cubic: CUDA-accelerated 3D Bioimage Computing

Alexandr A. Kalinin Anne E. Carpenter Shantanu Singh
Broad Institute of MIT and Harvard
Cambridge, MA 02142, USA
akalinin@broadinstitute.org

Matthew J. O'Meara
University of Michigan Medical School
Ann Arbor, MI 48109, USA
maom@umich.edu

Abstract

Quantitative analysis of multidimensional biological images is useful for understanding complex cellular phenotypes and accelerating advances in biomedical research. As modern microscopy generates ever-larger 2D and 3D datasets, existing computational approaches are increasingly limited by their scalability, efficiency, and integration with modern scientific computing workflows. Existing bioimage analysis tools often lack application programmable interfaces (APIs), do not support graphics processing unit (GPU) acceleration, lack broad 3D image processing capabilities, and/or have poor interoperability for compute-heavy workflows. Here, we introduce cubic*, an open-source Python library that addresses these challenges by augmenting widely-used SciPy and scikit-image APIs with GPU-accelerated alternatives from CuPy and RAPIDS cuCIM.* cubic*'s API is device-agnostic and dispatches operations to GPU when data reside on the device and otherwise executes on CPU—seamlessly accelerating a broad range of image processing routines. This approach enables GPU acceleration of existing bioimage analysis workflows, from preprocessing to segmentation and feature extraction for 2D and 3D data. We evaluate* cubic *both by benchmarking individual operations and by reproducing existing deconvolution and segmentation pipelines, achieving substantial speedups while maintaining algorithmic fidelity. These advances establish a robust foundation for scalable, reproducible bioimage analysis that integrates with the broader Python scientific computing ecosystem including other GPU-accelerated methods, enabling both interactive exploration and automated high-throughput analysis workflows.* cubic *is openly available at* https://github.com/alxndrkalinin/cubic.

1. Introduction

The growing scale and complexity of biological experiments—particularly in drug discovery, functional genomics, and systems biology—have elevated the importance of quantitative bioimage analysis [1, 4, 12]. Advances in microscopy now enable high-throughput imaging of single cells, organoids, and tissues in various imaging modalities and conditions, generating terabytes of multidimensional data [4, 5, 32]. Extracting information from these rich datasets is essential for identifying phenotypic signatures, quantifying cellular states, and linking molecular perturbations to observable outcomes [11, 53]. As a result, computational workflows for image-based profiling are becoming increasingly essential to modern biological discovery and translational research.

Despite this central role, existing bioimage analysis pipelines face significant bottlenecks. While there are robust and open-source image analysis platforms such as ImageJ/Fiji [45, 46] and CellProfiler [47] that have made image processing and morphological profiling broadly accessible to the community, these frameworks are challenging to scale to modern workloads. First, they tightly couple their graphical interfaces and APIs, making it challenging to integrate them into automated high-throughput workflows that employ modern scientific computing and machine learning technology stack, and second, their architecture makes it difficult to leverage GPU acceleration.

By contrast, general purpose image processing libraries like OpenCV [10] and scikit-image [49] offer programmatic flexibility via Python interfaces. Still, these have limited ability to handle the large-scale, high-dimensional datasets generated by modern imaging techniques. The shift toward 3D imaging, time-lapse data, and high-content screens exacerbates limitations in computational efficiency and scal-

ability, making processing pipelines slow, fragmented, or difficult to integrate with machine learning and data science workflows.

Over the last decade, the increasing availability of graphics processing units (GPUs) has enabled end-to-end training of complex neural network architectures on terabyte-scale image datasets [2, 40]. However, it has not translated into similar gains for classical image processing in bioimage analysis. While deep learning frameworks use GPU acceleration to speed up model training, they typically focus on implementing deep learning-specific differentiable modules and operations.

This leaves many commonly used traditional bioimage analysis routines locked to CPU workflows, with limited device flexibility and cumbersome APIs for large-scale data movement and hybrid pipelines. GPU-accelerated solutions are available for specific tasks or as separate plugins, but they often rely on different back-ends, involve complex installation procedures, and expose narrow, custom interfaces that do not always offer interoperability with the wider modern scientific computing and data analysis stack. Thus, researchers navigate a fragmented software landscape, assembling ad-hoc scripts and file conversions instead of concentrating on the biological questions at hand. While libraries mirroring scientific-Python APIs now provide GPU backends [17, 37], integrating them typically demands explicit device management—tracking data placement and routing each call to the matching CPU or GPU implementation.

To address these challenges, we introduce *cubic*, an open-source Python library for morphological analysis of multidimensional bioimages that provides a minimal-code-change interface for the device-agnostic execution with optional GPU acceleration. Specifically, *cubic* uses the CUDA-accelerated libraries CuPy and cuCIM to provide fast, efficient implementations of common image processing operations, including deconvolution, segmentation, and feature extraction from large 3D microscopy datasets. *cubic* is designed to be fully compatible with the Python scientific computing ecosystem, as it mirrors SciPy and scikit-image APIs, allowing users to leverage familiar functions and reducing the changes required to adapt existing codebases. It also supports zero-copy data exchange with PyTorch [40], enabling GPU-resident image processing operations to feed directly into deep learning models without extra memory copies or performance overhead. By transparently leveraging GPU acceleration when available, while remaining fully functional on CPU-only systems, *cubic* bridges the gap between ease of use, scalability, and computational performance, enabling robust and reproducible morphometric workflows for the next generation of bioimage analysis applications. *cubic* is open-source and is available at https://github.com/alxndrkalinin/cubic.

2. Related work

2.1. Traditional bioimage analysis tools

The bioimage analysis ecosystem is rich with widely used, mature platforms that have shaped the field [23]. ImageJ/-Fiji [45, 46] offer a vast plugin ecosystem and user-friendly graphical user interface (GUI) for both 2D and some 3D processing tasks. While widely adopted, it requires scripting macros for large-scale processing, making interoperability with the Python scientific computing stack challenging. CellProfiler [47] is implemented in Python and enables repeatable, modular pipelines for segmentation and feature extraction, and supports headless batch operation. However, it remains entirely CPU-bound and can be prohibitively slow for large volumetric images. Tools like ilastik and QuPath cater to interactive segmentation and classification tasks. Ilastik [8] provides user-friendly pixel/object classification using machine learning, with optional GPU acceleration only within deep learning modules. QuPath [6], designed for whole-slide pathology, now includes GPU support via PyTorch [40], but remains limited in core morphometric pipelines. While these packages excel in usability and community support, they usually lack large 3D image analysis capabilities and GPU acceleration. Moreover, implementation of image analysis operations in interactivity-first tools is usually tightly coupled to the GUI, making it difficult to assemble robust pipelines that involve other tools from the modern scientific Python stack [34].

More recently, napari [13, 16] has emerged as a Python-native, multi-dimensional image viewer that also provides a headless API and a rich plugin ecosystem exposing SciPy and scikit-image [55] APIs. While napari's programmatic interface enables both interactive and scriptable workflows within a single framework, it still relies on plugin registration and viewer-centric constructs even when run headlessly and does not aim to natively provide GPU support for available image analysis operations.

OpenCV [10] and scikit-image [49] are popular libraries for image processing and computer vision that offer programmatic flexibility and implement a wide range of algorithms. OpenCV, while providing a Python interface, primarily focuses on 2D image analysis. scikit-image is built on NumPy [25] and SciPy [50], making it easy to integrate into the scientific Python stack. For example, CellProfiler v4 [47] itself packaged most of its core image-processing routines into scikit-image. Similarly, many end-to-end bioimage analysis pipeline frameworks [3, 15, 38, 39, 41] leverage scikit-image routines under the hood to perform image preprocessing, segmentation, feature extraction, and analysis. However, scikit-image itself is CPU-bound and does not natively support GPU acceleration.

2.2. GPU-accelerated frameworks and tools

Specialized GPU-accelerated plugins for Imagej/Fiji and CellProfiler exist—such as AutoDeconJ [48] for 3D light-field deconvolution in ImageJ—but these remain task-specific and not integrated into the broader scientific computing environment. OpenCL-based CLIJ/CLIJ2 [22, 51] brings hundreds of classical 2D and 3D GPU-accelerated image operations into ImageJ/Fiji. However, CLIJ is also not natively Python-based and primarily operates within the ImageJ/Fiji ecosystem. clEsperanto and its Python binding pyclesperanto [24] address this issue by exposing CLIJ2 operations to Python users, enabling GPU-accelerated image processing in a device-agnostic manner. However, pyclesperanto implements a limited range of bioimage analysis operations with an interface specific to CLIJ2.

Deep learning frameworks such as PyTorch [40] and TensorFlow [2] natively support GPUs, but are tailored to neural network training and inference and do not implement most of the conventional image processing routines. Cytokit [18] implements TensorFlow-based GPU acceleration for image registration, deconvolution and quality scoring, but still relies on CPU-bound scikit-image, CellProfiler and OpenCV routines for preprocessing, segmentation and feature extraction. PyTorch-based libraries such as Kornia [42] and torchvision v2 [33] offer GPU-accelerated augmentations, resampling and basic preprocessing, but are primarily designed for deep-learning workflows and lack the full spectrum of segmentation and morphometric operations.

CuPy [37] and RAPIDS cuCIM [17] represent a different approach, providing GPU-accelerated implementations that closely mirror the APIs of NumPy, SciPy, and scikit-image. CuPy offers a drop-in replacement for NumPy arrays and lower-level signal and image processing operations, while cuCIM extends this paradigm to image processing routines from scikit-image. Unlike device-agnostic solutions, these libraries require explicit data management—users must be aware of which device their data resides on to call the appropriate CPU or GPU implementation accordingly.

Finally, several napari plugins provide GPU-accelerated routines for specific tasks. For example, the napari-accelerated pixel-and-object-classification plugin [20] implements OpenCL-based Random Forest pixel and object classifiers. The pycudadecon plugin [29] implements a CUDA-based accelerated Richardson–Lucy deconvolution [9]. However, these remain narrowly focused on their particular algorithmic domains. The napari-pyclesperanto-assistant [14] plugin integrates clEsperanto kernels, while inheriting pyclesperanto's API and the operation set. The napari-cupy-image-processing plugin [21] exposes GPU-accelerated signal and image processing rountines from CuPy and a handful of skimage-like operations within the napari ecosystem, but its coverage remains far narrower than cuCIM's comprehensive mapping.

3. Methods

3.1. Design principles

cubic implements a balanced design that seeks to address drawbacks of existing bioimage computing tools. We formulated the following design principles to guide the development of *cubic*:

- **P1 Comprehensive image processing operations.** Bioimage analysis workflows require diverse processing steps from preprocessing to quantitative measurement, but combining multiple specialized tool can be challenging. *cubic* aims to provide a wide range of operations including deconvolution, filtering, segmentation, feature extraction, morphological operations, and image metrics commonly used in microscopy workflows. This unified approach reduces dependency management and integration complexity.
- **P2 API decoupled from GUI.** Tight coupling of the user interface and the logic is an anti-pattern. *cubic* exposes all core functionality through programmatic interfaces, enabling seamless integration into custom pipelines and automated workflows without depending on graphical interfaces. This design maximizes flexibility and scriptability while requiring users to have programming expertise rather than relying on point-and-click interfaces.
- **P3 Multidimensional image support.** Modern microscopy generates complex multi-dimensional datasets including 3D image volumes that many tools handle inconsistently. *cubic* provides native support for 2D, 3D, and higher-dimensional image stacks with consistent APIs across all dimensionalities. This unified approach simplifies analysis of complex datasets but increases implementation complexity and memory requirements for higher-dimensional operations.
- **P4 GPU acceleration.** Computationally intensive bioimage processing can be prohibitively slow on CPU-only systems, especially for large 3D images. *cubic* leverages GPU hardware through CUDA libraries while maintaining automatic fallback to CPU implementations when GPU resources are unavailable. This approach delivers significant performance improvements for supported operations while maintaining broad compatibility.
- **P5 Scientific Python integration.** Python has emerged as the dominant language for data and image analysis due to its extensive ecosystem of scientific libraries, clear syntax, and strong community support. *cubic* ensures seamless interoperability with established libraries like NumPy, SciPy, and scikit-image through consistent array interfaces and data types.
- **P6 Device agnosticism.** Hardware-specific code creates maintenance burden and substantial refactoring of existing codebases. *cubic* enables identical code to ex-

ecute on both CPU and GPU hardware through unified array interfaces, allowing transparent acceleration without code modification. This approach maximizes code reusability and simplifies deployment.

3.2. Key features

Based on the survey of existing bioimage analysis tools in Section 2 and our design principles, we chose to implement *cubic* using SciPy [50] and scikit-image [49], combined with CuPy [37] and RAPIDS cuCIM [17]. Together, SciPy's lower-level signal and n-dimensional image processing capabilities (deconvolution, filtering) and scikit-image's extensive higher-level image analysis operations (segmentation, morphological operations, feature extraction) provide one of the most comprehensive collections of bioimage processing and analysis tools under a unified programmatic interface (**P1**, **P2**). Both libraries also natively support multidimensional arrays with consistent operations across 2D and 3D data (**P3**). Using CuPy and cuCIM as drop-in replacements for NumPy/SciPy and scikit-image enables GPU-accelerated array operations and image processing functions with mostly identical function signatures (**P4**). Supporting highly popular SciPy/scikit-image APIs enables seamless integration with the broader NumPy ecosystem and existing scientific Python workflows through consistent array interfaces and data types (**P5**).

Although CuPy and cuCIM mostly mirror SciPy/scikit-image's API, they still require placing the input array to the correct device and importing matching device-specific functions. We simplify this requirement by providing a device-agnostic interface that automatically dispatches calls to the array's current device, such that unmodified code executes on both CPU and GPU (**P6**). If a CUDA implementation is unavailable or no GPU is detected, *cubic* transparently falls back to the CPU version of each routine, preserving compatibility while delivering hardware-accelerated performance where possible. Through this wrapping approach, *cubic* provides access to the near-complete functionality of scikit-image's comprehensive API, encompassing the broad spectrum of image processing operations including color space conversions, feature detection, filtering, morphological operations, segmentation algorithms, geometric transformations, image registration, deconvolution, and feature extraction. By using the device-agnostic API and simply updating import statements, *cubic* easily allows adding GPU acceleration to existing scikit-image code.

cubic also implements custom algorithms not covered by existing libraries, including advanced image quality metrics such as Fourier Ring Correlation (FRC) and Fourier Shell Correlation (FSC) [7, 35] for resolution assessment, scale-invariant Peak Signal-to-Noise Ratio (PSNR) [54] and Structural Similarity Index Measure (SSIM) [52], segmentation quality metrics such as Average Precision (AP) [19], alternative implementations [36] of Lucy-Richardson deconvolution [31, 44], and various specialized image utility operations for bioimage analysis, all available in both 2D and 2D. This comprehensive coverage ensures that researchers have access to an even wider array of image processing operations needed for bioimage analysis workflows while maintaining the device-agnostic execution model.

3.3. Device-agnostic processing

To illustrate the disadvantages of device-specific APIs, the snippet below shows attempts to apply a Gaussian filter to a 3D image of cell nuclei using both CPU-based scikit-image and GPU-based cuCIM implementations:

```
import cupy as cp
import numpy as np
from skimage import data
import skimage.filters as filters_cpu
import cucim.skimage.filters as filters_gpu

# load example 3D image of cells
img = data.cells3d()
# select nuclei channel
img = img[:, 1] # ZYX (60, 256, 256)

# try running Gaussian filter on GPU
smooth = filters_gpu.gaussian(img)
> TypeError:
> Unsupported type <class 'numpy.ndarray'>

# move img to GPU
img = cp.asarray(img)
# now this works
smooth = filters_gpu.gaussian(img)

# try running Gaussian filter on CPU
smooth = filters_cpu.gaussian(img)
> TypeError: Implicit conversion
> to a NumPy array is not allowed.

# move img back to CPU
img = img.asnumpy()
# only now this works
smooth = filters_cpu.gaussian(img)
```

In this pattern, importing device-specific implementations of each module forces the developer to not only keep track of the current data location, but also to choose the correct device-specific implementation to match it, leading to redundant namespace clutter and cognitive overhead. Otherwise, mismatch between the data location and function implementation triggers unexpected runtime errors. As a result, extending existing scikit-image-based bioimage analysis pipelines to run on GPU demands extensive refactoring to alternate between backends, undermining readability and maintainability.

By contrast, *cubic* consolidates CPU and GPU functionality behind a single API, requiring only one import for each submodule and localizing device transfers to explicit boundary calls. For example:

```python
from cubic.cuda import ascupy
from cubic.skimage import filters

# load example 3D image of cells
img = data.cells3d()
# select nuclei channel
img = img[:, 1] # ZYX (60, 256, 256)

# run Gaussian filter on CPU using skimage
smooth = filters.gaussian(img)

# transfer image to GPU
img = ascupy(img)
# run Gaussian filter on GPU using cuCIM
smooth = filters.gaussian(img)
```

In this example, the GPU code matches exactly the widely-used scikit-image API, except for calling `cubic.skimage` instead of `skimage`. The function calls are identical, however, the operation runs on either CPU and GPU hardware depending on the input array location. The `ascupy` function transfers the input array to the GPU, allowing the use of GPU-accelerated implementations of the same algorithm. The output array is also a GPU array, which can be transferred back to the CPU using `asnumpy` if needed. Persistent output location allows running existing scikit-image code without further modifications. This design allows developers to write device-agnostic code that runs seamlessly on both CPU and GPU hardware, maximizing code reusability and simplifying deployment across different computing environments.

4. Benchmarks and examples

We demonstrate the capabilities of *cubic* using following examples:

- GPU-accelerated image rescaling benchmark.
- Re-implementation of a CellProfiler pipeline for 3D cell monolayer segmentation.
- Richardson-Lucy deconvolution with the optimal iteration selection procedure guided by image quality metrics.

4.1. 3D image rescaling benchmark

We demonstrate *cubic*'s performance advantages through benchmarking image rescaling operations, comparing CPU and GPU execution across different interpolation orders and input image sizes.

Dataset and experimental setup. The benchmark uses the `cells3d` dataset from scikit-image [49], a 3D fluorescence microscopy image stack with dimensions 60×256×256 pixels representing a typical confocal acquisition downscaled in XY to almost isotropic voxel size. To evaluate performance scaling with image size, we additionally test on a 2× XY-upsampled version (60×512×512 pixels), created using bilinear interpolation.

We benchmark upscaling the input image by 2× and then downscaling the result back by 0.5× using `cubic.skimage.trasnform.rescale`, which internally uses `scipy.ndimage.zoom` with optional Gaussian filtering for anti-aliasing. We test all interpolation orders (0-5) supported by SciPy: nearest-neighbor (order 0), linear (order 1), quadratic (order 2), cubic (order 3), quartic (order 4), and quintic (order 5). Anti-aliasing was enabled for downscaling operations to prevent aliasing artifacts.

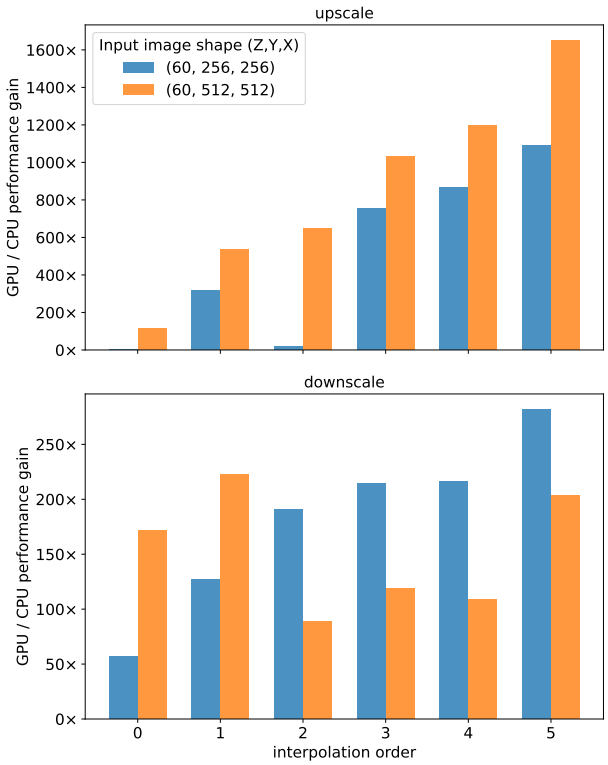
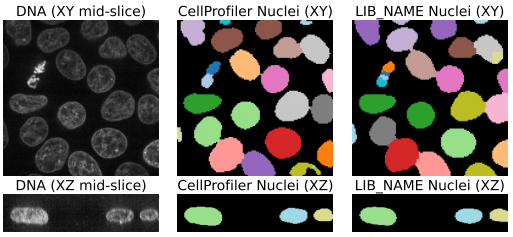
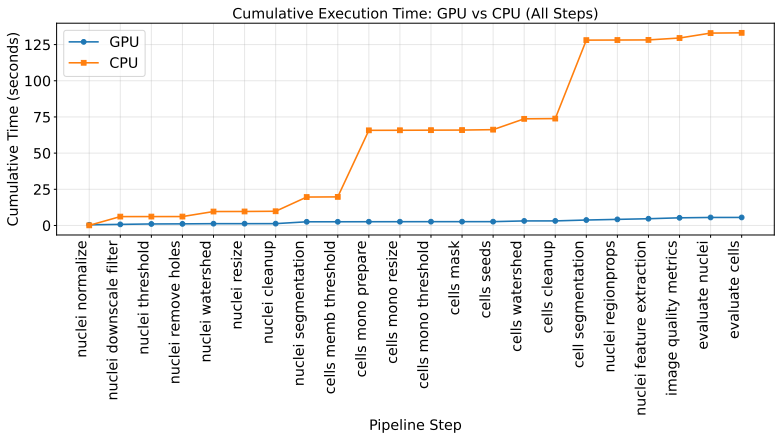


Figure 1. GPU performance gains for 3D image rescaling operations. Results show speedup factors comparing GPU versus CPU execution across interpolation orders (0-5) for two sizes of 3D input images.

Results. We found substantial GPU acceleration across all tested configurations (Figure 1). For upscaling operations, GPU speedup ranges from approximately 10× (order 0) for the original image size to over 1600× (order 5) for the larger upsampled input, with higher-order interpolation methods showing dramatically greater acceleration due to their increased computational complexity. Downscaling operations show more variable but still substantial performance improvements, ranging from 50-300× speedup across different interpolation orders. The performance scaling with input image size confirmes the known advantage of GPU acceleration for larger images, which consistently achieves a speedup of a least 100× across both operations and both input sizes. Even the simplest nearest-neighbor



(a) Visual comparison of DNA channel and nuclei segmentation results. Top row shows XY mid-slice views, bottom row shows XZ mid-slice views. From left to right: original DNA channel, CellProfiler nucleus segmentation, and `cubic` nucleus segmentation. *cubic* implementation produces segmentation results similar to those by the original CellProfiler pipeline (differences are due to CellProfiler internally setting some default parameter values differently from those in scikit-image without exposing them in the GUI).



(b) Cumulative execution time comparison between GPU and CPU implementations.

Figure 2. CellProfiler 3D monolayer segmentation benchmark results. (a) Visual validation showing similar segmentation quality between CellProfiler and *cubic* implementations. (b) Performance comparison demonstrating significant computational speedup using GPU.

interpolation (order 0) shows significant speedups of 10-170×, demonstrating that using *cubic* can provides substantial computational benefits across the entire spectrum of image rescaling operations.

4.2. CellProfiler 3D monolayer segmentation

We demonstrate *cubic*'s capabilities by reproducing a CellProfiler pipeline for 3D monolayer segmentation, while also comparing performance between CPU and GPU execution. Notably, CellProfiler v4 [47] relies on scikit-image for most of its core image processing operations, making our GPU-accelerated *cubic* wrapper a natural performance enhancement for such workflows. Note, that the goal of this example was not to perfectly reproduce CellProfiler pipeline, but to use similar operations to evaluate acceleration allowed by the use of GPU computation.

Dataset description. The experiment uses a 3D fluorescence microscopy image of a cell monolayer from the image set BBBC034v1 Thirstrup et al. 2018, available from the Broad Bioimage Benchmark Collection [30]. The 3D image stack has three channels: membrane (channel 0), mitochondria (channel 1), and DNA (channel 2). The images have pixel dimensions of 0.065 μm in X and Y, with 0.29 μm Z-spacing, representing typical confocal microscopy acquisition parameters for monolayer studies.

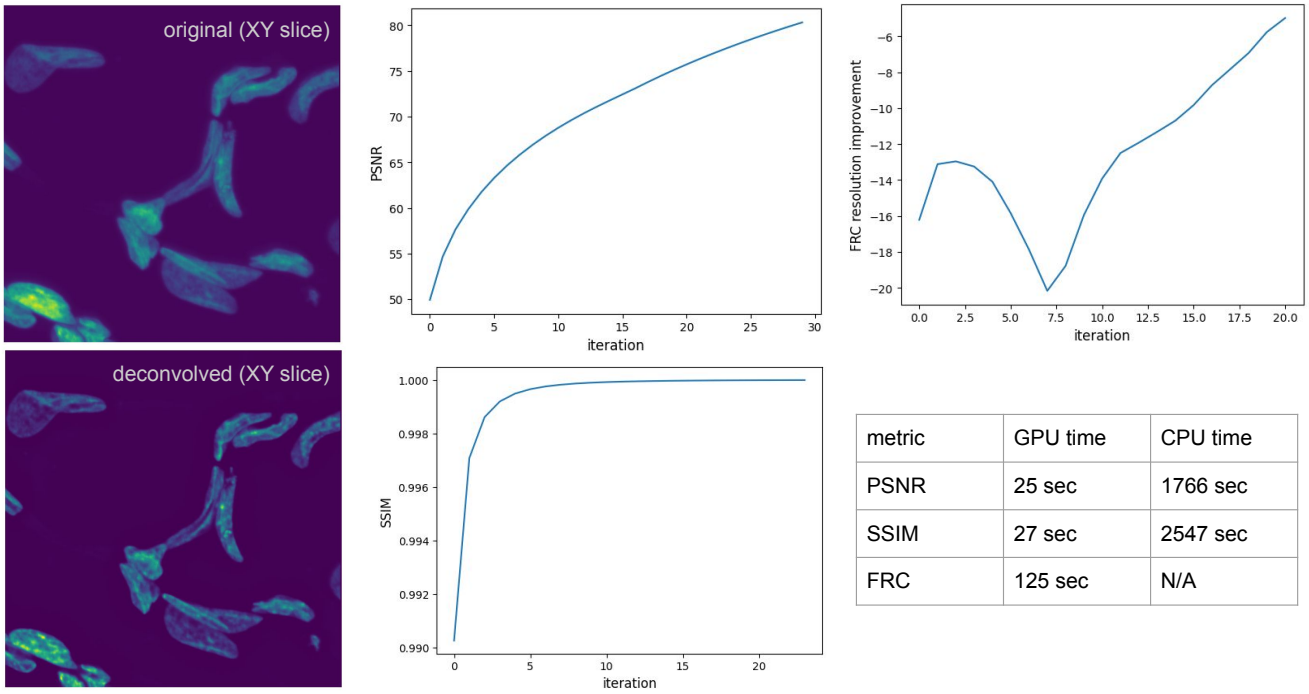
CellProfiler pipeline. The pipeline performs hierarchical segmentation to identify: (1) individual cell nuclei from the DNA channel, and (2) whole cells using membrane signal constrained by nuclear seeds. This two-step approach mirrors common workflows in cell biology where nuclear segmentation provides reliable seeds for subsequent cell boundary detection.

The reference CellProfiler implementation follows a multi-stage approach:

- *Nuclei segmentation*: DNA channel normalization, downscaling (0.5×), median filtering (ball radius 5), Otsu thresholding, hole filling (area threshold 20), watershed segmentation (ball radius 10), upscaling, and size filtering (minimum 50 pixels).
- *Cell segmentation*: Multi-Otsu thresholding of membrane channel (3 classes), hole removal, monolayer mask creation from combined channels with morphological closing (disk radius 17) at 0.25× resolution, seed generation from eroded nuclei (ball radius 5), and watershed segmentation (ball radius 8) with size filtering (minimum 100 pixels).

***cubic* implementation.** Our implementation reproduces this pipeline using GPU-accelerated *cubic* functions, maintaining identical parameter values and processing steps where possible. Additionally, we evaluated segmentation quality against CellProfiler results by calculating average precision (AP) on labeled masks. To both demonstrate performance on standard image-quality measures and tie them to segmentation agreement, we additionally computed peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) on intensity images masked with predicted segmentation labels. When evaluated within the masks, higher PSNR/SSIM values imply closer spatial agreement of segmentation results. The modular design allows easy switching between CPU and GPU execution while preserving numerical accuracy.

Results. Figure 2a shows visual comparison between CellProfiler and *cubic* segmentations in both XY and XZ views, demonstrating high fidelity reproduction of the original pipeline. Performance analysis (Figure 2b) reveals substantial speedup on GPU, with total pipeline acceleration of 25× compared to CPU execution (5.62 vs 133.43 sec). Individual steps show varying degrees of improvement, with



metric	GPU time	CPU time
PSNR	25 sec	1766 sec
SSIM	27 sec	2547 sec
FRC	125 sec	N/A

Figure 3. Richardson-Lucy deconvolution of a large 3D image volume ($30 \times 2160 \times 2560$) with per-iteration image quality metrics tracking and CPU vs GPU performance comparison.

morphological operations and image filtering achieving the largest gains due to their parallel nature.

This example demonstrates the potential ability to replace existing CellProfiler workflows that internally use scikit-image operations to provide significant computational advantages for high-throughput applications.

4.3. Richardson-Lucy deconvolution

We demonstrate *cubic*'s advanced deconvolution capabilities through Richardson-Lucy deconvolution with optimal iteration selection guided by image quality metrics, where both deconvolution and metric calculation happen on the same device.

Dataset description. The experiment uses a single 3D image stack of Hoechst-stained astrocyte nuclei acquired with a Yokogawa CQ1 confocal microscope [26]. Theoretical 3D point spread functions for each individual image volume were modeled using the Richards and Wolf algorithm [43] from the PSFGenerator plugin [27] for Fiji. Both the image and the PSF have the same ZYX size of $30 \times 2160 \times 2560$.

Richardson-Lucy algorithm. The Richardson-Lucy algorithm is an iterative deconvolution method that maximizes likelihood under Poisson noise assumptions. Each iteration involves forward convolution with the PSF, ratio computation with the observed image, back-convolution with the flipped PSF, and multiplicative update of the estimate. The algorithm's effectiveness depends critically on selecting the optimal number of iterations to balance noise suppression and detail preservation.

Traditional Richardson-Lucy implementations require manual iteration count selection, often leading to under- or over-deconvolution. We implemented automatic stopping criteria that monitor image quality improvement between consecutive iterations. The threshold indicates when further iterations provide diminishing returns, enabling automated optimal stopping without user intervention.

***cubic* implementation.** Our GPU-accelerated implementation leverages *cubic*'s FFT operations and element-wise arithmetic to achieve substantial speedup over CPU-based approaches. Key optimizations include:

- GPU-resident FFT operations for convolution steps
- Memory-efficient in-place operations to minimize data transfer
- Vectorized FRC computation using GPU-accelerated cross-correlation
- Automated convergence monitoring with configurable thresholds

Evaluation metrics. We assess deconvolution quality using multiple metrics: (1) peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) against ground truth, (2) single-image Fourier Ring Correlation-

derived resolution [28].

Results. Figure 3 demonstrates results of metric-guided iteration selection, with different metrics capturing various image quality changes during the deconvolution process.

Performance benchmarks (Figure 3) reveal GPU acceleration of 50× depending on the metric, with the FRC taking too long to measure on CPU.

This example showcases *cubic*'s capability to implement sophisticated, research-grade algorithms with both computational efficiency and methodological rigor, making advanced deconvolution accessible for high-throughput microscopy applications.

5. Discussion

In this study we introduced *cubic*, a lightweight Python library that unites standard SciPy and scikit-image routines and their GPU-accelerated alternatives in CuPy and cuCIM. Specficially, *cubic* retains the exact function signatures and patterns familiar to users of scikit-image and scipy.ndimage; switching a pipeline to GPU execution simply requires replacing imports and a single array transfer call. This lowers the barrier to entry for high-throughput, reproducible bioimage analysis and ensures that researchers can scale their 2D and 3D workflows across heterogeneous computing environments without sacrificing readability or maintainability. Through example benchmarks on representative 3D microscopy pipelines—covering 3D smoothing, multi-step segmentation and feature extraction, and deconvolution—we demonstrated speed-ups of 10–1500× over CPU-only workflows in each major processing stage.

Acknowledgements

This work was pertially supported by Chinese Key-Area Research and Development Program of Guangdong Province (2020B0101350001). This work was partially supported by the Human Frontier Science Program (RGY0081/2019 to S.S.) and a grant from the National Institutes of Health NIGMS (R35 GM122547 to A.E.C.). Xin Rong of the University of Michigan donated NVIDIA TITAN X GPU used for this research, and the NVIDIA Corporation donated the TITAN Xp GPU used for this research.

References

[1] What's next for bioimage analysis? *Nature Methods*, 20: 945–946, 2023. 1

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283. USENIX Association, 2016. 2, 3

[3] Analytic and Translational Genetics Unit. Microscopy computational tools. https://github.com/atgu/microscopy_computational_tools, 2024. 2

[4] Neda Bagheri, Anne E Carpenter, Emma Lundberg, Anne L Plant, and Rick Horwitz. The new era of quantitative cell imaging—challenges and opportunities. *Molecular Cell*, 82 (2):241–247, 2022. 1

[5] Harikrushnan Balasubramanian, Chad M Hobson, Teng-Leong Chew, and Jesse S Aaron. Imagining the future of optical microscopy: everything, everywhere, all at once. *Communications Biology*, 6(1):1096, 2023. 1

[6] Phil Bankhead, Michael B. Loughrey, Daniel Fernández, Yena Dombrowski, Daniel G. McArt, Peter D. Dunne, Sonia McQuaid, Ron Gray, Liam J. Murray, Helen G. Coleman, John A. James, Manuel Salto-Tellez, and Paul W. Hamilton. Qupath: Open source software for digital pathology image analysis. *Scientific Reports*, 7:16878, 2017. 2

[7] Niccolò Banterle, Khanh Huy Bui, Edward A Lemke, and Martin Beck. Fourier ring correlation as a resolution criterion for super-resolution microscopy. *Journal of structural biology*, 183(3):363–367, 2013. 4

[8] Stuart Berg, Dominik Kutra, Thorben Kroeger, Christoph N. Straehle, Bernhard X. Kausler, Chris Haubold, Marco Schiegg, Markus Ales, Thomas Beier, Babak Rudy, Martin Weigert, Vishwanathan Rajan, Urs Schmidt, Martin Weigert, Eugene W. Myers, Martin Kopf, Fred A. Hamprecht, and Anna Kreshuk. ilastik: interactive machine learning for (bio)image analysis. *Nature Methods*, 16(12):1226–1232, 2019. 2

[9] David SC Biggs and Mark Andrews. Acceleration of iterative image restoration algorithms. *Applied Optics*, 36(8): 1766–1775, 1997. 3

[10] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 1, 2

[11] Juan C Caicedo, Shantanu Singh, and Anne E Carpenter. Applications in image-based profiling of perturbations. *Current Opinion in Biotechnology*, 39:134–142, 2016. 1

[12] Srinivas Niranj Chandrasekaran, Hugo Ceulemans, Justin D Boyd, and Anne E Carpenter. Image-based profiling for drug discovery: due for a machine-learning upgrade? *Nature Reviews Drug Discovery*, 20(2):145–159, 2021. 1

[13] Chi-Li Chiu, Nathan Clack, and the napari community. Napari: a Python multi-dimensional image viewer platform for the research community. *Microscopy and Microanalysis*, 28 (S1):1576–1577, 2022. 2

[14] clEsperanto contributors. napari-pyclesperanto-assistant. https://github.com/clEsperanto/napari_pyclesperanto_assistant, 2020. 3

[15] Gabriel Comolet, Neeloy Bose, Jeff Winchell, Alyssa Duren-Lubanski, Tom Rusielewicz, Jordan Goldberg, Grayson

Horn, Daniel Paull, and Bianca Migliori. A highly efficient, scalable pipeline for fixed feature extraction from large-scale high-content imaging screens. *iScience*, 27(12), 2024. 2

[16] napari contributors. napari: a multi-dimensional image viewer for Python. https://doi.org/10.5281/zenodo.3555620, 2019. 2

[17] RAPIDSAI contributors. cuCIM: RAPIDS GPU-accelerated image processing library. https://github.com/rapidsai/cucim, 2025. Version 25.06.0. 2, 3, 4

[18] Eric Czech, Bulent Arman Aksoy, Pinar Aksoy, and Jeff Hammerbacher. Cytokit: a single-cell analysis toolkit for high dimensional fluorescent microscopy imaging. *BMC bioinformatics*, 20(1):448, 2019. 3

[19] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vis.*, 88(2):303–338, 2010. 4

[20] Robert Haase. napari-accelerated-pixel-and-object-classification. https://github.com/haesleinhuepf/napari-accelerated-pixel-and-object-classification, 2021. 3

[21] Robert Haase. napari-cupy-image-processing. https://github.com/haesleinhuepf/napari-cupy-image-processing, 2021. 3

[22] Robert Haase, Loic A. Royer, Peter Steinbach, Deborah Schmidt, Alexandr Dibrov, Uwe Schmidt, Martin Weigert, Nicola Maghelli, Pavel Tomancak, Florian Jug, and Eugene W. Myers. CLIJ: GPU-accelerated image processing for everyone. *Nature Methods*, 17(1):5–6, 2020. 3

[23] Robert Haase, Elnaz Fazeli, David Legland, Michael Doube, Siân Culley, Ilya Belevich, Eija Jokitalo, Martin Schorb, Anna Klemm, and Christian Tischer. A hitchhiker's guide through the bio-image analysis software universe. *FEBS Letters*, 596(19):2472–2485, 2022. 2

[24] Robert Haase, Sébastien Strïgaud, et al. pyclesperanto: GPU-accelerated image processing library. https://pypi.org/project/pyclesperanto/, 2025. Version 0.17.1. 3

[25] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with NumPy. *Nature*, 585(7825): 357–362, 2020. 2

[26] Alexandr A Kalinin, Paula Llanos, Theresa Maria Sommer, Giovanni Sestini, Xinhai Hou, Jonathan Z Sexton, Xiang Wan, Ivo D Dinov, Brian D Athey, Anne E Rivron, Nicolas Carpenter, Beth Cimini, Shantanu Singh, and Matthew J O'Meara. Foreground-aware virtual staining for accurate 3d cell morphological profiling. In *ICML 2025 Generative AI and Biology (GenBio) Workshop*, Vancouver, Canada, 2025. 7

[27] Hagai Kirshner, Franois Aguet, Daniel Sage, and Michael Unser. 3-D PSF fitting for fluorescence microscopy: implementation and localization application. *Journal of Microscopy*, 249(1):13–25, 2013. 7

[28] Sami Koho, Giorgio Tortarolo, Marco Castello, Takahiro Deguchi, Alberto Diaspro, and Giuseppe Vicidomini. Fourier ring correlation simplifies image restoration in fluorescence microscopy. *Nature Communications*, 10(1):3103, 2019. 8

[29] Talley Lambert. pycudadecon. https://github.com/tlambert03/pycudadecon, 2019. 3

[30] Vebjorn Ljosa, Katherine L Sokolnicki, and Anne E Carpenter. Annotated high-throughput microscopy image sets for validation. *Nature Methods*, 9(7):637, 2012. 6

[31] Leon B Lucy. An iterative technique for the rectification of observed distributions. *The Astronomical Journal*, 79:745, 1974. 4

[32] Ilya Lukonin, Marietta Zinner, and Prisca Liberali. Organoids in image-based phenotypic chemical screens. *Experimental & Molecular Medicine*, 53(10):1495–1502, 2021. 1

[33] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. https://github.com/pytorch/vision, 2016. 3

[34] Alán F. Muñoz, Tim Treis, Alexandr A. Kalinin, Shatavisha Dasgupta, Fabian Theis, Anne E. Carpenter, and Shantanu Singh. cp_measure: API-first feature extraction for image-based profiling workflows. In *Workshop on Championing Open-Source Development in Machine Learning, 42nd International Conference on Machine Learning*, Vancouver, Canada, 2025. 2

[35] Robert PJ Nieuwenhuizen, Keith A Lidke, Mark Bates, Daniela Leyton Puig, David Grünwald, Sjoerd Stallinga, and Bernd Rieger. Measuring image resolution in optical nanoscopy. *Nature Methods*, 10(6):557–562, 2013. 4

[36] Brian Northan. A collection of useful python utilities from true north intelligent algorithms, 2025. https://github.com/True-North-Intelligent-Algorithms/tnia-python. 4

[37] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. CuPy: A NumPy-compatible library for NVIDIA GPU calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017. 2, 3, 4

[38] Einar Olafsson. SpaCr: Spatial phenotype analysis of CRISPR-Cas9 screens. https://github.com/EinarOlafsson/spacr, 2025. 2

[39] Giovanni Palla, Hannah Spitzer, Michal Klein, David Fischer, Anna Christina Schaar, Louis Benedikt Kuemmerle, Sergei Rybakov, Ignacio L Ibarra, Olle Holmberg, Isaac Virshup, et al. Squidpy: A scalable framework for spatial omics analysis. *Nature Methods*, 19(2):171–178, 2022. 2

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 8024–8035. Curran Associates Inc., 2019. 2, 3

[41] Tobias M. Rasse, Réka Hollandi, and Peter Horváth. Opsef: Open source python framework for collaborative instance segmentation of bioimages. *Frontiers in Bioengineering and Biotechnology*, 8:558880, 2020. 2

[42] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3674–3683, 2020. 3

[43] Bernard Richards and Emil Wolf. Electromagnetic diffraction in optical systems, ii. structure of the image field in an aplanatic system. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 253 (1274):358–379, 1959. 7

[44] William Hadley Richardson. Bayesian-based iterative method of image restoration. *JoSA*, 62(1):55–59, 1972. 4

[45] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel J. White, Volker Hartenstein, Kevin Eliceiri, Pavel Tomancak, and Albert Cardona. Fiji: an open-source platform for biological-image analysis. *Nature Methods*, 9(7):676–682, 2012. 1, 2

[46] Caroline A Schneider, Wayne S Rasband, and Kevin W Eliceiri. NIH Image to ImageJ: 25 years of image analysis. *Nature Methods*, 9(7):671–675, 2012. 1, 2

[47] David R. Stirling, Anne E. Carpenter, and Beth A. Cimini. CellProfiler 4: improvements in speed, utility and usability. *BMC Bioinformatics*, 22:433, 2021. 1, 2, 6

[48] Changqing Su, Yuhan Gao, You Zhou, Yaoqi Sun, Chenggang Yan, Haibing Yin, and Bo Xiong. AutoDeconJ: a GPU-accelerated ImageJ plugin for 3D light-field deconvolution with optimal iteration numbers predicting. *Bioinformatics*, 2022. 3

[49] Stéfan van der Walt, Johannes L. Schönberger, and Juan et al. Nunez-Iglesias. scikit-image: Image processing in Python. *PeerJ*, 2:e453, 2014. 1, 2, 4, 5

[50] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020. 2, 4

[51] Daniela Vorkel and Robert Haase. GPU-accelerating imagej macro image processing workflows using CLIJ. *arXiv preprint*, 2008.11799, 2020. 3

[52] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4): 600–612, 2004. 4

[53] Gregory P Way, Ted Natoli, Adeniyi Adeboye, Lev Litichevskiy, Andrew Yang, Xiaodong Lu, Juan C Caicedo, Beth A Cimini, Kyle Karhohs, David J Logan, et al. Morphology and gene expression profiling provide complementary information for mapping cell state. *Cell Systems*, 13(11): 911–923, 2022. 1

[54] Martin Weigert, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, Deborah Schmidt, Coleman Broaddus, Siân Culley, et al. Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nature Methods*, 15(12):1090–1097, 2018. 4

[55] Guillaume Witz. napari-skimage. https://github.com/guiwitz/napari-skimage, 2024. 2