BRIEF COMMUNICATION
https://doi.org/10.1038/s41592-021-01262-9

nature methods

DeepImageJ: A user-friendly environment to run deep learning models in ImageJ

Estibaliz Gómez-de-Mariscal[1,5], Carlos García-López-de-Haro[1,5], Wei Ouyang[2], Laurène Donati[3], Emma Lundberg[2], Michael Unser[4], Arrate Muñoz-Barrutia[1] ✉ and Daniel Sage[4] ✉

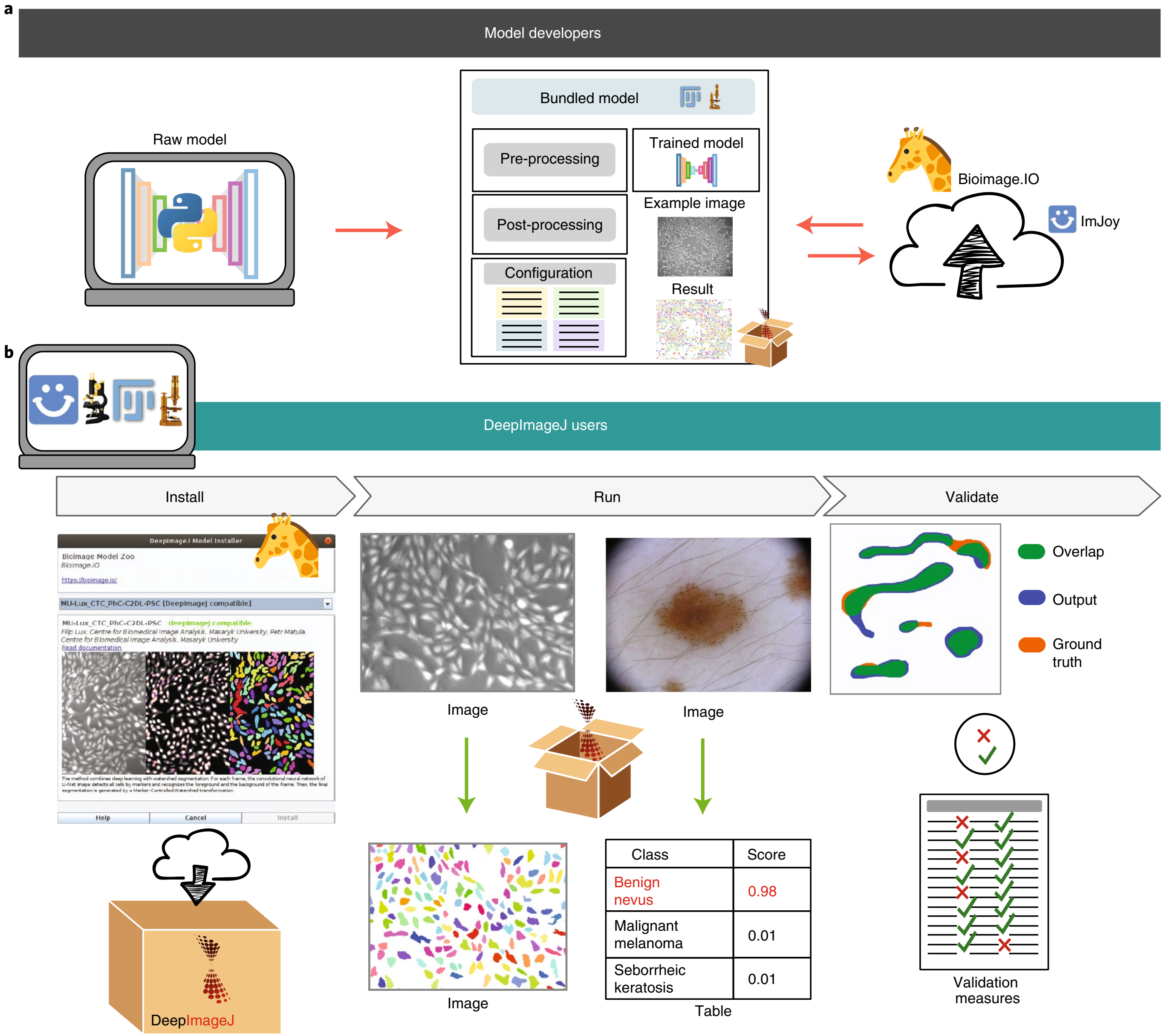
DeepImageJ is a user-friendly solution that enables the generic use of pre-trained deep learning models for biomedical image analysis in ImageJ. The deepImageJ environment gives access to the largest bioimage repository of pre-trained deep learning models (BioImage Model Zoo). Hence, nonexperts can easily perform common image processing tasks in life-science research with deep learning-based tools including pixel and object classification, instance segmentation, denoising or virtual staining. DeepImageJ is compatible with existing state of the art solutions and it is equipped with utility tools for developers to include new models. Very recently, several training frameworks have adopted the deepImageJ format to deploy their work in one of the most used softwares in the field (ImageJ). Beyond its direct use, we expect deepImageJ to contribute to the broader dissemination and reuse of deep learning models in life sciences applications and bioimage informatics.

Deep learning (DL) models have a profound impact on a wide range of imaging applications, including life sciences[1,2]. Unfortunately, their accessibility is often riddled with technical challenges for the nonexpert user. As most DL methods are available as source code, running them requires setting up a sophisticated software and hardware environment. The increasing use of image analysis workflows in biomedical research[1] and the willingness to disseminate trained DL models have pushed computer scientists to design more user-friendly solutions[3–5]. Currently, there exists an increasing number of active developer teams addressing this problem with different solutions: the CSBDeep team[6], the Ozcan Research Group, DeepClass4Bio[7], Ilastik[8], ImJoy[9], ZeroCostDL4Mic[10], YAPIC[11] and DeepTrack[12]. The CSBDeep team distributes their DL workflows via an ImageJ[3,13] toolbox[6], which lets nonexpert users perform a variety of microscopy image analysis using trained DL models. Through their plugin, it is possible to train denoising models in a local machine without any previous programming skills. The StarDist plugin[14] makes the most powerful tool for cell nuclei detection and segmentation in microscopy images accessible in ImageJ. Similarly, the Ozcan Research team has often made its trained models available in ImageJ[15]. DeepClass4Bio is an API to use image classification tasks in ImageJ using trained DL models. The Ilastik team has an early release of a neural network classification workflow equipped with both inference and training functionalities. ImJoy[9] is particularly suited for building and sharing interactive web interfaces for DL-based image analysis. ZeroCostDL4Mic[10] utilizes the free cloud GPU resources provided by Google Colaboratory and provides extensive documentation in a browser-based notebook interface, allowing nonexperts to train DL models such as a generic segmentation model (for example, the well-known U-Net[16]) or the super-resolution microscopy model (for example, DeepSTORM[17]). YAPIC is a Python library to train a U-Net on pixel classification and make predictions by writing few plain command lines. DeepTrack combines browser user interfaces to train different models in a noncoding fashion, together with a set of Python notebooks that support the easy training and use of their models.

The previously mentioned tools have started to boost the use of DL solutions for biomedical image analysis tasks. However, a user-friendly tool to disseminate trained models for image processing in a noncoding fashion and with a unified format is still missing[1,18]. We present deepImageJ, an open-source environment for ImageJ, which is the de facto standard image processing software in life sciences[3]. The open-source package ImageJ gives biologists access to a wide variety of user-friendly image analysis tools through third-party plugins and macros (Fig. 1). It contains most of the standard bioimage analysis methods and is continuously updated with the most recent techniques. The current integration of deepImageJ further contributes to the ImageJ ecosystem. Since the first release of deepImageJ, the framework has been widely used by developers to share their work with collaborators in the life sciences domain or to provide an easy way to test a DL solution for their biological imaging application (Fig. 1). DeepImageJ runs a variety of third-party models from the main DL libraries that are powering the DL framework (TensorFlow and PyTorch). Installing deepImageJ is straightforward compared to that of common Python environments thanks to the one-click installation facilitated by the ImageJ updates manager. DeepImageJ is designed as a standard ImageJ plugin with the technicalities hidden behind the user-friendly interface. It runs locally without the need of uploading data, thus providing better privacy protection.

DeepImageJ operates with several types of models for image processing tasks such as image-to-vector (for example, image classification), image-to-image (for example, image segmentation, deconvolution, virtual labeling, super-resolution) or pyramidal feature pooling networks (for example, region proposal networks for object detection, panoptic segmentation) (Fig. 2). Succinctly, the deepImageJ format is as general as possible so that the use of different models does not rely on any technical configuration or work

[1]Bioengineering and Aerospace Engineering Department, Universidad Carlos III de Madrid and Instituto de Investigación Sanitaria Gregorio Marañón, Madrid, Spain. [2]Science for Life Laboratory, School of Engineering Sciences in Chemistry, Biotechnology and Health, KTH – Royal Institute of Technology, Stockholm, Sweden. [3]EPFL Center for Imaging, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. [4]Biomedical Imaging Group and EPFL Center for Imaging, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. [5]These authors contributed equally: E. Gómez-de-Mariscal, C. García-López-de-Haro. ✉e-mail: mamunozb@ing.uc3m.es; daniel.sage@epfl.ch



DeepImageJ Model Installer

Bioimage Model Zoo
BioImage.IO

https://bioimage.io/

MU-Lux_CTC_PhC-C2DL-PSC [DeepImageJ compatible]

MU-Lux_CTC_PhC-C2DL-PSC deepimagej compatible
Filip Lux, Centre for Biomedical Image Analysis, Masaryk University, Petr Matula, Centre for Biomedical Image Analysis, Masaryk University
Read documentation

The method combines deep learning with watershed segmentation. For each frame, the convolutional neural network of U-Net shape detects all cells by markers and recognizes the foreground and the background of the frame. Then, the final segmentation is generated by a Marker-Controled Watershed transformation.

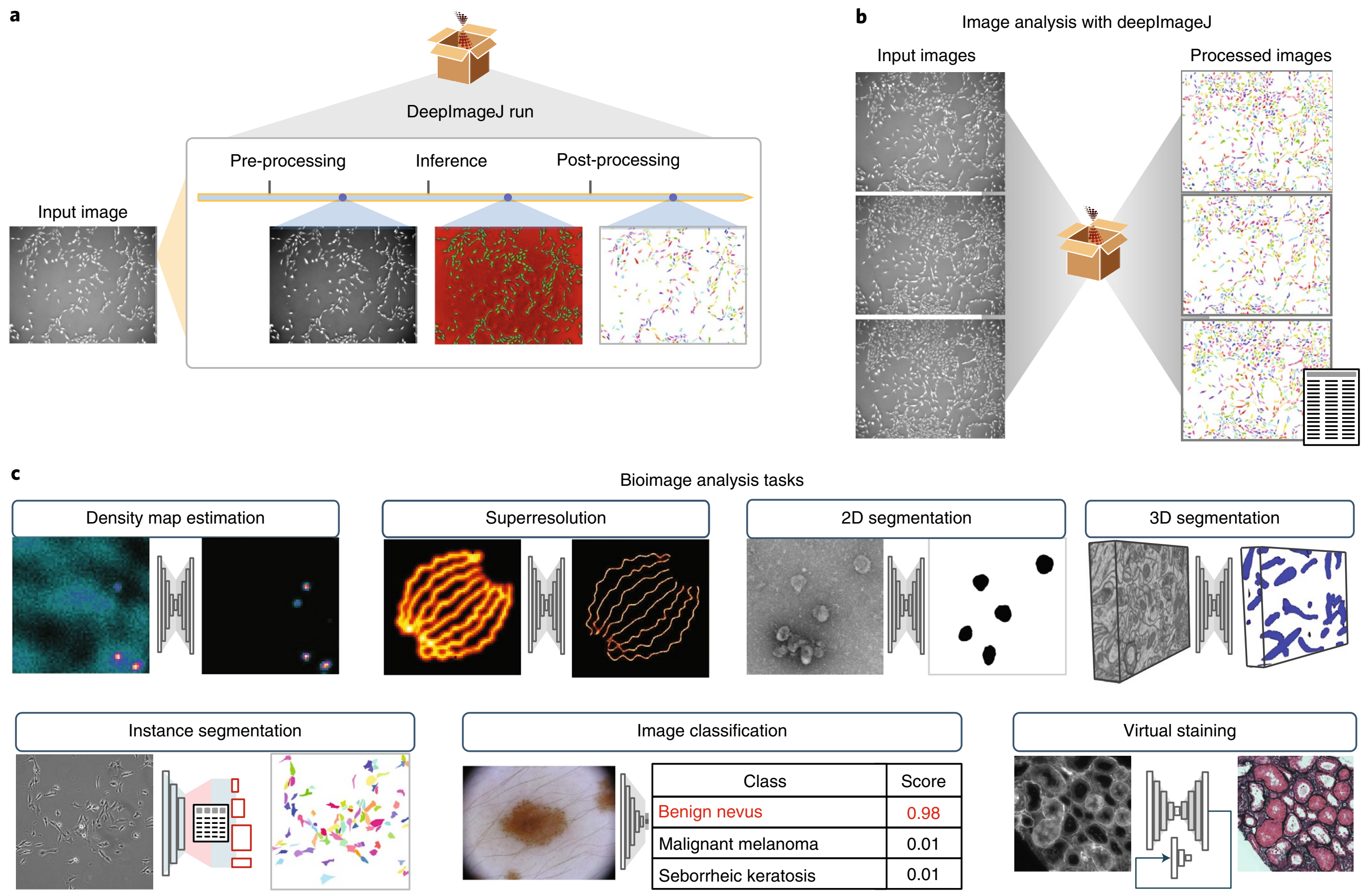
Help | Cancel | Install

Class	Score
Benign nevus	0.98
Malignant melanoma	0.01
Seborrheic keratosis	0.01

Fig. 1 | DeepImageJ environment and scope. DeepImageJ targets model developers and nonexpert biomedical image analysts. **a**, Developers train a DL model usually in Python and bundle it using the DeepImageJ_Build_Bundled Model plugin in ImageJ. The bundled model follows the BioImage Model Zoo format: it contains the trained model architecture and weights, pre and post-processing routines, testimonial images for reproducibility purposes and a specifications file. The latter gathers all the technicalities that allow cross-compatibility among the BioImage Model Zoo consumer software, deepImageJ among others. The bundled model package can be disseminated through the public model repository in the cloud synchronized with the BioImage Model Zoo, or directly sending it to the final user through a private communication channel. **b**, Life scientists can run a deepImageJ model online using ImJoy, or install it locally. The model can be used as any regular ImageJ plugin to analyze images. The deepImageJ model package allows the automatic processing of any input image and is compatible with outputs of different formats and dimensions such as images or tables. The DeepImageJ_Validate plugin provides a set of evaluation measures to compare the resulting image with a ground-truth image provided by the user.

(that is, utilizing one model or another, makes no difference to the user). The latter goes in hand with the recent BioImage Model Zoo (https://bioimage.io/) initiative, which is meant to unify the effort of the biomedical image analysis community to make accessible, open and usable the trained DL model through different consumers. As part of the initiative, we are building an open-source repository to deliver trained DL models, notebooks and datasets in a standard manner, resulting in a combined effort for the democratization of DL in the biomedical image analysis field. The deepImageJ format follows the specifications of the BioImage Model Zoo, allowing interoperability between the community partners software and seamless access to the model repository.

The main plugin of the deepImageJ toolbox, DeepImageJ_Run, allows the execution of DL models in one click. It ensures the consumption of pre-processing, inference and post-processing as detailed by the model developer (Fig. 2). The deepImageJ toolbox is complemented by three companion plugins for model bundling, model installation and validation of the results (Methods).



Class	Score
Benign nevus	0.98
Malignant melanoma	0.01
Seborrheic keratosis	0.01

Fig. 2 | Functionalities of deepImageJ. a, DeepImageJ_Run plugin processes an input image with a locally installed deepImageJ compatible trained model. It automatically carries out the pre-processing, inference and post-processing steps required and written previously by a developer. **b**, The DeepImageJ_Run plugin can be called from ImageJ macros, so locally stored image datasets can be automatically processed. Furthermore, the use of deepImageJ trained models can be integrated into extended bioimage analysis pipelines. **c**, DeepImageJ targets those DL models that have an image as input and it is compatible with outputs of different formats and dimensions. Therefore, it is suitable for tasks such as density estimation (https://github.com/LEB-EPFL/DEFCoN-ImageJ), super-resolution[17], two-dimensional (2D) and three-dimensional (3D) segmentation[16,19], instance segmentation[20], image classification (https://www.isic-archive.com/) and virtual staining[15].

To facilitate deepImageJ model testing, we ported DeepImageJ_Run plugin to Javascript such that it can run in a web browser via ImageJ.JS (https://ij.imjoy.io). This enables the integration with ImJoy, which allows testing models in the BioImage Model Zoo website without downloading any model or installing the plugin locally. This is especially helpful for users to compare and select models based on their data, making the dissemination of DL models more effective.

When running DL models, it is crucial to pre-process the input image so the model is fed with an image that has the same precise features as the ones employed in the training process (for example, data normalization pre-processing). To handle this, deepImageJ gives the user the flexibility to run pre and post-processing routines written in an ImageJ scripting language: ImageJ macro or Java plugins. The latter is the main bridge between com (Fig. 2).

Although we made every effort to lay solid foundations to use the deepImageJ toolbox, the correctness of a DL model's output ultimately depends on its appropriate usage. Hence, it is critical that the user pays close attention to the information given by the DL developers before running a model, and that all the results obtained are thoroughly inspected. For this, users can take advantage of DeepImageJ_Validate to assess the accuracy of the results whenever ground-truth data are available. Unlike in other computer vision niches, most DL models for microscopy image processing still lack the ability to generalize across datasets[18]. While we are confident about the future developments to get general and data cross-compatible models, it is currently recommended to fine tune the DL models. Such (re)training and evaluation of a model is only possible when there is ground-truth data, proper infrastructure (software and hardware) and, most often, knowledge about machine learning. Because model training and fine tuning are out of the scope of deepImageJ, we would like to highlight already existing user-friendly tools such as Ilastik, ImJoy, ZeroCostDL4Mic or YAPIC, which can perform such tasks. The efforts to ensure cross-compatibility with the above tools to provide combined solutions are ongoing. For example, well-established DL models for biomedical image processing such as the U-Net[16], DeepSTORM[17] and StarDist[14] can be trained and automatically exported to deepImageJ using ZeroCostDL4Mic. The YAPIC team has integrated a new feature in the software to export any of their models into the deepImageJ format. We encourage all users with a potential need for fine tuning their models to integrate these combinations in their pipelines. Moreover, such combinations are becoming popular in bioimage analysis as they exploit the computational capacity of Python easily to (re)train DL models and the flexibility given in tools such as ImageJ to concatenate different image processing steps.

The use of trained DL models without a deeper understanding of the method could potentially become the source of unreliable results. Aware of this, we remark the very recent effort made by the community to define and recommend good praxis in bioimage analysis, and more specifically to train life scientists on both the promises and risks of using this new technique. We trust in the potential of deepImageJ as a user-friendly tool to bridge the current gap between computer vision and biomedical image analysis. Moreover, the deepImageJ environment is highly suitable for educational activities due to its easy installation and model execution, and the flexibility achieved thanks to access to all the ImageJ resources.

The experience gathered through the development of deepImageJ has seeded the scientific collaborations among different actors involved in the bioimage analysis field and set a roadmap for future developments, with key aspects being interoperability and smart guidance towards automated machine learning (AutoML)[1]. Because DL models consist of numerous layers and operations, using them to process images requires considerable memory consumption. DeepImageJ integrates a smart feature that allows adjusting the patch size when possible to smaller image size and to complete the tiling strategy automatically without further changes in the model specifications. Nonetheless, further improvements can still be made to design an intelligent memory management strategy for running the DL models, such as adjusting the input patch shape automatically according to the user's resources. DeepimageJ, for example, provides relevant information for the user to know how much memory the model execution will consume and, therefore, to determine if the model can be used in a certain machine. However, it will not set up the model execution configuration according to the memory capacity. A straightforward example is the smart configuration of the tiling strategy considering the capacity of the user's machine. Yet another challenge is the development of personalized guides based on the user's data to choose the best-suited model and proper fine tuning strategies. So, it is that AutoML could turn into the new hot topic in bioimage analysis.

Notwithstanding the current tendency to build general models for image processing tasks[18], there is a pressing need to release user-friendly and model-adjustable environments to run DL models[1,4,18]. The latter goes in hand with a higher-level target of reducing human interaction in the routinely performed bioimage analysis tasks. That said, deepImageJ is a contribution to the field for the accessibility of DL in bioimage analysis and the dissemination of DL models in a standardized manner. The deepImageJ environment facilitates the work exchange between developers and end users, which the ongoing integration of deepImageJ in the BioImage Model Zoo seeks to reinforce. Therefore, we expect deepImageJ to enlighten the work of life sciences researchers and to become yet another standard open-source tool for the dissemination of DL image processing models.

Online content

Any methods, additional references, Nature Research reporting summaries, source data, extended data, supplementary information, acknowledgements, peer review information; details of author contributions and competing interests; and statements of data and code availability are available at https://doi.org/10.1038/s41592-021-01262-9.

Received: 28 October 2019; Accepted: 5 August 2021;
Published online: 30 September 2021

References

1. Meijering, E. A bird's-eye view of deep learning in bioimage analysis. *Computational Struct. Biotechnol. J.* **18**, 2312 (2020).
2. Moen, E. et al. Deep learning for cellular image analysis. *Nat. Methods* **16**, 1233–1246 (2019).
3. Schroeder, A. B. et al. The ImageJ ecosystem: Open-source software for image visualization, processing, and analysis. *Protein Sci.* **30**, 234–249 (2020).
4. Deep learning gets scope time. *Nat. Methods* **16**, 1195 (2019).
5. Lucas, A. M. et al. Open-source deep-learning software for bioimage segmentation. *Mol. Biol. Cell* **32**, 823–829 (2021).
6. Weigert, M. et al. Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nat. Methods* **15**, 1090–1097 (2018).
7. Inés, A., Domínguez, C., Heras, J., Mata, E. & Pascual, V. DeepClas4Bio: Connecting bioimaging tools with deep learning frameworks for image classification. *Computers Biol. Med.* **108**, 49–56 (2019).
8. Berg, S. et al. Ilastik: interactive machine learning for (bio)image analysis. *Nat. Methods* **16**, 1226–1232 (2019).
9. Ouyang, W., Mueller, F., Hjelmare, M., Lundberg, E. & Zimmer, C. ImJoy: an open-source computational platform for the deep learning era. *Nat. Methods* **16**, 1199–1200 (2019).
10. von Chamier, L. et al. Democratising deep learning for microscopy with ZeroCostDL4Mic. *Nat. Commun.* **12**, 2276 (2021).
11. Fäßler, F. et al. Cryo-electron tomography workflows for quantitative analysis of actin networks involved in cell migration. *Microsc. Microanalysis* **26**, 2518–2519 (2020).
12. Midtvedt, B. et al. Quantitative digital microscopy with deep learning. *Appl. Phys. Rev.* **8**, 011310 (2021).
13. Schneider, C. A., Rasband, W. S. & Eliceiri, K. W. NIH Image to ImageJ: 25 years of image analysis. *Nat. Methods* **9**, 671–675 (2012).
14. Schmidt, U., Weigert, M., Broaddus, C. & Myers, G. Cell detection with star-convex polygons. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018 – 21st International Conference, Granada, Spain, September 16–20, 2018, Proceedings, Part II* 265–273 (Springer, 2018).
15. Rivenson, Y. et al. Virtual histological staining of unlabelled tissue-autofluorescence images via deep learning. *Nat. Biomed. Eng.* **3**, 466–477 (2019).
16. Falk, T. et al. U-Net: deep learning for cell counting, detection, and morphometry. *Nat. Methods* **16**, 67–70 (2019).
17. Nehme, E., Weiss, L. E., Michaeli, T. & Shechtman, Y. Deep-STORM: super-resolution single-molecule microscopy by deep learning. *Optica* **5**, 458–464 (2018).
18. Caicedo, J. C. et al. Nucleus segmentation across imaging experiments: the 2018 data science bowl. *Nat. Methods* **16**, 1247–1253 (2019).
19. Gómez-de-Mariscal, E. et al. Deep-learning-based segmentation of small extracellular vesicles in transmission electron microscopy images. *Sci. Rep.* **9**, 13211 (2019).
20. Tsai, H.-F., Gajda, J., F.W. Sloan, T., Rares, A. & Shen, A. Q. Usiigaci: Instance-aware cell tracking in stain-free phase contrast microscopy enabled by machine learning. *SoftwareX* **9**, 230–237 (2019).

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Methods

In the following lines technical descriptions and software requirements are provided. The ImageJ user guide for this version of the plugin (deepImageJ v.2.1.0) is given in the Supplementary Material. As deepImageJ is an ongoing project, we strongly recommend users to check the documentation at deepImageJ's website (https://deepimagej.github.io/deepimagej/) and the plugin's Wiki in GitHub (https://github.com/deepimagej/deepimagej-plugin/wiki).

DeepImageJ plugins. The main plugin of the deepImageJ toolbox, DeepImageJ_Run, allows the execution of DL models for an image processing task in a few clicks without DL expertise or programming skills. During this process, users have access to a description and the complete documentation of the trained model. If a GPU is locally available and set up following the guidelines in the website of deepImageJ, the plugin will automatically connect with it. Image-to-image processing tasks such as segmentation, denoising, deconvolution or super-resolution need a tiling strategy when using DL models (Methods). The plugin guides the user through this step by recommending a specific tile size whenever possible. In addition, the plugin ensures the consumption of pre-processing, inference and post-processing routines as detailed by the model developer (see Fig. 2). While the process is automatically executed, the plugin informs the user about each performed step. This provides flexibility to users who want to test different model configurations and understand more in depth the entire process. Moreover, DeepImageJ_Run can be called from the popular scripting ImageJ macro, which allows the use of the models in larger image processing workflows. This feature facilitates a variety of setups such as the automatic processing of a locally stored set of raw microscopy images[21] (Fig. 2).

The deepImageJ toolbox is complemented by three companion plugins for model bundling, model installation and validation of the results:

- DeepImageJ_Build_BundledModel guides model developers to bundle their trained model and provide all the necessary information (meta-data) for its easy use in ImageJ (Methods). The plugin creates the package required to upload the model to the online repository (Fig. 2). Alternatively, model developers can use the pydeepimagej (https://github.com/deepimagej/pydeepimagej) library to create the bundled models directly from the source code in Python.
- DeepImageJ_Install_Model allows the BioImage Model Zoo repository to be accessed to install a model stored in the cloud (Fig. 2).
- DeepImageJ_Validate tool gives access to a variety of perceptual and segmentation validation measures that guide the user through an evaluation of the obtained result (Fig. 2).

Software/network compatibility. DeepImageJ is compatible with both Fiji[22], ImageJ[13] and ImageJ2 (ref. [23]). It is self-sufficient on any operating system: MacOS, Linux and Windows and on 64-bit operating systems (32-bit operating systems are not supported). It supports TensorFlow models until version 1.15, and PyTorch 1.6. Keras version 2 or lower is also supported as long as the models are compatible with Tensorflow version 1.15 or lower. The same as CSBDeep[6], deepImageJ uses a TensorFlow Java API manager to ensure TensorFlow version compatibiltiy. The latter can be upgraded in ImageJ2 through the ImageJ-TensorFlow manager (https://github.com/imagej/imagej-tensorflow), developed by Curtis Rueden and Deborah Schmidt. The Deep Java Library (https://djl.ai/) ensures the compatibility with PyTorch. Nonetheless, those users with a Windows operating system need to install Visual Studio (https://visualstudio.microsoft.com/) for the deployment of this library in ImageJ/Fiji.

The Java libraries used to load TensorFlow and PyTorch models point to the same source code as the respective Python packages. This implies that regardless the code (Python or Java), the same results and execution times are ensured.

DeepImageJ bundled models. DeepImageJ_Run processes folders (models) that contain one of the following:

- saved_model.pb and variables: TensorFlow model in Protocol Buffer format (saved bundled model).
- pytorch_model.pt: PyTorch model stored in TorchScript format.
- and all the following files:

- file (.ijm/.jar/.class): ImageJ macros or Java code written by the model's author for the pre and post-processing. Ready to use ImageJ macros can be found at https://github.com/deepimagej/imagej-macros.
- exampleImage.tiff: Example of input of the image.
- result(.tif/.csv): Output of the model after the post-processing.
- model.yaml: Bioimage Model Zoo configuration specifications, containing details about the related publication and technical characteristics of the model.

All previous files are created by the author of the model, which makes the bundled model self-sufficient. Their content is described in the next paragraph. Further details about loading bundled models in deepImageJ are given in the Supplementary Material.

Any DL model is determined by a graph (the architecture of the network) and its weights (specific values for all the parameters in the network obtained after training). The TensorFlow's Java API is only compatible with the SavedModel format, which is obtained using an in-house Python routine (https://github.com/deepimagej/python4deepimagej/). Namely, the deepImageJ models are defined by a protocol buffer format file (called saved_model.pb) that contains the architecture of the model and a series of text files storing the weights that are kept in a folder called variables. PyTorch models should be exported in TorchScript format (pytorch_model.pt), which contains both the architecture and the weights compressed in a single file. ImageJ macros (.ijm) or compiled Java code (.jar/.class) are optional pre and post-processing steps. The pre-processing routine transforms the image into a specific input type for which the model was trained. Typical pre-processing operations are normalization of the pixel intensity values, change of the bit depth and image resizing. The post-processing routine curates the output of the network. Optional post-processing operators are, for example, thresholding, resizing or extraction of objects features. At least two files, an input image (exampleImage.tiff) and an output result (.tif/.csv), are also stored in each of the bundled model folders to facilitate model testing.

The configuration file (model.yaml) has descriptive information about the model and it is synchronized with the Bioimage Model Zoo configuration specifications (https://github.com/bioimage-io/configuration):

- General information: Name of the author(s), title, description, reference to the publication or GitHub repository, license and framework.
- Technical characteristics of the model: Input and output specifications (dimensions shape, size and axis order), pre and post-processing information and model format.
- DeepImageJ specific information: Example image name and pixel size, TensorFlow signature name, pre and post-processing file names, minimum amount of memory required to process the example input image, estimated execution time on the PC.

Input and output size calculations: tiling strategy. The model developer needs to specify the following information when uploading their convolutional neural network (CNN) model:

- Q, I: Whether the input size of the model (Q) is predetermined or not. If it is predetermined, Q needs to be provided, and it will be compared with the size of the image to process (I). Q corresponds to the field called shape in the model.yaml.
- m,s: If the network has an auto-encoder architecture, the size of each dimension of the input image has to be a multiple of a minimum size s defined as $s=p^d$ where d is the number of poolings (down-sampling operations) and p their size. In addition, the input should have a minimum size m so the shape of the tensor that will enter the model satisfies the equation

$$Q = m + ns, \quad n \in \mathbb{N}^* \tag{1}$$

m and s correspond to the fields called min and step, respectively, in the model.yaml.

- P: To preserve the input size at the output, convolutions are usually calculated using zero padding boundary conditions (Supplementary Fig. 1) is an illustration of the 2D case). Namely, additional void pixel values are added along the borders of the image. Hence, the size (per dimension) of the valid domain of the output is given by $R=Q-2P$ with Q, the model input size and P, the size of the network padding, sometimes denoted as halo. The size of the padding is equivalent to the receptive field of one pixel in the CNN. For a symmetrical encoder-decoder architecture, namely, the same number of down and up-samplings. It can be computed as

$$P = 2^p \left(l \left(\frac{k_p - 1}{2} \right) \right) + 2 \sum_{i=0}^{p-1} 2^i \left(l \left(\frac{k_i - 1}{2} \right) \right) \tag{2}$$

where k_i is the kernel size for each convolutional layer, p is the number of poolings and l is the number of convolutional layers at each level of the encoder-decoder. Usually, k_i is an odd number. If the kernel is not square, then P and R have different values on each dimension. P is given in the field halo of the model.yaml.

To handle input images with a large size, deepImageJ follows a common strategy called tiling:

- If the network has not a predetermined input size (Q), the algorithm calculates what is the smallest size t that satisfies equation (1) and is still larger or equal to the size of the input image I and the total padding $2P$:

$$ts = \underset{t=m+ns, n\in\mathbb{N}^*}{\text{argmin}} \{t \geq (I + 2P)\} \tag{3}$$

Then, the image is augmented by mirroring along the borders up to a size t per dimension, and it is processed. Finally, the output is cropped to the initial size I. See Supplementary Fig. 1 for an illustration.
- If the network input size (Q) is predetermined, then the algorithm compares the size of the image (I) with it taking into account the padding (P). If it is

smaller ($I+2P\leq Q$), then the image is augmented by mirroring until the desired size Q is reached. If the opposite is true ($I+2P>Q$), the optimal number of tiles to process (N) is calculated as follows:

$$N = \text{ceil}\left(\frac{I}{Q-2P}\right) \qquad (4)$$

where the function ceil(x) outputs the smallest integer number that is equal to or larger than x. Note that N can vary on each dimension. Then, the image will be covered by patches of size

$$T = \text{floor}\left(\frac{I}{N}\right) \qquad (5)$$

where the function floor(x) outputs the largest integer number that is equal to or smaller than x and $T\leq(Q-2P)$. From each processed patch of size Q, a patch of size T is cropped and placed accordingly to reconstruct a valid output (tiling strategy). The patches along the borders are filled by mirroring as shown in Supplementary Fig. 1. As the quotient in equation (5) may not be an entire number, the last patch on each dimension has exactly size $I-(N-1)T$.

Both the input size of the network (Q) and the padding of the CNN (P) are critical parameters for good results and they are directly related to the time spent by the plugin to process one image. Large input images and deeper networks that have a larger receptive field imply longer computations.

The current deepImageJ version also handles inputs and outputs of different sizes, such as, for example, the case of models for super-resolution. To determine the shape of the output image given a certain input image, we do

$$\tilde{Q} = aQ + 2b. \qquad (6)$$

where a and b are the scale and offset parameters, respectively, given in the model.yaml file, Q the shape of the input image and $\tilde{Q}$ the shape of the output image. Likewise, in the model.yaml file, the reference image for Q needs to be indicated. The latter is given in the field reference_input.

Reporting Summary. Further information on research design is available in the Nature Research Reporting Summary linked to this article.

Data availability

All data that were used to generate the figures in this paper are available at https://deepimagej.github.io/deepimagej/.

Code availability

We used TensorFlow and PyTorch libraries for Python to create the in-house models shown in the figures. The deepImageJ plugin can be used in ImageJ and Fiji. The source code for the plugin together with its releases is provided at https://github.com/deepimagej/deepimagej-plugin. The pydeepimagej Python package is provided at https://github.com/deepimagej/pydeepimagej. The ImageJ macro files can be accessed at https://github.com/deepimagej/imagej-macros. All source code is under a BSD 2-Clause License. The web page https://deepimagej.github.io/deepimagej/provides free access to the ImageJ plugin, along with the bundled models and user guide for image processing.

References

21. Gómez-de-Mariscal, E., Franco, D., Muñoz-Barrutia, A. & Arganda-Carreras, I. in *Bioimage Analysis Components and Workflows* (eds Sladoje, N. & Miura, K.) (Springer, 2021).
22. Schindelin, J. et al. Fiji: an open-source platform for biological-image analysis. *Nat. Methods* **9**, 676–682 (2012).
23. Rueden, C. T. et al. ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics* **18**, 529 (2017).

Acknowledgements

We thank P. M. Gordaliza, I. Arganda-Carreras (tested the beta-versions), D. F. González-Obando (tested the beta-versions), C. Rueden, S. Tosi, T. Pengo (tested the beta-versions), R. Henriques (ZeroCostDL4Mic), R. F. Laine (ZeroCostDL4Mic), G. Jacquemet (ZeroCostDL4Mic), D. Krentzel (ZeroCostDL4Mic) and C. Möhl (YAPIC) for the fruitful discussions and enriching feedback about the deepImageJ project. We would also like to thank NEUBIAS for supporting the project, the NEUBIAS symposium and NEUBIAS Academy@Home and P. Rasti and S. Bollmann for including deepImageJ in their tutorials. We would like specially to mention all the contributors and community partners at the BioImage Model Zoo for the time they have spent to get a cross-compatible model format. We acknowledge the support of Ministerio de Ciencia, Innovación y Universidades, Agencia Estatal de Investigación, under grant nos TEC2016-78052-R and PID2019-109820RB-I00, MINECO/FEDER, UE, co-financed by European Regional Development Fund (ERDF), 'A way of making Europe' (E.G.M., C.G.L.H., A.M.B.), and a 2017 Leonardo Grant for Researchers and Cultural Creators, BBVA Foundation (A.M.B.). This work was also supported by the EPFL Center for Imaging (C.G.L.H., L.D., D.S., M.U.). We would like to thank the Science for Life Laboratory, Erling-Persson Foundation and Knut and Alice Wallenberg foundation (grant no. 2018.0172) (W.O., E.L.). We thank the program 'Short Term Scientific Missions' of NEUBIAS (network of European bioimage analysts) (E.G.M., C.G.L.H.). We also want to acknowledge the support of NVIDIA Corporation with the donation of the Titan X (Pascal) GPU card used for this research (A.M.B.).

Author contributions

E.G.-M. and C.G.-L.-H. contributed to the design of the experimental framework, and reviewed, trained and exported existing image processing methods. C.G.-L.-H. and D.S. developed and implemented the toolbox and worked on the supporting documentation with input from the rest of the authors. C.G.-L.-H. and W.O. built the connection between the toolbox and ImJoy. E.G.-M. wrote the code lines of the supplementary Python notebooks, Python library and ImageJ macros. E.G.M. and W.O. worked on the synchronization with the BioImage Model Zoo. E.G.-M., W.O. and L.D. wrote the manuscript with help from E.L., M.U., A.M.-B. and D.S. E.G.-M., A.M.-B. and D.S. created the website of deepImageJ. M.U., A.M.-B. and D.S. initiated the project. A.M.-B. and D.S. supervised the project. All the authors contributed to the conception of the study, the design of the experimental framework and took part in the literature review. All authors revised the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41592-021-01262-9.

Correspondence and requests for materials should be addressed to Arrate Muñoz-Barrutia or Daniel Sage.

Peer review information Rita Strack was the primary editor on this article and managed its editorial process and peer review in collaboration with the rest of the editorial team.

Reprints and permissions information is available at www.nature.com/reprints.

nature research

Corresponding author(s): Arrate Muñoz Barrutia
Daniel Sage

Last updated by author(s): Jul 26, 2021

Reporting Summary

Nature Research wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Research policies, see our Editorial Policies and the Editorial Policy Checklist.

Statistics

For all statistical analyses, confirm that the following items are present in the figure legend, table legend, main text, or Methods section.

n/a | Confirmed

☒ □ The exact sample size (*n*) for each experimental group/condition, given as a discrete number and unit of measurement

☒ □ A statement on whether measurements were taken from distinct samples or whether the same sample was measured repeatedly

☒ □ The statistical test(s) used AND whether they are one- or two-sided
Only common tests should be described solely by name; describe more complex techniques in the Methods section.

☒ □ A description of all covariates tested

☒ □ A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons

☒ □ A full description of the statistical parameters including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)

☒ □ For null hypothesis testing, the test statistic (e.g. *F*, *t*, *r*) with confidence intervals, effect sizes, degrees of freedom and *P* value noted
Give P values as exact values whenever suitable.

☒ □ For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings

☒ □ For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes

☒ □ Estimates of effect sizes (e.g. Cohen's *d*, Pearson's *r*), indicating how they were calculated

Our web collection on statistics for biologists contains articles on many of the points above.

Software and code

Policy information about availability of computer code

Data collection	No data was collected for this work.
Data analysis	TensorFlow 1.15 and PyTorch 1.16 libraries for Python 3. DeepImageJ 2.1.7 in ImageJ (1.53e) and Fiji (1.53e).

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors and reviewers. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Research guidelines for submitting code & software for further information.

Data

Policy information about availability of data

All manuscripts must include a data availability statement. This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A list of figures that have associated raw data
- A description of any restrictions on data availability

All data that were used to generate the figures in this paper are available at: https://deepimagej.github.io/deepimagej/.

Field-specific reporting

Please select the one below that is the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☒ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/documents/nr-reporting-summary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	This work does not report on experimental work. The software presented in this work loads TensorFlow and PyTorch models into ImageJ/Fiji. No descriptive statistics have been used in this work.
Data exclusions	No data were excluded from the analysis.
Replication	There was no data replication.
Randomization	We did not randomize data.
Blinding	We did not use blinding when analyzing the data.

Reporting for specific materials, systems and methods

We require information from authors about some types of materials, experimental systems and methods used in many studies. Here, indicate whether each material, system or method listed is relevant to your study. If you are not sure if a list item applies to your research, read the appropriate section before selecting a response.

Materials & experimental systems

n/a	Involved in the study
☒	☐ Antibodies
☒	☐ Eukaryotic cell lines
☒	☐ Palaeontology and archaeology
☒	☐ Animals and other organisms
☒	☐ Human research participants
☒	☐ Clinical data
☒	☐ Dual use research of concern

Methods

n/a	Involved in the study
☒	☐ ChIP-seq
☒	☐ Flow cytometry
☒	☐ MRI-based neuroimaging