative adaptive strategy [9]. Previous versions of CellProfiler 2 shipped a version of the Otsu thresholding method that log-transformed the data before applying the threshold; this assisted in the thresholding of dim images, but led users to question why our Otsu values did not match those from other libraries such as scikit-image [10]. This inconsistent behavior could be confusing to users, so we began the process of updating that implementation in CellProfiler 3 and completed it in CellProfiler 4. We added a dedicated setting to log transform image data during application of any thresholding method. These new options will assist users in segmenting challenging images.

New measurements

We overhauled some measurement modules in CellProfiler 4. We redesigned MeasureObjectSizeShape to record additional measurements now available in scikit-image, including bounding box locations, image moments and inertia tensors, producing up to 60 new shape measurements per object. We anticipate that these new features may be of particular value for training machine learning models, which play an increasingly important role in performing object classification on large data sets. In addition to new features, several of the previously 2D-exclusive measurements, such as Euler Number and Solidity, are now also available when working with 3D images. Together these expanded measurements provide researchers with even more metrics with which to investigate cellular phenotypes.

Results

Performance improvements

A key focus in producing CellProfiler 4 has been improving performance of the software and addressing common issues encountered by users. We revised our build packaging process to more reliably bundle CellProfiler's Java dependencies so that additional software and system configuration is no longer required to use the program. In doing so we also optimized the program's startup sequence, which provided a substantial improvement in the time taken to initialize the software (Fig. 3a). Another critical area of focus for improvement has been in file loading (input/output, or I/O operations). Combined improvements in Python's underlying directory scanning functions and optimizations to CellProfiler's image loading procedures have dramatically reduced the time needed to add large folders of images to the file list. This is particularly noticeable when using networked storage.

In our performance testing of an example analysis pipeline, overall performance was similar to CellProfiler 3 (Fig. 3b). However, executing this pipeline in Test Mode was inhibited by unnecessary user interface updates between running individual modules. Optimizing the UI updates sent during test mode reduced the time taken to run an image set in this mode (Fig. 3c).

Running more complex analysis workflows such as 3D segmentation and the commonly used Cell Painting assay [11] was time-consuming in CellProfiler 3. We therefore aimed to identify and refine modules which displayed long execution times in these scenarios.

Optimizations across all modules produced a 50% performance improvement when running 3D pipelines such as the 3D monolayer tutorial dataset (Fig. 3d) [12]. Within 3D workflows we had identified the MedianFilter module as being particularly slow to process. By switching to the new scipy.ndimage filter implementation we were able to substantially reduce the time taken to process each image (Fig. 4a).

Another key target was the MeasureTexture module, which exhibited long run times when performing per-object measurements. Analysis revealed that this was caused by per-object functions processing full-size masked arrays for each object to be measured. To improve performance we adjusted these functions to produce and process arrays cropped down to the particular region of interest for each object. In our testing this reduced the time taken to analyze each image from minutes down to seconds, without any change in the resulting measurements (Fig. 4b).

Major gains were also made in measurement of the Costes Colocalization Coefficient in the MeasureColocalization module. This statistic requires the calculation of Costes' automatic threshold, which is generated by thresholding the two images to be compared and then serially reducing the thresholding value until the Pearson

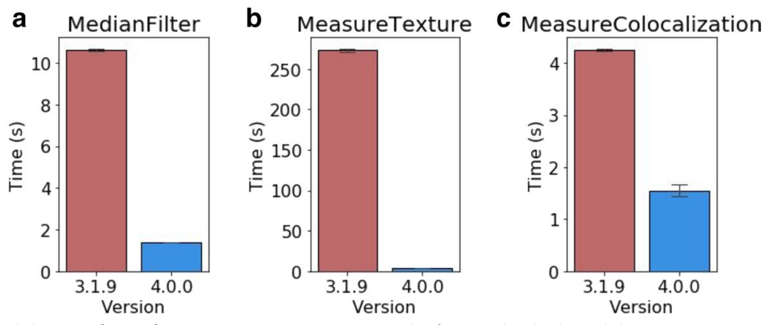

Fig. 4 Module-specific performance improvements. Results from individual module testing on a machine running Windows 10. **a** Execution time for the MedianFilter module running within the 3D Monolayer pipeline (n = 5). **b** Execution time when running per-object texture measurements on data from the ExampleFly pipeline (n = 5). **c** Execution time when running MeasureColocalization on 8-bit images from the ExampleFly pipeline (n = 5)

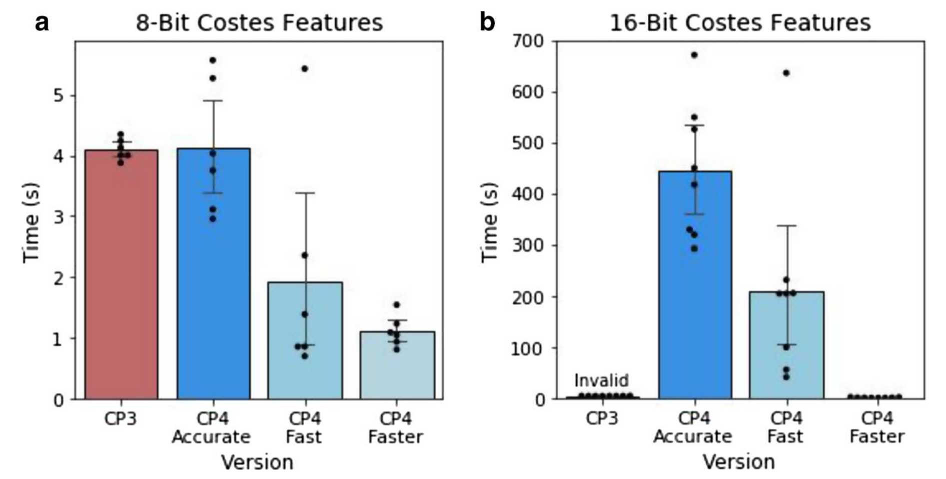

Fig. 5 Performance of alternative Costes automated thresholding strategies. Execution times for the MeasureColocalization module performing 1 pairwise comparison with Costes features enabled, using each algorithm on **a** 8-bit images from the ExampleFly pipeline (n = 6) or **b** 16-bit images from the example Cell Painting dataset (n = 8). On 16-bit images results from CellProfiler 3 are calculated incorrectly, but shown to illustrate relative performance

R correlation between the two thresholded images drops below a value of 0. Our original implementation would reduce the candidate value in images scaled 0–1 by 1/255 at each step, which was inappropriate for 16-bit images containing 65,536 grey levels rather than the 256 present in 8 bit images. Testing 65,536 candidate thresholds in 16-bit images would be excessively slow, so we introduced optional alternative implementations of the Costes automated thresholding method to resolve this inefficiency. Our first optimization maintained the canonical strategy of evaluating every possible threshold, but only measured the Pearson R correlation of the thresholded images if the new value produced a different total number of thresholded pixels than the previous value. We termed this “accurate” mode, but in images with large numbers of unique pixel values performance was unacceptably slow. We therefore introduced “fast” mode to the module, in which the candidate threshold is decreased in larger steps if the previous Pearson R value was substantially higher than 0. This improved performance when working with 8-bit images (Fig. 5a), but was still inefficient with 16-bit images (Fig. 5b). We subsequently devised an alternative implementation, dubbed “faster” mode, in which a weighted bisection search algorithm is used to consecutively narrow a window of possible target thresholds. By reducing the candidate window by 1/6 each cycle, we were able to calculate identical thresholds to the “accurate” method in seconds rather than hours. This opens up the ability to perform efficient Costes Colocalization calculations on 16-bit images (Fig. 5b). In theory these accelerated methods could ‘overshoot’ the target threshold by a small margin in rare instances, but in our testing they consistently produced identical results to the “accurate” implementation. Nonetheless we have made all three strategies (“accurate”, “fast” and “faster”) available within the module settings. Other colocalization methods did not suffer from the same degree of performance issues, but additionally updating them to newer implementations reduced the time taken for the module to process without Costes features enabled (Fig. 4c).

Together, these improvements will substantially reduce the computational time and power necessary to process images, particularly when working with large, complex data sets. This will have the added benefits of reducing resource costs for researchers, making large-scale analysis with CellProfiler more affordable and accessible. The reduction in analysis time will also reduce the environmental impact of running such pipelines.

Performance in common workflows

To examine the impact of our changes on performance on a large heterogeneous workflow, we compared the performance of CellProfiler 3 to CellProfiler 4 when running the Cell Painting assay protocol [11]. This workflow is typically performed on large datasets in a cluster environment, so we selected a sample of 48 image sets from a published dataset and measured processing times on a single machine. Execution times were captured for each module across three independent runs of this dataset. The sum of these timings represents the total workload executed by each module, excluding file I/O operations. These measurements revealed a tenfold reduction in total CPU time required to analyze each image (Fig. 6a).

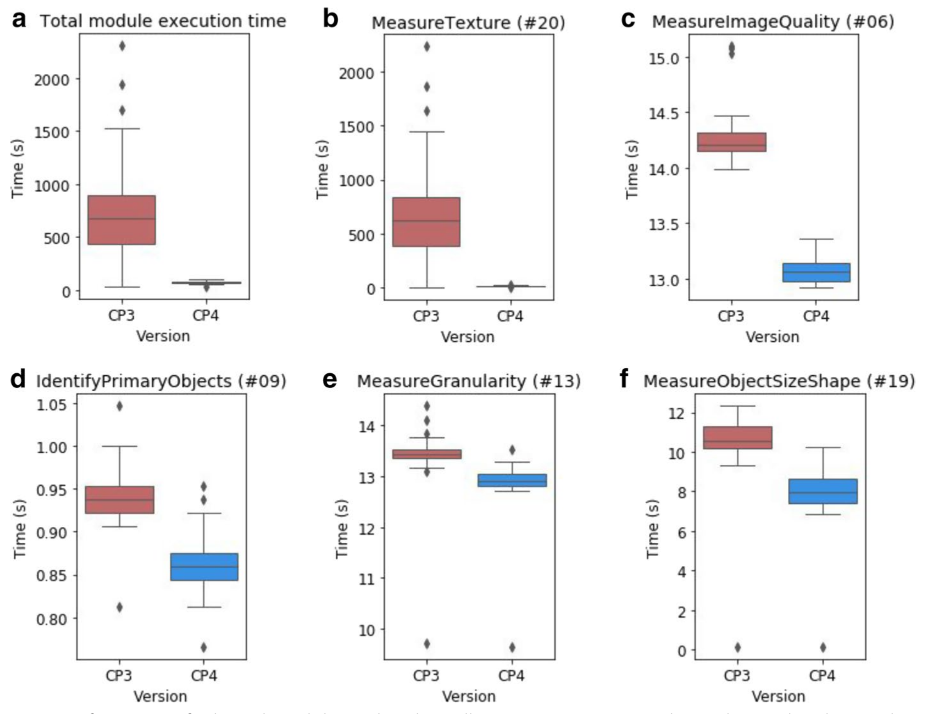

Fig. 6 Performance of selected modules within the Cell Painting assay protocol**.** Numbers in brackets within panel titles correspond to modules in Additional file 1: Figure S1. **a** Total module execution time (measured in CPU time) per image set for all modules in the pipeline. **b** Execution time for the MeasureTexture module per image set. **c** Execution time for the MeasureImageQuality module per image set. **d** Execution time for the IdentifyPrimaryObjects module per image set. **e** Execution time for the MeasureGranularity module per image set. **f** Execution time for the MeasureObjectSizeShape module per image set

In keeping with our expectations, the refinements to MeasureTexture contributed the majority of the performance improvements that we observed (Fig. 6b). We also noted small improvements in the MeasureImageQuality (Fig. 6c), IdentifyPrimaryObjects (Fig. 6d), MeasureGranularity (Fig. 6e) and MeasureObjectSizeShape (Fig. 6f) modules. Other modules in the pipeline exhibited similar performance in both versions or took negligible time to execute (Additional file 1: Figure S1).

Discussion

As the adoption of high content microscopy methods continues to expand there may be several areas where CellProfiler could be expanded with new functionality. Analysis of tissue sections stands out as a potential area of improvement. The large file sizes associated with tissue specimens pose a challenge for image analysis, as system memory typically is not sufficient to load the entire image at once, a bottleneck which could be avoided by adopting packages such as Dask [13] as a means of handling such images by loading subsections of an image on-demand. This would expand the utility of CellProfiler within the digital pathology field.

We also aim to continue adding support for 3D analysis to modules that currently only support 2D workflows. While segmentation is possible in 3D pipelines, additional tools and measurements will be valuable for laboratories using CellProfiler. Alongside this, further performance improvements will continue to benefit researchers, particularly when working with large datasets.

The splitting of cellprofiler-core into a standalone package has also laid the groundwork for producing a stable API for use in other Python-based applications. This will eventually allow users to modify and execute pipelines from within environments such as Jupyter, which may be of benefit to researchers looking to automate complex workflows. This API could provide a higher-level interface for common image processing tasks such as object segmentation, which would simplify the workflow for analyzing images directly within a Python environment and could serve as a bridge to Python tools whose GUI is incompatible with CellProfiler's, such as Napari [14]. The current implementation provides access to all of CellProfiler's important classes and the ability to run pipelines or individual modules. For future development we would like to introduce a more convenient system for programmatically generating image sets without the need for the original input modules or CSV files.

In recent years there has been considerable development towards deep learning models which can perform image segmentation in an automatic manner. Providing access to these algorithms would be of substantial benefit to CellProfiler's users, however the need for dedicated hardware and software to run these models poses a challenge for packaging and distribution. To avoid compatibility issues with older hardware, as well as to minimise the software dependencies needed to run CellProfiler, our approach has been to develop independent plugin modules which are distributed separately from the main CellProfiler program. For CellProfiler 3 we previously released a plugin for NucleAIzer [15], and in the future we hope to investigate integrations with other popular models such as Cellpose [16] and Stardist [17].

Conclusions

The migration of CellProfiler to Python 3 will ensure that the software will remain accessible and maintainable in the coming years. In CellProfiler 4 we have further refined the user interface and introduced new modules and features to help scientists to develop and execute their analysis workflows. The targeted performance improvements in this version will substantially reduce computational costs associated with high throughput image analysis, broadening the potential applications for this open-source software package.

Availability and requirements

Project name: CellProfiler.

Project home page: https://cellprofiler.org/

Operating system(s): Windows, MacOS, Linux.

Programming language: Python 3.

Other requirements: Java 1.6+(JDK 14 bundled with builds).

License: BSD 3-Clause License.

Any restrictions to use by non-academics: None.

Abbreviations

API: Application Programming Interface; CP3: CellProfiler 3; CP4: CellProfiler 4; I/O: Input/output.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s12859-021-04344-9.

Additional file 1: Figure S1. Execution times of all modules within the Cell Painting example pipeline. Measured as per-image CPU time taken for each module in the Cell Painting assay protocol (n = 48). I/O loading operations in the Images module are not recorded by these measurements.

Acknowledgements

The authors would like to thank Katrin Leinweber, Vito Zanotelli, Erin Weisbart, Chris Allen, Nasim Jamali, and Pearl Ryder for contributions to bug fixes and testing of pre-release versions of this software. We also thank all the members of the bioimaging community who have provided feedback and suggestions which have helped to guide this work.

Authors' contributions

DRS, MJS-B, AML, BAC, and AG developed the software. DRS wrote the manuscript, with editorial contribution and supervision from AEC and BAC. All authors read and approved the final manuscript.

Funding

This work was supported by National Institutes of Health Grants (R35 GM122547 and P41 GM135019 to AEC). This project has been made possible in part by Grant Numbers 2018-192059 to AG and 2020-225720 to BAC from the Chan Zuckerberg Initiative DAF, an advised fund of Silicon Valley Community Foundation. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Availability of data and materials

CellProfiler 4 is open-source software which has been made freely available to the scientific community. Pre-compiled builds for Windows and MacOS, as well as documentation manuals, are available at http://cellprofiler.org. Source code is available at https://github.com/CellProfiler/CellProfiler. Benchmarking and visualizations presented in Figs. 1c, d, f, 3a–c, 4b, c, and 5a was performed with the publicly available pipelines and image set "ExampleFly" (https://github.com/CellProfiler/examples/tree/master/ExampleFly)). Benchmarking and visualizations presented in Figs. 1a, 3d, and 4a were performed with the publicly available "3D Monolayer" pipeline and image set which can be accessed at https://github.com/CellProfiler/tutorials/tree/master/3d_monolayer. Cell Painting benchmarking experiments in Figs. 5b, 6, and Additional file 1: Figure S1 made use of a previously published data set (Plate 37983 from https://bbbc.broadinstitute.org/BBBC025) and pipeline (analysis.cppipe from https://github.com/carpenterlab/2016_bray_natprot/blob/master/supplementary_files/cell_painting_pipelines.zip, cited in [11]). The pipeline for this data set was originally written for CellProfiler 2, and so was adjusted to run on CellProfiler 3 and CellProfiler 4 with comparable outputs. These adjusted pipelines as well as the sample data and pipeline used to produce Fig. 2 are provided in a public GitHub repository (https://github.com/carpenterlab/2021_Stirling_BMCBioInformatics).

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Imaging Platform, Broad Institute of MIT and Harvard, Cambridge, MA, USA. [2]Software Department, Allen Institute for Cell Science, Seattle, WA, USA.

Received: 7 July 2021 Accepted: 27 August 2021
Published online: 10 September 2021

References
1. Schneider CA, Rasband WS, Eliceiri KW. NIH Image to ImageJ: 25 years of image analysis. Nat Methods. 2012;9(7):671–5.
2. Bankhead P, Loughrey MB, Fernández JA, Dombrowski Y, McArt DG, Dunne PD, et al. QuPath: open source software for digital pathology image analysis. Sci Rep. 2017;7(1):1–7.
3. Berg S, Kutra D, Kroeger T, Straehle CN, Kausler BX, Haubold C, et al. ilastik: interactive machine learning for (bio) image analysis. Nat Methods. 2019;16(12):1226–32.
4. Carpenter AE, Jones TR, Lamprecht MR, Clarke C, Kang IH, Friman O, et al. Cell profiler: image analysis software for identifying and quantifying cell phenotypes. Genome Biol. 2006;7(10):R100.
5. Wiesmann V, Franz D, Held C, Münzenmayer C, Palmisano R, Wittenberg T. Review of free software tools for image analysis of fluorescence cell micrographs. J Microsc. 2015;257:39–53. https://doi.org/10.1111/jmi.12184.
6. McQuin C, Goodman A, Chernyshev V, Kamentsky L, Cimini BA, Karhohs KW, et al. Cell profiler 30: next-generation image processing for biology. PLoS Biol. 2018;16(7):e2005970.
7. Beg M, Taka J, Kluyver T, Konovalov A, Ragan-Kelley M, Thiery NM, et al. Using Jupyter for reproducible scientific workflows. Comput Sci Eng. 2021;23:36–46. https://doi.org/10.1109/mcse.2021.3052101.
8. Kamentsky L, Jones TR, Fraser A, Bray M-A, Logan DJ, Madden KL, et al. Improved structure, function and compatibility for Cell Profiler: modular high-throughput image analysis software. Bioinformatics. 2011;27(8):1179–80.
9. Sauvola J, Pietikäinen M. Adaptive document image binarization. Pattern Recognit. 2000;33(2):225–36.
10. van der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yager N, et al. scikit-image: image processing in Python. PeerJ. 2014;2:e453.
11. Bray M-A, Singh S, Han H, Davis CT, Borgeson B, Hartland C, et al. Cell Painting, a high-content image-based assay for morphological profiling using multiplexed fluorescent dyes. Nat Protoc. 2016;11(9):1757–74.
12. Ljosa V, Sokolnicki KL, Carpenter AE. Annotated high-throughput microscopy image sets for validation. Nat Methods. 2012;9(7):637–637.
13. Dask RM. Parallel computation with blocked algorithms and task scheduling. In: Proceedings of the 14th python in science conference. Kathryn Huff and James Bergstra; 2015. p. 130–6.
14. napari contributors. napari: a multi-dimensional image viewer for python. 2019. https://zenodo.org/record/4968798.
15. Hollandi R, Szkalisity A, Toth T, Tasnadi E, Molnar C, Mathe B, et al. nucleAIzer: a parameter-free deep learning framework for nucleus segmentation using image style transfer. Cell Syst. 2020;10(5):453-8.e6.
16. Stringer C, Wang T, Michaelos M, Pachitariu M. Cellpose: a generalist algorithm for cellular segmentation. Nat Methods. 2021;18(1):100–6.
17. Schmidt U, Weigert M, Broaddus C, Myers G. Cell detection with star-convex polygons. Medical image computing and computer assisted intervention—MICCAI 2018. 2018. p. 265–73. https://doi.org/10.1007/978-3-030-00934-2_30.