

1 **CellProfiler 4: Improvements in Speed, Utility and Usability**

2

3 David R. Stirling¹, Madison J. Swain-Bowden², Alice M. Lucas¹, Anne E. Carpenter¹, Beth A. Cimini^{1,*},
4 Allen Goodman^{1*}

5 ¹=Imaging Platform, Broad Institute of MIT and Harvard, Cambridge MA, USA

6 ²= Software Department, Allen Institute for Cell Science, Seattle, WA, USA

7 * = These authors contributed equally

8 ^= To whom correspondence should be addressed, bcimini@broadinstitute.org

1 **CellProfiler 4: Improvements in Speed, Utility and Usability**

2 Abstract

3 Background

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

10 Results

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

16 Conclusions

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

20 **Keywords**

21 Image Analysis, Microscopy, Image Segmentation, Image Quantitation, Bioimaging

1 **Background**

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

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

21 In 2005 we introduced CellProfiler, an open-source image analysis program which allows users
22 without specific training to automate their image analysis by using modular processing pipelines (4).
23 CellProfiler has been widely adopted by the community, and is currently referenced more than 2000

1 times per year. Built-in modules provide a diverse array of algorithms for analyzing images, which can be
2 further extended through the use of community-developed plugins. In an independent analysis of 15
3 free image analysis tools CellProfiler scored highly in both usability and functionality (5). Our previous
4 release, CellProfiler 3, introduced support for analysis of 3D images to further expand the tool's
5 applications (6). However, some popular features from CellProfiler 2 could not be brought forward into
6 that release and certain modules struggled to operate efficiently in 3D pipelines.

7 **Implementation**

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

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

19 **User interface refinements**

20 Guided by feedback from biologists, we have made several improvements to the CellProfiler
21 user interface with the goal of making the software more accessible and easier to use. The basic 3D
22 viewer introduced in CellProfiler 3.0 has now been replaced with a more fully-featured viewer which
23 allows users to inspect any plane in a volume (Figure 1A). We have also expanded the figure contrast

1 dialogs to give users more granular control over how images are displayed in both 2D and 3D mode
2 (Figure 1B). These changes will help users to better visualize and understand their data.

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

10 We also restored some features which were previously lost in the migration from CellProfiler 2
11 to CellProfiler 3. Most notably, we rebuilt the Workspace Viewer, where users construct a customized
12 view of their data and can stay focused on a specific region of interest as the pipeline is modified (Figure
13 1F), making it much simpler to monitor and refine segmentation of problematic regions of an image. In
14 addition, new icons in the Test Mode pipeline interface provide a stronger visual indication of which
15 module is currently about to be executed, and provide the ability to return to and execute earlier
16 modules in the pipeline. This replicates and replaces the functionality of the slider widget from
17 CellProfiler 2, which could not be carried forward into CellProfiler 3 but was popular with users.

18 **New and restored features**

19 In CellProfiler 4 we introduced several new analysis features and settings. A common workflow
20 issue we identified was that analysts often segment highly variable objects in multiple stages (such as
21 segmenting and masking out bright objects to aid segmentation of similar-but-dimmer objects), but
22 previous versions could not simply treat resulting segmentations as a single object set when performing
23 and exporting measurements. To resolve this we added the CombineObjects module to allow users to

1 merge sets of objects which have been defined separately. A key issue when designing this module was
2 how to handle objects that would overlap if the sets were merged, therefore we built several strategies
3 detailed in Figure 2. The resulting merged set can then be carried forward throughout the pipeline
4 without the need to merge measurement tables outside of CellProfiler.

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

14 We also upgraded several existing modules. We rewrote the Threshold module to allow all pre-
15 existing threshold strategies to be used in 'adaptive' mode, giving users more options in images with
16 highly-variable background. We have also added the Sauvola local thresholding method as an alternative
17 adaptive strategy (9). Previous versions of CellProfiler 2 shipped a version of the Otsu thresholding
18 method that log-transformed the data before applying the threshold; this assisted in the thresholding of
19 dim images, but led users to question why our Otsu values did not match those from other libraries such
20 as scikit-image (10). This inconsistent behavior could be confusing to users, so we began the process of
21 updating that implementation in CellProfiler 3 and completed it in CellProfiler 4. We added a dedicated
22 setting to log transform image data during application of any thresholding method. These new options
23 will assist users in segmenting challenging images.

1 **New Measurements**

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

11 **Results**

12 **Performance improvements**

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

22 In our performance testing of an example analysis pipeline, overall performance was similar to
23 CellProfiler 3 (Figure 3B). However, executing this pipeline in Test Mode was inhibited by unnecessary

1 user interface updates between running individual modules. Optimizing the UI updates sent during test
2 mode reduced the time taken to run an image set in this mode (Figure 3C).

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

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

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

17 Major gains were also made in measurement of the Costes Colocalization Coefficient in the
18 MeasureColocalization module. This statistic requires the calculation of Costes' automatic threshold,
19 which is generated by thresholding the two images to be compared and then serially reducing the
20 thresholding value until the Pearson R correlation between the two thresholded images drops below a
21 value of 0. Our original implementation would reduce the candidate value in images scaled 0-1 by 1/255
22 at each step, which was inappropriate for 16-bit images containing 65,536 grey levels rather than the
23 256 present in 8 bit images. Testing 65,536 candidate thresholds in 16-bit images would be excessively

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

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

1 **Performance in common workflows**

2 To examine the impact of our changes on performance on a large heterogeneous workflow, we
3 compared the performance of CellProfiler 3 to CellProfiler 4 when running the Cell Painting assay
4 protocol (11). This workflow is typically performed on large datasets in a cluster environment, so we
5 selected a sample of 48 image sets from a published dataset and measured processing times on a single
6 machine. Execution times were captured for each module across three independent runs of this dataset.
7 The sum of these timings represents the total workload executed by each module, excluding file I/O
8 operations. These measurements revealed a 10-fold reduction in total CPU time required to analyze
9 each image (Figure 6A).

10 In keeping with our expectations, the refinements to MeasureTexture contributed the majority
11 of the performance improvements that we observed (Figure 6B). We also noted small improvements in
12 the MeasureImageQuality (Figure 6C), IdentifyPrimaryObjects (Figure 6D), MeasureGranularity (Figure
13 6E) and MeasureObjectSizeShape (Figure 6F) modules. Other modules in the pipeline exhibited similar
14 performance in both versions or took negligible time to execute (Figure S1).

15 **Discussion**

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

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

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

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

1

2 **Conclusions**

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

9 **Availability and Requirements**

10 Project name: CellProfiler
11 Project home page: <https://cellprofiler.org/>
12 Operating system(s): Windows, MacOS, Linux
13 Programming language: Python 3
14 Other requirements: Java 1.6+ (JDK 14 bundled with builds)
15 License: BSD 3-Clause License
16 Any restrictions to use by non-academics: None

17 **List of Abbreviations**

18 API - Application Programming Interface
19 CP3 - CellProfiler 3
20 CP4 - CellProfiler 4

1 I/O - input/output

2 **Declarations**

3 **Ethics approval and consent to participate**

4 Not applicable.

5 **Consent to publish**

6 Not applicable.

7 **Availability of data and materials**

8 CellProfiler 4 is open-source software which has been made freely available to the scientific community.

9 Pre-compiled builds for Windows and MacOS, as well as documentation manuals, are available at

10 <http://cellprofiler.org>. Source code is available at <https://github.com/CellProfiler/CellProfiler>.

11 Benchmarking and visualizations presented in Figures 1C, 1D, 1F, 3A-C, 4B-C, and 5A was performed

12 with the publicly available pipelines and image set

13 “ExampleFly”(<https://github.com/CellProfiler/examples/tree/master/ExampleFly>). Benchmarking and

14 visualizations presented in Figures 1A, 3D, and 4A were performed with the publicly available “3D

15 Monolayer” pipeline and image set which can be accessed at

16 https://github.com/CellProfiler/tutorials/tree/master/3d_monolayer . Cell Painting benchmarking

17 experiments in Figures 5B, 6, and S1 made use of a previously published data set (Plate 37983 from

18 <https://bbbc.broadinstitute.org/BBBC025>) and pipeline (analysis.cppipe from

19 https://github.com/carpenterlab/2016_bray_natprot/blob/master/supplementary_files/cell_painting_p

20 [ipelines.zip](https://github.com/carpenterlab/2016_bray_natprot/blob/master/supplementary_files/cell_painting_p), cited in (11)). The pipeline for this data set was originally written for CellProfiler 2, and so

21 was adjusted to run on CellProfiler 3 and CellProfiler 4 with comparable outputs. These adjusted

1 pipelines as well as the sample data and pipeline used to produce Figure 2 are provided in a public
2 GitHub repository (https://github.com/carpenterlab/2021_Stirling_submitted).

3 **Competing interests**

4 The authors declare that they have no competing interests.

5 **Funding**

6 This work was supported by National Institutes of Health grants (R35 GM122547 and P41 GM135019 to
7 AEC). This project has been made possible in part by grant numbers 2018-192059 to AG and 2020-
8 225720 to BAC from the Chan Zuckerberg Initiative DAF, an advised fund of Silicon Valley Community
9 Foundation. The funders had no role in study design, data collection and analysis, decision to publish, or
10 preparation of the manuscript.

11 **Authors' contributions**

12 DRS, MJS-B, AML, BAC, and AG developed the software. DRS wrote the manuscript, with editorial
13 contribution and supervision from AEC and BAC.

14 **Acknowledgements**

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

19

20

1 References

- 2 1. Schneider CA, Rasband WS, Eliceiri KW. NIH Image to ImageJ: 25 years of image analysis. *Nat Methods*. 2012 Jul;9(7):671–5.
- 4 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 Dec 4;7(1):1–7.
- 6 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 Dec;16(12):1226–32.
- 8 4. Carpenter AE, Jones TR, Lamprecht MR, Clarke C, Kang IH, Friman O, et al. CellProfiler: image analysis software for identifying and quantifying cell phenotypes. *Genome Biol*. 2006 Oct 31;7(10):R100.
- 11 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 [Internet]. Vol. 257, *Journal of Microscopy*. 2015. p. 39–53. Available from: <http://dx.doi.org/10.1111/jmi.12184>
- 14 6. McQuin C, Goodman A, Chernyshev V, Kamentsky L, Cimini BA, Karhohs KW, et al. CellProfiler 3.0: Next-generation image processing for biology. *PLoS Biol*. 2018 Jul;16(7):e2005970.
- 16 7. Beg M, Taka J, Kluyver T, Konovalov A, Ragan-Kelley M, Thiery NM, et al. Using Jupyter for Reproducible Scientific Workflows [Internet]. Vol. 23, *Computing in Science & Engineering*. 2021. p. 36–46. Available from: <http://dx.doi.org/10.1109/mcse.2021.3052101>
- 19 8. Kamentsky L, Jones TR, Fraser A, Bray M-A, Logan DJ, Madden KL, et al. Improved structure, function and compatibility for CellProfiler: modular high-throughput image analysis software. *Bioinformatics*. 2011 Apr 15;27(8):1179–80.
- 22 9. Sauvola J, Pietikäinen M. Adaptive document image binarization. *Pattern Recognit*. 2000 Feb;33(2):225–36.
- 24 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 Jun 19;2:e453.
- 26 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 Sep;11(9):1757–74.
- 29 12. Ljosa V, Sokolnicki KL, Carpenter AE. Annotated high-throughput microscopy image sets for validation. *Nat Methods*. 2012 Jun 28;9(7):637–637.
- 31 13. Rocklin M. Dask: 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.
- 33 14. napari contributors (2019). napari: a multi-dimensional image viewer for python [Internet]. Available from: <https://zenodo.org/record/4968798>

1 15. Hollandi R, Szkalisity A, Toth T, Tasnadi E, Molnar C, Mathe B, et al. nucleAlzer: A Parameter-free
2 Deep Learning Framework for Nucleus Segmentation Using Image Style Transfer. *Cell Syst.* 2020
3 May 20;10(5):453–8.e6.

4 16. Stringer C, Wang T, Michaelos M, Pachitariu M. Cellpose: a generalist algorithm for cellular
5 segmentation. *Nat Methods.* 2021 Jan;18(1):100–6.

6 17. Schmidt U, Weigert M, Broaddus C, Myers G. Cell Detection with Star-Convex Polygons [Internet].
7 Medical Image Computing and Computer Assisted Intervention – MICCAI 2018. 2018. p. 265–73.
8 Available from: http://dx.doi.org/10.1007/978-3-030-00934-2_30

9

10

1 **Figure Legends**

2

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

11

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

17

18 **Figure 3 – General performance in CellProfiler 3 vs CellProfiler 4.** Results represent independent runs
19 on a machine running Windows 10, using 1 worker process. (A) Time from launching the CellProfiler
20 executable to display of the full GUI (n=5). (B) Time taken to run the ExampleFly pipeline in Analysis
21 Mode (n=3). (C) Time to run the ExampleFly pipeline in Test Mode (n=5). (D) Time to run the 3D
22 monolayer tutorial pipeline in Analysis Mode (n=3).

23

24 **Figure 4 – Module-specific performance improvements.** Results from individual module testing on a

1 machine running Windows 10. (A) Execution time for the MedianFilter module running within the 3D
2 Monolayer pipeline (n=5). (B) Execution time when running per-object texture measurements on data
3 from the ExampleFly pipeline (n=5). (C) Execution time when running MeasureColocalization on 8-bit
4 images from the ExampleFly pipeline (n=5).

5

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

11

12 **Figure 6 – Performance of selected modules within the Cell Painting assay protocol.** Numbers in
13 brackets within panel titles correspond to modules in Figure S1. (A) Total module execution time
14 (measured in CPU time) per image set for all modules in the pipeline. (B) Execution time for the
15 MeasureTexture module per image set. (C) Execution time for the MeasureImageQuality module per
16 image set. (D) Execution time for the IdentifyPrimaryObjects module per image set. (E) Execution time
17 for the MeasureGranularity module per image set. (F) Execution time for the MeasureObjectSizeShape
18 module per image set.

19

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

23

24

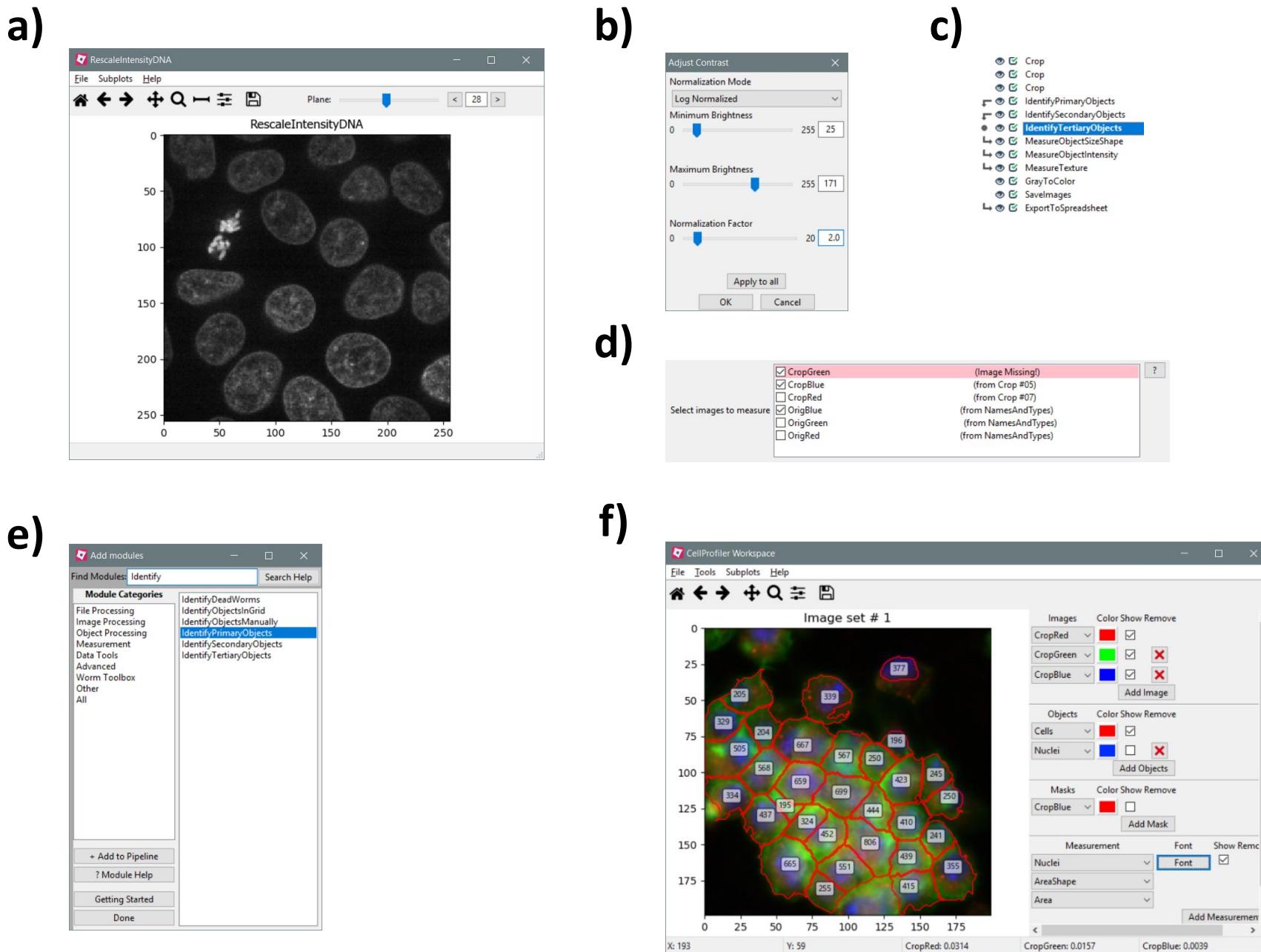
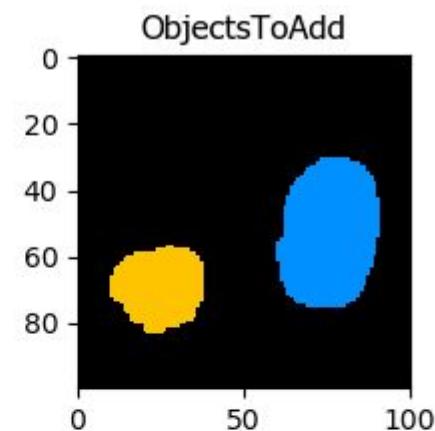
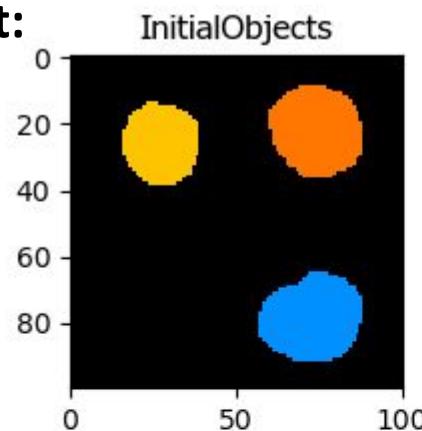


Figure 1

Input:



Method:

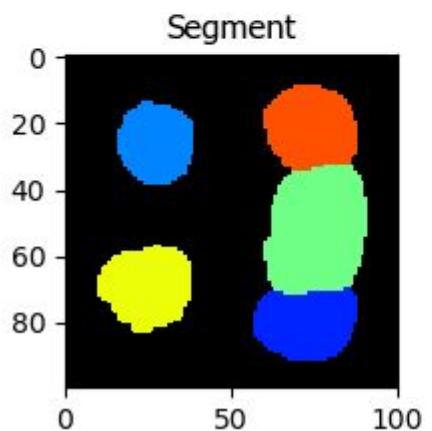
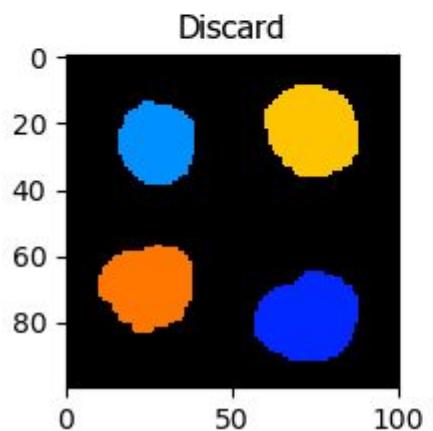
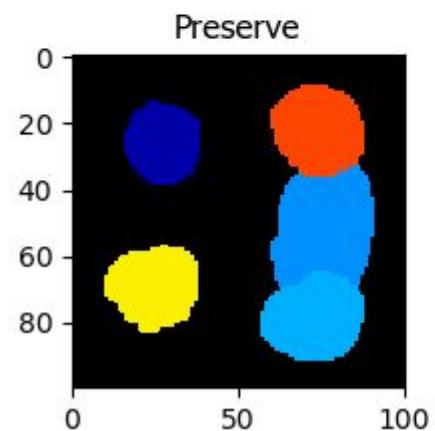
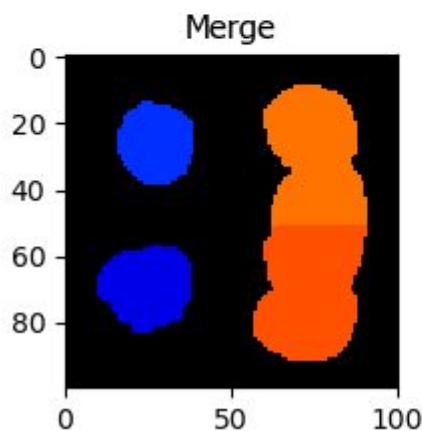


Figure 2

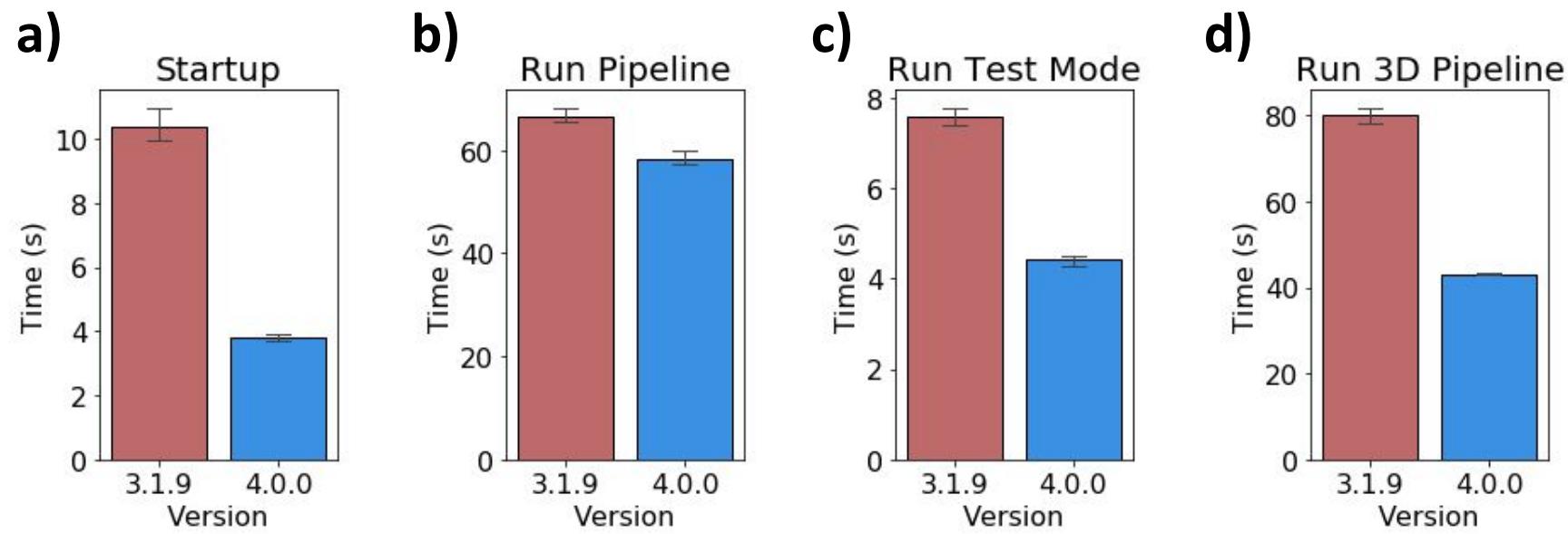


Figure 3

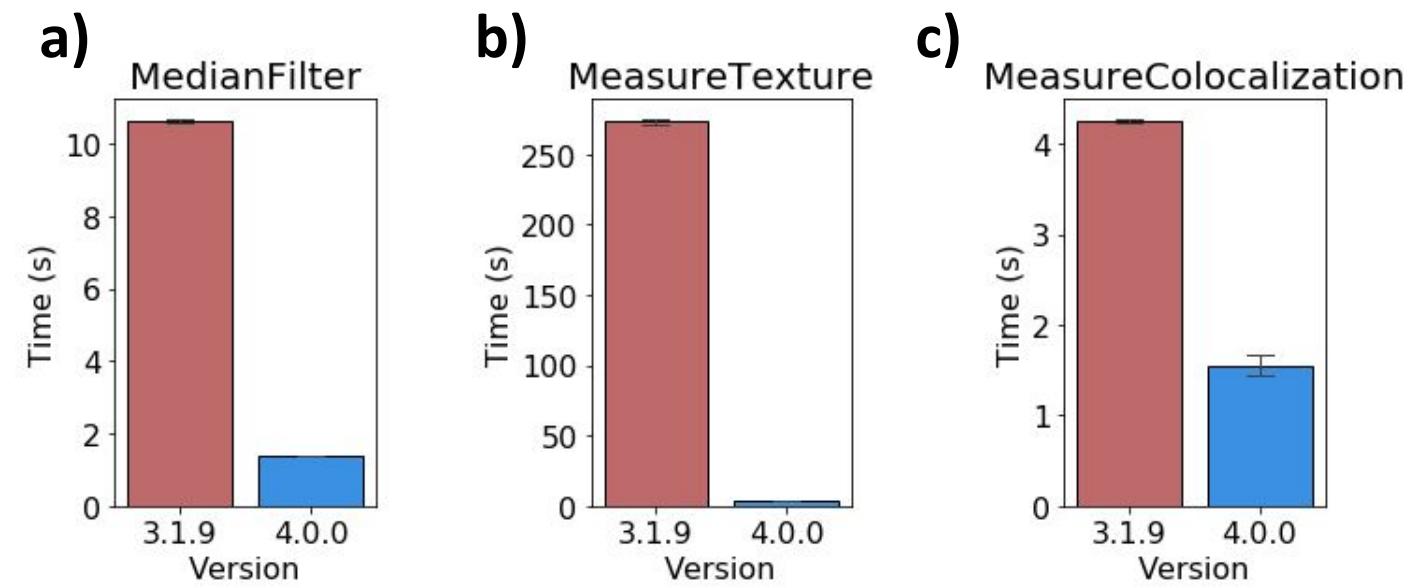


Figure 4

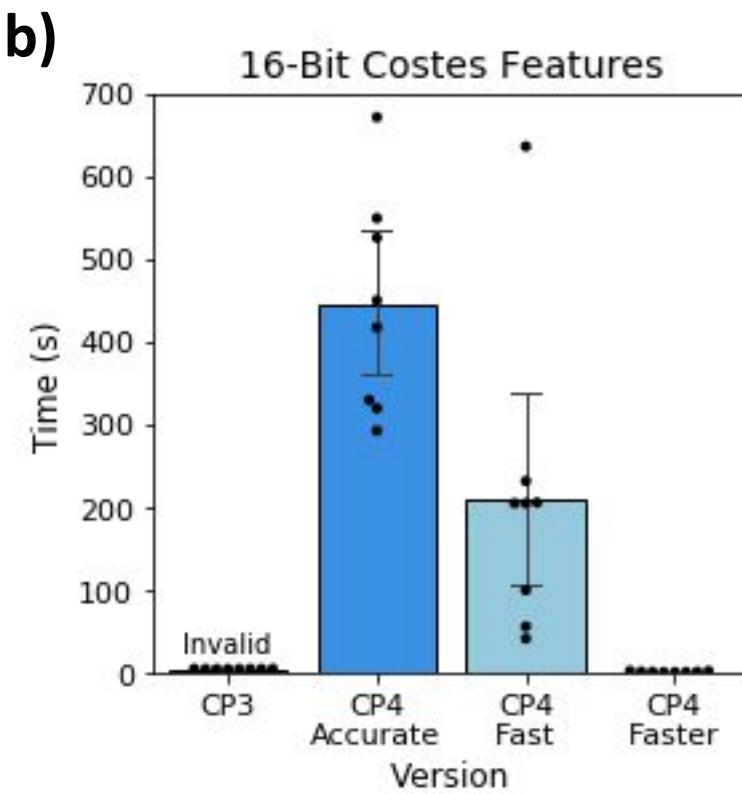
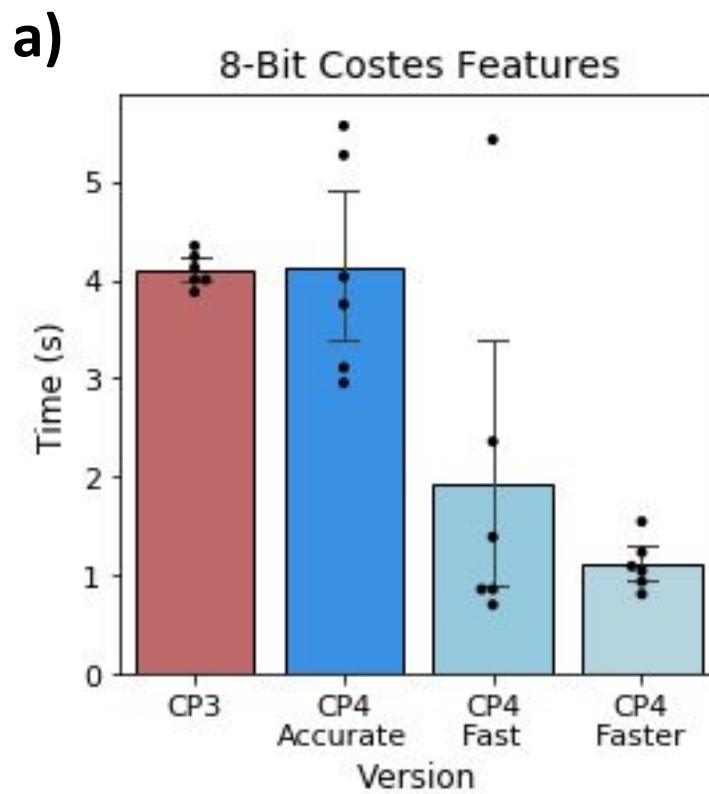


Figure 5

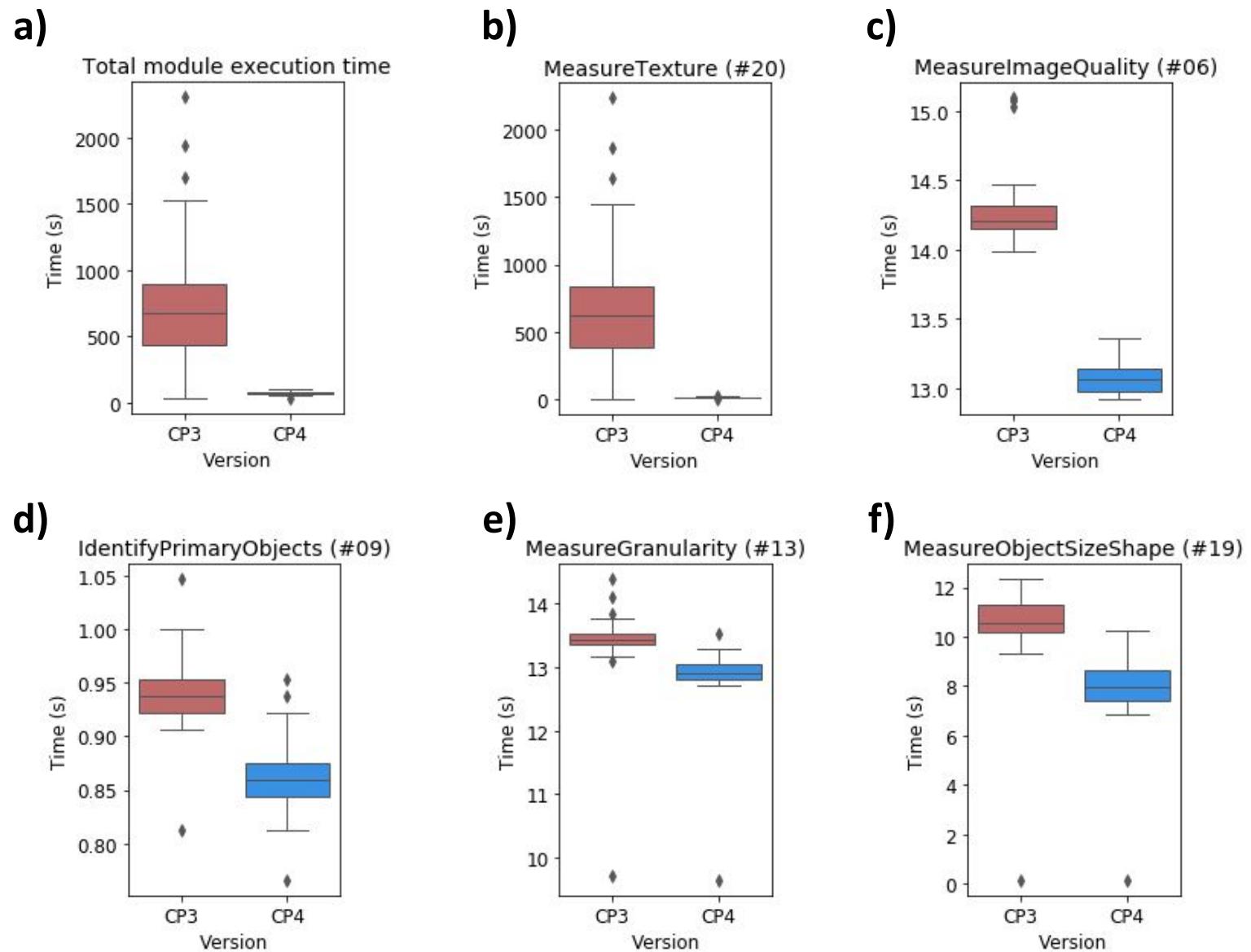


Figure 6

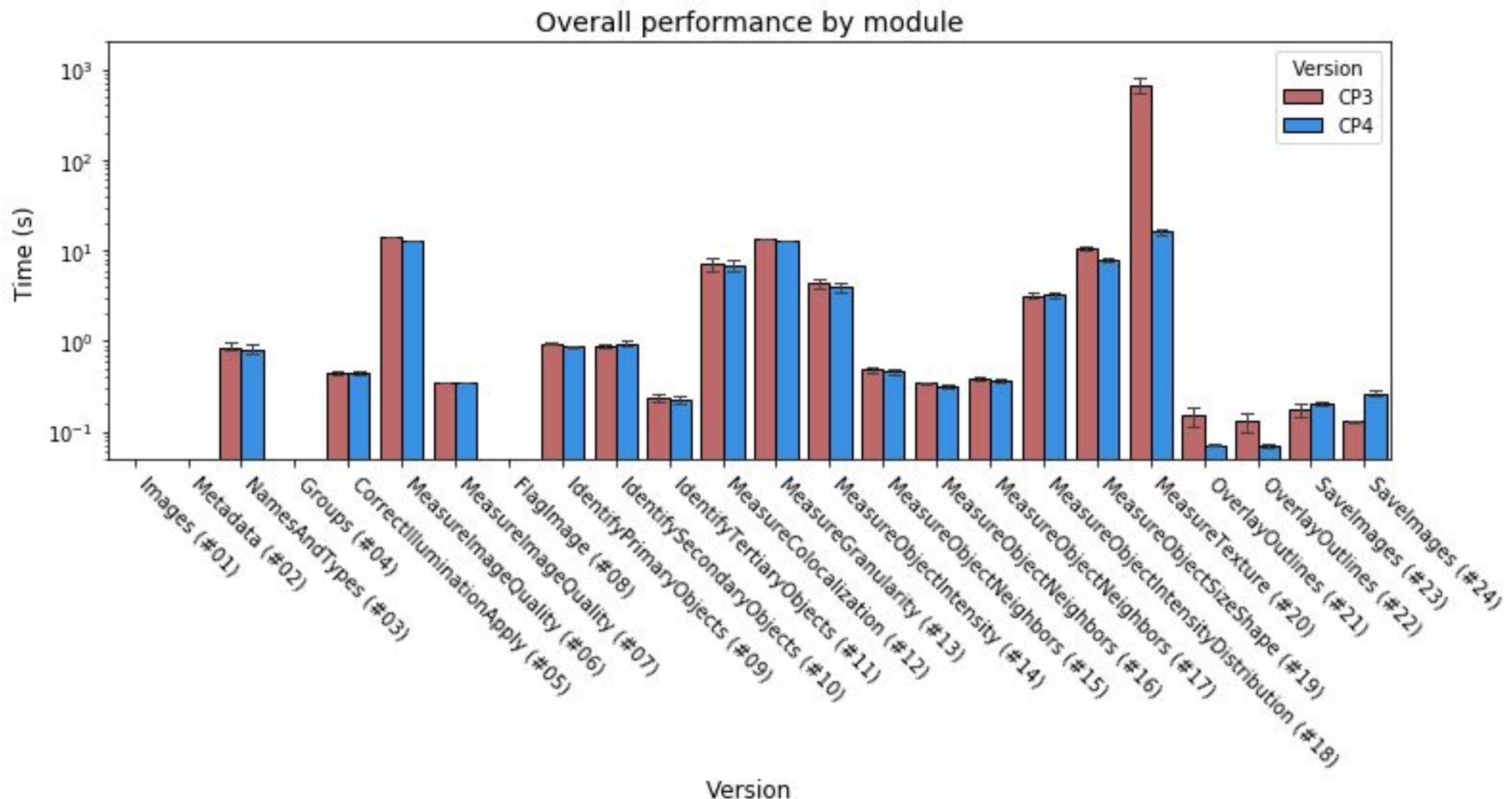


Figure S1