AnnotatorJ: an ImageJ plugin to ease hand-annotation of cellular compartments

Réka Hollandi[a], Péter Horváth[ab*]

Affiliations

a. Synthetic and Systems Biology Unit, Biological Research Center (BRC), Temesvári körút 62, 6726 Szeged, Hungary.
b. Institute for Molecular Medicine Finland (FIMM), University of Helsinki, Tukholmankatu 8, 00014 Helsinki, Finland.
* Correspondence: horvath.peter@brc.hu

Running head:

AnnotatorJ

Abbreviations

DL:	deep learning
ROI:	region of interest
DL4J:	Deeplearning4j
IoU:	intersection over union

Number of characters: 21.655

Abstract

When single-cell identification and deep learning meet, AnnotatorJ arises. Cellular analysis quality depends on accurate and reliable detection and segmentation of cells so that the subsequent steps of analyses e.g. expression measurements may be carried out precisely and without bias. Deep learning has recently become a popular way of segmenting cells, performing unimaginably better than conventional methods. However, such deep learning applications may be trained on a large amount of annotated data to be able to match the highest expectations. High-quality annotations are unfortunately expensive as they require field experts to create them, and often cannot be shared outside the lab due to medical regulations.

We propose AnnotatorJ, an ImageJ plugin for the semi-automatic annotation of cells (or generally, objects of interest) on (not only) microscopy images that helps find the true contour of individual objects by applying U-Net pre-segmentation. The manual labour of hand-annotating cells can be significantly accelerated by using our tool. Thus, it enables users to create such datasets that could potentially increase the accuracy of state-of-the-art solutions, deep learning or otherwise, when used as training data.

Introduction

Single-cell analysis pipelines begin with an accurate detection of the cells. Even though microscopy analysis software tools aim to become more and more robust to various experimental setups and imaging conditions, most lack efficiency in complex scenarios such as label-free samples or unforeseen imaging conditions (e.g. higher signal-to-noise ratio, novel microscopy or staining techniques), which opens up a new expectation of such software tools: adaptation ability (Hollandi et al., 2019). Another crucial requirement is to maintain ease of usage and limit the number of parameters the users need to fine-tune to match their exact data domain.

Recently, deep learning (DL) methods have proven themselves worthy of consideration in microscopy image analysis tools as they have also been successfully applied in a wider range of applications including but not limited to face detection (Taigman et al., 2014, Sun et al. 2014, Schroff et al. 2015), self-driving cars (Badrinarayanan et al., 2017, Redmon et al., 2016, Grigorescu et al., 2019) and speech recognition (Hinton et al., 2012). Caicedo et al. (Caicedo et al., 2019) and others (Hollandi et al 2019, Moshkov et al 2019) proved that single cell detection and segmentation accuracy can be significantly improved utilizing DL networks. The most popular and widely used deep convolutional neural networks (DCNNs) include Mask R-CNN (He et al., 2017): an object detection and instance segmentation network, YOLO (Redmon et al., 2016): a fast object detector and U-Net (Ronneberger et al., 2015): a semantic segmentation approach specifically intended for bioimage analysis purposes.

As robustly and accurately as they may perform, these networks rely on sufficient data, both in amount and quality, which tends to be the bottleneck of their applicability in certain cases such as single-cell detection. While in more industrial applications (see (Grigorescu et al., 2019)for an overview of autonomous driving) a large amount of training data can be collected relatively easily: see the cityscapes dataset (Cordts et al., 2016) (available at https://www.cityscapes-dataset.com/) of traffic video frames using a car and camera to record and potentially non-expert individuals to label the objects, clinical data is considerably more difficult, due to ethical constraints, and expensive to gather as expert annotation is required. Datasets available in the public domain such as BBBC (Ljosa et al., 2012) at https://data.broadinstitute.org/bbbc/, TNBC (Naylor et al., 2017, 2019) or TCGA (Cancer Genome Atlas Research Network, 2008; Kumar et al., 2017) and detection challenges including ISBI (Coelho et al., 2009), Kaggle (https://www.kaggle.com/), ImageNet (Russakovsky et al., 2015) etc. contribute to the development of genuinely useful DL methods; however, most of them lack heterogeneity of the covered domains and are limited in data size. Even combining them one could not possibly prepare their network/method to generalize well (enough) on unseen domains that vastly differ from the pool they covered. On the contrary, such an adaptation ability can be achieved if the target domain is represented in the training data, as proposed in (Hollandi et al 2019), where an image style transfer method was included in the cell segmentation pipeline that utilizes exactly the missing target domain information by generating synthetic training examples for them.

Eventually, similar DL approaches' performance can only be increased over a certain level if we provide more training examples. The proposed software tool was created for this purpose: the expert can more quickly and easily create a new annotated dataset in their desired domain and feed the examples to DL methods with ease. The user-friendly functions included in the plugin help organize data and support annotation e.g. multiple annotation types, editing, classes etc. Additionally, a batch exporter is provided offering different export formats matching typical DL models'; supported annotation and export types are visualized on **Figure 1**, open source code is available at https://bitbucket.org/biomag/annotatorj.

In AnnotatorJ we initialize annotations with DL pre-segmentation using U-Net that suggests contours from as little as a quickly drawn line over the object (see **Supplementary Material** and **Figure 2**). U-Net estimates a semantic region belonging to the target class with the highest probability within a small bounding box (a rectangle) around the initially drawn contour and returns a fine approximation of the true object boundary; this is referred to as the suggested contour. The user then manually refines the contour to create a pixel-perfect annotation of the object.

Methods

Motivation

We propose AnnotatorJ, an ImageJ (Abramoff et al 2004, Schneider et al 2012) plugin for the annotation and export of cellular compartments that can be used to boost DL models' performance. The plugin is mainly intended for bioimage annotation but could possibly be used to annotate any type of objects on images. During development we kept in mind that the intended user should be able to get comfortable with the software really quickly and fasten the otherwise truly time-consuming and exhausting process of manually annotating single cells or their compartments (such as individual nucleoli, lipid droplets, nucleus or cytoplasm).

The performance of DL segmentation methods is significantly influenced by both the training data size and its quality. Should we feed automatically segmented objects to the network, errors present in the original data will be propagated through the network during training and bias the performance, hence such training data should always be avoided. Hand-annotated and curated data, however, will minimize the initial error boosting the expected performance increase on the target domain to which the annotated data belongs.

Features

AnnotatorJ helps organize the input and output files by automatically creating folders and matching file names to the selected type and class of annotation. Currently, the supported annotation types are 1) instance, 2) semantic and 3) bounding box; see **Figure 1**. Each of these are typical inputs of DL networks; instance annotation provides individual objects separated by their boundaries (useful in case of e.g. clumped cells of cancerous regions) and can be used to provide training data for instance segmentation networks such as Mask R-CNN (He et al., 2017). Semantic annotation means foreground-background separation of the image without distinguishing individual objects (foreground), a typical architecture using such segmentations is U-Net (Ronneberger et al., 2015). And finally, bounding box annotation is done by identifying the object's bounding rectangle, and is generally used in object detection networks (like YOLO (Redmon et al., 2016) or R-CNN (Girshick et al., 2014)).

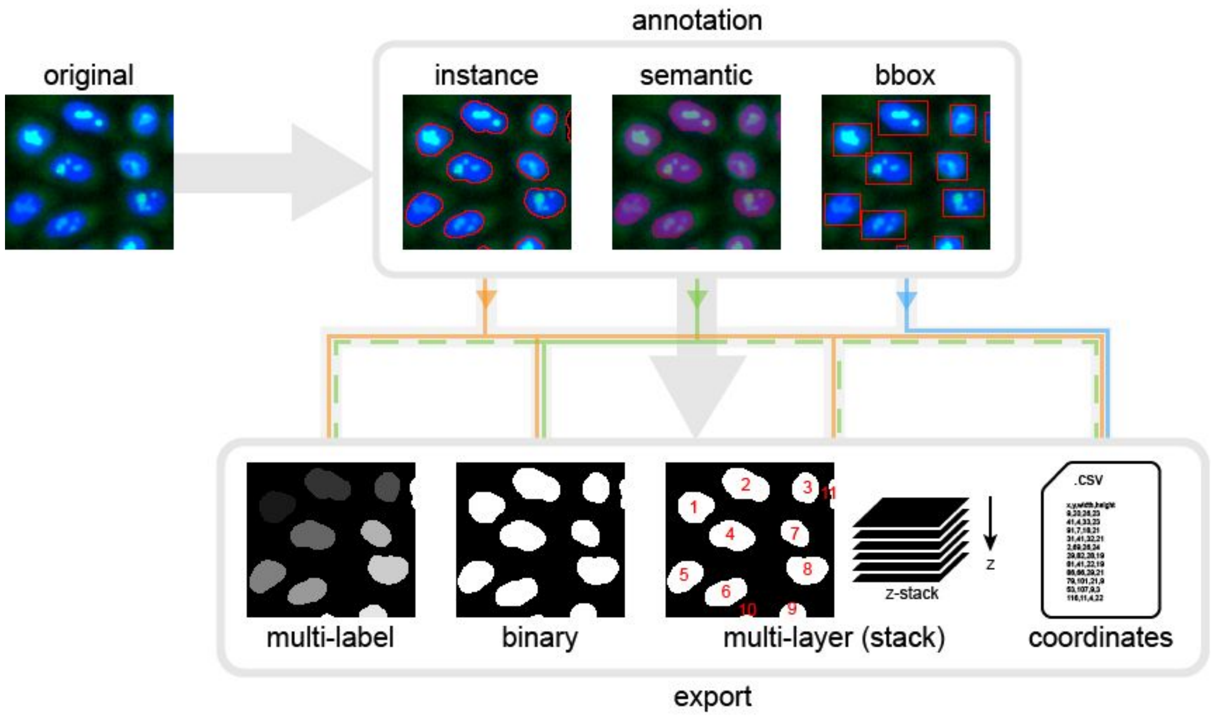


Figure 1. Annotation types. The top row displays our supported types of annotation: instance, semantic and bounding box (noted as 'bbox' on the figure) based on the same objects of interest, in this case nuclei, shown in red. Instances mark the object contours, semantic overlay shows the regions (area) covered, while bounding boxes are the smallest enclosing rectangles around the object borders. Export options are shown in the bottom row: multi-label, binary, multi-layer images and coordinates in a text file. Lines mark the supported export options for each annotation type by colours: orange for instance, green for semantic and blue for bounding box. Dashed lines indicate additional export options for semantic that should be used carefully.

Semantic annotation is done by painting areas on the image overlay. All necessary tools to operate a given function of the plugin are selected automatically. Contour or overlay colours can be selected from the plugin window. For a detailed description and user guide please see the documentation of the tool (available at https://bitbucket.org/biomag/annotatorj repository).

Annotations can be saved to default or user-defined "classes" corresponding to biological phenotypes (e.g. normal or cancerous) or object classes – used as in DL terminology (such as person, chair, bicycle etc.), and later exported in a batch by class. Phenotypic differentiation of objects can be supported by loading a previously annotated class's objects for comparison as overlay to the image and toggling their appearance by a checkbox.

We use the default ImageJ ROI Manager to handle instance annotations as individual objects (ROI(s): region of interest). Annotated objects can be added to the ROI list automatically (without the bound keystroke "t" as defined by ROI Manager) when the user releases the mouse button used to draw the contour by checking its option in the main window of the plugin. This ensures that no annotation drawn is missing from the ROI list.

Contour editing is also possible in our plugin using “Edit mode” (by selecting its checkbox) in which the user can select any already annotated object on the image by clicking on it, then proceed to edit the contour and either apply modifications with the shortcut ‘Ctrl’+’q’ or discard them with ‘escape’. The given object selected for edit is highlighted in inverse contour colour.

In the options (button “…” in the main window) the user can select to use either U-Net or a classical region-growing method to initialize the contour around the object marked. Currently only instance annotation can be assisted.

Contour suggestion using U-Net

Our annotation helper feature “*Contour assist*” (see **Figure 2**) allows the user to work on initialized object boundaries by roughly marking an object’s location on the image which is converted to a well-defined object contour by a U-Net (Ronneberger et al., 2015) model trained on nucleus or other compartment data. We refer this as the *suggested contour* and expect the user to refine the boundaries to match the object border precisely. The suggested contour can be further optimized by applying *active contour* (AC) (Kass et al., 1988) to it. We aim to avoid fully automatic annotation (as previously argued) by only enabling one object suggestion at a time and requiring manual interaction to either refine, accept or reject the suggested contour. These operations are bound to keyboard shortcuts for convenience (see **Fig.2**). When using the *Contour assist* function automatic adding of objects is not available to encourage the user to manually validate and correct the suggested contour as needed.

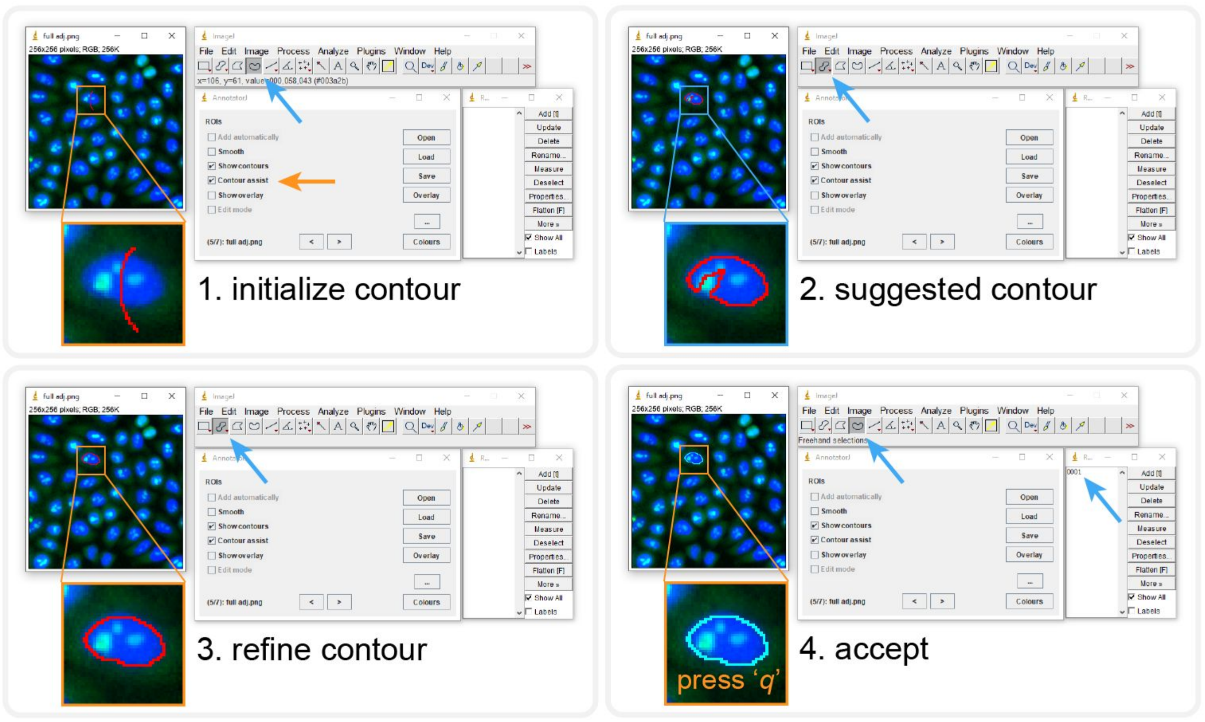


Figure 2. Contour assist mode of AnnotatorJ. The blocks show the order of steps, the given tool needed is automatically selected. User interactions are marked with orange arrows, automatic steps with blue. 1) Initialize the contour with a lazily drawn line, 2) the suggested contour appears (a window is shown until processing completes), brush selection tool is selected automatically, 3) refine the contour as needed, 4) accept it by pressing the key 'q' or reject with 'Ctrl'+'delete'. Accepting adds the ROI to ROI Manager with a numbered label. See also **Supplementary Material** for a demo video (figure2.mov).

On **Figure 2** we demonstrate *Contour assist* using a U-Net model trained on versatile microscopy images of nuclei in Keras and on a fluorescent microscopy image of a cell culture where the target objects, nuclei are labelled with DAPI (in blue). This model is provided at https://bitbucket.org/biomag/annotatorj/downloads/models.zip in the open-source code repository of the plugin.

Contour suggestions can be efficiently used for proper initialization of object annotation, saving valuable time for the expert annotator by suggesting a nearly perfect object contour that only needs refinement (as shown on **Figure 2**). Using a U-Net model accurate enough for the target object class the expert can focus on those image regions where the model is rather uncertain (e.g. around the edges of an object or the separating line between adjacent objects) and fine-tune the contour accurately while sparing considerable effort on more obvious regions (like an isolated object on simple background) by accepting the suggested contour after marginal correction.

The framework of the chosen U-Net implementation, Deeplearning4j (DL4J, available at http://deeplearning4j.org/ or https://github.com/eclipse/deeplearning4j), supports

Keras model import, hence custom, application-specific models can be loaded in the plugin easily by either training them in DL4J (Java) or Python (Keras) and saving the trained weights and model configuration in .h5 and .json files. This vastly extends the possible fields of application for the plugin to general object detection or segmentation tasks.

Exporter

The annotation tool is supplemented by an exporter, AnnotatorJExporter plugin also available in the package. It was optimized for the batch export of annotations created by our annotation tool. For consistency, one class of objects can be exported at a time. We offer 4 export options: 1) multi-labelled, 2) multi-layered and 3) semantic images and 4) coordinates; see **Figure 1**. Instance annotations are typically expected to be exported as multi-labelled (instance-aware) or multi-layered (stack) grayscale images, the latter of which is useful for handling overlapping objects such as cytoplasms in cell culture images. Semantic images are binary foreground-background images of the target objects while coordinates (top-left corner (x,y) of the bounding rectangle appended by its width and height in pixels) can be useful training data for object detection applications including astrocyte localization (Suleymanova et al., 2018) or in a broader aspect, face detection (Taigman et al., 2014). All export options are supported for semantic annotation, however, we note that in instance-aware options (multi-labelled or multi-layered mask and coordinates) only such objects are distinguished whose contours do not touch on the annotation image.

Results

Performance evaluation

We quantitatively evaluated annotation performance and speed in AnnotatorJ (see **Figures 3-4**) with the help of three annotators who had experience in cellular compartment annotation. Both annotation accuracy and time was measured on the same two test sets: a nuclei and a cytoplasm image set (see also **Figure S1** and **Supplementary Material**). Both test sets contained images of various experimental conditions, including fluorescently labelled- and brightfield-stained samples, tissue section and cell culture images. We compared the effectiveness of our plugin using *Contour assist* mode to only allowing the use of *Automatic adding*. Even though the latter is also a functionality of AnnotatorJ, it ensured that the measured annotation times correspond to a single object each. Without this option the user must press the key "t" after every contour drawn to add it to the ROI list which can be unintendedly missed, increasing its time as the same contour must be drawn again.

For the annotation time test presented on **Figure 3** we measured the time passed between adding new objects to the annotated object set in ROI Manager for each object, then averaged the times for each image and each annotator, respectively. Time was measured in the Java implementation of the plugin in milliseconds. **Figures 3-4** show SEM (standard error of the mean) error bars for each mean measurement (see **Supplementary Material** for details).

In the case of annotating cell nuclei, results confirm that hand-annotation tasks can be significantly accelerated using our tool. Each three annotators were faster by using *Contour assist*, two of them nearly double their speed.

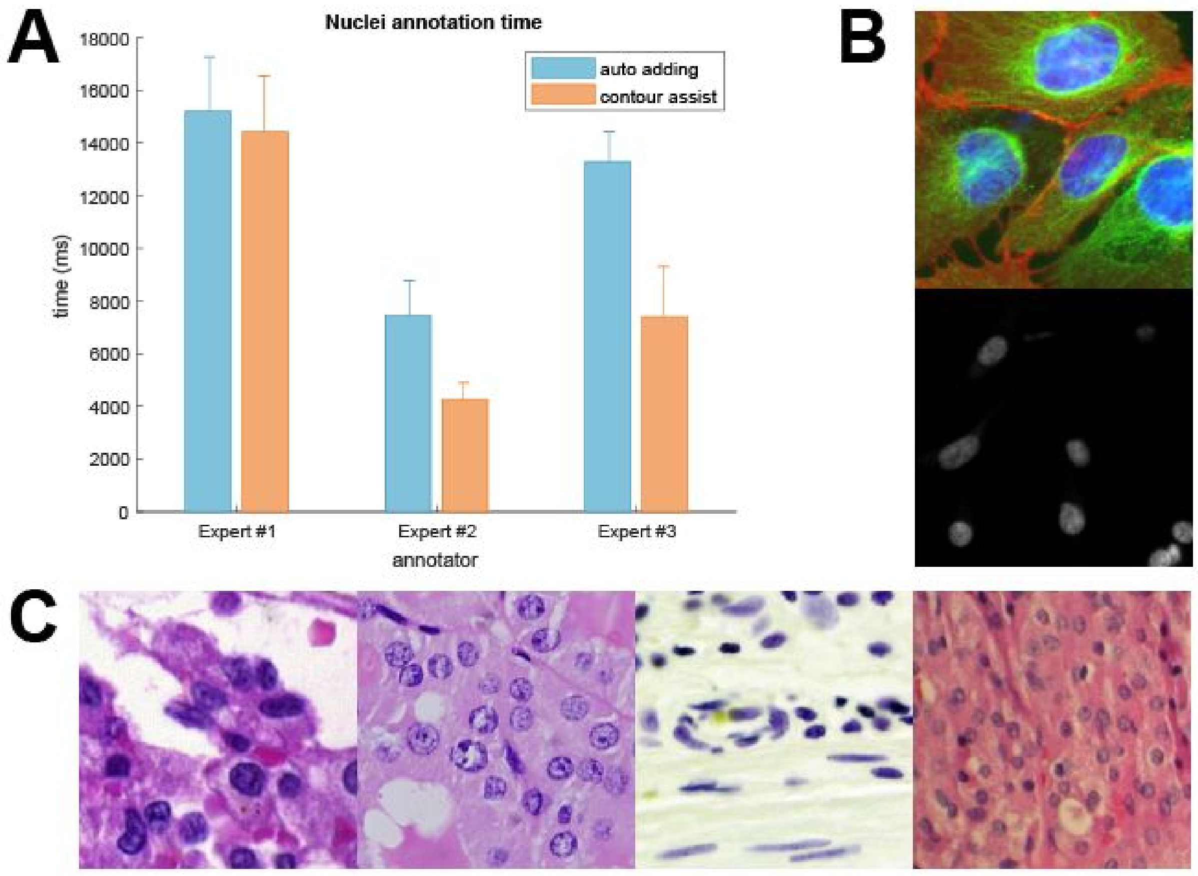


Figure 3. Annotation times on nucleus images. AnnotatorJ was tested on sample microscopy images (both fluorescent and brightfield, as well as cell culture and tissue section images), annotation time was measured on a per-object (nucleus) level. Bars represent the mean annotation times on the test image set, error bars show SEM (standard error of the mean). Orange corresponds to Contour assist mode and blue to only allowing the Automatic adding option. A) Nucleus test set annotation times, B) example cell culture test images, C) example histopathology images. Images shown on B-C) are 256x256 crops of original images. Some images are courtesy of Kerstin Elisabeth Dörner, Andreas Mund, Viktor Honti and Hella Bolck.

To ensure efficient usage of our plugin in annotation assistance, we also evaluated the accuracies achieved in each test case by calculating mean intersection over union (IoU) scores of the annotations as segmentations compared to ground truth masks previously created by different expert annotators. We used the mean IoU score defined in the Data Science Bowl 2018 competition (https://www.kaggle.com/c/data-science-bowl-2018/overview/evaluation) and in (Hollandi et al 2019):

$$IoU\ score\,(t) = \frac{TP(t)}{TP(t)+FP(t)+FN(t)+\varepsilon} \qquad (1)$$

IoU determines the overlapping pixels of the segmented mask with the ground truth mask (intersection) compared to their union. IoU score is calculated at 10 different thresholds from 0.5 to 0.95 with 0.05 steps, at each threshold true positive (TP), false positive (FP) and false negative (FN) objects are counted. An object is considered true positive if its IoU is greater than the given threshold t. IoU scores calculated at all 10 thresholds were finally averaged to yield a single IoU score for a given image in the test set.

An arbitrarily small $\varepsilon = 10^{-40}$ value was added to the denominators for numerical stability. (1) is a modified version of mean average precision (mAP) typically used to describe the accuracy of instance segmentation approaches. Precision is formulated as

$$precision\,(t) = \frac{TP(t)}{TP(t)+FP(t)+\varepsilon} \qquad (2)$$

Nucleus and cytoplasm image segmentation accuracies were averaged over the test sets, respectively. We compared our annotators using and not using *Contour assist* mode (**Figure 4**). The results show greater inter-expert than intra-expert differences allowing us to conclude that the annotations created in AnnototarJ are nearly as accurate as free-hand annotations.

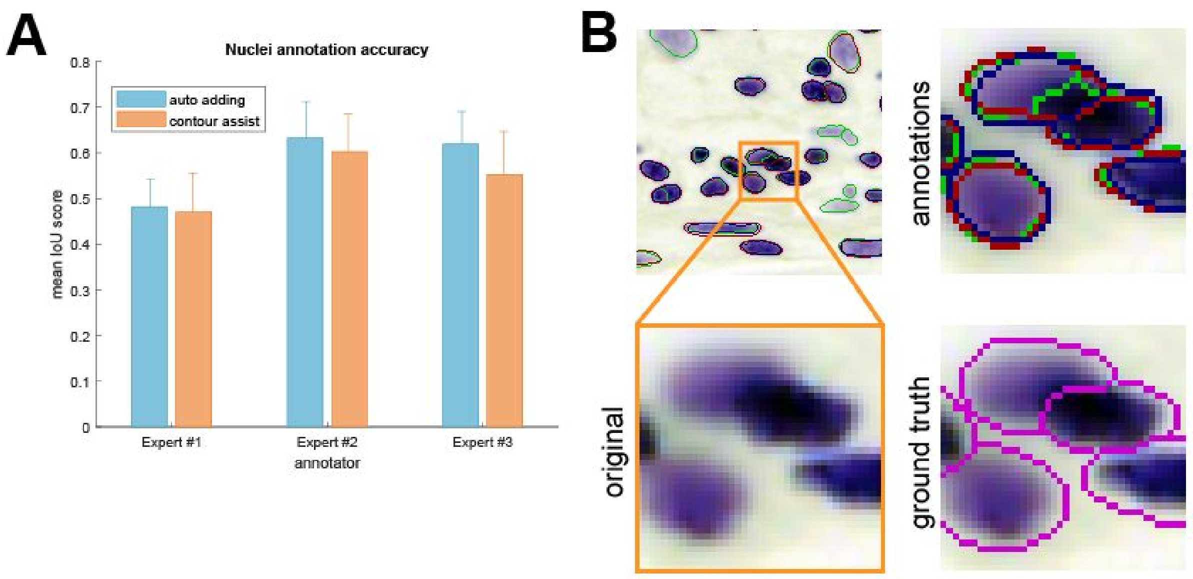


Figure 4. Annotation accuracies. Annotations created in the same test as the times measured in Figure 3 were evaluated using mean IoU scores for the nucleus test set. Error bars show SEM, colours are as in Figure 3. A) Nucleus test set accuracies, B) example contours drawn by our expert annotators. Inset

highlighted in orange is zoomed in showing the original image, annotations marked in red, green and blue corresponding to experts #1-3, respectively, and ground truth contours (in magenta) are overlayed on the original image for comparison..

Export evaluation

As the training data annotation process for deep learning applications requires the annotated objects to be exported in a manner that DL models can load them, which typically covers the four types of export options offered in AnnotatorJExporter, it is also important to investigate the efficiency of export. We measured export times similarly to annotation times. For the baseline results each object defined by their ROIs was copied to a new empty image, filled and saved to create a segmentation mask image file. Exportation from AnnotatorJExporter was significantly faster and only required a few clicks: it took 4 orders of magnitude less time to export the annotations (about 60 ms). Export times reported correspond to a randomly selected expert so that computer hardware specifications remain the same.

Comparison to other tools and software packages

The desire to collect annotated datasets has arisen with the growing popularity and availability of application specific DL methods. Object classification on natural images (photos) and face recognition are frequently used examples of such applications in computer vision. We discuss some of the available software tools created for image annotation tasks and compare their feature scope in the following table **(Table 1, see also Table S1)** and in **Supplementary Material**.

Table 1. Comparison of annotation software tools.

feature		**tool**							
		LabelImg	Lionbridge. AI	Hive	VGG Image Annotator	Diffgram	CytoMine	SlideRunner	AnnotatorJ
Open source		✓	x	x	✓	✓	✓	✓	✓
Cross-platform		✓	service	service	✓	✓	Ubuntu	✓	✓
Implementation		Python	N/A	N/A	web	Python, web	Docker	Python	Java
Annotation	Bounding box	✓	✓	✓	✓	✓	✓	✓	✓
	Freehand ROI	x	x	N/A	x	x	✓	x	✓
	Polygonal region	x	✓	✓	✓	✓	✓	✓	✓ (ImageJ)
	Semantic	x	✓	✓	x	x	✓	x	✓
	Single click*	x	x	x	x	x	✓ (magic wand)	✓	✓ (drag)
	Edit selection	x (drag)	N/A	N/A	✓	✓	✓	N/A	✓
Class option		✓	✓	✓	✓	✓	✓	✓	✓
DL	Support	x	AI assist	N/A	x	✓	x	x	✓
	Model import	x	N/A	N/A	x	Tensorflow	x	x	Keras, DL4J

**: bounding box drawing with a single click and drag is not considered a single click annotation.*

We collected our list of methods to compare following (Morikawa 2019) and (“The best image annotation platforms”, 2018). While there certainly is a considerable amount of annotation tools for object detection purposes, most of them are not open source. We included Lionbridge.AI (https://lionbridge.ai/services/image-annotation/) and Hive (https://thehive.ai/), two service-based solutions because of their wide functionality and artificial intelligence support. Both of them work in a project-management way and outsource the annotation task to enable fast and accurate results. Their main application spectra covers more general object detection tasks like classification of traffic video frames. LabelImg (https://github.com/tzutalin/labelImg) on the other hand, as well as the following tools, is open source but offers a narrower range of annotation options and lacks machine learning support making it a lightweight but free alternative. VGG Image Annotator (Dutta and Zisserman, 2019) comes on a web-based platform therefore makes it really easy for the user to familiarize with the software. It enables multiple types of annotation with class definition. Diffgram (https://diffgram.com/) is available both online and as a locally installable version (Python) and adds DL support which speeds up the annotation process significantly - that is, provided the intended object classes are already trained and the DL predictions only need minor edit. A similar, also web-based approach is provided by supervise.ly (https://supervise.ly/, see in **Supplementary Material**) which is free for research purposes. Even though web-hosted services offer a convenient solution for training new models (if supported), handling sensitive clinical data may be problematic. Hence, locally installable software are more desirable in biological and medical applications. A software closer to the bioimage analyst community is CytoMine (Marée et al., 2016), a more general image processing tool with a lot of annotation options that unfortunately does not provide DL support. SlideRunner (Aubreville et al., 2018) was created for large tissue section (slide) annotation specifically but similar to others it does not integrate machine learning methods to help annotation and rather focuses on the classification task.

AnnotatorJ on the other hand, as an ImageJ (Fiji) plugin should provide a familiar environment for biomage annotators to work in. It offers all the functionality available in similar tools (such as different annotation options: bounding box, polygon, free hand drawing, semantic segmentation and editing them) while also incorporates support for a popular DL model, U-Net. Furthermore, any user-trained Keras model can be loaded into the plugin with ease because of the DL4J framework, extending its use cases to general object annotation tasks. Due to its open source implementation the users can modify or extend the plugin to even better fit their needs.

Discussion

We presented an ImageJ plugin, AnnotatorJ for convenient and fast annotation and labelling of objects on digital images. Multiple export options are also offered in the plugin.

We tested the efficiency of our plugin with three experts on two test sets comprising of nucleus and cytoplasm images. We found that our plugin accelerates the hand-annotation process on average and offers up to four orders of magnitude faster export. By integrating the DL4J Java framework for U-Net contour suggestion in *Contour assist* mode any class of object can be annotated easily: the users can load their own custom models for the target class.

Materials and methods

ImageJ

ImageJ (or Fiji: Fiji is just ImageJ (Schindelin et al., 2012)) is an open-source, cross-platform image analysis software tool in Java that has been successfully applied in numerous bioimage analysis tasks (segmentation (Arganda-Carreras et al., 2017; Legland et al., 2016), particle analysis (Abramoff et al. 2004) etc.) and is supported by a broad range of community, comprising of biomage analyst end users and developers as well. It provides a convenient framework for new developers to create their custom plugins and share them with the community. Many typical image analysis pipelines have already been implemented as a plugin, e.g. U-Net segmentation plugin (Falk et al., 2019).

U-Net implementation

We used the Deeplearning4j (DL4J, http://deeplearning4j.org/) implementation of U-Net in Java. DL4J enables building and training custom DL networks, preparing input data for efficient handling and supports both GPU and CPU computation throughout its ND4J library.

The architecture of U-Net was first developed by Ronneberger *et al* (Ronneberger et al., 2015) and was designed to learn medical image segmentation on a small training set when limited amount of labelled data is available, which is often the case in biological contexts. To handle touching objects as often the case in nuclei segmentation, it uses a weighted cross entropy loss function to enhance the object-separating background pixels.

Region-growing

A classical image processing algorithm, region-growing (Adams and Bischof, 1994; Haralick and Shapiro, 1985) starts from initial seed points or objects and expands the regions towards the object boundaries based on the intensity changes on the image and constraints on distance or shape. We used our own implementation of this algorithm.

Acknowledgements

R.H. and P.H. acknowledge support from the LENDULET-BIOMAG Grant (2018-342) and from the European Regional Development Funds (GINOP-2.3.2-15-2016-00006, GINOP-2.3.2-15-2016-00026, GINOP-2.3.2-15-2016-00037). We thank Krisztián Koós for the technical help, and Ákos Diósdi, Máté Görbe and Tamás Monostori for testing the plugin.

References

"The best image annotation platforms for computer vision (+ an honest review of each)" (2018, October 30), https://hackernoon.com/the-best-image-annotation-platforms-for-computer-vision-an-honest-review-of-each-dac7f565fea

Abramoff, M.D., Magalhaes, P.J., Ram, S.J. "Image Processing with ImageJ". Biophotonics International, volume 11, issue 7, pp. 36-42, 2004.

Adams, R. and Bischof, L. (1994), "Seeded region growing", IEEE Transactions on Pattern Analysis and Machine Intelligence.

Arganda-Carreras, I., Kaynig, V., Rueden, C., Eliceiri, K.W., Schindelin, J., Cardona, A. and Sebastian Seung, H. (2017), "Trainable Weka Segmentation: a machine learning tool for microscopy pixel classification", Bioinformatics , Vol. 33 No. 15, pp. 2424–2426.

Aubreville, M., Bertram, C., Klopfleisch, R. and Maier, A. (2018), "SlideRunner", Bildverarbeitung Für Die Medizin 2018.

Badrinarayanan, V., Kendall, A. and Cipolla, R. (2017), "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 39 No. 12, pp. 2481–2495.

Caicedo, J.C., Roth, J., Goodman, A., Becker, T., Karhohs, K.W., Broisin, M., Molnar, C., et al. (2019), "Evaluation of Deep Learning Strategies for Nucleus Segmentation in Fluorescence Images", Cytometry. Part A: The Journal of the International Society for Analytical Cytology, Vol. 95 No. 9, pp. 952–965.

Cancer Genome Atlas Research Network. (2008), “Comprehensive genomic characterization defines human glioblastoma genes and core pathways”, Nature, Vol. 455 No. 7216, pp. 1061–1068.

Chenyang Xu and J. L. Prince, "Gradient vector flow: a new external force for snakes," Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Juan, Puerto Rico, USA, 1997, pp. 66-71. doi: 10.1109/CVPR.1997.609299

Coelho, L.P., Shariff, A. and Murphy, R.F. (2009), “Nuclear segmentation in microscope cell images: A hand-segmented dataset and comparison of algorithms”, 2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro, available at:https://doi.org/10.1109/isbi.2009.5193098 .

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., et al. (2016), “The Cityscapes Dataset for Semantic Urban Scene Understanding”, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), available at:https://doi.org/10.1109/cvpr.2016.350 .

Dutta, A. and Zisserman, A. (2019), “The VIA Annotation Software for Images, Audio and Video”, Proceedings of the 27th ACM International Conference on Multimedia - MM ’19, available at:https://doi.org/10.1145/3343031.3350535 .

Eclipse Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0. http://deeplearning4j.org

F. Schroff, D. Kalenichenko and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 815-823. doi: 10.1109/CVPR.2015.7298682

Falk, T., Mai, D., Bensch, R., Çiçek, Ö., Abdulkadir, A., Marrakchi, Y., Böhm, A., et al. (2019), “U-Net: deep learning for cell counting, detection, and morphometry”, Nature Methods, Vol. 16 No. 1, pp. 67–70.

Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014), “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”, 2014 IEEE Conference on Computer Vision and Pattern Recognition, available at:https://doi.org/10.1109/cvpr.2014.81 .

Grigorescu, S., Trasnea, B., Cocias, T. and Macesanu, G. (2019), “A survey of deep learning techniques for autonomous driving”, Journal of Field Robotics, available at:https://doi.org/10.1002/rob.21918 .

Haralick, R.M. and Shapiro, L.G. (1985), "Image Segmentation Techniques", Applications of Artificial Intelligence II, available at:https://doi.org/10.1117/12.948400 .

He, K., Gkioxari, G., Dollar, P. and Girshick, R. (2017), "Mask R-CNN", 2017 IEEE International Conference on Computer Vision (ICCV), available at:https://doi.org/10.1109/iccv.2017.322 .

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-R., Jaitly, N., Senior, A., et al. (2012), "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups", IEEE Signal Processing Magazine.

Hollandi, R. et al. A deep learning framework for nucleus segmentation using image style transfer. bioRxiv 580605 (2019). doi: 10.1101/580605

J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement", arXiv (2018), https://arxiv.org/abs/1804.02767

Kass, M., Witkin, A. and Terzopoulos, D. (1988), "Snakes: Active contour models", International Journal of Computer Vision.

Kumar, N., Verma, R., Sharma, S., Bhargava, S., Vahadane, A. and Sethi, A. (2017), "A Dataset and a Technique for Generalized Nuclear Segmentation for Computational Pathology", IEEE Transactions on Medical Imaging, Vol. 36 No. 7, pp. 1550–1560.

Legland, D., Arganda-Carreras, I. and Andrey, P. (2016), "MorphoLibJ: integrated library and plugins for mathematical morphology with ImageJ", Bioinformatics , Vol. 32 No. 22, pp. 3532–3534.

Ljosa, V., Sokolnicki, K.L. and Carpenter, A.E. (2012), "Annotated high-throughput microscopy image sets for validation", Nature Methods, Vol. 9 No. 7, p. 637.

Marée, R., Rollus, L., Stévens, B., Hoyoux, R., Louppe, G., Vandaele, R., Begon, J.-M., et al. (2016), "Collaborative analysis of multi-gigapixel imaging data using Cytomine", Bioinformatics , Vol. 32 No. 9, pp. 1395–1401.

Moshkov, N. et al. Test-time augmentation for deep learning-based cell segmentation on microscopy images. bioRxiv 814962; doi: https://doi.org/10.1101/814962

Naylor, P., Lae, M., Reyal, F. and Walter, T. (2017), "Nuclei segmentation in histopathology images using deep neural networks", 2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017), available at:https://doi.org/10.1109/isbi.2017.7950669 .

Naylor, P., Lae, M., Reyal, F. and Walter, T. (2019), “Segmentation of Nuclei in Histopathology Images by Deep Regression of the Distance Map”, IEEE Transactions on Medical Imaging, Vol. 38 No. 2, pp. 448–459.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016), “You Only Look Once: Unified, Real-Time Object Detection”, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), available at:https://doi.org/10.1109/cvpr.2016.91

Rei Morikawa, “24 Best Image Annotation Tools for Computer Vision” (2019, July 18), https://lionbridge.ai/articles/image-annotation-tools-for-computer-vision/

Ronneberger, O., Fischer, P. and Brox, T. (2015), “U-Net: Convolutional Networks for Biomedical Image Segmentation”, Lecture Notes in Computer Science.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., et al. (2015), “ImageNet Large Scale Visual Recognition Challenge”, International Journal of Computer Vision.

Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., et al. (2012), “Fiji: an open-source platform for biological-image analysis”, Nature Methods, Vol. 9 No. 7, pp. 676–682.

Schneider, C.A., Rasband, W.S., Eliceiri, K.W. "NIH Image to ImageJ: 25 years of image analysis". Nature Methods 9, 671-675, 2012.

Suleymanova, I., Balassa, T., Tripathi, S., Molnar, C., Saarma, M., Sidorova, Y. and Horvath, P. (2018), “A deep convolutional neural network approach for astrocyte detection”, Scientific Reports, Vol. 8 No. 1, p. 12878.

Taigman, Y., Yang, M., Ranzato, M. ’aurelio and Wolf, L. (2014), “DeepFace: Closing the Gap to Human-Level Performance in Face Verification”, 2014 IEEE Conference on Computer Vision and Pattern Recognition, available at:https://doi.org/10.1109/cvpr.2014.220 .

Y. Sun, X. Wang and X. Tang, "Deep Learning Face Representation from Predicting 10,000 Classes," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1891-1898. doi: 10.1109/CVPR.2014.244

Supplementary Material

Test data

We tested the efficiency in terms of both annotation time and accuracy on two test image sets: nucleus and cytoplasm. A sample of the test sets is displayed on **Figures 3B-C** and **S1B-C**; images are collected from publicly available datasets and our collaborators (see sources in (Hollandi et al. 2019)). We collected images from various modalities including different microscopy imaging (brightfield, fluorescent), sample origin (cell lines, histological tissue sections), staining technique (label-free, immunohistochemical or fluorescent markers for specific cellular compartments) that show the target object (nucleus or cytoplasm) in a heterogeneous environment so that we can test our plugin in general cases.

Evaluation

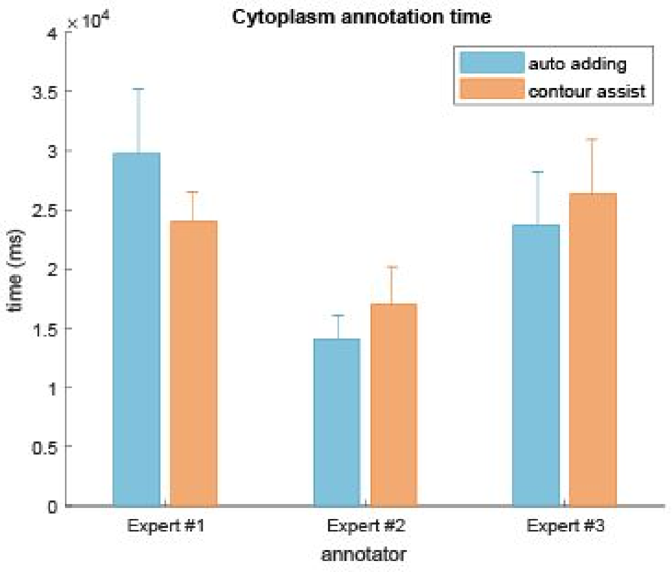
We used SEM (standard error of the mean) in our evaluation of annotation time and accuracy which is formulated as

$$SEM(x) = \frac{\sigma(x)}{\sqrt{N}} \tag{3}$$

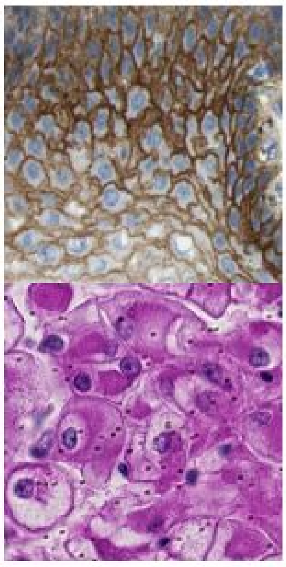
where the sample is denoted as x, $\sigma(x)$ marks the standard deviation of x and N is the number of elements in x.

We also tested annotation on a cytoplasm test image set containing more complex regions e.g. overlapping objects (see **Figure S1**). Our cytoplasm model aided annotation, however, due to its semantic nature it could not always separate touching objects properly hence the increased annotation time in certain cases.

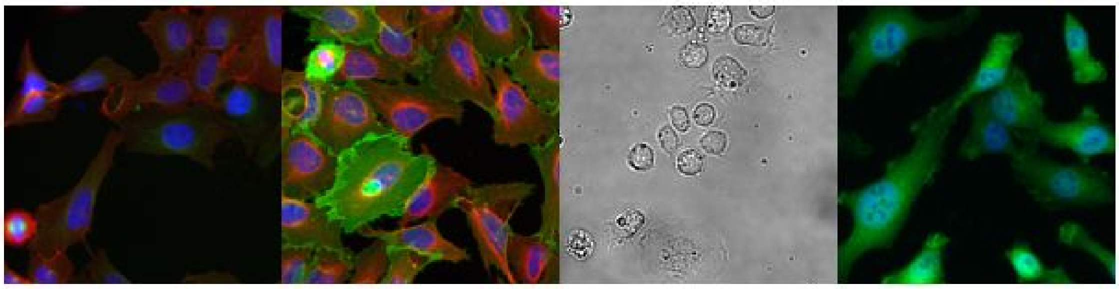
A



B



C



Supplementary Figure S1. Annotation times on cytoplasm images. AnnotatorJ was tested on sample microscopy images (both fluorescent and brightfield, as well as cell culture and tissue section images), annotation time was measured on a per-object (cytoplasm) level. Bars represent the mean annotation times on the test image set, error bars show SEM (standard error of the mean). Orange corresponds to Contour assist mode and blue to only allowing the Automatic adding option. A) Cytoplasm annotation times, B) example histopathology test images, C) example cell culture images. Images shown on B-C) are 256x256 crops of original images. Some images are courtesy of Kerstin Elisabeth Dörner, Andreas Mund, Viktor Honti and Hella Bolck.

Additional methods

Active contours (AC)

Active contours (Kass et al., 1988) is a generally well applicable image processing technique to fit an initial contour to the boundaries of an object by evolving a so-called *snake* contour driven by 2 energy functions as follows.

$$E = \int_0^1 E_{int}\,[\nu\,(s)]ds + \int_0^1 E_{ext}\,[\nu\,(s)]ds \tag{4}$$

$$E_{int}\,[\nu\,(s)] = \tfrac{1}{2}\left[\alpha\,(s)\left|\tfrac{\partial v(s)}{\partial s}\right|^2 + \beta\,(s)\left|\tfrac{\partial^2 v(s)}{\partial s^2}\right|^2\right] \tag{5}$$

$$E_{ext}\ [\mathrm{v}\,(s)] = -\gamma \cdot \left| I\,(x\,(s)\,,y\,(s)) * \nabla^2 G_\sigma \right|^2 \qquad (6)$$

(4) is the combination of the internal (5) and external (6) energy functions, and is minimized during the iterative evolution process. The internal energy is responsible for extending and smoothing the contour while the external energy restrictively drives it towards the edges of the image which are enhanced by the LoG (Laplacian of Gaussian) filter-convolved image (data term). α, β and γ are the regularization parameters.

Gradient vector flow (GVF)

GVF was first published in (Xu and Prince 1997) as a method to define a vector field based on the image gradient, that is the derivatives of the image, as enhanced by edges. It is attracted to bright regions (ideally edges) on the image.

Additional annotation tools

Since open source or local software is of primary interest in medical research (see in Introduction and *Comparison to other tools and software packages*), we extend our previous list of compared tools here.

We briefly discussed supervise.ly (https://supervise.ly/) above, a service-based web-hosted solution for data labelling with DL model support. Training of various DL models including Mask R-CNN, U-Net, YOLO and others is possible. It supports annotation as bounding box, semantic- and polygonal drawing, editing points in polygons and class labelling. However, its free community edition intended for research purposes provides limited functionality.

FastAnnotationTool (https://github.com/christopher5106/FastAnnotationTool) is a lightweight, open source bounding box annotation tool with editing option in C++.

Make Sense (https://www.makesense.ai/) is another open source labelling tool that also adds DL support for annotation suggestion (using pre-trained models) and labelling. Point, polygon and bounding box annotation is possible and classes can be assigned. Implemented in TypeScript, it offers both a web-based and a local (https://github.com/SkalskiP/make-sense) version.

We collected the sources of annotation tools in Table S1.

Table S1. Sources of the compared software tools.

tool	source
LabelImg	https://github.com/tzutalin/labelImg

Lionbridge.AI	https://lionbridge.ai/services/image-annotation/
Hive	https://thehive.ai/hive-data
VGG Image Annotator	http://www.robots.ox.ac.uk/~vgg/software/via/via-1.0.6.html
Diffgram	https://diffgram.com/ https://github.com/diffgram/diffgram/tree/master/sdk
CytoMine	http://www.cytomine.org/ https://github.com/cytomine/Cytomine-bootstrap
SlideRunner	https://github.com/maubreville/SlideRunner
supervise.ly	https://supervise.ly/
FastAnnotationTool	https://github.com/christopher5106/FastAnnotationTool
Make Sense	https://www.makesense.ai/ https://github.com/SkalskiP/make-sense

Supplementary video

A short demonstration of *Contour assist* is provided as the video *figure2.mov*; available in higher resolution at https://drive.google.com/file/d/1dk3FrX-KIhTpaNYSKv-zVsoAoVUGtEYp/view?usp=sharing.