SVETLANA: A SUPERVISED SEGMENTATION CLASSIFIER FOR NAPARI

Clément Cazorla, Renaud Morin *

IMACTIV-3D
1 place Pierre Potier, 31100 - Toulouse

Pierre Weiss †

CNRS & Université de Toulouse
31400 - Toulouse

ABSTRACT

We develop and present a Napari plugin called SVETLANA (SuperVised sEgmenTation cLAssifier for NapAri). It is dedicated to the manual or automatic classification of segmentation results in bio-medical imaging. While many open-source softwares now make it possible to automatically segment complex 2D and 3D objects such as cells in biology, the subsequent analysis of the results is not yet accessible to non specialists. This plugin allows end-users to train and run efficient neural network classifiers such as residual networks. The resulting network can be used as a post-processing tool to improve the segmentation, or as a classifier for various tasks (e.g. separating different cell populations). We showcase its practicality through various real cell biology problems in 2D, 3D and multi-spectral imaging.

***Index Terms*—** Segmentation, Classification, Convolutional Neural Networks, bio-medical imaging

1. INTRODUCTION

The last decade has made automatic segmentation of bio-medical images much more accessible to users not familiar with signal processing. This is the result of progress in machine learning, to the creation of open training databases and to the development of ergonomic open-source softwares. Technologies such as neural networks provide unprecedented segmentation results. They make it possible to avoid setting hyperparameters which are often hard to tune and interpret. Examples of powerful and popular tools for segmentation in biology include Ilastik [1], CellPose [2], StarDist [3] or Deep-ImageJ [4].

1.1. Our motivation

Unfortunately, segmentation masks – as good as they are – are rarely directly exploitable to answer biological questions. In particular, it is often necessary to classify the detected objects in order to perform statistical analyses that give a concrete meaning to the results. While these excellent segmentation tools have solved an important problem, a difficult part of the analysis remains inaccessible to most users.

1.2. Our contribution

The goal of this work is to continue filling the gap between methodological advances and end-users, by providing a convenient software for the classification of segmentation results. We resort to the newborn Napari environment [5] which allows visualizing and analyzing complex multi-dimensional images (e.g. 2D, 3D, 3D+t, hyperspectral) in Python. Our software takes the form of a Napari plugin called SVETLANA. It is separated in three different modules:

Annotation This module picks connected components of the segmentation masks at random and displays them within their neighborhood, so that the user can label them. In our 2D experiments, it allows labeling about 1000 cells in 15 minutes.

Training This module allows to pick an arbitrary Pytorch [6] neural network architecture (possibly pretrained) and to further train it with the annotations generated by the previous module.

Prediction This module uses the trained network to classify the connected components of the segmentation mask.

The results are stored in files widely accessible formats for the forth-coming analyzes. This plugin meets a need to further enhance the excellent results obtained with recent wide purpose segmentation tools.

1.3. Related works

Different options can be adopted to segment and classify objects in images. Before 2010, most of the works relied on the following pipeline (see e.g. [7]): 1) Segment the image 2) Annotate the resulting masks 3) Extract features within the masks (e.g. edges, textures, ...) 4) Design a classifier based on the extracted features. Each of the above step was carried out with carefully hand-crafted methods. Supervised and unsupervised learning then progressively entered in the game.

*C. Cazorla is partially funded by ANR CIFRE 2020/0843. He acknowledges the image.sc community for their precious help.

†P. Weiss acknowledges a support from ANR-3IA Artificial and Natural Intelligence Toulouse Institute and ANR Micro-Blind

In many cases, they outperformed man-made routines by allowing to explore a wider range of decision routes.

An effort was then pursued to make these technologies available to the larger number. One remarkable example is Ilastik [1]. There, a few annotations by the user are usually enough to perform complex classification tasks with an arbitrary number of classes. Its backbone is a random forest classifier with a fixed number of features (convolutions with different filter types). It is widely praised for its ease of use. A few clicks are enough to solve many real-world problems. Unfortunately, this strength is also a limitation in certain cases: the performance of random forests falls short in comparison with the most advanced segmentation and classification routines trained with vast collection of carefully labelled data.

When precision is critical, the rapidly evolving state-of-the-art is rather based on neural networks and especially convolutional neural networks [8, 9]. The downside of these technologies is the need to create large data sets, which are usually just not accessible. Each biology laboratory explores a different organism, at a different scale with a different modality and focus. Each collected image can be costly both in terms of money, know-how and time. To address this issue, new initiatives emerge to collect large heterogeneous training databases. For instance the Data Science Bowl [10] allowed to train a single neural network, which is now capable of segmenting cells of nearly any type. This tool, embedded in neat graphical interfaces (e.g. CellPose [2] or StarDist [3]) is a huge asset for biology. Unfortunately, as of now, it does not provide classification tools.

2. PLUGIN DESCRIPTION

The objective of SVETLANA as a whole is to provide a tool to ”sort” the segmentation results, either manually or automatically. It is separated in 3 different stages, similar to what is done in Ilastik [1].

2.1. The choice of Napari

Just as Fiji [11], Napari is a completely free and open-source project. The two programs are similar in a number of ways, but we chose Napari for several reasons:

- Napari is based on Python and Qt whereas Fiji is based on Java. As we all know, Python is much more efficient than Java for scientific computing. Moreover, it is compatible with all the most recent Deep Learning and optimization libraries which are developed in Python, not in Java. It is also very fast at loading huge images as it supports image pyramids.
- The ability to use cookiecutter to generate a plugin template easily makes the integration very pleasant and intuitive. This partly explains the important growth of the number of plugins and the rapid expansion of Napari. Moreover, the developer community is extremely responsive on `https://forum.image.sc/` and helped us a lot in the development of Svetlana.
- It also already embeds several segmentation reference methods such as Cellpose [2] and allows the user to easily load several plugins in series and create his custom processing pipeline, which makes the integration Svetlana natural since it is in line with a segmentation method.

2.2. The annotation mode

This first sub-plugin takes as input two images: the image itself and its segmentation mask. Each object ot the latter should be given a different label. For instance, a 20-cell image would have a mask with values in a range of 0-20.
There are two philosophies for using this software:

- It can be seen as an annotation tool in its own right. For an image with a reasonable amount of segmented objects, it could be used to classify them in a simple and efficient way, in order to perform statistical analyses afterwards.
- Nevertheless, if the image contains an astronomical number of detected objects, this software becomes an aid for the creation of a training data set for a classifier. Indeed, the tool extracts a subset of thumbnails of a size chosen by the user and randomly selected in order to maximize the chances of obtaining a minimal and representative data set.

As shown in 1, since it is not easy for the user to determine the optimal patch size, there is a feature to suggest it to the user, as well as the maximum number of thumbnails that can be extracted, i.e. the number of objects of which the segmentation mask is composed.

It works on 2D and 3D images, whatever the channels number and each patch represents the object in its neighborhood, as shown in 2. The user can choose the maximum label number he wants, and each label is attributed clicking on the number we want to give him. As soon as the patch is labeled, the next image is automatically loaded. If an error is committed, a backward step is possible by clicking on the ”r” key.

Once the patches have been extracted and the annotation completed, all the information which are necessary for the training are stored in a binary file:

- The image path
- The labels image path
- The coordinates list for each object
- The labels list

- The chosen patch size

Additionally, it is possible to save a new mask per existing label, as well as a binary file that can be reloaded using the torch.load() function. It contains morphological features of interest of each of the objects that were extracted from the connected component analysis.

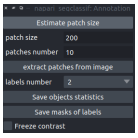


Fig. 1: Annotation interface

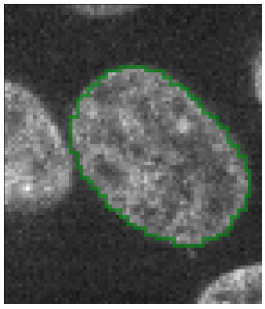


Fig. 2: Visualization of a patch to be annotated

2.3. The training mode

Parler de Deep Image Prior (i.e. on peut entraîner avec peu de labels, rasoir d'Ockham, Yann Ollivier).

2.3.1. The light-weight approach

For the classification part, we used relatively classical convolutional neural networks such as ResNet18. Nonetheless, the paradigm goes against the classical approach of training neural networks using very large databases.Indeed, we were inspired by the approach put forward in the article Deep Image Prior[12] in which they fit a generator network to a single degraded image. They assimilate the reconstruction to a conditional image generation problem and demonstrate that all the information requisite to resolve it is contained in the single degraded input image and in the network chosen for reconstruction.

In section 3, we will show that it is possible to train an efficient classifier model with a few hundred images.

2.3.2. The features

In this section, we are going to detail the different features of the training interface shown in figure 3. The example image shown in the latter is 3D, composed of two channels, and it has been segmented using Cellpose[2].

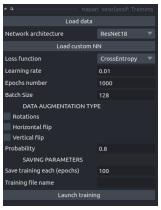


Fig. 3: The training sub plugin

First of all, the "load data" button aims at loading the content of the binary file saved at the end of the annotation and whose content we have detailed above.

Then, we propose a wide range of network architectures, but the user can also load the custom neural network of one's choice. We suggest the use of the cross entropy loss, but other classical losses are available.

The learning rate is chosen at the beginning and there is no scheduler for now, which means it will remain constant the whole training.

As the possible batch size depends on the performance of the machine the user works on, it can be chosen by the user. If it is selected too large, it will be automatically adjusted to the highest power of 2 tolerated by the GPU ram.

Basic data augmentations such as vertical and horizontal flips and rotations of angle between -90° and 90° are available, and their probability of occurrence can be adjusted.

Finally, we can choose the name of the ".pth" file containing the training result and all the times we want to save. This gives the possibility to access the network at different stages since we have no preconceived notion of the number of epochs necessary for convergence.

The saved files is also a binary containing several data:

- The model
- The optimizer (Adam) state
- The loss
- The epochs number
- The loss value list
- the image path
- The labels image path
- The patch size

2.4. The prediction mode

This plugin is very minimalist, as shown in figure 4. Indeed, it allows to load the training result file and to launch the prediction by choosing the batch size. As for the training, the optimal batch size will depend on the capabilities of the GPU.

In 2D, there are two representations of the classified mask as shown in figures 4 and 5.

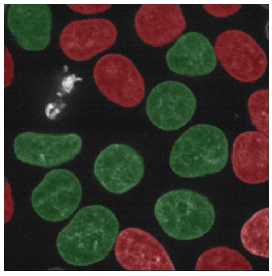


Fig. 4: Example of a 2D prediction with two labels

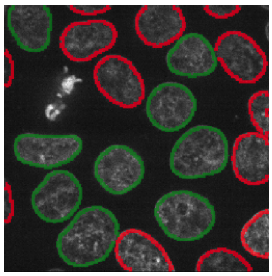


Fig. 5: An alternative representation of the labels

As for the annotation plugin, the user has the opportunity to save a mask per label and a binary file containing relevant morphological information about the classified objects to perform further statistical analysis.

3. NUMERICAL EXPERIMENTS

3.1. First experiment: artificial texture classification

For this first experiment, we generated an image composed of 345 circular objects differentiable only by their texture that we seek to order into two classes. The distribution of the two textures in the image is about 70/30 and we extract 100 thumbnails to annotate. As we have very few data, we decided to demonstrate the efficiency of shallow convolutional networks. As shown in Figure 6, we designed a tiny 2-layer CNN of 3000 parameters.

We have a misclassified objects rate of 0.8%

4. CONCLUSION

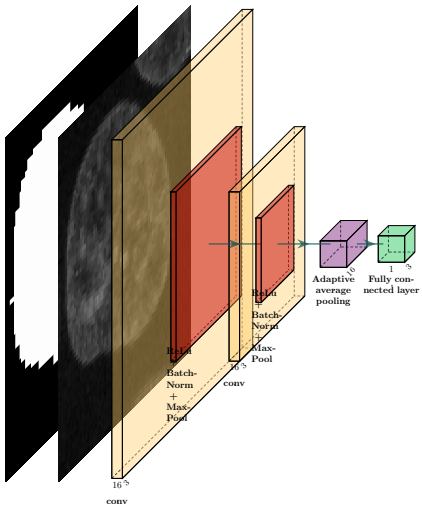
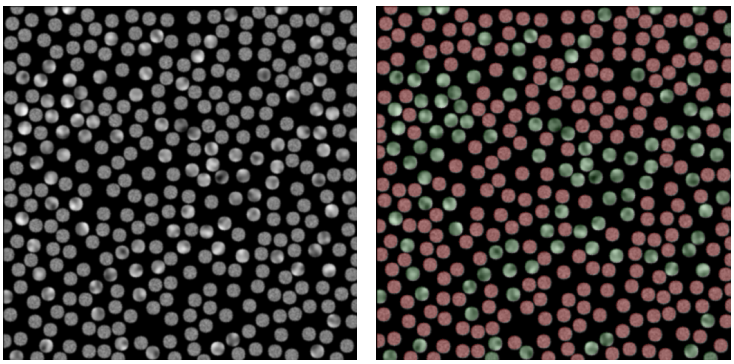


Fig. 6: Our 2-layer custom CNN



(a) Original texture image (b) Classified mask

Fig. 7: Result of the textures classification

5. REFERENCES

[1] Stuart Berg, Dominik Kutra, Thorben Kroeger, Christoph N Straehle, Bernhard X Kausler, Carsten Haubold, Martin Schiegg, Janez Ales, Thorsten Beier, Markus Rudy, et al., "Ilastik: interactive machine learning for (bio) image analysis," *Nature Methods*, vol. 16, no. 12, pp. 1226–1232, 2019.

[2] Carsen Stringer, Tim Wang, Michalis Michaelos, and Marius Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," *Nature methods*, vol. 18, no. 1, pp. 100–106, 2021.

[3] Elnaz Fazeli, Nathan H Roy, Gautier Follain, Romain F Laine, Lucas von Chamier, Pekka E Hänninen, John E Eriksson, Jean-Yves Tinevez, and Guillaume

Jacquemet, "Automated cell tracking using stardist and trackmate," *F1000Research*, vol. 9, 2020.

[4] Estibaliz Gómez-de Mariscal, Carlos García-López-de Haro, Wei Ouyang, Laurène Donati, Emma Lundberg, Michael Unser, Arrate Muñoz-Barrutia, and Daniel Sage, "Deepimagej: A user-friendly environment to run deep learning models in imagej," *Nature Methods*, vol. 18, no. 10, pp. 1192–1195, 2021.

[5] Jeffrey M Perkel et al., "Python power-up: new image tool visualizes complex data," *Nature*, vol. 600, no. 7888, pp. 347–348, 2021.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[7] Humayun Irshad, Antoine Veillard, Ludovic Roux, and Daniel Racoceanu, "Methods for nuclei detection, segmentation, and classification in digital histopathology: a review—current status and future potential," *IEEE reviews in biomedical engineering*, vol. 7, pp. 97–114, 2013.

[8] Anamika Dhillon and Gyanendra K Verma, "Convolutional neural network: a review of models, methodologies and applications to object detection," *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, 2020.

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, "Mask r-cnn," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988.

[10] Juan C Caicedo, Allen Goodman, Kyle W Karhohs, Beth A Cimini, Jeanelle Ackerman, Marzieh Haghighi, CherKeng Heng, Tim Becker, Minh Doan, Claire McQuin, et al., "Nucleus segmentation across imaging experiments: the 2018 data science bowl," *Nature methods*, vol. 16, no. 12, pp. 1247–1253, 2019.

[11] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, et al., "Fiji: an open-source platform for biological-image analysis," *Nature methods*, vol. 9, no. 7, pp. 676–682, 2012.

[12] Victor Lempitsky, Andrea Vedaldi, and Dmitry Ulyanov, "Deep image prior," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 2018, pp. 9446–9454.