MedImages.jl: A comprehensive Julia library for standardized 3D and 4D medical imaging data handling

Jakub Mitura[a,*], Divyansh Goyal[a]

[a] *JuliaHealth*

Abstract

MedImages.jl is a Julia library designed to standardize the handling of 3D and 4D medical imaging data, such as CT, MRI, and PET scans. By providing a unified data structure inspired by the Brain Imaging Data Structure (BIDS), MedImages.jl addresses the complexities associated with diverse medical imaging formats like DICOM and NIfTI. The library facilitates the loading, saving, and manipulation of medical images while rigorously preserving critical spatial metadata (origin, spacing, and orientation). It leverages Julia's high-performance capabilities to offer efficient spatial transformations and integrates with the broader ecosystem, including Python-based tools via PyCall. MedImages.jl aims to democratize access to advanced medical image processing by simplifying the interaction with complex metadata and fostering reproducible research.

Keywords: Medical Imaging, Julia, DICOM, NIfTI, Spatial Metadata, Open Source

*Corresponding author

Email addresses: `jakub.mitura@gmail.com` (Jakub Mitura), `divital2004@gmail.com` (Divyansh Goyal)

Code metadata

Nr.	Code metadata description	Metadata
C1	Current code version	2.0.1
C2	Permanent link to code/repository used for this code version	`https://github.com/JuliaHealth/MedImages.jl`
C3	Permanent link to reproducible capsule	`https://juliahealth.org/MedImages.jl/`
C4	Legal Code License	Apache License 2.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Julia
C7	Compilation requirements, operating environments & dependencies	Julia 1.10.3+, Linux/macOS/Windows
C8	If available Link to developer documentation/manual	`https://juliahealth.org/MedImages.jl/`
C9	Support email for questions	jakub.mitura@gmail.com

Table 1: Code metadata

1. Motivation and significance

1.1. How Julia helps in scientific computing

Julia is a high-level, dynamic programming language specifically designed to bridge the gap between easy-to-use, productive languages like Python, R, and MATLAB, and high-performance execution languages like C, C++, and Fortran [1, 2]. It helps researchers by providing a robust environment suitable for a wide range of scientific and technical computing needs [1, 3].

The language's design focuses on overcoming major systemic challenges. A primary goal is solving the "Two-Language Problem," where researchers traditionally prototype in high-productivity languages but must rewrite performance-critical sections in low-level languages [4, 5, 6]. Julia combines the ease of a productivity language with the speed of a performance language in a single environment, eliminating this need [6, 5]. Additionally, Julia solves the "Expression Problem" through multiple dispatch, allowing developers to easily add new functions to existing data types or integrate new custom data types with existing algorithms without modifying original code [1, 4, 7].

Julia acts like a universal adapter in scientific computing, streamlining the workflow from initial idea to high-performance production code. Its ecosystem includes specialized packages for scientific machine learning (SciML), biological sciences (BioJulia), and medical imaging [1, 7].

1.2. Potential for Julia in medical image analysis

Julia presents significant potential for transforming the analysis of medical images by combining high performance with developer productivity. This is demonstrated through unique, domain-specific libraries. For instance, in MRI reconstruction, packages like MRIReco.jl and KomaMRI.jl achieve speeds comparable to established, highly optimized C/C++ libraries [5, 8]. GIRFReco.jl provides an end-to-end, self-contained solution for spiral MRI reconstruction [9], and BlochSimulators.jl offers highly optimized simulations [8].

The language's high performance enables new scientific methods, such as rapid parameter estimation in quantitative MRI, which has shown drastic reductions in computational cost compared to MATLAB implementations (e.g., 90x speed boost for whole brain fitting), making advanced parameters accessible in clinical settings [10, 11]. Furthermore, Julia's support for automatic differentiation (AD) facilitates the rapid implementation and optimization of complex mathematical models [12].

1.3. Difficulties related to medical imaging formats

Handling medical imaging data involves navigating the complexities of formats like DICOM and the critical nature of spatial metadata. The DICOM standard is ubiquitous but criticized for its elaborate structure and complexity [13]. Accessing spatial metadata is often error-prone due to the highly nested and interdependent structure of annotations within DICOM datasets [13].

A major challenge lies in handling spatial coordinates and reference frames. Medical images often have varying voxel spacing (non-isotropic pixels) and lack a universal coordinate system [14, 15]. The physical relationship between image data and patient anatomy—defined by origin, spacing, and a direction cosine matrix—must be preserved [14]. Ignoring this critical physical spatial information causes simple operations like adding two images together to be invalid [14]. Furthermore, registering images with different domains requires precise resampling and coordinate alignment [16].

2. Software description

MedImages.jl addresses these challenges by providing a standardized, high-performance library for medical image handling in Julia.

2.1. Architecture

The library is organized into modular components, designed to separate concerns between data representation, I/O, and processing (Figure 1).

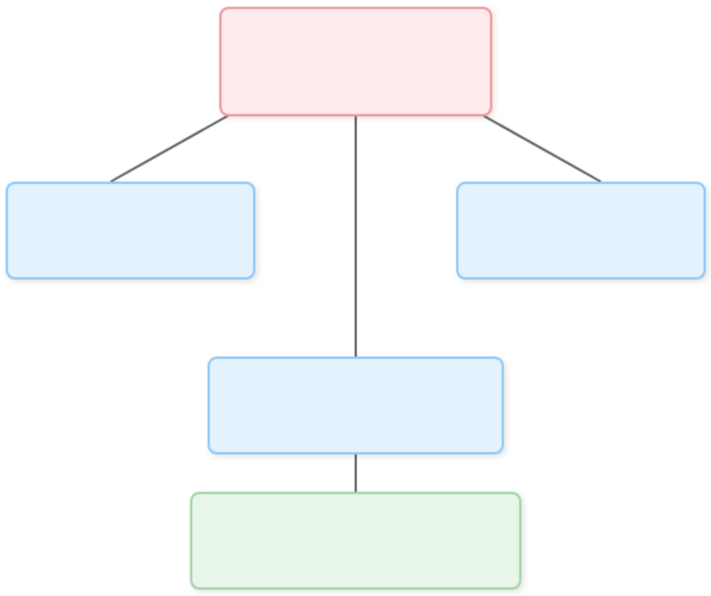

Figure 1: Architecture of MedImages.jl showing the core data structure and interacting modules. This diagram illustrates the relationship between the core data structure (MedImage_data_struct), input/output operations (Load_and_save), transformation modules (Basic_transformations, Spatial_metadata_change), and external ecosystem integration (PyCall/SimpleITK).

- **MedImage_data_struct**: Defines the core `MedImage` structure, which encapsulates voxel data, spatial metadata (origin, spacing, direction), and image classification types. The metadata structure is loosely based on the BIDS format.

- **Load_and_save**: Handles input/output operations for standard formats like NIfTI (via Nifti.jl) and DICOM (via Dicom.jl).
- **Basic_transformations**: Provides functions for image manipulation such as rotation, cropping, and padding.
- **Spatial_metadata_change**: Manages critical spatial operations, including resampling to new spacings and changing orientation (e.g., to RAS).

2.2. Functionality

MedImages.jl simplifies the complexity of medical image formats. It allows users to load images from disparate sources into a unified `MedImage` object. Key functionalities include:

- **Standardized I/O**: seamless loading and saving of 3D and 4D datasets.
- **Spatial Awareness**: The library enforces the physical space tenet, ensuring that operations respect the image's physical location and orientation.
- **Resampling**: Tools to resample images to specific voxel spacings or reference images, handling interpolation automatically.
- **Integration**: Through `PyCall.jl`, it can leverage established tools like SimpleITK, bridging the gap between Julia's emerging ecosystem and existing high-performance libraries.

3. Illustrative Examples

The following example demonstrates how to load a medical image, resample it to a standard isotropic spacing, and save the result. This workflow is common in preprocessing pipelines for machine learning.

```
using MedImages

# Load an image (NIfTI or DICOM)
im_path = "path/to/image.nii.gz"
med_img = load_image(im_path)
```

```
# Inspect current spacing
println("Original Spacing: ", med_img.spacing)

# Resample to isotropic 1x1x1 mm spacing
target_spacing = (1.0, 1.0, 1.0)
resampled_img = resample_to_spacing(med_img, target_spacing)

# Save the processed image
save_med_image(resampled_img, "output_isotropic.nii.gz")
```

4. Impact

4.1. Positive Impact on the Scientific Community

The development and adoption of MedImages.jl are poised to have a substantial positive impact on the scientific community, particularly for researchers in medical imaging and computational biology.

Enhancing Reproducibility. Reproducibility is a cornerstone of scientific progress, yet it remains a challenge in medical imaging due to proprietary formats and opaque processing pipelines. MedImages.jl enhances reproducibility by providing an open-source, standardized framework where every step of the image processing pipeline—from loading to spatial transformation—is transparent and code-based [17]. By relying on explicit spatial metadata management, the library reduces the risk of "silent errors" (such as misaligned coordinate systems) that can invalidate research findings.

Accelerating Discovery via the Two-Language Solution. Scientific innovation is often slowed by the need to translate prototype code (written in Python/MATLAB) into high-performance languages (C++) for deployment. MedImages.jl leverages Julia's ability to solve the "Two-Language Problem," allowing researchers to write high-level, mathematically expressive code that compiles to machine-native performance [5]. This acceleration allows for more rapid iteration of algorithms, faster processing of large datasets (like 4D fMRI or PET series), and the ability to deploy research code directly into production environments without rewriting it [3].

Democratizing Access to Advanced Tools. Historically, robust medical image processing has required deep expertise in C++ (e.g., ITK) or reliance on expensive proprietary software. MedImages.jl democratizes access by wrapping complex functionalities—such as precise spatial resampling and orientation management—into intuitive Julia functions. This lowers the barrier to entry for biologists, clinicians, and data scientists, allowing them to perform sophisticated image analysis without needing to be experts in software engineering [18]. Furthermore, its integration with the Python ecosystem via `PyCall` ensures that users can transition gradually, leveraging existing tools like SimpleITK while adopting Julia's performance benefits [14].

Fostering a Collaborative Ecosystem. By adhering to open standards and open-source licensing (Apache 2.0), MedImages.jl encourages community contribution. It serves as a foundational block upon which more specialized tools (e.g., for tumor segmentation, registration, or radiomics) can be built. This modularity fosters a collaborative ecosystem where improvements in the core library immediately benefit all downstream applications, accelerating the collective pace of innovation in the field [17].

5. Conclusions

MedImages.jl represents a significant step forward for the medical imaging community in Julia. By solving the two-language problem and providing a standardized, spatially-aware framework, it empowers researchers to build high-performance, reproducible analysis pipelines. As the ecosystem grows, MedImages.jl will play a pivotal role in enabling advanced medical image analysis and machine learning workflows.

CRediT authorship contribution statement

Jakub Mitura: Conceptualization, Methodology, Software, Writing - Original Draft, Project administration. **Divyansh Goyal:** Software, Validation, Writing - Review & Editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code and data are available on GitHub: `https://github.com/JuliaHealth/MedImages.jl`.

Acknowledgements

The authors would like to thank the JuliaHealth community for their support and contributions.

References

[1] S. Pal, M. Bhattacharya, S. Dash, S.-S. Lee, C. Chakraborty, A next-generation dynamic programming language julia: Its features and applications in biological science, Journal of Advanced Research 64 (10 2024). `doi:10.1016/j.jare.2023.11.015`.
URL `http://dx.doi.org/10.1016/j.jare.2023.11.015`

[2] J. Belyakova, B. Chung, J. Gelinas, J. Nash, R. Tate, J. Vitek, World age in julia: Optimizing method dispatch in the presence of eval (extended version), arXiv (2020). `doi:10.48550/ARXIV.2010.07516`.
URL `https://arxiv.org/abs/2010.07516`

[3] S. Ahn, S. G. Ross, E. Asma, J. Miao, X. Jin, L. Cheng, S. D. Wollenweber, R. M. Manjeshwar, Quantitative comparison of osem and penalized likelihood image reconstruction using relative difference penalties for clinical pet, Physics in Medicine and Biology 60 (7 2015). `doi:10.1088/0031-9155/60/15/5733`.
URL `http://dx.doi.org/10.1088/0031-9155/60/15/5733`

[4] J. Eschle, T. Gál, M. Giordano, P. Gras, B. Hegner, L. Heinrich, U. H. Acosta, S. Kluth, J. Ling, P. Mato, M. Mikhasenko, A. M. Briceño, J. Pivarski, K. Samaras-Tsakiris, O. Schulz, G. A. Stewart, J. Strube, V. Vassilev, Potential of the julia programming language for high energy physics computing, Computing and Software for Big Science 7 (10 2023). `doi:10.1007/s41781-023-00104-x`.
URL `http://dx.doi.org/10.1007/s41781-023-00104-x`

[5] T. Knopp, M. Grosser, Mrireco.jl: An mri reconstruction framework written in julia, arXiv (2021). `doi:10.48550/ARXIV.2101.12624`.
URL `https://arxiv.org/abs/2101.12624`

[6] J. Bezanson, J. Chen, B. Chung, S. Karpinski, V. B. Shah, J. Vitek, L. Zoubritzky, Julia: dynamism and performance reconciled by design, Proceedings of the ACM on Programming Languages 2 (10 2018). `doi:10.1145/3276490`.
URL `http://dx.doi.org/10.1145/3276490`

[7] E. Roesch, J. G. Greener, A. L. MacLean, H. Nassar, C. Rackauckas, T. E. Holy, M. P. H. Stumpf, Julia for biologists, Nature Methods 20 (4 2023). `doi:10.1038/s41592-023-01832-z`.
URL `http://dx.doi.org/10.1038/s41592-023-01832-z`

[8] O. van der Heide, C. A. T. van den Berg, A. Sbrizzi, Gpu-accelerated bloch simulations and mr-stat reconstructions using the julia programming language, Magnetic Resonance in Medicine 92 (3 2024). `doi:10.1002/mrm.30074`.
URL `http://dx.doi.org/10.1002/mrm.30074`

[9] A. Jaffray, Z. Wu, S. J. Vannesjo, K. Uludağ, L. Kasper, Girfreco.jl: An open-source pipeline for spiral magnetic resonance image (mri) reconstruction in julia, Journal of Open Source Software 9 (5 2024). `doi:10.21105/joss.05877`.
URL `http://dx.doi.org/10.21105/joss.05877`

[10] N. J. Sisco, P. Wang, A. Stokes, R. Dortch, Rapid parameter estimation for selective inversion recovery myelin imaging using an open-source julia toolkit (9 2021). `doi:10.1101/2021.09.27.461996`.
URL `http://dx.doi.org/10.1101/2021.09.27.461996`

[11] N. J. Sisco, P. Wang, A. M. Stokes, R. D. Dortch, Rapid parameter estimation for selective inversion recovery myelin imaging using an open-source julia toolkit, PeerJ 10 (3 2022). `doi:10.7717/peerj.13043`.
URL `http://dx.doi.org/10.7717/peerj.13043`

[12] D. Hofmann, A. G. Chesebro, C. Rackauckas, L. R. Mujica-Parodi, K. J. Friston, A. Edelman, H. H. Strey, Increasing spectral dcm flexibility and speed by leveraging julia's modelingtoolkit and automated differentiation (11 2023). `doi:10.1101/2023.10.27.564407`.
URL `http://dx.doi.org/10.1101/2023.10.27.564407`

[13] C. P. Bridge, C. Gorman, S. Pieper, S. W. Doyle, J. K. Lennerz, J. Kalpathy-Cramer, D. A. Clunie, A. Y. Fedorov, M. D. Herrmann, Highdicom: a python library for standardized encoding of image annotations and machine learning model outputs in pathology and radiology, Journal of Digital Imaging 35 (8 2022). doi:10.1007/s10278-022-00683-y.
URL http://dx.doi.org/10.1007/s10278-022-00683-y

[14] Z. Yaniv, B. C. Lowekamp, H. J. Johnson, R. Beare, Simpleitk image-analysis notebooks: a collaborative environment for education and reproducible research, Journal of Digital Imaging 31 (11 2017). doi:10.1007/s10278-017-0037-8.
URL http://dx.doi.org/10.1007/s10278-017-0037-8

[15] Advances in spatial and temporal databases, Lecture Notes in Computer Science (2011). doi:10.1007/978-3-642-22922-0.
URL http://dx.doi.org/10.1007/978-3-642-22922-0

[16] R. Sandkühler, C. Jud, S. Andermatt, P. C. Cattin, Airlab: Autograd image registration laboratory, arXiv (2018). doi:10.48550/ARXIV.1806.09907.
URL https://arxiv.org/abs/1806.09907

[17] J. Schindelin, C. T. Rueden, M. C. Hiner, K. W. Eliceiri, The imagej ecosystem: An open platform for biomedical image analysis, Molecular Reproduction and Development 82 (7 2015). doi:10.1002/mrd.22489.
URL http://dx.doi.org/10.1002/mrd.22489

[18] B. C. Lowekamp, D. T. Chen, L. Ibáñez, D. Blezek, The design of simpleitk, Frontiers in Neuroinformatics 7 (2013). doi:10.3389/fninf.2013.00045.
URL http://dx.doi.org/10.3389/fninf.2013.00045