https://doi.org/10.1038/s41592-023-02129-x

JDLL: a library to run deep learning models on Java bioimage informatics platforms

Check for updates

The advancements in artificial intelligence (AI) technology over the past decade have been a breakthrough in imaging for life sciences, paving the way for novel methods in image restoration[1], reconstruction[2] and segmentation[3]. However, the wide adoption of deep learning (DL) techniques by end users in bioimage analysis is hindered by the complexity of their deployment. These techniques stem from a variety of rapidly evolving frameworks (for example, TensorFlow 1 or 2, PyTorch) that come with distinct and often conflicting setups, which can discourage even proficient developers. This has led to integration difficulties or even absence in mainstream bioimage informatics platforms such as ImageJ, Icy and Fiji, many of which are primarily developed in Java.

We present JDLL (Java Deep Learning Library), a Java library that provides a comprehensive toolkit and application programming interface (API) for crafting advanced scientific applications and image analysis pipelines with DL capabilities. JDLL streamlines the installation, maintenance and execution of DL models across any major DL frameworks. JDLL is based on two main components (Fig. 1). The first, the DL-engine-installer, acts as a framework manager. It facilitates the download and integration of leading DL frameworks such as Tensorflow 1 and 2, PyTorch (from v1.4) and ONNX (from v1.3). It is capable of functioning with both CPU and GPU across a vast array of supported versions. Notably, DL-engine-installer automatically identifies the required engine and version from the model specifications, handling downloads or loads seamlessly for the end user. The second component, DL-model-runner, offers a Java API for performing inference on models built on the aforementioned DL frameworks. It can work around memory (CPU or GPU) limitations when processing large images, thanks to its tiling feature. Furthermore, it offers a user-friendly API for model loading or downloading, presenting comprehensible details about its training process and the underlying DL framework.

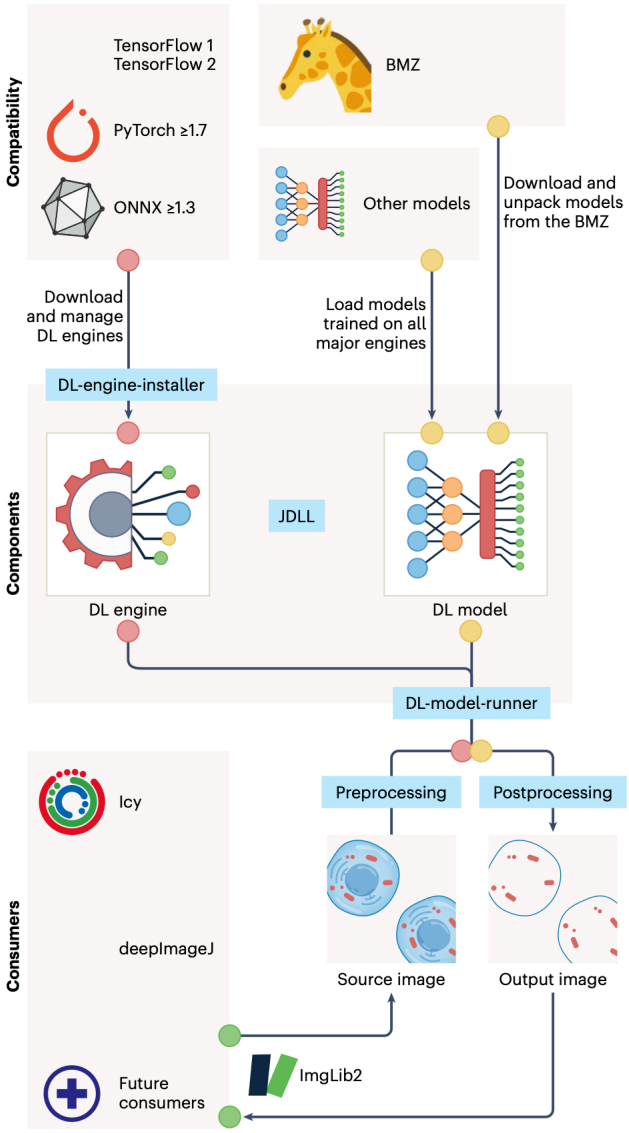

Fig. 1 | The JDLL architecture. JDLL is a Java API that can manage and load models created with a wide range of DL frameworks or engines (top). It can also download and deploy the models shared on the BMZ repository following BMZ conventions. These models are unpacked with JDLL, along with the engines required to run inference with the models, in a manner transparent to API consumers. JDLL provides to these consumers (such as Icy and deepImageJ, bottom) a simple and unified API to run inference on images via a custom ImgLib2 wrapper for their specific image data models. Because the API is generic and relies on an ImgLib2 component, JDLL can be used by any Java software platform, fostering the reproducibility of DL model deployment.

The flexibility and adaptability of these components are pivotal in facilitating the effortless installation and utilization of the full spectrum of DL models prevalent in the life sciences. JDLL is streamlined, carrying minimal dependencies – namely, ImgLib2 (ref. 4) and specific libraries for processing JSON and YAML files, commonly used for DL model specifications. It is built to be compatible with existing and potential future DL frameworks, leveraging ImgLib2 to create a framework-agnostic tensor implementation. Consequently, applications built on JDLL will not require engine-specific code. JDLL offers a universal, singular implementation compatible with all engines across their various versions. Its cutting-edge management of DL engines empowers it to operate multiple models within a single program, even if they were trained on normally incompatible DL engines. For example, JDLL can effortlessly sequence a style-transfer model operating on Tensorflow 2 with an instance segmentation model running on Tensorflow 1 within a single script.

The Bioimage Model Zoo (BMZ)[5] represents a collaborative endeavor aimed at streamlining the utilization and sharing of DL models in life sciences, facilitating their download or retrieval and running inference easily. It provides specifications for the models and the engines they require to run, along with pre- and postprocessing steps. Several community partners in BMZ, such as Icy, ImageJ/Fiji (via deepImageJ), ZeroCostDL4Mic and ImJoy[6–9], already target end-user applications. JDLL aspires to serve as the common core component for DL across the Java-based platforms in this list, introducing distinctive features to support this integration. The BMZ specifications detail both pre- and postprocessing steps, and for true reproducibility, it is imperative that these steps yield consistent outputs across all platforms. Whereas the existing Java platforms rely on their own custom implementations, these processing steps are now addressed generically thanks to the universal tensor capabilities provided by JDLL and ImgLib2. As a result, platforms incorporating JDLL can depend on a shared code base for all processes tied to running a DL model. This approach minimizes redundant coding efforts and ensures pixel-by-pixel reproducibility.

JDLL serves as a foundational library designed to support a growing community of developers in creating scientific tools for end users (such as bioimage analysts or biologists). Furthermore, its efficient and streamlined API reduces the complexities of executing a DL model to just a handful of code lines. This makes it compatible for use in analysis scripts (such as those in MATLAB, Fiji or Icy) or through connectors that integrate it with visual programming languages (we can foresee KNIME and Icy Protocols). This positions JDLL as an asset for the biological community. Thanks to its generality and versatility, JDLL offers the means to imbue scientific Java image analysis platforms with DL capabilities. Given the popularity of these platforms within the biological research community, JDLL stands poised to bolster the widespread adoption of DL in life sciences.

Code availability

The source code, documentation, tutorials and examples implementing JDLL can be found at https://github.com/bioimage-io/JDLL. JDLL is made available under the open-source Apache software license.

Carlos García López de Haro[1], Stéphane Dallongeville[1,2], Thomas Musset[1,2], Estibaliz Gómez-de-Mariscal[3], Daniel Sage[4], Wei Ouyang[5], Arrate Muñoz-Barrutia[6], Jean-Yves Tinevez[7] ✉ & Jean-Christophe Olivo-Marin[1,2] ✉

[1]Bioimage Analysis Unit, Institut Pasteur, Université Paris Cité, Paris, France. [2]CNRS UMR 3691, Institut Pasteur, Paris, France. [3]Instituto Gulbenkian de Ciência, Lisbon, Portugal. [4]Biomedical Imaging Group and Center for Imaging, Ecole Polytechnique de Lausanne (EPFL), Lausanne, Switzerland. [5]Science for Life Laboratory, KTH Royal Institute of Technology, Stockholm, Sweden. [6]Biomedical Sciences and Engineering Laboratory, Universidad Carlos III de Madrid, Leganés, Spain. [7]Image Analysis Hub, Institut Pasteur, Université Paris Cité, Paris, France. ✉e-mail: jean-yves.tinevez@pasteur.fr; jean-christophe.olivo-marin@pasteur.fr

Published online: 8 January 2024

References

1. Weigert, M. et al. *Nat. Methods* **15**, 1090–1097 (2018).
2. Belthangady, C. & Royer, L. A. *Nat. Methods* **16**, 1215–1225 (2019).
3. Moen, E. et al. *Nat. Methods* **16**, 1233–1246 (2019).
4. Pietzsch, T., Preibisch, S., Tomancak, P. & Saalfeld, S. *Bioinformatics* **28**, 3009–3011 (2012).
5. Ouyang, W. et al. Preprint at *bioRxiv* https://doi.org/10.1101/2022.06.07.495102 (2022).
6. de Chaumont, F. et al. *Nat. Methods* **9**, 690–696 (2012).
7. Gómez-de-Mariscal, E. et al. *Nat. Methods* **18**, 1192–1195 (2021).
8. von Chamier, L. et al. *Nat. Commun.* **12**, 2276 (2021).
9. Ouyang, W., Mueller, F., Hjelmare, M., Lundberg, E. & Zimmer, C. *Nat. Methods* **16**, 1199–1200 (2019).

Acknowledgements

This work has been partially supported by the Agence Nationale de la Recherche through the LabEx IBEID (ANR-10-LABX-62-IBEID), the Institut Carnot Pasteur Microbes & Santé (ANR 16 CARN 0023-01), the programs PIA INCEPTION (ANR-16-CONV-0005) and France-BioImaging (ANR-10-INBS-04); by DIM ELICIT Région Ile-de-France; by the European Commission through the H20 20-FET-OPEN-2018–2019-2020-01 grant no. 862840 ("FREE@POC") (J.-C.O.-M.); by additional internal funding from the Bioimage Analysis unit and the Institut Pasteur (J.-C.O.-M. and J.-Y.T.); by the Ministerio de Ciencia, Innovación y Universidades, Agencia Estatal de Investigación, under grant PID2019-109820RB-I00, MCIN / AEI / 10.13039/501100011033/, co-financed by European Regional Development Fund (ERDF), "A way of making Europe," and the European Commission through the Horizon Europe program (AI4LIFE project, grant agreement 101057970-AI4LIFE) (A.M.-B.); and by Fundação Calouste Gulbenkian and EMBO Postdoctoral Fellowship (EMBO ALTF 174-2022) (E.G.M.). Funded by the European Union. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Author contributions

Code concept and design were done by C.G.L.d.H., J.-Y.T., S.D. and J.-C.O.-M., with contributions from E.G.d.M., D.S., W.O. and A.M.-B. JDLL coding development and implementation was done by C.G.L.d.H., J.-Y.T., T.M. and S.D.; manuscript organizing and writing by C.G.L.d.H., J.-Y.T., A.M.-B. and J.-C.O.-M. with all authors contributing comments and revisions; funding and project administration by J.-Y.T. and J.-C.O.-M.

Competing interests

The authors declare no competing interests.

Additional information

Peer review information *Nature Methods* thanks David Barry and Edward Evans, III for their contribution to the peer review of this work.