Developing a User-Friendly and Modular Framework for Deep Learning Methods in 3D Bioimage Segmentation

Guillaume Mougeot

OXFORD BROOKES UNIVERSITY

THESIS

submitted in partial fulfilment of the requirements of the award of

Doctor of Philosophy

of

Oxford Brookes University

authored by

Guillaume MOUGEOT

Developing a User-Friendly and Modular Framework for Deep Learning Methods in 3D Bioimage Segmentation

---

October 2023

Carried out in collaboration with

Université Clermont Auvergne

THÈSE

En vue de l'obtention du

Doctorat de l'Université Clermont Auvergne

Présentée et soutenue publiquement **le 8 décembre 2023** par :

Guillaume MOUGEOT

Développement d'une infrastructure logicielle conviviale et modulaire pour des méthodes d'apprentissage profond appliquées à la segmentation de bio-images tridimensionnelles

---

Thèse dirigée par **Frédéric Chausse, Sophie Desset et Katja Graumann**

Cotutelle :	Oxford Brookes University Department of Health and Life Science
Ecole doctorale :	Sciences Pour l'Ingénieur n°70
Spécialité :	Image, Système de perception, Robotique
Unités de Recherche :	Institut Génétique Reproduction et Développement, Institut Pascal

---

JURY

Carole FRINDEL	Maître de Conférence HDR	INSA, Lyon	Rapportrice
Nicolas PASSAT	Professeur	Université Reims Champagne Ardenne	Rapporteur, Président
Jack SUNTER	Senior Lecturer	Oxford Brookes University	Examinateur
David ROUSSEAU	Professeur	Université d'Angers	Examinateur
Frédéric CHAUSSE	Professeur	Université Clermont Auvergne	Directeur
Sophie DESSET	Ingénieure de recherche	Inserm, Clermont-Ferrand	Co-directrice
Katja GRAUMANN	Senior Lecturer	Oxford Brookes University	Co-directrice

Acknowledgments

Writing acknowledgments is a difficult exercise. This is a new retrospective on three intensive years, to which is added the prism of emotion. For this particular project, this exercise was even more difficult as each of the numerous facets of this beautiful journey has hidden a multitude of people, linked to its multidisciplinary and international character. Two languages were spoken, and a plethora of areas studied. Unfortunately, I was not able to spend as much time as I would have liked with all these people, and I will not be able to thank them all for their undeniable contribution. Yet, I will do my best to thank those who have contributed most to forging the essence of this work.

Maybe unconventionally, as this project has involved two languages and as words do not carry the same weight in different mother tongues, I would like to write these acknowledgements in these two languages.

First of all, I would like to thank all the members of the jury who came from far away and gave me the privilege of spending several hours talking with them: David Rousseau, Carole Frindel, Jack Sunter and Nicolas Passat. From kind criticisms to scientific discussions pushing the boundaries of imagination, this moment of sharing will remain engraved in my memory. At the end of the discussion, I even caught myself exclaiming, "we should do a viva every month!"

And that clearly was the aim of the thesis committees, which, on an annual rather than a monthly basis, offered a fresh scientific perspective from experts who also came from far away. I would therefore like to thank David Rousseau once again, as well as Sébastien Tosi, Anaïs Badoual and Emmanuel Faure. I met them several times along the way, at conferences, workshops, and lab visits, and enjoyed some exhilarating moments of scientific curiosity.

Of all the exhilarating scientific moments during this thesis, those spent with my five supervisors are second to none. Thanks to them, I learned a lot. Really a lot. Perhaps more than they could possibly imagine. Much more than the purely technical-scientific knowledge involved in this project. I have exchanged a lot. It could have been easy to get lost, but each of the people involved was a pillar, a rock, a solid anchor through the

misty jungle of the great vagaries of scientific research. Thanks to them, this project has become a new archipelago, a new land of innovation, young yet fertile.

Ayant constitué l'ancrage principal dans le monde de la recherche en ingénierie, j'aimerais tout d'abord remercier Frédéric. Merci Fred pour avoir prêté l'oreille dans les moments d'incertitude scientifique, d'avoir apporté une lumière nouvelle lorsqu'une figure brouillonne, un article émergent, ou une réflexion balbutiante avaient besoin de recul. Merci aussi pour nos points d'échange moins scientifiques comme sur le mode fonctionnement de la recherche mondiale, française ou clermontoise ou sur les panels de façons qu'il y a d'aborder l'écriture d'une thèse. Merci beaucoup.

L'essence quintessentielle de ce projet est sa multidisciplinarité. Sans elle, ce projet aurait été tout autre. Et la personne qui, sans conteste, à participer à rendre l'impossible possible est Sophie. Motivée par une détermination scientifique forte, presque poétique, à vouloir se faire rencontrer deux mondes qui n'ont, aux premiers abords, que peu d'affinités réciproques, elle a réussi à maintenir à flot un navire sans cesse chahuté par d'omniprésentes contradictions. Etant maintenant, je l'espère, à même, moi aussi, de tenir la barre de ce navire impossible, je n'ai pas assez de mot pour exprimer ma gratitude. Merci Sophie pour nos indénombrables heures de discussion. Tu y as déployé une inégalable énergie à enseigner tout ce qu'il était possible de transmettre à un néophyte total en biologie. Tu as su répondre à la myriade, au miasme même, de mes milliers questions, plus ou moins intelligibles je l'admets, avec une patience incomparable. Merci d'avoir pris le temps de m'accompagner et relever dans tous mes moments de doute, qui, irrémédiablement, frappent tout chercheur en herbe. Merci d'avoir pris le temps d'écouter et interpréter tous ce charabia scientifique que je m'échignais à dessiner, et qui, au prix de moulte efforts communs, a pu germer, grandir et aboutir à un résultat dont je suis aujourd'hui très fier. J'espère de tout cœur que tous ces échanges, ces « brainstormings », ces pingpongs intellectuels, et ces aboutissements communs t'auront apporté autant qu'à moi. Et évidemment merci pour tous ces moments de partage en dehors du cadre du labo qui ont, sans le moindre doute, participé à mon plus grand bonheur et émerveillement pendant ce grand voyage de trois ans. J'aimerais ici aussi glisser un grand merci à Stéphane, Malvina et Sévan et à toute ta grande famille pour nos échanges passionnants et, très souvent, poilant.

Mes remerciements s'adressent aussi à Christophe qui n'a eu de cesse de s'intéresser à ce projet flirtant constamment avec la frontière de ses sentiers battus. Tu as su garder une écoute et une parole systématiquement authentique et un œil toujours rempli d'une joie de continuellement apprendre et d'enrichir ce projet. Surtout n'arrête jamais de poser toutes ces brillantes questions, et de sans cesse tenter de créer de nouveaux objectifs collaboratifs avec (presque toute) la communauté scientifique, biologique comme informatique. J'étais très heureux de voir ton implication aussi à partager ces savoirs émergents et très admiratif de t'avoir vu prendre les rênes du master de bio-informatique. Merci aussi pour ces échanges sportifs très motivants, qui nous ont menés jusqu'au sommet du siège du roi Arthur.

This multidisciplinary adventure was also international. I would therefore also like to thank my supervisors at Oxford, who were greatly involved in this project. I would like to thank Katja for her attentiveness and for actively helping to ensure that my exchanges with the university and my stay in Oxford went as smoothly as possible. A huge thank you to David, without whom this project would certainly not have seen the light of day. For me, you are proof that the flame of curiosity can never be extinguished. You were there, always with a big smile, from the beginning to the end of this project. Your pertinent questions, your insightful remarks, your wise advice and your unfailing guidance have without the slightest doubt made me dream, carried me and helped me to go all the way. Thank you very much David.

But the framework of this thesis certainly does not stop at that of my supervisors. An immeasurable network of contacts has inspired, consciously or unconsciously, hundreds of ideas, comments, advice, and has also, in many ways, contributed to shaping this journey.

Mes remerciements vont donc aussi à toute l'équipe CODED qui a accepté la présence de cet étrange projet, parasitant parfois leurs tranquilles réunions de biologie. Merci à Aline, Manu, Sylvie, Sylviane, Sam, Simon, Aurélia, Manon, Lauriane, Sarah, Léa, et Tristan. Merci particulièrement à Aline pour ses longues heures passées à annoter des images 3D, qui, finalement, ont eu l'air de payer par de beaux graphiques et images et par un modèle de deep learning plus que performant. Merci aussi à Manu pour ses nombreuses irruptions souriantes dans le bureau, ses fabuleuses quiches aux champignons et les soirées jeux de société. Merci à Sarah d'avoir partagé et égaillé ce

petit bureau obscur qu'était le nôtre, et d'avoir accepté de répondre à toutes mes questions, pour la plupart administrative, et, très probablement, ennuyantes à souhait. Merci à Floriane, Valentine, Miguel, Etienne, Lauriane d'avoir aussi partager ces murs trop chauds en été et glacials en hiver. J'ai aussi eu la chance d'avoir à mes côtés 4 compagnons de routes que sont Adama, Pedro, Abderrahime et Sami. Merci à eux d'avoir accepté d'être embarqué dans une aventure aux retombées incertaines et d'avoir contribué à lui donner une plus grande solidité scientifique. Je remercie de tout cœur aussi un autre électron libre qui, poussé par une curiosité sans faille, à décider de se remonter les manches et de tenter d'utiliser le brouillon informe qu'était Biom3d au moment de sa création : Hervé. Tu as passé des heures et des heures, à tenter d'éplucher et, pire, de comprendre tous les bugs qui truffaient l'outil de l'époque. Grâce à toi, et tes centaines d'emails, le résultat est d'une propreté inespérée, c'est beau, merci beaucoup. Pas très loin, se trouvait aussi un autre interlocuteur très précieux qui m'a transmis une quantité colossale de savoir dont je rêverais d'en avoir retenu le dixième (je serais un « vrai » savant-informaticien si c'était le cas) : Pierre. Merci beaucoup pour ces heures d'échange qui, pourtant, semblaient s'écouler à une allure folle. Ce grand laboratoire a aussi abrité et fait grandir de nombreuses autres personnes qui sont devenus des amis. Merci à Victor, Ronan (et Nono !), Nico, Elodie, Romane grâce à qui les couloirs du GReD et les jeudi soirs (mais pas que !) étaient remplis de rires inoubliables. Merci beaucoup à Marine et Lili pour leurs inarrêtable passions à égailler la vie de ce laboratoire, par des vidéos, des quizz en tout genre, des jeux de piste capillotractés, des magazines loufoques, et bien sûr des acrobaties et galipettes inoubliables. Merci à Jonas, Prikshit, Fabiana, Vincent et tous ceux qui, dans l'ombre participent à rendre chaque journée de ce laboratoire plus lumineuse que la précédente. Je suis plus qu'admiratif de tous ces efforts et cette générosité bénévole. Merci à Phillipe pour m'avoir fait découvrir les hauts plateaux auvergnats à coup de pédales. Au sein du GReD, j'aimerais aussi remercier Margaux, Steph, Emilie (B.), Isa, Aline, Caro et toutes les personnes qui, sans le savoir peut-être, m'ont donné le sourire et ont fait résonner en moi cette passion pour le métier de chercheur. Enfin, j'aimerais remercier Maryse et Marie-Jo pour avoir accepté de suivre les rocambolesques démarches administratives de ce projet tripartite.

Le deuxième des parties clermontois de ce projet était l'Institut Pascal au sein duquel j'ai aussi pu faire des rencontres plus qu'enrichissantes. Je pense notamment aux

irremplaçables (à leur grand regret d'ailleurs) membres et organisateurs des animations scientifiques, réunions qui, pour moi, ont été une révélation : Elie, Rémi, Mathieu, Johann et Céline. Merci beaucoup aussi à Emilie (P.) pour ta présence, tes retours et conseils.

Despite the short time I was able to spend in Oxford, I had the joy of discovering not only a very healthy working environment, shaking up my vision of the profession of researcher, but also an extremely welcoming and dynamic team. I warmly thank Verena, Nadine, and Charlotte for their genuine curiosity, for their overflowing motivation, and for all those beers and chicken wings. Thanks also to Flavia for her patience and her investment beyond all expectations in making the Biom3d tool usable within the team.

The sixty-eight names cited above represented sixty-eight new encounters that I had the chance to make during this long stay, but they are only a very small glimpse of the submerged surface of the iceberg. I would also like to conclude with a big thank you to everyone I was unable to mention. Hundreds of people got involved in this adventure. I am thinking in particular of all the beta testers, users and curious people who appreciated Biom3d as well as the multiple moments of discussion following MIFOBIO, at the IABM, at the SEB meeting or at all the other conferences, seminars, congresses, and workshops at which I had the chance to participate. Thank you to all of you.

Cependant, cette épopée n'aurait certainement pas été colorée de la même façon sans les innombrables amitiés qu'ils l'ont faite vivre, à commencer par toutes les nouvelles rencontres clermontoises et oxoniennes. Merci encore à tous les copains du GReD, de l'Institut Pascal et du Nuclear Plant Group. Merci aussi à tous ceux que j'ai eu la chance de rencontrer en dehors des murs des laboratoire, avec une pensée particulière pour les membres du club de gym.

Parce que la force des amitiés se joue du temps et de l'espace, j'aimerais aussi remercier Jérôme, Sebastian, Gojko, Teresa, Emre, Matthieu, Jimmy, Ludo, Céline, Justine pour tous ces moments importants de vie passés à échanger et à rire, que se fussent à Shanghai à Nantes ou à bien d'autres endroit de cette petite planète. Merci d'avoir été présent et soutenu aussi bien le jour de ma soutenance que pendant toutes les années qui l'ont précédées. Plus amont encore dans le temps, je dis merci aux

fameux « geeks » alsaciens, Alexandre, Alexandre et Melinda, pour ce lien d'amitié qui défie, menton levé, les douze années qui séparent notre première rencontre.

J'aimerais aussi dire merci de toute mon âme à la colonne vertébrale de mon esprit, l'indestructible cœur et pilier de ma stabilité émotionnelle, j'ai nommé : ma famille. Je ne peux malheureusement en énumérer tous les membres ici. Néanmoins, mille milliards de mercis à mes parents, ma sœur, mes frères, oncles et tantes, et grands-parents. Sans vous je ne serais rien, littéralement.

Enfin, au cœur de ces aventures et chemins semés d'embuches, j'ai eu l'indescriptible chance de rencontrer un petit être qui en a illuminé la voie et qui est devenu une famille au cœur de ma famille : Marianne. Merci plus que tout.

Abstract

The emergence of deep learning has breathed new life into image analysis, especially for the segmentation, a challenging step required to quantify bidimensional (2D) and tridimensional (3D) objects. Despite deep learning promises, these methods are only slowly spreading in the biological field. In this PhD project, the 3D nucleus of the cell is used as the object of interest to understand how its shape variations contribute to the organisation of the genetic material. First a literature survey showed that very few publicly available methods for 3D nucleus segmentation provide the minimum requirements for their reproducibility. These methods were subsequently benchmarked and only one of them called nnU-Net surpassed the best specialized computer vision tool. Based on these observations, a new development philosophy was designed and, from it, Biom3d, a novel deep learning framework emerged. Biom3d is a user-friendly tool successfully used by biologists involved in 3D nucleus segmentation and provides a new alternative for automatically and accurately computing nuclear shape parameters. Being well optimized, Biom3d also surpasses the performance of cutting-edge methods on a wide variety of biological and medical segmentation problems. Being modular, Biom3d is a sustainable framework compatible with the latest deep learning innovations, such as self-supervised methods. Self-supervision aims at tackling the important need for deep learning methods in manual annotations by pretraining models on large unannotated datasets to extract information first before retraining them on annotated datasets. In this work, a self-supervised approach based on pretraining an entire U-Net model with the Triplet and Arcface losses was developed and demonstrates significant improvements over supervised methods for 3D segmentation. The performance, modularity and interdisciplinary nature of the tools developed during this project will serve as an innovation platform for a wide panel of users ranging from biologist users to future deep learning developers.

Résumé

L'émergence de l'apprentissage profond a donné un nouveau souffle à l'analyse d'images, en particulier pour la segmentation, une étape difficile mais nécessaire pour quantifier des objets bidimensionnels (2D) et tridimensionnels (3D). Malgré les promesses de l'apprentissage profond, ces méthodes ne se répandent que lentement dans le domaine de la biologie. Au cours de ce projet de thèse, le noyau 3D de la cellule est utilisé comme objet d'intérêt pour comprendre comment ses variations de forme contribuent à l'organisation du matériel génétique. Tout d'abord, une étude bibliographique a montré que très peu de méthodes disponibles publiquement pour la segmentation du noyau 3D répondent aux exigences minimales de reproductibilité. Ces méthodes ont ensuite été évaluées et seule l'une d'entre elles, appelée nnU-Net, a surpassé le meilleur outil spécialisé de vision par ordinateur. Sur la base de ces observations, une nouvelle philosophie de développement a été élaborée et, à partir de celle-ci, Biom3d, une nouvelle infrastructure logicielle pour l'apprentissage profond, a vu le jour. Biom3d est un outil convivial utilisé avec succès par les biologistes impliqués dans la segmentation des noyaux 3D et offrant une nouvelle alternative pour mesurer automatiquement et avec précision les paramètres morphologiques des noyaux. Bien optimisé, Biom3d surpasse également les performances des méthodes de pointe sur une grande variété de problèmes de segmentation biologique et médicale. Modulaire, Biom3d est un cadre durable compatible avec les dernières innovations en matière d'apprentissage profond, telles que les méthodes auto-supervisées. L'auto-supervision vise à répondre au besoin important en annotation manuelle des méthodes d'apprentissage profond, en pré-entraînant les modèles sur de grands ensembles de données non annotées pour extraire des connaissances *a priori* avant de les réentraîner sur des ensembles de données annotées. Dans ce travail, une approche auto-supervisée basée sur le pré-entraînement d'un modèle U-Net entier avec les fonctions de coût Triplet et Arcface est développée et démontre des améliorations significatives par rapport aux méthodes supervisées au regard de la segmentation 3D. La performance, la modularité et la nature interdisciplinaire des outils développés au cours de cette thèse serviront de plateforme d'innovation pour un large panel d'utilisateurs allant des utilisateurs biologistes aux futurs développeurs de méthodes d'apprentissage profond.

Contents

Chapter 1

Context, challenges, and motivations

1.1 Tridimensional segmentation of biomedical images

Many fundamentally different fields have developed the use of tridimensional (3D) images for a variety of very different applications. Biologists, physicians, physicists, engineers, and computer scientists have worked together to develop the tools needed; biological and physiological markers, physical and mathematical theories, cameras and microscopes, computers and algorithms to image and analyse 3D objects. Each of these fields has been enriched by rapid advances of knowledge and discoveries. For example, since the invention of the confocal microscope by Marvin Minsky [1] and the discovery of the properties of the green fluorescent protein (GFP) by Roger Y. Tsien, Osamu Shimomura, and Martin Chalfie [2], the number of publications using the derivatives of these techniques have soared. The original goal of capturing 3D images was, and still is, to observe 3D objects, to understand their roles, their mechanisms, and interactions. Visualizing internal mechanisms of life in action has allowed simultaneously to confirm biological and medical hypotheses and to ease the formulation of novel ones. Biologists now often alternate between benchtop experiments and microscope visualization. For instance, a combination of the biochemical properties of a protein and knowledge of its location by 3D microscopy could be used to confirm its function. In medicine, 3D imaging has allowed physicians to understand the structure and mechanisms of organs inside a living body. They are now able to detect early stages of cancers before the onset of severe symptoms.

Measurements in 3D images. To go beyond simple observations and intuitions, the image analyst must take measurements in the 3D images. For example, a biologist could be interested in counting fluorescence *in situ* hybridization (FISH) spots to compare a mutant and a wild-type. A radiologist could measure the size and location of a set of lung nodules. Once these measurements have been made, the analyst can then perform statistical evaluations and thus reinforce the results brought by the observation. Indeed, when dealing with living objects observations are often non-binary, meaning that it is often hard to tell if what an observer sees is not very different from what one might want to see, thus incorporating subjectivity into the observation

and its interpretation. Measurements combined with statistical analysis help the biologist or physician manage confirmation bias. Too many measurements can be made on 3D images to provide an exhaustive list. Some frequently used examples are counting objects, distance between two objects, position and orientation of objects, volume, surface, elongation, curvature, and signal intensity. By adding a dimension when moving from 2D to 3D, the number and the difficulty of possible measurements increase considerably. For instance, in 2D, a "surface" is the perimeter of each object and can be found with simple algorithms such as [3]. Computing a surface of a 3D volume might imply a series of more complicated techniques and more computing resources for 3D reconstruction such as the marching cubes algorithm [4] for mesh computation or a method for surface normal estimation such as in [5]. 3D measurements from images require each voxel (3D pixel) to be identifiable, for instance to differentiate from background or other object of interest (Figure 1-1). This step is called *segmentation* and is often the inevitable step prior to any 3D measurement.

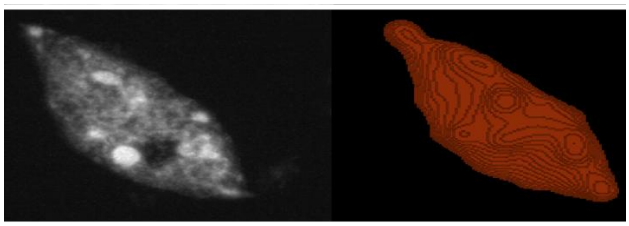


Figure 1-1 – **A tridimensional plant nucleus taken with a confocal microscope (*left*) and its corresponding segmentation (*right*).** Illustration captured using *napari*, a 3D visualization tool.

Manual segmentation. To create such a segmented image, the expert, biologist or physician, must attribute manually a label to each voxel, a task called *annotation* or *labelling*. The experts can use tools such as napari [6] to annotate their 3D images. As 3D images often contain several tens of millions of voxels, experts have also been helped by semi-automatic tools, such as Labtik, ilastik [7] (Figure 1-2), or Weka [8], [9], which are based on machine learning algorithms such as Random Forest [10] to propose a segmentation result which the expert uses to assist them as they annotate. Manual annotation of 3D biological and medical images is difficult due to, first, the nature of objects that must be annotated and, second, the annotation tools. Objects in biological and biomedical imaging are seldom well defined and discrete. A cell

membrane does not follow the mechanical rules of either a fluid or a solid. The intrinsic disorder and the blurry edges of biological and medical objects thus require well-trained eyes and are often a source of disagreement between experts. The second difficulty of manual annotation is caused by hardware and software limitations. Annotating 3D images on a 2D screen requires software adaptations that are non-trivial. The current publicly available annotation tools offer only the possibility to annotate one by one each 2D slice of the 3D image. The problem with this approach is that the transition between two slices is often incorrect which causes the segmentation to look serrated when visualized with a 3D viewer, such as napari. The solution is then to correct the annotation by going through 2D slices cut orthogonally to the first 2D slices, which is a very time-consuming process. Another solution could be to use Virtual Reality combined with a tool such as DIVA proposed by the Institut Pasteur [11], unfortunately not publicly available yet.

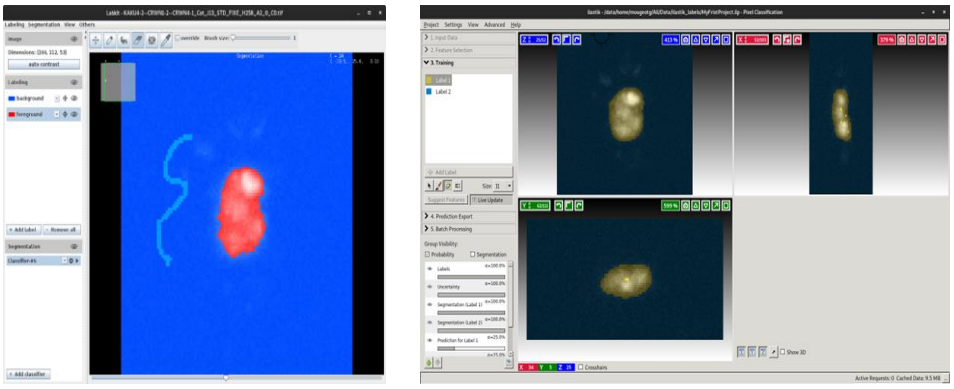


Figure 1-2 – Semi-automatic annotation tools for 3D segmentation. (left): Labtik. (right): ilastik

Manipulating 3D images. After capturing a 3D image, an expert may face various challenges when attempting to manipulate it. Two prominent obstacles are the considerable memory space required for 3D images and the scarcity of free and open-source tools available for their transformation. Some 3D images can have a size above 200 gigabytes which requires powerful hardware to open and more powerful ones to manipulate. This constraint has pushed computer scientists to develop memory-optimized tools such as 3DSlicer [12], Paraview [13], 3DMod [14], webKnossos [15], Dask [16] or Fiji/ImageJ [17] to manipulate these large images. These tools rely on intelligently compressing the images to fit them into computer memory and to display them on screen. To analyse 2D biomedical images, the most used toolbox is

Fiji/ImageJ, a free, open-source and user-friendly software. Fiji/ImageJ provides a range of image processing and analysis tools, including image filtering, segmentation, registration, and measurement. It also supports a variety of file formats commonly used in scientific imaging, such as TIFF, JPEG, and DICOM, and more thanks to the Bio-format plugin which lets microscopists import images from any type of microscope. Fiji/ImageJ is highly extensible, with a large collection of plugins and macros available to users. Even though Fiji/ImageJ includes a set of plugins for 3D images such as a 3D viewer, a 3D segmentation toolbox, notably provided by the MorpholibJ plugin [18], or a 3D object tracker, it is mostly designed for 2D images. For example, in Fiji/ImageJ, measurements of segments in 3D images are possible only on 2D slices. The programming language of Fiji/ImageJ is Java which represents another limit of the software as the computer vision community is now shifting to Python, especially with the arrival of deep learning. To create novel perspectives for 3D images, napari is currently leading the movement with a versatile Python framework, integrating a memory-efficient 3D viewer and, noticeably, a friendly programming framework to easily add new plugins.

Segmentation diversity. Napari allows simultaneous visualisation of an input image and its segmentation. This tool can give access the vast diversity of possible segmentations in 3D images. To classify these segmentations, computer scientists have created two main categories of segmentation. First, *semantic segmentation* involves annotating each voxel in an image with a class label and corresponds to what was previously described in this Section. Second, *instance segmentation* is similar to semantic segmentation but additionally assigns to each voxel of individual objects another label in order to separate them. For example, a 3D image representing a cluster of cell nuclei can undergo a semantic segmentation by assigning each voxel the label “nucleus” or the label “background”. If subsequently going through instance segmentation, the voxels are additionally assigned with the label “nucleus 1”, “nucleus 2”, “nucleus 3” and so on, thus discriminating each individual nucleus. Instance segmentation is harder than semantic segmentation and can only been applied to objects with a distinguishable border, such as cells, nuclei, or organs. It is not possible to perform instance segmentation on cell membranes, for instance, or on the image background. The work described in this thesis will focus mainly on semantic segmentation. The thesis will present mature technologies available for both

categories, but also present gaps in current applications for 3D biological and biomedical image segmentation.

An important context. Despite the previous important categorization, the almost infinite number of possible experiments in biology or in medicine, causes each segmentation problem to be unique and deeply dependent on the context. To extract significant information from biological samples, biologists, microscopists, and image analysists must work together. Performing biological experiments intended to be analysed by microscopy without having in mind the end image processing could have severe consequences that might lead to a dead end and waste of time and money. One classic example of such consequences arises when determining the image sampling rate. A 3D image can be understood as a three-dimensional array of numerical values, where each value represents the intensity level of a small region within the observed object (see Section 1.2.3). This 3D image is essentially a sampled representation of the real object, where regions between each sample are not captured. Therefore, selecting an appropriate sampling rate is crucial to ensure that the object of interest is fully captured in the resulting image. Choosing a sampling rate that is too low can lead to incomplete or inaccurate representations of the object, resulting in loss of important information, a problem called *under sampling*. Another example that would involve both the biologist and the microscopist, is a cell surface estimation problem in an epithelium of Drosophila ovaries where cells are densely packed together. In this case, a mistake could be to mark with a fluorescent protein the cytoplasm only, which would result in the inability to distinguish each cell from its adjacent counterparts. Instead, marking the cell membrane and then segmenting the inner regions could solve this issue. These examples point out how intricate the relationship between all the actors involved in imaging should be. If some of these actors perceive their counterparts as "external service providers" or view their contributions as merely illustrative applications of their own work, it increases the likelihood of the final project being flawed. Interdisciplinarity undoubtedly plays a pivotal role in the advancement of science, particularly in the fields of biological and medical imaging.

1.2 The biological origins of the project and its development

Intending to make ends meet between fundamentally orthogonal fields of research, biology and computer science, implies taking risks and, sometimes, daring to break down barriers, fostering collaborations and dialogues, and encouraging fruitful

exchanges of knowledge and expertise. This work has been made between two research laboratories in biology, one in the Institut de Génétique, Reproduction et Développement (iGReD) in the Université Clermont Auvergne, Clermont-Ferrand, France and one at Oxford Brookes University, United-Kingdom and one research laboratory in computer vision in the Institut Pascal in the Université Clermont Auvergne, France. Following our interdisciplinary goals and despite the strong computer science content of this project, most of the work has been done at one of the two laboratories of biology. The first seeds for present works were planted in the thesis of Axel Poulet (2016) and developed in the work of Tristan Dubos (2021) the iGReD. Thus, its early inception and developments was by biologists and have now slowly grown to reach the field of computer vision. In this Section the biological starting points of the project and their evolution are presented.

1.2.1 Plant cells, nuclei and chromocentres

This project was born studying plant nuclei, more specifically *Arabidopsis thaliana* nuclei. This plant is a genetical model, historically selected for its ability to reproduce in four weeks and for its quite short genome (~150 mega-base-pair) with few repeated sequences compared to other plants. Apart from their apparent agricultural applications, plant cells are the subject of an active field of fundamental research, which has produced relatively fewer studies when compared to research on human cells. A lot is still to be discovered about the functioning of plants. So where to start? Certainly, the most notable part is the plant cell nucleus where the DNA (deoxyribonucleic acid) is compartmented. The DNA is a long sequence of pairs of only four types of nucleotides, a so-called *base-pair*. Parts of the DNA sequence are called *genes* which are transformed into a sequence of amino acids called *proteins*, which are the building blocks of cell machinery. The level of production of each protein is called *gene expression*. Genome structure is not random and is organised in sub-structures at different scales, from the nucleosomes, tiny loops in the DNA strands, to the chromosomes, dividing the entire genome into separated large regions, the positioning of which influences the level of gene expression. The DNA and its structuring proteins are called *chromatin.* The nucleus is an *organelle* inside the cell, isolating the chromatin with a double membrane. The nucleus plays a vital role in safeguarding the integrity of DNA throughout the lifespan of the eukaryotic cell, especially during cell division. The transfer of molecules between the DNA and the

cytoplasm is restricted by the nuclear membrane and specialized structures called *nuclear pores*, which may have an impact on protein production. The nucleus may adopt a large variety of shapes depending on cell location in the organism, on cell development, on external stresses, or on diseases (Figure 1-3). For instance, an abnormal morphology of nucleus has been shown to be the biomarker of certain pathologies such as cancer [19]. The morphology of the plant nucleus, the structure of its envelope and sub-compartments, the level of gene expression, and the cell environment are the primary focus of investigation for both teams of biologists involved in this work.

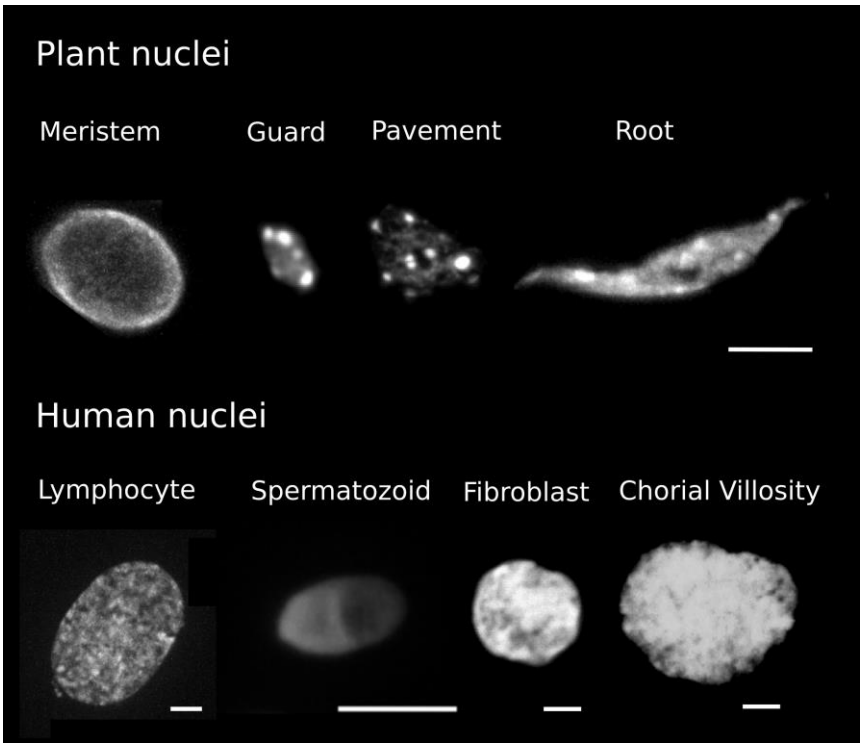


Figure 1-3 – **Diversity of nucleus shapes.** Each image represents one isolated nucleus stained by DAPI. Scale 5 µm.

Nuclear Organization in Plant Cells. Just like animal cells, plant cells feature a complex nucleus composed of various subdomains and structures (Figure 1-4). Key components of the nuclear organization include the outer membrane, nuclear pores, inner membrane, nucleoskeleton, and nucleoplasm. The outer membrane is an extension of the endoplasmic reticulum (ER), another crucial organelle in eukaryotic cells. It plays a vital role in connecting the nucleus with the cytoskeleton, which provides mechanical support to the cell. This connection is indispensable for processes like cell division, cell differentiation, cell polarization, and, in the case of plants,

nucleus migration during day-night cycles. In the context of *A. thaliana*, a model plant species, research teams have contributed to a deeper understanding of the role of KASH proteins in this nuclear-cytoskeleton link. Studies have revealed that this bridging complex has multiple functions such dynamics roles during cell division [20] and structural roles in positioning the nucleus within the cell, conveying signals across the membrane and organizing chromatin in the 3D nuclear space with impact on gene transcription [21]. Studies have revealed that this bridging complex has multiple functions such dynamics roles during cell division [22] and structural roles in positioning the nucleus within the cell, conveying signals across the membrane and organizing chromatin in the 3D nuclear space with impact on gene transcription [21]. Finally, the innermost part of the nucleus contains the nucleoplasm and the chromatin whose tridimensional structure has been the subject of recently growing communities such as the *International Nucleome Consortium* (INC, https://inc-cost.eu/) for animals and the *Impact of nuclear domains on gene expression and plant traits* for plants (INDEPTH, https://www.brookes.ac.uk/indepth/) [23].

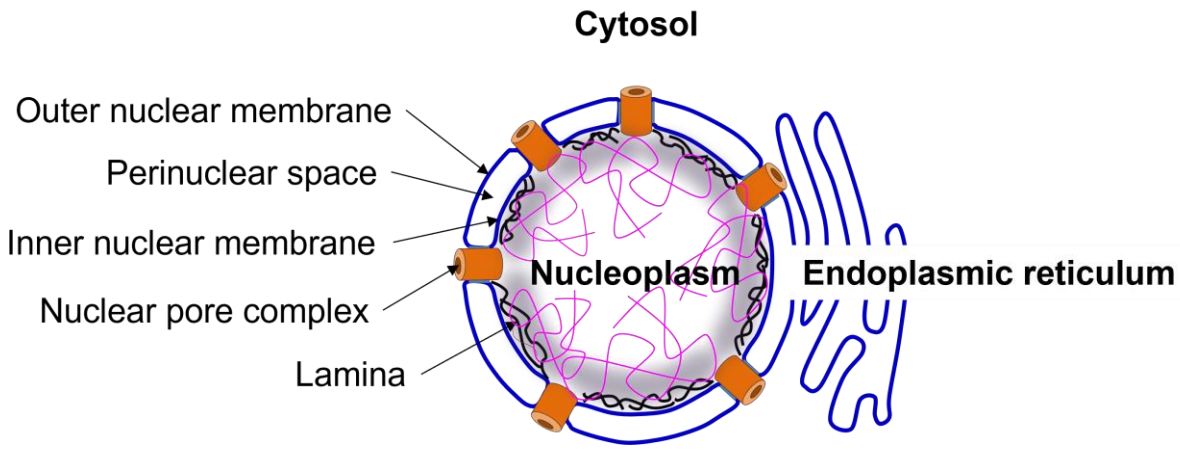


Figure 1-4 – **Organization of the nuclear envelope.** The nuclear envelope is made of membrane (*blue*) including outer nuclear membrane, inner nuclear membrane separated by the perinuclear space, interrupted by numerous nuclear pore complexes (*orange*) and is connected with the endoplasmic reticulum. The plasmina (*black*) is the putative plant lamina. The nuclear envelope, the nuclear pore complexes and the plasmina are believed to interact with chromatin (*purple*).

DNA organization. The genetic material of the cell is structured in chromosomes located in the nucleoplasm. Chromosomes form small, looped structures made up of DNA called *nucleosomes*, rolled around a set of eight proteins termed *histones*. The

nucleosome structure compacts the DNA and impacts gene expression [24], with more compact regions, known as *heterochromatin*, containing less expressed genes than less compact regions, *euchromatin*. On a global scale, the 3D structure of the whole DNA sequence is organised in the nucleus into chromosome territories (Figure 1-5). This has been shown in *A. thaliana* [25] by marking each of the 5 pairs of chromosomes with a different fluorescent stain. Each chromosome includes two linked copies, named *sister chromatids*, of the same DNA sequence. Marking plant DNA with a fluorescent stain, such as DAPI, also highlights the density distribution of the DNA. In such an image (Figure 1-5), it can be seen denser spots of DNA called *chromocentres*. These heterochromatin regions are the *centromeric* and *pericentromeric* regions of each chromosome, which are located at the link between the two sister chromatids. Changes in gene expression not involving changes in DNA base sequence, such as the study of DNA compactness, are termed epigenetic. Epigenetics has shown that the gene sequences are not sufficient to proper cell functioning and that the surroundings of these sequences are as important. DNA methylation, post-translation modification of histones, histone variants, non-coding RNA, chromatin spatial organization are all part of epigenetic field of study. These modifications are so important that they could, for example, cause the malfunction of daughter cells, even if the DNA sequence has correctly been reproduced.

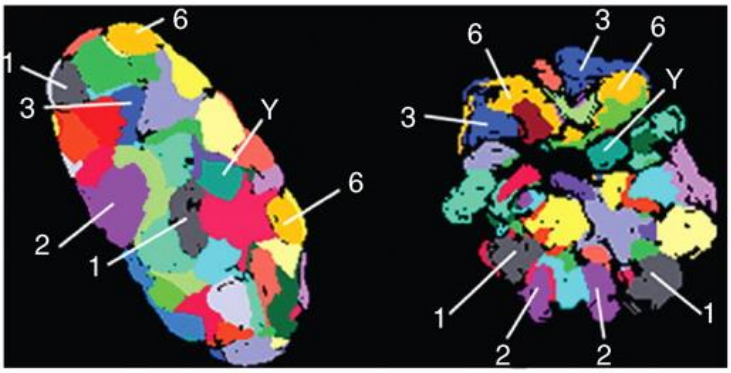


Figure 1-5 – **Chromosome territories in human nucleus.** Each colour corresponds to a different chromosome. Illustration adapted from [26].

Biological questions and experiments. There is an intimate relationship between the local and the global scale of the organisation of the nucleus. For instance, when biologists alter the genes encoding histone proteins by stopping the expression of one of them, the size and the location of chromocentres change. On the contrary, during

cell development or following certain stresses, the nucleus organisation changes, and the expression of chromatin proteins is also impacted. There is thus a strong correlation between nucleus shape change and gene expression. But is there a causal relationship? More specifically, the epigenetic question that is being explored by biologists involved in this project could be formulated as follows: to what extent does the re-organization of nuclear space influence gene expression? This re-organization could be caused by stresses, mutations, or cell differentiation. During their experiments, biologists submit their plants to a series of stresses and mutations and then try to measure modifications in plant characteristics, a step called phenotyping. These plant characteristics are mainly visual and could be macroscopic, such as the size of leaves or root hairs, or could be microscopic, such as nucleus volume or number of chromocentres. While macroscopic phenotypes can be measured with a simple camera, microscopic phenotypes generally require a microscope. The first studies led by the biologists taking part in this project focused on SUN proteins [27], [28] and on KAKU4 and CRWN proteins [29], [30] of *A. thaliana*. In these studies, a multidisciplinary approach was chosen, including genetics, tridimensional microscopy, and image processing. Image processing specifically was the subject of the work of two PhD students preceding the work presented in this thesis. They both participated in the development of NucleusJ and NODeJ, plugins of Fiji/ImageJ, designed to help biologists automatically extract nucleus and chromocentre characteristics, such as volume or elongation, in 3D images containing potentially thousands of them. These tools are based on “classical” computer vision techniques such as thresholding or watershed segmentation [31]. Their limitations have been the starting point of this work.

1.2.2 Microscopy and medical imaging

To understand these limitations and the objects of study of this work, let us delve into the nature of microscopy images and, more generally, 3D imaging techniques. The primary objective of this research is to develop an image analysis system capable of analysing volumetric images, irrespective of their origin or type, thereby enabling the analysis of 3D images related to any biomedical problem. However, there are two reasons for not detaching from the imaging technique the biological or medical object of interest. First, it is to avoid falling into the pitfall of becoming an “external service provider” and, in this way, missing key features in the images, pivotal for their future analysis. Second, it is arduous to create a comprehensive dataset that would represent

the almost infinite variations of imaging techniques and types of biomedical problems. Therefore, research in this work will initially focus on microscopy images and gradually expand to other possibilities if the developed image analysis systems exhibit flexibility. In-depth explanations will be presented on microscopy image analysis and then on medical imaging techniques as a potential application.

Confocal microscopy. Microscopes, regardless of their type, are typically composed of three fundamental components: an emitter that projects particles, a transmitter that guides the particles towards the sample, and a receptor that detects the particles after they have interacted with the sample by reflection or transmission. If the emitted particles are photons, then the microscope is a light microscope. Among all types of light microscopes, the confocal microscope is widely recognized as one of the most powerful tools for capturing high-resolution 3D images. For confocal microscopes, the emitter is a laser, the transmitter is a set of lenses and mirrors, and the receptor is a photomultiplier tube converting the light into electrical signal. Confocal microscopy is a special type of fluorescence microscopy, which means that the sample contains a substance that can remit laser light [32]. In biology, this substance is usually a set of fluorescent molecules called *fluorophores*. Fluorophores emit light in all directions once exposed to laser light, which has the good property of allowing biologists to see through tissue but the disadvantage of often producing images with background noise. The main particularity of confocal microscopes is a special device, called *pinhole*, that is placed in front of the receptor and that removes the out-of-focus and noisy light emission. The final image is thus clearer. However, the pinhole limits the field of view to one tiny region of the sample. In order to get a view of the whole sample, the sample must be moved in the three spatial directions: up and down, left and right, and forward and backward. Once the whole sample scanned, tridimensional image is obtained with a high resolution and a high contrast. As these tridimensional movements are prone to adding noise the images and slowness in the acquisition process, many improvements of confocal microscope have been done to improve the acquisition accuracy and speed, among which the most notable ones are probably light-sheet microscopes and super-resolution microscopes. Once again, these improvements have been possible thanks to a close relationship between biologists and chemists, designing new fluorophores, microscopists and physicists, building new microscopes, and computer scientists and image analysists, developing image processing software. Compared with other types

of microscopies, confocal microscopes can capture high-resolution 3D images in a relatively short amount of time and have less requirements regarding sample conditioning, which allow, for instance, to image living bodies. For these reasons, confocal microscopy is usually the method of choice for biological imaging. Confocal imaging of the nucleus represents the original application of the image analysis developments presented in this work.

Electron microscopy. When seeking for sub-nanometre resolutions or imaging the entire cell environment, rather than selected fluorescent regions, biologists often utilize an electron microscope (EM) as an alternative to confocal microscopy [33]. Instead of light, electron microscopes use an electron gun to emit a beam of electrons which have a much shorter wavelength than visible light, allowing them to image structures at the atomic level. The electrons are then transmitted toward the sample with a series of magnetic lenses. Electrons are then either transmitted through the sample or remitted by the sample surface. If the receptor detects transmitted electrons, then the microscope will be called Transmission Electron Microscope (TEM) or, if the receptor detects remitted electrons, Scanning Electron Microscope (SEM) (Figure 1-6). For cell imaging, electron microscopes allow biologists to image tiny structures such as nucleus double membranes or nuclear pores that are extremely difficult to detect with confocal microscopes. The main drawback of electron microscopy is the imposed stringent conditioning of the samples before imaging. The sample preparation might involve chemical fixation, dehydration, resin embedding, and slicing into ultra-thin sections. Additionally, during imaging, the sample must be placed in a high-vacuum environment. All of these steps might irreversibly damage the sample. Another drawback of electron microscopy is the acquisition time, especially for 3D images, that might last as long as a week. A special type of electron microscope called Serial Block Face-SEM (SBF-SEM) is present in Oxford Brookes University. This microscope creates 3D images by, first, capturing a 2D image of the surface of the sample, then automatically slicing a tiny layer of it and capturing the newly revealed surface. This process is repeated until reaching a sufficiently deep layer and the resulting set of 2D images are finally stacked together to obtain a 3D image. This time-consuming process can create an extremely detailed image with a computer memory size often exceeding 200 gigabytes. Contrary to confocal images which display bright regions only where the fluorophore is located, the rest of it being black, electron

microscope produces grayscale images with a lower contrast where almost each molecule in the sample is detected with a different grey intensity. The resulting image thus contains a lot of regions which must be sorted to extract relevant information. This is one of the main limitations of the image analysis tools previously developed by the teams involved in this project which cannot be applied on EM image.

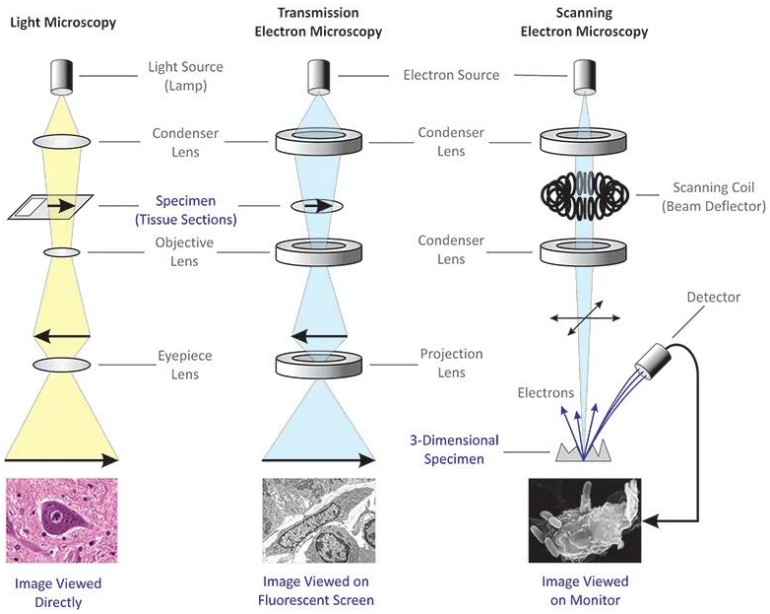


Figure 1-6 – **Light microscopy (*left*) versus TEM (*middle*) versus SEM (*right*).** TEM captures an image by detected the electrons transmitted through the sample while SEM captures remitted electrons. Illustration taken from [34].

Medical imaging. In the realm of 3D medical imaging, analogous difficulties arise as those encountered with electron microscopy images. 3D medical images, typically portraying organs or other macroscopic structures within the human body, are grayscale representations. Each location within these images is assigned a unique grayscale value, enabling distinction between objects of interest and the background. However, this results in a surfeit of information that must be meticulously processed in order to extract the relevant information. Probably due to their large number and to their practical application, medical images have been the objects of many software developments. Many online challenges have been created to serve the enhancement of medical image processing and the number of new ones constantly increases. Most of them are listed on https://grand-challenge.org/ website. Among the long list of existing medical imaging techniques, work is this project will mainly focus on Computed Tomography (CT-scan) and Magnetic Resonance Imaging (MRI) as they are the two

most common imaging techniques on this website (Figure 1-7). Even though the developments in this work will be tested on only these two imaging techniques, they should be applicable to other ones such as ultrasounds or Positron Emission Tomography (PET). CT-scans use a rotating X-ray scanner to capture multiple images from different angles. The set of resulting 2D images are then numerically combined to form a 3D image of the body parts being scanned. MRI uses a powerful magnetic field to align protons of hydrogens atoms contained in the body tissue. Radio waves are then sent through the body, which causes these protons to absorb the energy and temporarily move out of alignment. When the radio waves are turned off, the hydrogen atoms release the absorbed energy, and this release is detected by the scanner. Like CT-scan, MRI relies on numerical post-processing to compile the set of generated images into one 3D image. Although the primary focus of this work is microscopy, the similarities between electron microscope images and medical imaging prompt to explore developments in the latter field. It will here be demonstrated that tools developed for 3D medical images can be applied to confocal and electron microscopy. Furthermore, the software developed in this work has several applications in medical imaging.

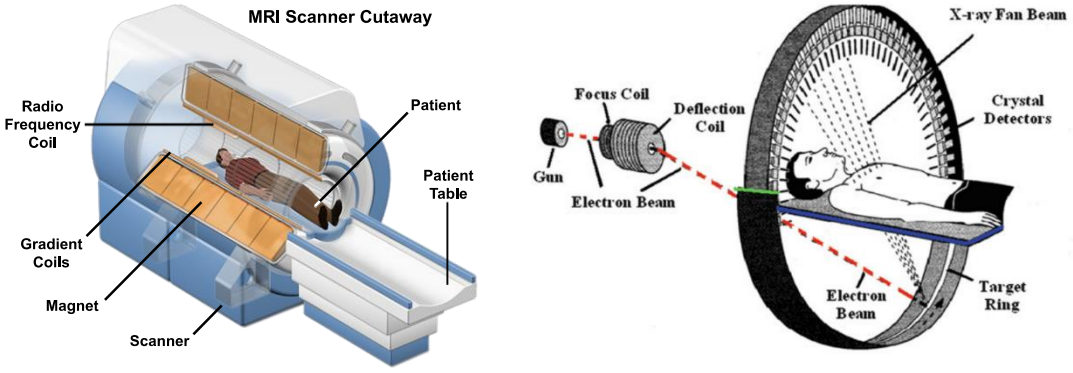


Figure 1-7 – **MRI (*left*) versus CT-scan (*right*).** MRI captures the photons released by hydrogen atoms of the body after a radio waves impulse while CT-scan relies on X-ray. 3D Images from both modalities will be analysed in this thesis. Illustrations taken from [35] and [36].

1.2.3 Digital image processing and analysis

What is a 3D image? An image is a visual representation of a bidimensional (2D) or tridimensional (3D) object, scene, or phenomenon. Since modern computers operate only using bits, 0 or 1 values, continuous visual data must be discretized before being

stored. To accomplish this, the captured space is divided into a 2D or a 3D grid. Each cell in the grid is assigned with one or more discrete values that represent the intensity of the signals within that cell. There is thus a double sampling, first in space and then in intensity. The grid is called a *digital image*, which will be abbreviated in this thesis to *image*, and a grid cell is called a *picture element*, which will be abbreviated to *pixel* for 2D images or *voxel* for 3D images. In this configuration, there are several parameters that must be defined if performing measurements is intended in the image:

- **The image size** is the number of picture elements in each dimension of the image. For instance, in 2D, the term Full HD is commonly used to qualify images with a grid of 1920 by 1080 pixels. In 3D, typical images can have 512 by 512 by 128 voxels. The number of picture elements is directly linked to the size of the image in the computer storage. In the previous 2D example, the number of pixels to store is $1920 \times 1080 = 2\,073\,600$. For the 3D image, the number of voxels is $512 \times 512 \times 128 = 33\,554\,432$, which is 16 times bigger than the 2D image. The commonly bigger size of 3D images is an important constraint both to store and to analyse them.
- **The number of channels** is the number of intensity values stored for each picture element. The channel dimension represents the last dimension of the image. Indeed, 2D images have, in reality, 3 dimensions and 3D images have 4. The number of channels in standard 2D images is typically 3: one channel for red intensities, one for green intensities, and one for blue intensities. In the previous Full HD example, an image will thus have an actual dimension of 1920 by 1080 by 3. In medical imaging, CT-scan images usually contain only one channel of grey values and MRI images are often multimodal, meaning that they have several channels. For example, the BraTS 2020 dataset [37] contains 3D images of brains with 4 channels (T1, T1-weighted, T2-weighted, and T2-FLAIR). For microscopy images, biologists typically use several fluorescent proteins which emit photons with different wavelengths. Each wavelength is captured independently and stored in a different channel in the resulting image. A microscope image can thus have potentially tens of channels, but large numbers of channels rarely occur, and typical numbers of channels stay below 5.

- **The spatial sampling (or *spacing*)** is the spatial size of each picture element. Each picture element represents a physical region in 3D space and this region has a certain height, width and, eventually, depth. When taking pictures with a standard camera, a 2D projection of the 3D objects is captured. Even if the final image looks "real", this projection step causes the objects to be deformed and any measurement done in this type of image must take it into account. Images captured with microscopes, or any medical imaging technique usually do not have such deformations and thus facilitate measures of the objects they represent. In biomedical images, there is a direct link between the number of picture elements and the size of the object. To perform one measure, a user can simply count the number of picture elements in the measure and multiply it by the size in meters of each picture element. Due to some technical constraints, 3D images often do not have a similar size in all 3 dimensions. Typically, for a 3D microscopy image the spatial size of each voxel could be $0.1\ \mu m \times 0.1\ \mu m \times 0.2\ \mu m$, which means that the depth of the image is less sampled than the other two dimensions. This property is called *anisotropy*, and this results in an image that looks "flat" in the depth dimension.
- **The intensity sampling** is the number of possible intensity values of each picture elements. For one channel and one picture element, there is one sampled intensity value. The number of possible intensity values range between only 2, the resulting image is called "binary", and $2^{32} = 4\ 294\ 967\ 296$. For 2D images, the standard is to sample with $2^8 = 256$ values. Why such a power of 2 sampling? Because computers can only store bits, 0 or 1, which means that a base 10 numbers, such as 87, which have 2 digits, "8" and "7", will be encoded by the computer in a base 2 number, 1010111 which has 7 digits. For computer storage reason, it is often appropriate to store intensity values with only base 2 numbers with a maximum of 8 digits. The resulting image has only $2^8 = 256$ possible values for each intensity. In such a configuration, 0 represents the absence of signal and 255 the maximum signal intensity. This type of image is called 8-bits. Although 8-bits images are sufficient for standard 2D images because the human eye cannot distinguish colours with a lower sampling, for medical and microscopy images the signal intensity variations are sometimes so subtle and important that the number of

sampled intensity values must be increased. Typical biomedical images can be 16-bits or 32-bits. The choice of intensity sampling also affects the storage space required for the image; a higher sampling rate leads to a larger image size.

These four characteristics are determinant in the choice of the method to store, visualize and analyse the images. For example, anisotropic images will often require some post-processing to "unfold" the represented object. However, even though important, focusing on these characteristics only is not sufficient to properly analyse the images, as the nature of the represented objects is probably even more important. Although image characteristics are alike, methods developed to analyse objects of the macroscopic world are not always applicable to the world of biomedical objects. The laws of physics change between these two worlds and, consequently, the morphology of the objects as well. The large variety of human postures in 2D images is not similar to the finer variety of shapes in a kidney or a nucleus image.

Development of image analysis methods. To decipher the subtle and specific variations in biomedical images is one of the main objectives of this work. NucleusJ [29], the ImageJ/Fiji plugin previously developed by the teams involved in this project, is specific to confocal images of nuclei. Initially developed for plant nucleus taken in wide field images, it has been shown to be applicable to other types of nuclei such as sperm nuclei. NucleusJ basic principle is based on applying a threshold to the image intensities. The resulting image, a so-called segmentation mask, is black and white, where white voxels are located on the nuclei and black voxels represent the background. The "ideal" threshold is found via the Otsu method [38] and a post-processing based on the Gift-wrapping method [39] is applied to fill the eventual holes in the segmentation mask. NODeJ, another ImageJ/Fiji plugin developed by the biologist teams, can then be applied to segment the chromocentres inside each nucleus. NODeJ combines a modified Laplacian filter with a thresholding method [40] to segment each chromocentre. Both tools come handy in analysing 3D nuclear images, mainly images of plant nuclei, as they are quite accurate, fast and do not require any manual annotations. However, they are both limited to confocal images and to nucleus and chromocentres segmentation. Both tools would be useless to segment nuclei in electron microscope images or to segment epithelium cell borders in a confocal image. A simple intensity threshold cannot be applied in electron microscope images and the

Gift-wrapping method cannot be used to fill the eventual gaps in the segmentation of epithelium cell borders. Two approaches can be taken when facing new imaging challenges: either create customized solutions for each problem or develop a more versatile method that can adapt to any new problem as long as there is a set of manual annotations. The former requires significant engineering efforts, while the latter only requires novel annotations. In this work, the second approach will be pursued. It is often called *data-driven* approach, and it will leverage recent advances in a novel class of computer science methods called deep learning.

Deep learning. The emergence of deep learning and, more generally, machine learning has been inspired by how humans learn. If you intend to teach someone to do a certain task, let say a segmentation task, one of the easiest ways is to show both the original image and the expected results, the handmade annotation. Links will then be created in the neural network of the trainee between the original input and the expected output. An artificial neural network works the same way. An image is input to a series of operations, which successively transform the image to reach the expected output format. The predicted output is compared to the expected output with a similarity score. Each operation has a set of parameters that are then automatically adjusted based on this score. The higher the similarity score is, the less the parameters will be adjusted. This process is repeated until the artificial neural network reaches a sufficiently good score. In this configuration, it is assumed the existence of a "good" artificial neural network and that only novel pairs of original images and manual annotations must be provided to adapt this network to a new task. As finding such a "good" artificial neural network can be extremely difficult, this apparently simple approach is the basic principle of many recent developments in machine learning. It was only in 2011 [41] that the first artificial neural network, based on a special type of operation called *convolution*, reached human-like performance on a basic task, image classification. This performance was made possible thanks to the emergence of powerful Graphical Processing Units (GPUs) originally designed for video game rendering. Deep learning has become drastically popular among image analysists but, yet, has often not reached the end users, such as biologists or radiologists. In this work will be first presented an attempt to understand why there is such as gap between deep learning developers and biologists and then light will be shed on a newly developed tool, named *biom3d*, that

go beyond the proof of concept of most of the available deep learning methods and that can be user-friendly for both the end users and the developers.

Reducing the need for manual annotations. Despite the impressive flexibility and performance of deep learning methods, their effectiveness heavily relies on the availability of extensive manual annotations. This dependency is called *supervision*, regarding how deep learning models are *supervised* by the man-made data. In domains like biology and medicine, where expert-level annotations are necessary, the time and cost involved in generating such large datasets often exceed practical limits. Consequently, this becomes a significant impediment to the widespread adoption of deep learning methods in these fields. In response to this emerging challenge, several techniques have emerged to address the limitations associated with supervised deep learning methods. These techniques, including active learning, generative methods, weakly supervised learning, and self-supervised learning, have garnered significant attention in recent years. More details about each these methods will be provided in Section 2.3.2. While these methods have proven to be helpful on certain datasets such as ImageNet [42] for 2D image classification or COCO [43] for 2D image segmentation, they are yet to prove their effectiveness on a broader scale. This work will delve deeper into the potential applications of self-supervised learning for 3D biomedical images. This choice is motivated by the recent promising advancements in this field and the specific characteristics of biomedical datasets. Self-supervised learning has shown particular suitability in scenarios where a vast amount of unannotated data is available alongside a limited number of annotated samples. The rapid progress in biomedical imaging technologies has enabled biologists and radiologists to generate a wealth of high-quality 3D images with relative ease and efficiency. However, harnessing the full potential of this abundant unlabelled data for training artificial neural networks can be a challenging task. Self-supervised learning presents a promising approach to leverage this surplus of unannotated data by pretraining models, using pretext tasks that exploit inherent structures and patterns within the data, before training them on annotated datasets.

1.3 Outline and contributions

The initial aims of thesis were to develop an AI method to segment nuclei in microscopy images and to compare it with existing methods. The structure of the thesis follows almost chronologically the evolution of questions raised along the way.

In Chapter 2 is first overviewed the promises of deep learning methods for 3D imaging as presented in the literature. A deeper exploration will define criterion to find reusable methods and a set of such methods for 3D nuclear segmentation will be benchmarked.

In Chapter 3, Biom3d is introduced, an easy-to-use and modular tool, developed during this PhD, that automatically configures the complex workflow of deep learning model training. Biom3d default configuration is for volumetric segmentation of multiple classes of objects in multi-channel images, such as microscopy and medical images. In segmentation mode, Biom3d is thus able to reach state-of-the-art performance on multiple segmentation challenges and, in particular, on 3D nucleus segmentation. The new software development philosophy adopted during Biom3d creation is one of the major achievements of this thesis project and has permitted Biom3d to satisfy the expectations of a continuum of users, from Non-Programmers to Deep Learning Developers.

Finally, in Chapter 4 is presented a series of experiments aiming at adapting existing self-supervised learning methods to 3D bioimages. Novel contributions based on the Triplet and the Arcface losses will demonstrate promising results. All these adaptations and new contributions were made possible by the Biom3d framework.

Chapter 2

Related work, review, and benchmarking

Code is like humour.
When you have to explain it, it's bad.

- Cory House-

2.1 Classical computer vision methods for 3D segmentation of bio-images

Segmentation, as a computer vision problem, has been closely intertwined with advancements in imaging techniques. Its early mentions can be traced back to the 1970s [44]–[46], where researchers already recognized its relevance in biological and medical applications. This Subsection introduces classical computer vision methods for image segmentation that predate the emergence of artificial intelligence-based approaches. It is important to note that this introduction is not exhaustive, as the primary focus of this thesis lies on the latter, AI-driven techniques.

2.1.1 Classical computer vision approaches

Classical computer vision approaches to segmentation involved the application of various techniques such as thresholding, clustering, region growing, and edge detection. These methods relied on predefined rules and heuristics to identify and delineate objects or regions of interest within an image. They aimed to exploit low-level image characteristics such as intensity, colour, texture, and spatial relationships.

Thresholding, for instance, involves setting a pixel intensity value to separate foreground and background regions. Finding the best threshold automatically usually involves the computation of the histogram of intensity values in the image. For example, Otsu's method [38], probably one of the most widely used thresholding method, hypothesises the existence of two probability distributions in the histogram and the best threshold is the one that minimizes their intra-class variance of each distribution. **Clustering methods** seek to group picture elements into K clusters based on their intensity and location. Neighbouring picture elements of similar intensities will thus be assigned to the same cluster with an algorithm such as K-means [47]. The

number K of clusters can be determined randomly or with a heuristic [48]. **Region growing algorithms** aim to group picture elements with similar characteristics, such as the aforementioned clusters, into larger coherent regions. This category of algorithms may involve techniques such as graph partitioning with, for instance, Markov-random fields [49] or the watershed transformation [31] which considers the gradient of the intensities as a topographic surface where the region boundaries are defined by the picture elements with the highest intensity value. **Edge detection methods** focus on identifying abrupt changes in picture element intensity to locate object boundaries. Once edges are located, regions can then be deduced. While local approaches such as gradient-based methods can often be efficient, they might often miss edges in noisy conditions. More global approaches might thus be more appropriate such as methods based on the minimization of an energy function as active contours [50], fitting a user-defined contour to the image edges, or variational methods [51], segmenting objects without clearly defined boundaries.

These classical techniques play a crucial role in early image segmentation research and lay the foundation for subsequent developments. However, they often face challenges in handling complex image structures, variations in lighting conditions, noise, and the presence of overlapping or touching objects. They heavily rely on handcrafted features and predefined rules, making them less adaptable to diverse and complex real-world scenarios. The emergence of artificial intelligence, particularly deep learning, has revolutionized image segmentation by enabling the development of data-driven approaches which automatically learn and extract meaningful features from images for segmentation tasks. They have shown great promise in addressing the limitations of classical methods and achieving state-of-the-art performance in various applications. While this thesis primarily focuses on artificial intelligence-based segmentation techniques, it is important to recognize and appreciate the contributions made by classical computer vision methods.

2.1.2 NucleusJ and NODeJ

Preceding this thesis, the work of the PhD students Axel Poulet and Tristan Dubos has focused on the development of two ImageJ/Fiji plugins, NucleusJ and NODeJ, to segment plant nuclei in confocal 3D images. NucleusJ [29] (Figure 2-1) takes as input a wide-field 3D image and output a Comma Separated Value (CSV) file containing, for each nucleus in the image, a list of 14 characteristics such as the volume, the

surface, the sphericity, or the median intensity value. To get these values, it does a series of transformations on the images based on adaptations of classical computer vision algorithms. An Otsu's thresholding combined with a connected component algorithm simultaneously find large volumes, the nuclei, and remove small volumes, the background noises. Each nucleus can thus be cropped and isolated. Isolated nuclei are segmented again with a modified version of the Otsu thresholding. This segmentation is improved by a Gift-wrapping algorithm which fills holes in the segmentation results caused, for instance, by the nucleolus not been marked by the DAPI staining. The final segmentation result is used to compute the 14 characteristics. To retrieve even more information, NODeJ plugin [52] (Figure 2-1) proceeds to a subsequent chromocentre segmentation starting from the segmentation result of NucleusJ. First, the gradient of the image is obtained by subtracting from each picture element the intensities of its neighbours. Then a threshold is applied to segment the chromocentres. The resulting chromocentre segmentation can finally be used to compute more characteristics such as the number and location of each chromocentre. When phenotyping plant nuclei, NucleusJ and NODeJ have, for instance, demonstrated significance differences between wild type nuclei and KAKU4 and CRWN mutants.

NucleusJ and NODeJ typically illustrate the constant need of adaptation required by the classical computer vision methods which would fail if directly applied. However, despite this lack of flexibility, this type of method is still developed as it can be very efficient and performant in some cases, when properly configured. Methods such as GIANI [53], for mouse embryo image analysis, or TRUEFAD (https://github.com/AurBrun/TRUEFAD), for myotube segmentation, are other examples of recent computer vision-based tools. Artificial intelligence-based methods have indeed the potential of being even more performant but often at the cost of requiring extensive efforts, skills, time, and expensive pieces of hardware or software. It will be seen later in this work, that, in some cases, classical computer vision methods can overpass AI approaches (see Section 2.2.4).

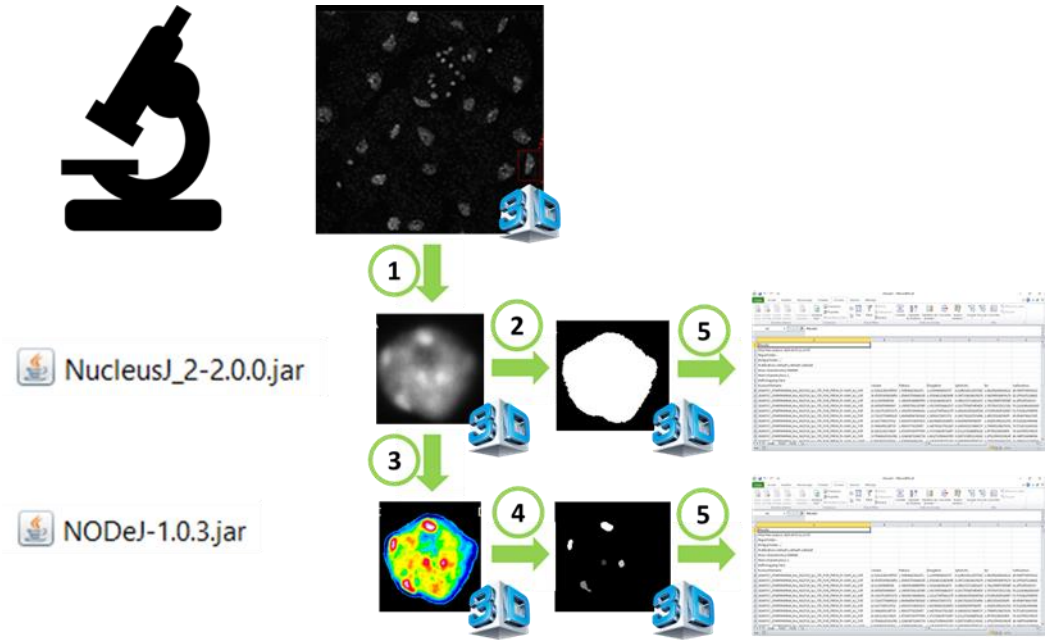


Figure 2-1 – **Nucleus*J*/NODe*J* workflow: from 3D wide-field images to CSV file.** (1) Crop of nuclei into individual files. (2) Generation of nucleus masks. (3) Intensity gradient of nucleus content. (4) Generation of chromocentre masks. (5) Generation of csv files with parameters describing nucleus morphology and chromocentres.

2.2 Supervised deep learning methods

The main core of artificial intelligence methods is now probably deep learning, which could be viewed as a convergent evolution that has quickly dominated. Indeed, preceding deep learning methods, AI field have been the realm of a Cambrian-like explosion of novel methods every year. Discoveries in biology and medicine has deeply influenced these evolutions first by inspiring novel ideas and offering novel insights on how natural neural networks work and, second, by being an important case of study regarding the increasing amount of new data and challenges it provides. This Section mainly focuses on supervised deep learning methods, still one of the most chosen approaches regarding deep learning model training. It will first introduce deep learning concepts and functioning and then overview the large panel of applications for biology and medicine. It will be concluded by a comparative analysis of deep learning methods for 3D nucleus segmentation, an important part of the work achieved during this thesis.

2.2.1 General background

Even though some may distinguish them, the terms *machine learning* (ML) and *artificial intelligence* (AI) will be used indistinctly in this thesis. The latter has been chosen so far for it to be often preferred by mainstream media and commonly used, the former is probably more accurate and generally chosen by experts. This Subsection will develop the general introduction of Chapter 1 about deep learning by enriching it with a mathematical background.

Machine learning. AI and ML both refer to the art of programming a set of parameterized operations for a computer to achieve a certain task. This set is often named *model* and the operations are named *layers*. A task is usually defined by a set of inputs, such as images, that will be noted x, and a set of expected outputs, such as categories, for instance "wild-type nucleus" and "mutant nucleus" (Figure 2-2-A), that will be noted y. The term *learning* implies that some of the layers, noted f_w, have parameters, noted w, that will be adjusted automatically so the predicted output, noted $\hat{y} = f_w(x)$, match the expected output y, also called *ground truth*. To do so, the *loss function*, noted L_w, measures the difference between the output $\hat{y}$ and the ground truth y (Figure 2-2-B). The goal is to minimize this difference. The *optimizer*, noted g_α, then adjusts the parameters w to new values w' depending on the loss value $L_w(\hat{y}, y)$: $w' = g_\alpha(w, L_w(\hat{y}, y))$. This step is named *training*. The definition of loss function L_w and the optimizer g_α is often dependent on the task definition. However, there are certain standards, particularly for the optimizer. Probably, the most used optimizer is the *Stochastic Gradient Descent*: $g_\alpha\big(w, L_w(\hat{y}, y)\big) = w - \alpha \frac{dL_w}{dw}$. In this case, the operation parameters are adjusted by subtraction of the gradient of the loss function $\frac{dL_w}{dw}$. The sign of gradient $\frac{dL_w}{dw}$, positive or negative, represents the variations of the loss function L_w, increasing or decreasing, if the operation parameters w were slightly increased or decreased. Subtracting the operation parameter w with the gradient $\frac{dL_w}{dw}$ means that, if the gradient is positive, w will be decreased, thus moving toward a loss decrease and, similarly, if the gradient is negative, w will be increased, thus moving again toward a loss decrease. It is important to note that there are two different types of parameters here. First, the *trainable* parameters w are the operation parameters adjusted automatically by the optimizer and, second, the so-called *hyper-parameters* must be manually defined, such as the learning rate α in the optimizer, controlling the

update intensity of the operation parameters. Other examples of hyper-parameters are the number and the type of each layer (if the model has several), the number of data given simultaneously to the model during training (*batch size*), or the number of times the model sees the entire dataset (*epoch*). Defining properly the hyper-parameters is often regarded as one of the main challenges of machine learning. It will be shown later in this thesis (see Chapter 3) that hyper-parameter configuration for 3D images is even more challenging due their sizes. Biom3d framework, one of the main contributions of this thesis, tackles this challenge.

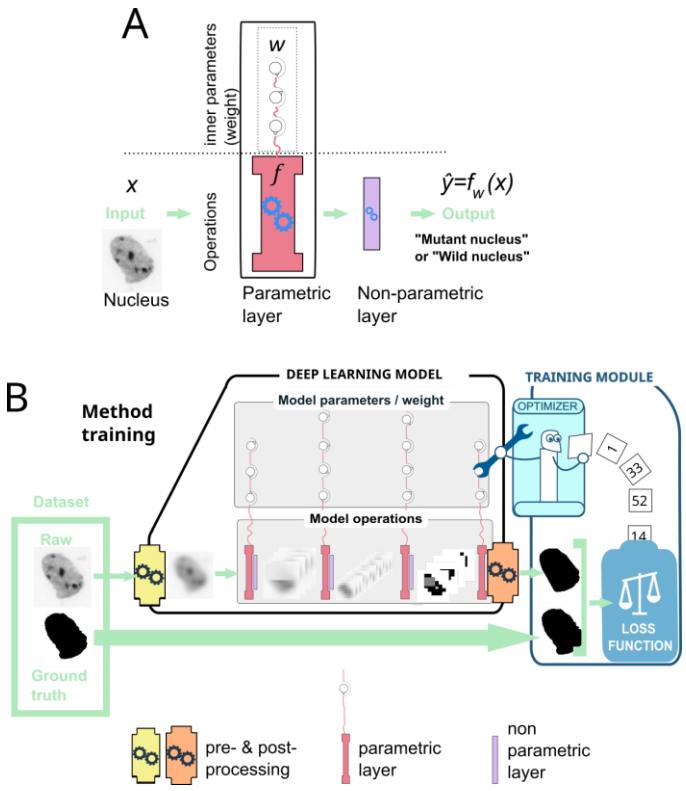


Figure 2-2 – **Basic principles of machine learning. (A) A basic units of machine learning model**. A nucleus image (*left*) is input to a parametrized operation (*red*) and to a non-parametrized operation (*purple*). The output is here a *class*, either "Mutant" or "wild-type" nucleus. **(B) Training of a deep learning model.** A raw input of a nucleus (*left*) is pre-processed (*yellow box*) and fed to the deep learning model (*middle grey box*) which transforms it, via a series of operations, into an expected output, here a segmentation. After a post-processing (*orange box*) the expected output is compared to the ground truth segmentation with a loss function (*blue box*). The optimizer (*bot*) will adjust the parameters of the deep learning model depending on this loss. Illustration taken from [40].

Machine learning versus Deep learning. The distinction between classical machine learning models and deep learning models lies in the presence of multiple successive parametric layers in the latter. In deep learning models, if two or more consecutive layers with learnable parameters are present, the model is considered *deep* (Figure 2-2-B). This stacked structure of layers offers two key advantages compared to single-layer models. Firstly, deep learning models can achieve comparable performance to single-layer models with a seemingly lower number of trainable parameters. This parameter efficiency is attributed to the hierarchical nature of deep models, where each layer learns and refines representations of increasing complexity. By leveraging the hierarchical structure, deep models can efficiently capture and encode information from the input data, leading to better performance with fewer parameters. Secondly, the layered structure of deep models enables them to extract more complex and abstract features. While shallow models may be limited to detecting simple features like edges or corners, deep models have the capacity to learn hierarchical representations, allowing them to recognize more intricate patterns and concepts. For example, a deep model can learn to identify not only the individual components of a dog, such as edges or corners, but also the overall shape and appearance of the complete dog (Figure 2-5).

Short history of deep learning for vision. The use of deep learning models can be traced back to the early 1980s with, for instance, the work of Kunihiko Fukushima on a model called *Neocognitron* [54]. Fukushima introduced a special type of model, later called Convolutional Neural Network (CNN or ConvNets) inspired by works on the visual cortex [55]. Although this method using brain like networks worked, it was very slow to train. It was only in 2011 that deep learning reached performance exceeding that of a human experimenter on visual task problems [41] owing to their implementation on fast Graphical Processing Units (GPUs) and large-scale handmade datasets. CNNs were made famous in 2012 by winning several well-known competitions in the field including ImageNet 2012 [42] and MICCAI 2013 Grand Challenge [56] (Figure 2-3). Afterwards, deep learning methods usage shortly exploded in communities of biological and medical image analysts [57], most of which deal with image segmentation relying on the U-Net model [58].

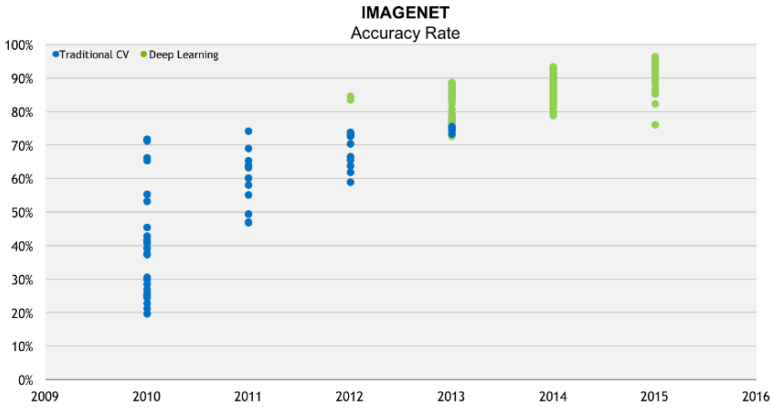


Figure 2-3 - **ImageNet classification challenge results between 2010 and 2015.** A shift occurred in 2012 with the appearance of deep learning methods (*green*) rapidly replacing the traditional computer vision approaches (*blue*). Illustration adapted from [59] from Jensen Huang talk at CES 2016.

Deep learning operations. The base parametric layer of most of deep learning methods for visual task is called *convolution* and consists in a weighted sum of a small set of pixels that belong to a small sub-window (usually 3×3 pixels or 3×3×3 voxels) within the input image (Figure 2-4-A). The resulting digit is stored in an output array. This weighted sum is reiterated using the same set of weights, called *kernel*, but applied to a small sub-window shifted by one pixel. The resulting array is completed by browsing through the entire input image. Mathematically, this operation can be written as follows for 2D images:

$$Image_{output}^{(c)}(h,w) = \sum_{i=1}^{N} \sum_{j=-d}^{d} \sum_{k=-d}^{d} Kernel^{(c)}(i,j,k) * Image_{input}^{(i)}(h+j, w+k)$$

where $Image_{x}^{(c)}(h,w)$ is the pixel value of the image at (h,w,c), h being the height coordinate, w being the width coordinate and c being the channel coordinate, N represents the number of channels in the input image and d the kernel size. This operation can easily be adapted to 3D images by addition of one dimension.

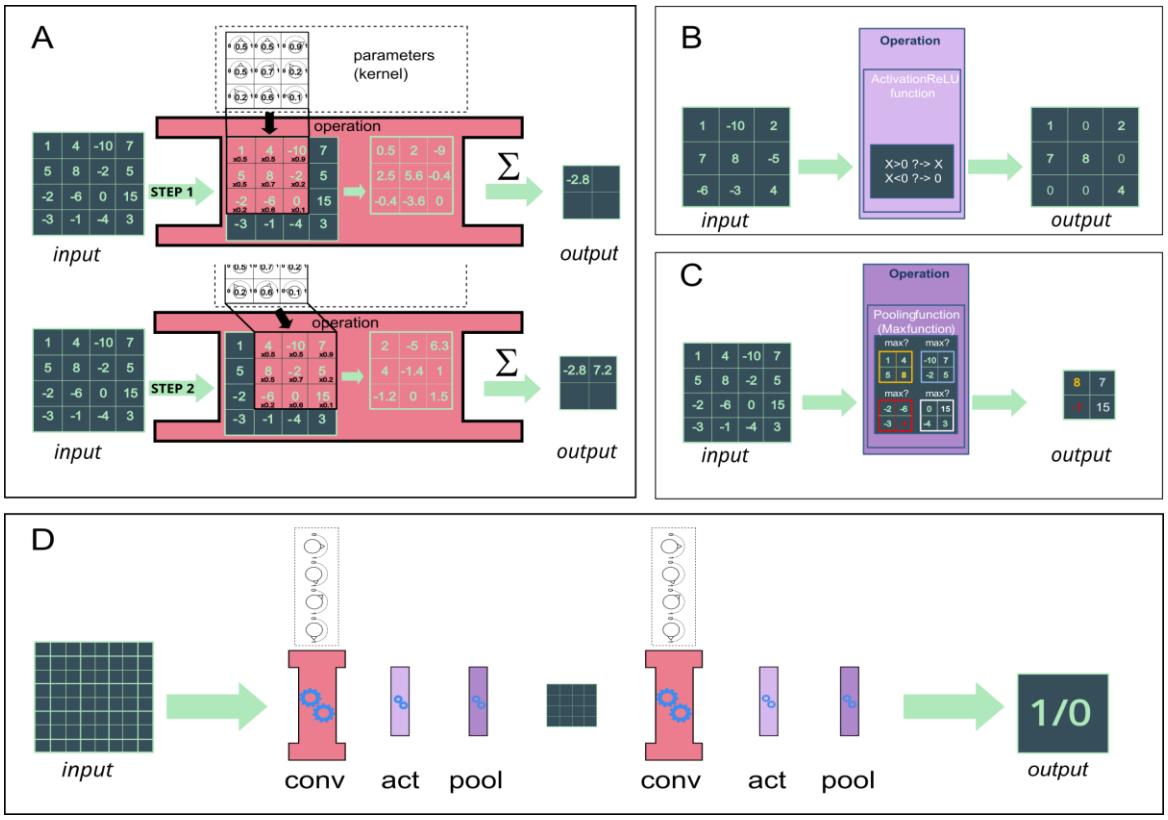


Figure 2-4 – **Deep learning layers and a minimalist model for image analysis. (A) A convolution layer** is the weighted sum between a set of parameters (*kernel*) and a small sub-window within the input image (*left, dark green*). This gives a single digit stored in the output image (*right, dark green*). The small sub-window is then slid to the left (*step 2*) and the process is reiterated. Illustration taken from [40]. **(B) An activation layer**, here a Rectified Linear Unit (ReLU), applies a threshold to the input image, transforming all negative values into zeros. **(C) A pooling layer**, here a MaxPooling, reduce the input image size by a factor of two, slicing the image in 2×2 sections, selecting the maximum values of each section. **(D) A minimalist deep learning model** classifies the input image into two classes. Illustration taken from [40].

Often placed after convolution layers, the *activation layer* (Figure 2-4-B) are non-parametric layers applying a threshold to their input. Some values in the output will be turned to zero forcing the model to make a focus on specific region and patterns in the image (Figure 2-5). Additionally, to encourage the model to extract hierarchical features from the data with increasingly complex shapes, the output of each activation layer is compacted by being shrunken down, usually by a factor of 2, with a *pooling layer* (Figure 2-4-C). A succession of sets of convolution, activation and pooling layers

can form a deep learning model (Figure 2-4-D). Figure 2-4-D shows a minimalist example where the input image is transformed into a single binary digit. This simple model can be used to classify images into two categories. This task is called *image classification* and often consider as one of the simplest in image processing. The ImageNet challenge [42] is, for instance, an image classification challenge which involves classifying more than a million images into one thousand classes. In deep learning, classification models are considered as *backbones* onto which additional layers can be attached to address other tasks such as object detection or image segmentation. These additional layers build upon the learned representations of the backbone model to perform more specific and intricate tasks. Common backbones include CNNs such as VGG [60], ResNet [61], EfficientNet [62] or Vision Transformers [63], [64].

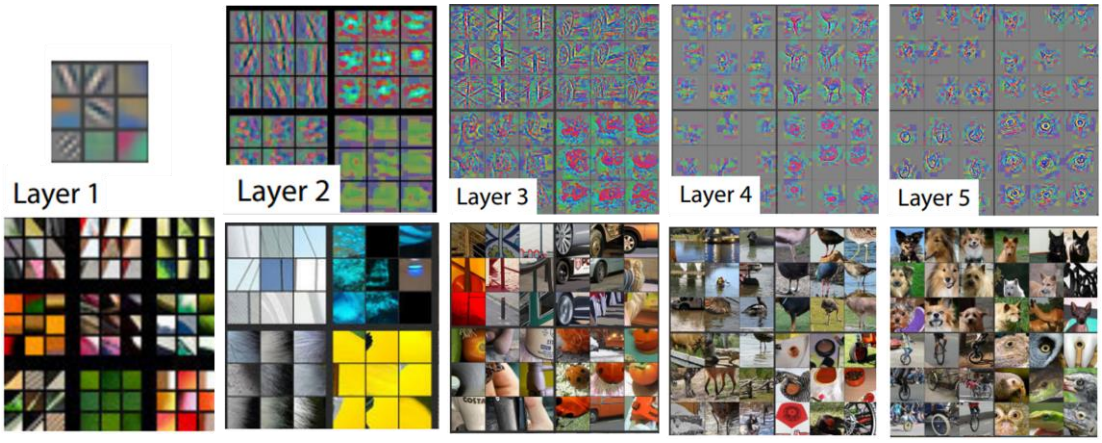


Figure 2-5 – **What does the deep learning model "sees"?** Visualization of the output of some activation layers at different depth in the deep learning model (*top row*). These views have been reconstructed with a deconvolution model (inverted convolution) using the samples displayed in the bottom line. It can be noted that the deeper the layer is in the model, the more complex the patterns of attention are, shallow layers focusing on simple edges and deep layers focusing on complete objects. Illustration adapted from [65].

Vision Transformer. The Vision Transformer (ViT) models have recently set a significant milestone in deep learning for visual tasks. This type of model originate from Natural Language Processing [66] (Figure 2-6-A) and have achieved remarkable performance on well-known computer vision challenges such as ImageNet for image classification [42] (up-to-date of the state-of-the-art available here:

https://paperswithcode.com/sota/image-classification-on-imagenet) or COCO for object detection and image segmentation [43] (https://paperswithcode.com/dataset/coco). The main parametric layer of these model is not the convolution layer but the *attention layer*. The introduction of attention mechanisms aims to reduce the "inductive bias" imposed by convolutions. Indeed, convolutional layers assume that objects or parts of objects within an image are in a relative proximity to each other. However, this assumption may not always hold in certain scenarios, such as object occlusions. Attention layers provide a more flexible and adaptive approach to capture long-range dependencies and relationships between image regions. Attention mechanisms assign different weights to different regions in the input, allowing the model to focus on more informative regions and disregard irrelevant ones. More formally, the input image x is first split into smaller squares $x_1, x_2, \dots, x_n$, each being a vector with a fixed size (Figure 2-6-B). This sequence is then transformed into another sequence $a_1, a_2, \dots, a_n$ with the attention mechanism defined by the following formula:

$$a_i = \sum_{j=1}^{n} softmax\left(\frac{(W_q x_i)^T (W_k x_j)}{\sqrt{d_q}}\right) W_v x_j$$

where W_q, W_k and W_v are three matrices of trainable parameters, d_q is the number of rows of W_q and $z \longmapsto softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$. The weights of this weighted sum are determined by the $softmax$ function which tends to output either values close to one or values close to zero. For instance, if the model finds a useful link between regions x_1 and x_2 of the image, then this mutual information could be merged into a_1 by setting the matrices W_q and W_k so that the $softmax$ function outputs one if $j = 2$ and zero otherwise. As many different links might exist between image regions, the attention layer contains many of these attention operations and is thus called *multi-head attention layer* (Figure 2-6-A). To leverage the power of deep learning, many of these multi-head attention layers are stacked together to create the final Vision Transformer model. It is worth noting that while attention layers have shown great promise in Vision Transformers, convolutional layers still play a vital role in many computer vision applications. Different architectures, such as hybrid models that combine both convolutional and attention layers, are also being explored to leverage the strengths of both approaches (Figure 2-6-C).

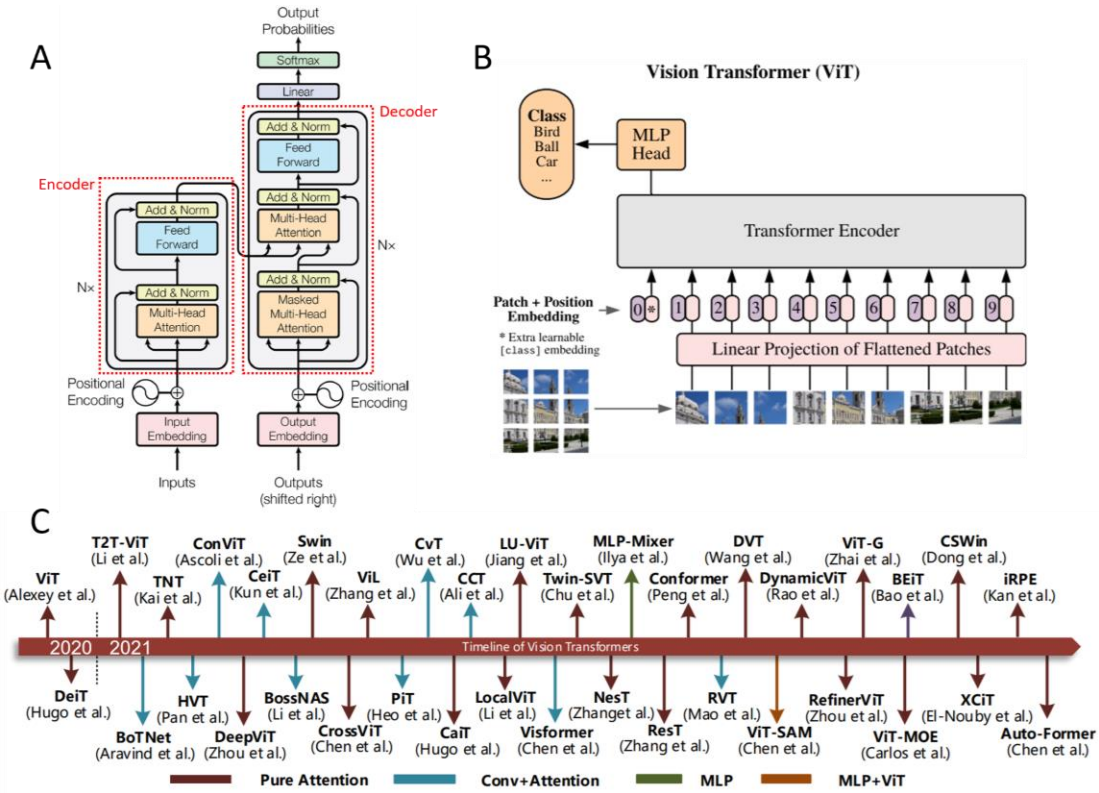


Figure 2-6 – **Vision Transformer. (A) The original transformer model** is designed for sequence analysis, originally textual sequences. Its main parametric layer is the multi-head attention layer (*orange*). Illustration adapted from [66]. **(B) The vision transformer** takes an image as input and transforms it into a sequence by splitting the image into small patches and flattening them into vectors. Illustration taken from [63]. **(C) The vision transformer evolution in 2021.** A large panel of derivative of the original transformer model appeared in 2021 quickly making it one of the most popular types of models. Illustration adapted from [67].

2.2.2 Applications in biology and medicine

To build upon this general background, the following Subsection will provide a quick and non-exhaustive overview of deep learning methods used in biology and medicine for image analysis. The choice of presenting a certain method here has been made regarding their popularity and availability. To valorise open-science contributions, the focus will exclusively be made on free and open-source tools that provides a Graphical User Interface for non-programmers. A distinction will be made between biological and medical tools. However, some tools are generic enough to be used for both applications such as Fiji/ImageJ [17] or 3D Slicer [12].

Biological applications. In matter of free and open-source tools for biological image analysis (Table 2-1), probably the most used ones are Fiji/ImageJ, Weka [8], ilastik [7] and napari [6], all of which including machine learning tools either by default or in form of plugins. Fiji/ImageJ and Weka are both coded in Java, the former being more popular for its large set of traditional computer vision tools and the latter for its collection of machine learning algorithms. Ilastik and napari are developed with Python, an increasingly popular language in the computer vision community. Ilastik main purpose is to segment, detect, and classify objects in 2D and 3D bioimages while napari is mostly a 3D viewer for bioimages. Napari has yet a highly flexible software architecture which ease both the installation and the development of plugins in Python. For instance, deep learning segmentation tools such as StarDist [68] (see Section 2.2.3) has been integrated into napari with an easy-to-use graphical user interface (development helped by MagicGUI https://github.com/pyapp-kit/magicgui).

As deep learning tools are often difficult to use due to the difficulty to find properly trained model, the website Bioimage.io offers to host a number of ready-to-use deep learning model in DeepImageJ, a plugin of Fiji/ImageJ [69]. It is important to note that a downloaded model from Bioimage.io must be applied on images that are like the ones used during model training or the output will be aberrant (Figure 2-7). To help train a deep learning model on novel images, ZeroCostDL4Mic website [70] (https://github.com/HenriquesLab/ZeroCostDL4Mic) hosts a set of online notebooks for bioimage segmentation, object detection, object classification and image denoising, by leveraging the computers of Google Colab.

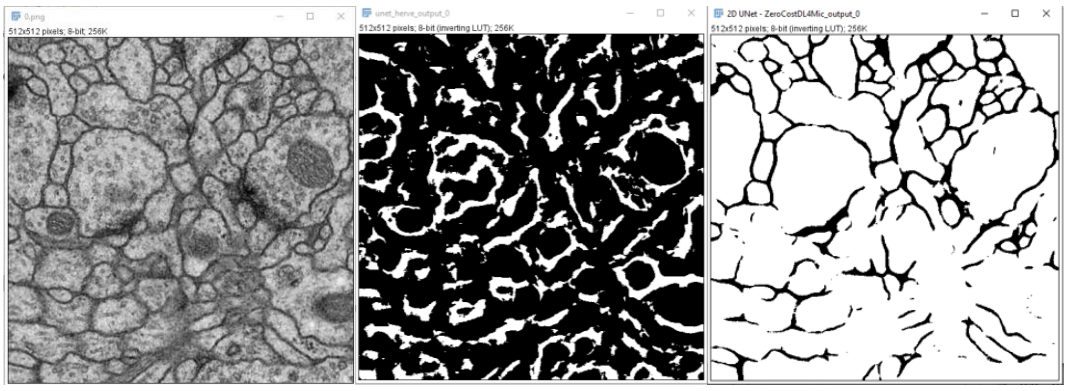


Figure 2-7 – **Segmentations done with DeepImageJ of membranes captures with an electron microscope.** The raw image (*left*) is segmented first by an inappropriate model (*middle*) and an appropriate model (*right*).

Besides the previous general-purpose pieces of software, some others are more modality-specialized such as 3Dmod [14] which focuses on 3D modelling of electron microscopy images or OpenOrganelle [71] for 3D organelle segmentation in the same modality. Others are more tasks specific, such as Cellpose [72], [73] for cell and nucleus segmentation, Mastodon and ELEPHANT [74] for cell lineage, 3DeeCellTracker [75] for cell tracking, or CSBDeep toolkit (http://csbdeep.bioimagecomputing.com/), including CARE [76], DenoisSeg [77] and Noise2Void [78], for confocal image denoising.

Table 2-1 – **Free and open-source programs for 3D microscopy image analysis.** Each program integrates a graphical user interface.

Software	Modality	Tasks	ML/DL	3D viewer	online/ offline	Ref.
Fiji/ImageJ	Generic	Computer vision	Plugins: ML and DL (DeepImageJ)	Yes, via plugin	offline	[17], [69]
Icy	Generic	Generic	Plugins: ML and DL	Yes	offline	[79]
napari	Generic	None by default	Plugins: ML and DL	Yes, by default	offline	[6]
Ilastik	Generic	Segmentation, detection, classification	ML	No, 2D sections only	offline	[7]
Weka	Generic	Segmentation, detection, classification	ML	No, 2D sections only	offline	[8]
Cellprofiler	Light microscopy	Cell image analysis	Plugins: ML and DL	No, 2D only	offline	[80]
Cellpose	Light microscopy	Cell and nucleus segmentation	DL	No, 2D only	offline and online	[72], [73]
ZeroCost-DL4Mic	Generic	Segmentation, detection, classification, denoising	DL	No	online	[70]
CSBDeep	Light microscopy	Image restoration (CARE, Noise2Void, DenoiSeg),	DL	No	offline	[68], [76]–[78]

		Nucleus Segmentation (StarDist)				
Imjoy	Generic	Generic	DL	No, 2D only	online	[81]
Mastodon	Light Microscopy	Cell tracking	ML and DL (ELEPHANT)	No, 2D sections only	offline	[82]
Open-Organelle	Electron Microscopy	Multi-organelle segmentation	DL	No, 2D sections only	offline	[71]
3dmod	Electron Microscopy	3D modelling	None	Yes	offline	[14]

Medical applications. Even though the main starting point of this thesis is biology, similarities shared between biological and medical images implies that applications for one type of image could potentially be applied to other. Table 2-2 showcases a non-exhaustive list of free and open-sources programs for medical applications. This list, partially extracted from [83], includes well-known applications such as 3D Slicer [12], ITK-SNAP [84] and MITK [85] which offer a wide range of functionalities, including image analysis, processing, visualization, tracking in 3D videos, and segmentation. Among these, 3D Slicer stands out for its versatility and extensive collection of extensions and plugins. Many of these extensions have been developed during challenges like Grand Challenges (https://grand-challenge.org/) or as a result of research presented at conferences such as ISBI (https://2023.biomedicalimaging.org/en/) or MICCAI (http://www.miccai.org/). However, these challenges and conferences prioritize the exploration of ideas and implementations of proof-of-concepts, which may not always result in fully operational and reusable applications. Although there has been a recent emphasis on code publication and sharing, the integration of these tools for practical applications is often overlooked, leading to a gap between published research and usable software. Surprisingly published code is not ever a reproducible code and, even less, a reusable code as is. Yet, some recent methods for medical image segmentation have made commendable efforts in this direction. Notable examples include nnU-Net [86],

TotalSegmentor [87], and Universal [88] Model for image segmentation, as well as nnDetection [89] for object detection.

Table 2-2 – **Free and open-source programs for 3D medical image analysis and visualization.** Each program integrates a graphical user interface with a 3D viewer. They are roughly classified by the number of features (and, consequently, by popularity) offered for medical image analysis.

Software	Modality	Task	Language	Ref.
3D Slicer	Generic: medical and biological	Generic: 3D, 3D+time, Processing, Analysis, Virtual Reality, Segmentation…	Python, C++, Matlab	[12]
ITK-SNAP	Generic	Generic	C++	[84]
MITK	Generic	Generic	C++	[85]
MIPAV	Generic	Generic	Java	[90]
MedInria	Medical	Generic	C++	[91]
Anatomist	Brain MRI	Neuroimaging	Python, C++	[92]
FreeSurfer	Brain MRI	Neuroimaging	C++	[93]
FSL	Brain MRI	Neuroimaging	C++	[94]
Seg3D	Generic	Segmentation	C++, Python, Matlab	[95]
Tomviz	Tomographic data	Visualization	C++, Python	[13]
Paraview	Generic: medical, biological, engineering…	Visualization	C++	[96]
Vaa3D	Bioimages	Visualization	C++, C	[97]

2.2.3 Applications to the nucleus

Out of more than 150 published methods, we identify fewer than 12 that biologists can use (quoted from [40]). Despite the proficiency of deep learning methods publication for analysing nucleus images, only few are reusable. This quote is extracted from [40], work produced during this thesis and targeting the availability of deep learning methods for nuclear image analysis. As previously mentioned, the cell nucleus is the focus of investigation of the two teams of biologists involved in this project as well as the object of still prominent number of publications in biology. It

represents also the first field of investigation of this work on deep learning which will be further extended to biology and medicine in following chapters. Despite this nuclear focus, some of the contributions presented below, such as the deep learning sharing criteria, could probably be applied to a broad spectrum of applications. This sub-section mostly delineates the work done in [40]. It first presents the challenges of sharing deep learning methods by defining precise availability criteria, then sheds light on one of these criteria, namely the dataset, and finally overviews the actually available methods for nuclear image classification, detection, segmentation and denoising.

Deep learning method development. Making a deep learning method accessible to biologists is a long and often underestimated process. This coding journey is illustrated in Figure 2-8 and starts with a general deep learning method developed by researchers in computer science only and made to solve general visual tasks such as the ImageNet challenge [42]. This method can then be shared, reused, and adapted by a multidisciplinary team of biologists, bio-informaticians and computer scientists onto biological problems. The resulting applied method must be general enough for other multidisciplinary teams to reuse it on their own dataset. As often being the first step of applied science following a method issued from fundamental research, it will probably require two substantial refinements for *method applicability* and *method usability*. These two refinements are labelled “fine-tuned” and “easy-to-use” in Figure 2-8 and follows each other to simplify the representation. A method applicability tends to be universal if it can be applied to a broad range of data with only minor user inputs. This is one of the main goals of deep learning methods as they are supposed to work well on novel data by only providing novel manual annotations and then re-training/fine-tuning the deep learning model. Some knowledge in computer science is yet still required to achieve this. The method usability tends to be universal if the method is user-friendly, meaning that a clear graphical user interface and tutorials are provided. To develop such an interface, a consequent team of computer scientists might be needed. All these steps of development require the deep learning method to be properly shared.

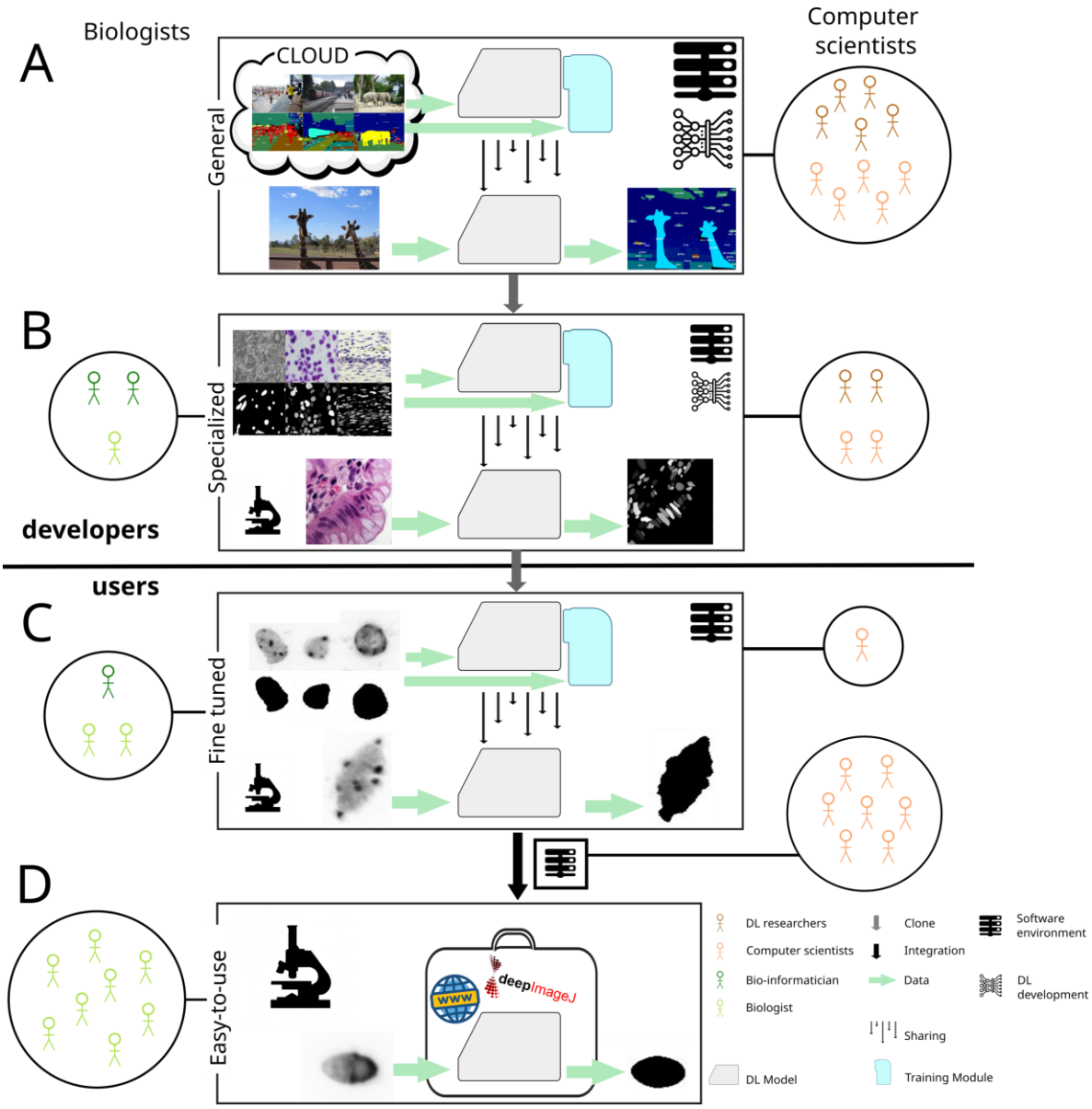


Figure 2-8 – **Deep learning method development steps, from a fundamental idea in computer vision to an easy-to-use tool for biologists. (A)** A new DL method is designed to solve general imaging problems and is trained on large standard datasets. **(B)** New developments are conducted by a multidisciplinary team to specialize the model to the images produced by the biologists. **(C)** A specialised model can then be used by biologists on their images. However, the model is often not completely adapted to them, and creation of a new small dataset and model fine-tuning is required. At this stage, retraining is often facilitated by packaging tools, such as Docker, reducing the need for IT-skills. **(D)** For this fine-tuned method to be used by a non-IT aware user, a team of software engineers should integrate it into an easy-to-use interface such as software, a web page, or a plugin.

Deep learning method sharing. Sharing a deep learning method is a more complex process compared to shallow machine learning methods. To consider a deep learning model viably shared, the following components should be made available (Figure 2-9-A):

- **Reusing criteria**:
    - *Detailed explanations in the publication*: The publication should provide comprehensive explanations of the deep learning method, including any relevant theoretical and practical background.
    - *Commented code for prediction and training*: The complete code of the method should be provided with detailed comments to aid understanding. The code must include the definition of loss functions, evaluation metrics, optimizer settings, hyper-parameter values, data pre-processing steps, and data post-processing steps.
    - *Clear user documentation*: Documentation should include installation steps and a clear explanation of the prediction procedure, enabling users to easily set up and run the model.
    - *Trained model* or *Dataset*: The trained model, which contains all the learned parameters, should be shared in a file or format that allows for inference. If not available, a dataset alongside a training procedure should be provided.
    - *Deep learning environment*: The software and hardware requirements for running the model should be specified, and ideally packaged with tools like Anaconda or Docker (Figure 2-9-B).
- **Reproduction and improvement criteria**:
    - *Training and testing datasets*: The datasets used for training and testing the model should be made available, ensuring both reproducibility and the ability to fine-tune the model.
    - *Developer documentation*: Detailed information about the code architecture and how to contribute to the code development should be provided.
- **Accessibility criteria**:
    - *Interfaces*: To make the tool accessible to non-developer users, the sharing of the inference procedure should be complemented with a

user interface, such as an Application Programming Interface (API) or a Command Line Interface (CLI) for programmers, or a Graphical User Interface (GUI) for non-programmers.

During the review process made in [40], a method will be considered as viably shared if it follows the five “reusing criteria”. It will be shown below, and it is illustrated in Figure 2-9-A, that only few provide the required components.

Deep learning environment. The last of these components, the *deep learning environment*, deserves a bit more explanations. If a code is made available, and if one tries to install it and use it, it is not unusual to find out that the code is not working, throwing some errors about missing requirements or drivers. This type of problem is linked to the deep learning environment. A piece of software with a high level of functionality, such as a deep learning-based software including a nice graphical user interface, will almost always be coded with the help of existing sets of tools. These sets of tools will also be coded with the help of other sets of tools. This stack of tools on top of one another is called a *software stack*. A simplified representation of the software stack is depicted on Figure 2-9-B and is divided into three levels: the *kernel*, the *operating system*, and the *applications*. The kernel is what makes physical, hardware components function together. The operating system (OS), on top of the kernel, can contain an interface and tools to create applications. It can include a Graphical User Interfaces such as Windows or Ubuntu-Desktop. Finally, the application level is where the final applications will be developed, installed, launched, and used. This last level is itself divided into several levels that will not be detailed in this thesis. The important information to take from this representation is that each of these levels have several *versions* and that each version of one level is dependent on very specific versions on the level below. A deep learning application for instance, could require a specific version of the Python programming language, of the Pytorch library, of a Linux-based OS and of Graphical Processing Unit drivers. The range of versions that properly work together is often small. To tackle this, a solution is to use packaging solutions, such as Anaconda, which packages tools in the application level, Docker, which includes the OS as well, or a Virtual Machine including every level. A naïve approach could be to consider using Virtual Machines always, but they tend to be slower than the two others and to have a larger computer memory footprint. Docker is often a good solution even though it is harder to use than Anaconda.

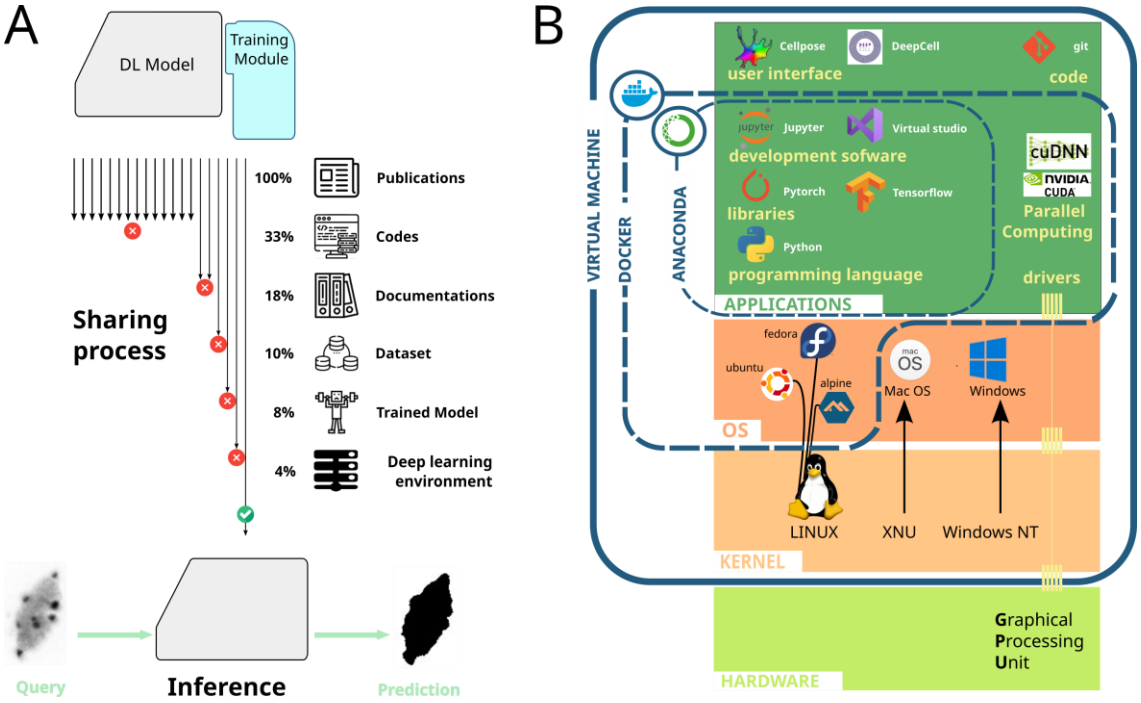


Figure 2-9 – **Sharing and use of deep learning models. (A) Sharing requirements** must be fulfilled for a deep learning method to be properly shared. For 3D nucleus segmentation, only 4% of published methods encompass the minimum requirements. **(B) Simplified view of the deep learning development environment.** Several layers of programs are stacked on top of each other based on the computer hardware (*light green, bottom*). The first software layer, the kernel (*light orange*), is a platform defining basic functions to leverage the hardware computation power and can be integrated in virtual machine definition. The next layer, the operating system (*OS*, *dark orange*) is another set of functions defined on top the kernel which includes more features such as a Graphical User Interface. It can be included in the definition of a Docker container. The top layer, the application layer (*dark green*), defines the programming environment and deep learning frameworks where Python applications can be isolated in an Anaconda environment. Illustration adapted from [40].

Importance of datasets. Deep learning method are called *data-driven*, meaning that the core of their proper functioning is in the constitution of a dataset with as few mistakes as possible. The dataset is the solely source of knowledge for deep learning models which means that, before resolving the required task, the model must extract some general knowledge about the images, such as an abrupt change in colour might represent an edge or that some regions of the images are noises. One solution to help the model extracting some of this information is to pre-train it either on a *pretext task*,

a process called *self-supervision*, or on a large set of pre-annotated images, a process called *transfer learning*. Transfer learning is often done using ImageNet [42], a large dataset including a broad range of daily life objects. Even though one could think that biological or medical objects look different, deep learning models pre-trained on ImageNet have shown significant improvements on cell images [98] or medical images [99]. Pre-training on large datasets of biological images similar to the images of interest is yet preferable. A complete list of manually annotated datasets for nuclear image analysis have been gathered in Table 2-3. As manually annotating can be time-consuming, computer scientists have also designed software to create artificial dataset of nucleus images and annotations, such as CytoPacq [100] (https://cbia.fi.muni.cz/simulator/index.php). For more details, Section 2.3.2 provides an overview of methods to overcome the lack of annotated data sets.

Data annotation and sharing. However, in most cases, manually annotating a dataset is required and thus a good annotation tool is needed. Free and open-source tools for bioimage annotation have been classified in

Table 2-4 depending on the targeted task: image classification, object detection or image segmentation. Some programs incorporate semi-automatic tools such as ilastik and Weka which can speed up the process but can sometimes bias it by suggesting wrong annotations. Paintera can also include Segment Anything [101], a deep learning tool developed by Meta to propose segmentations in 2D images. Nevertheless, for this thesis, the napari annotation tool has been selected as it offers an intuitive and efficient 3D viewer, along with helpful plugins.

Once annotated, a dataset should be made publicly available. Sharing annotated datasets allows other researchers to evaluate and build upon the results but is yet often more challenging than sharing codes which only requires a small amount of computer memory space and online versioning tools such as GitHub or GitLab. The first challenge with data sharing concerns the size of the data and its online accessibility. Sharing a dataset on an online server may require storage resources as well as web hosting capabilities not always accessible to scientists. A free and easy-to-use solution is Zenodo. Another solution is to use a cloud-storage platform such as Google Drive, OneDrive, Baidu Drive, or WebKnossos [15] but, for intellectual property reasons, those are not recommended. For bioimages, one could also use the services offered by

the Broad Bioimage Benchmark Collection (BBBC) [102] or the Image Data Resource [103] maintained by the OMERO developers [104]. However, these two services require the user to send an application and to follow certain criteria which could be a limiting. The second challenge with data sharing is the uprising recommendations to following the FAIR (Findable, Accessible, Interoperable, Reusable) criteria. Adhering to these principles ensures that the shared data can be easily reused by the scientific community. For bioimages, one of the key components to share along with the images is the metadata, which includes all the contextual information surrounding the capture of the image, such as the microscope characteristics, the time of capture or the image resolution. FAIR data sharing brings to the fore front the writing of "Data Management Plans" which are now strongly recommended by national research institutes such as the CNRS in France.

Table 2-3 – Complete list of publicly available datasets for nuclear image analysis.

Topic	Name	Description
Nucleus classification 2D	Mitos-Atypia-14 - Grand Challenge	Classification of nuclear atypia in breast cancer biopsy slides. Approximately 10400 frames. https://mitos-atypia-14.grand-challenge.org/Dataset/
Nucleus segmentation 2D	MoNuSeg - Grand Challenge	Multi-organ nuclei segmentation challenge. Challenge of MICCAI 2018. 30 images with approximately 22000 nuclear boundary annotations. https://monuseg.grand-challenge.org/Data/
Nucleus segmentation and classification 2D	MoNuSAC - Grand Challenge	31000 annotated nuclei from 4 different organs (Lungs, Prostate, Kidney and Breast) stained with H&E. https://monusac-2020.grand-challenge.org/Data
Nucleus segmentation 2D	Segmentation of Nuclei in Histopathology Images by deep regression of the distance map	50 annotated histopathology images. https://zenodo.org/record/1175282
Nucleus segmentation 2D	NucleusSegData: Cell Nucleus Segmentation Dataset for Fluorescence Microscopy Images	Fluorescence microscopy images. 2661 cell nuclei of 37 fluorescence microscopy images http://www.cs.bilkent.edu.tr/~gunduz/downloads/NucleusSegData/
Nucleus segmentation 2D	2018 Kaggle Data Science Bowl	A large variety nuclei images under a variety of conditions (small or large nuclei, from colored or grayscale images of different resolutions). Kaggle competition proposed by Booz Allen Hamilton https://www.kaggle.com/c/data-science-bowl-2018/data
Nucleus segmentation 2D	A Dataset and a Technique for Generalized Nuclear Segmentation for Computational Pathology	21000 nuclear boundaries in H&E-stained tissue. Used in many publications. https://nucleisegmentationbenchmark.weebly.com/dataset.html
Nucleus segmentation 2D	Deep learning for digital pathology image analysis: A	Tissue images containing nuclei segmentation, epithelium segmentation, tubule segmentation, lymphocyte detection, mitosis detection, invasive ductal carcinoma detection and lymphoma

	comprehensive tutorial with selected use cases.	classification http://andrewjanowczyk.com/wp-static/
Nucleus segmentation 2D	Dataset from "Immunohistochemistry (IHC) Image Analysis Toolbox"	52 images of clustered stained nuclei https://www.dropbox.com/s/9knzkp9g9xt6ipb
Nucleus segmentation 2D	Hand-segmented 2D Nuclear Images	100 images of clustred stained nuclei http://murphylab.web.cmu.edu/data/2009_ISBI_Nuclei.html
Nucleus segmentation 2D	EVICAN Dataset - a balanced dataset for algorithm development in cell and nucleus segmentation	Grayscale images from multiple bright field microscopes. 4600 images and 26000 segmented cells https://edmond.mpdl.mpg.de/imeji/collection/l45s16atmi6Aa4sI
Nucleus segmentation 2D	An annotated fluorescence image dataset for training nuclear segmentation methods	Annotated fluorescent nuclear images of different tissue origins https://www.ebi.ac.uk/biostudies/files/S-BSST265/dataset.zip
Nucleus segmentation and classification 2D	PanNuke: An Open Pan-Cancer Histology Dataset for Nuclei Instance Segmentation and Classification	205343 semi-automatically segmented nuclei from 19 different tissues stained by H&E. https://warwick.ac.uk/services/its/intranet/projects/webdev/sandbox/juliemoreton/research-copy/tia/data/pannuke
Nucleus segmentation and classification 2D	Dataset of segmented nuclei in hematoxylin and eosin stained histopathology images of ten cancer types	Biggest annotated dataset of the list. From The Cancer Genome Atlas: 5060 whole slide tissue images from 10 cancer types (approximately 5 billion segmented nuclei) automatically segmented and quality controlled and 1356 manually segmented patches from the TCGA from 14 cancer types (10 same and 4 new) https://app.box.com/s/yd4pbndk2bxtnourzpbvopga8dczsnes/folder/99392899243
Nucleus segmentation 2D and 3D	Broad Bioimage Benchmark Collection	Database of various medical image analysis problems. https://bbbc.broadinstitute.org/image_sets Some of the image sets focus on nuclei segmentation in 2D and 3D: -Nuclei of U2OS cells in a chemical screen (2D): https://bbbc.broadinstitute.org/BBBC039 -Drosophila Kc167 cells (cells and nuclei outlined in 2D): https://bbbc.broadinstitute.org/BBBC007 -Human U2OS cells (out of focus) (2D): https://bbbc.broadinstitute.org/BBBC006 -Human HT29 colon-cancer cells (diverse phenotypes) (2D): https://bbbc.broadinstitute.org/BBBC018 -Murine bone-marrow derived macrophages (2D): https://bbbc.broadinstitute.org/BBBC020 -Nuclei of mouse embryonic cells (3D): https://bbbc.broadinstitute.org/BBBC050
Nucleus segmentation 3D	EPFL - Electron microscopy dataset	Segmented nuclei of CA1 hippocampus brain region captured with electron microscopy. 2 segmented 3D images (1 for training and 1 for testing) https://www.epfl.ch/labs/cvlab/data/data-em/
Nucleus segmentation 3D	Neurosphere_Dataset	52 cells with stained nuclei captured with a LSFM http://opensegspim.weebly.com/download.html
Nucleus segmentation 3D	Asynchronous fate decisions by single cells collectively ensure consistent lineage composition in the mouse blastocyst	Confocal microscopy images of mouse embryo. 545 segmented 3D images. http://dx.doi.org/10.6084/m9.figshare.c.3447537.v1
Nucleus segmentation 3D	*A. thaliana* cotyledon cell nuclei	Confocal microscopy images of individual plant cell nuclei. 413 3D-images. https://OMERO.bio.fsu.edu/webclient/?show=project-3451

Table 2-4 –**Programs for annotating 2D and 3D images.**

Software	Task	Semi-automatic	2D	3D	3D viewer	Pixel or vector annotations?	Ref.
Qualitative Annotations	Classification		✓	✓		Not applicable	[105]
3D Slicer	Detection and Segmentation	✓(plugin)	✓	✓	✓	Vector	[12]
ITK-SNAP	Segmentation			✓	✓		[84]
3D-Bat	Detection			✓	✓	Vector	[106]
VGG Image Annotator	Detection and Segmentation		✓			Vector	[107]
LabelImg	Segmentation		✓			Vector	[108]
QuPath	Detection and Segmentation	✓(plugin)	✓			Pixel	[109]
Labtik (Fiji)	Segmentation		✓	✓		Pixel	[17]
ilastik	Segmentation	✓	✓	✓		Pixel	[7]
Weka	Segmentation	✓	✓	✓		Pixel	[8]
Paintera	Segmentation	✓	✓	✓	✓	Pixel and Vector	[110]
napari	Segmentation	✓(plugin)	✓	✓	✓	Pixel	[6]

Deep learning methods for nuclear image analysis. Once annotated and shared, dataset can then be used to train a deep learning method. This paragraph represents the core of the work presented in [40] which gathers 151 published deep learning methods for denoising, classification, detection, and segmentation of nuclear images (Figure 2-10). These publications, selected between 2014 and 2021, have been sorted following the criteria displayed in Figure 2-9-A and stored in a large XLSX file (Figure 2-11).

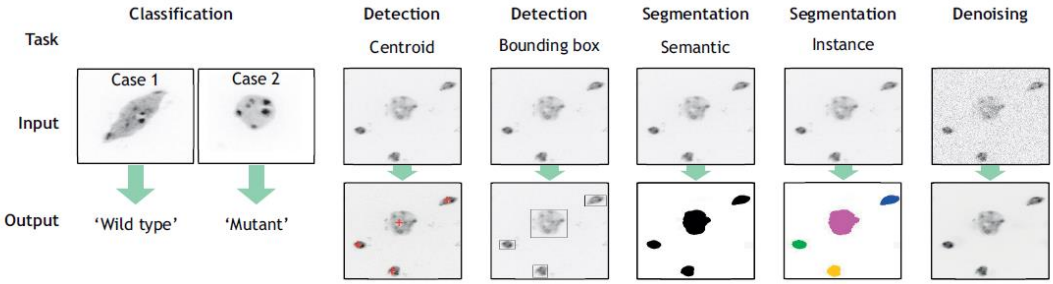


Figure 2-10 – **Examples of tasks typically solved by deep learning method for nuclear image analysis.** From left to right: a nucleus is classified in 2 categories; red crosses mark the detected centroids of nuclei; bounding boxes surrounds the perimeters of each nucleus; the image is segmented, nuclei in black and background in white; each nucleus is individually segmented with a different colour; an image is denoised.

	TASK	PUBLICATION DETAILS	CODE	DOCUMENTATION	DATA	MODEL	ENVIRONMENT	2D or 3D
91	SEGMENTATION	Song, J., Xiao, L., Molaei, M. & Lian, Z. Multi-layer boosting sparse convolutional model for generalized nuclear segmentation from histopathology images. Knowledge-Based Syst. 176, 40–53 (2019).	no					
92	SEGMENTATION	Yi, J. et al. Multi-scale cell instance segmentation with keypoint graph based bounding boxes. in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) vol. 11764 LNCS 369–377 (2019).	https://github.com/yijingru/KG_Instance_Segmentation	no	no	no	no	2D
93	SEGMENTATION	Lv, G. et al. Nuclei R-CNN: Improve Mask R-CNN for Nuclei Segmentation. in 2019 2nd IEEE Int. Conf. Inf. Commun. Signal Process. ICICSP 2019 357–362 (2019). doi:10.1109/ICICSP48821.2019.8958541.	no					
94	SEGMENTATION	Liu, D. et al. Nuclei segmentation via a deep panoptic model with semantic feature fusion. IJCAI Int. Jt. Conf. Artif. Intell. 2019–Augus, 861–868 (2019).	https://github.com/dliu5812/PFFNet	yes (but very succinct)	yes (public dataset)	no	yes (but not clear setup)	2D
95	SEGMENTATION	Caicedo, J. C. et al. Nucleus segmentation across imaging experiments: the 2018 Data Science Bowl. Nat. Methods 16, 1247–1253 (2019).	1st place: https://github.com/selimsef/dsb2018_topcoders 2nd place: https://github.com/jacobkie/2018DSB 3rd place: https://github.com/Lopezurrutia/DSB_2018	yes	yes (public dataset)	yes	no	2D
96	SEGMENTATION	Chakravarty, A. & Sivaswamy, J. RACE-Net: A Recurrent Neural Network for Biomedical Image Segmentation. IEEE J. Biomed. Heal. Informatics 23, 1151–1162	https://github.com/arunava555/RACE-net	no	no	no	no	2D
97	SEGMENTATION	Zeng, Z., Xie, W., Zhang, Y. & Lu, Y. RIC-Unet: An Improved Neural Network Based on Unet for Nuclei Segmentation in Histology Images. IEEE Access 7, 21420–21428	no					
98	SEGMENTATION	Xie, X., Li, Y., Zhang, M. & Shen, L. Robust Segmentation of Nucleus in Histopathology Images via Mask R-CNN BT - Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries. in (eds. Crimi, A. et al.) 428–436 (Springer International Publishing, 2019).	no					
99	SEGMENTATION	Abdolhoseini, M., Kluge, M. G., Walker, F. R. & Johnson, S. J. Segmentation of Heavily Clustered Nuclei from Histopathological Images. Sci. Rep. 9, 4551 (2019).	https://github.com/Mahmoud-Abdolhoseini/GliaTrace-toolkits	no	yes https://bioimage.ucsb.edu/research/bio-segmentation	no	no	2D
100	SEGMENTATION	Naylor, P., Laé, M., Reyal, F. & Walter, T. Segmentation of Nuclei in Histopathology Images by Deep Regression of the Distance Map. IEEE Trans. Med. Imaging 38, 448–459 (2019).	https://github.com/PeterJackNaylor/DRFNS	yes (but succinct)	yes	no	no	2D
101	SEGMENTATION	Turan, S. & Bilgin, G. Semantic nuclei segmentation with deep learning on breast pathology images. in 2019 Sci. Meet. Electr. Biomed. Eng. Comput. Sci. EBBT 2019 1–4 (2019). doi:10.1109/EBBT.2019.8741715.	no					
102	SEGMENTATION	Xu, Z., Sobhani, F., Moro, C. F. & Zhang, Q. US-net for robust and efficient nuclei instance segmentation. Proc. - Int. Symp. Biomed. Imaging 2019–April, 44–47	no					
		Zhao, Z. et al. A coarse-to-fine data generation method for 2D and 3D cell nucleus						

Figure 2-11 – Extract from the XLSX table of [40] sorting relevant deep learning methods for nuclear image analysis. Table downloaded from: http://www.biologists.com/JCS_Movies/JCS258986/TableS1.xlsx.

Enhanced visualization of nuclei, appealing to both human interpretation and deep learning methods, can significantly facilitate image analysis. Deep learning-based **denoising** have been investigated to improve image quality. Five methods providing the underlying code have been identified: Noise2Void [78], VoidSeg [111], DenoiSeg [77], DecoNoising [112], and 3D-RCAN [113]. All of these methods employ a U-Net model [58] trained with artificially noised images, to reproduce the original image. The first three solutions are based on the CSBDeep toolbox [76] and have demonstrated their efficiency with 2D nucleus images. Notably, the DenoiSeg method

combines denoising and segmentation in the same model, resulting in substantial improvements in 2D nuclear segmentation compared to other methods, such as StarDist [68]. Furthermore, 3D-RCAN is also suitable for denoising 3D images. Programmers may find interest in exploring these methods, while non-programmers can leverage user-friendly denoising models integrated into ZeroCostDL4Mic, provided they have access to a suitable dataset of denoised raw images.

Image classification involves categorizing each image into different classes. Most nuclear image analysis studies predominantly concentrate on 2D histopathology images derived from human tissues, typically stained with Haematoxylin and Eosin. Regrettably, out of the 24 methods reviewed, only one study [114] provides both code and datasets. To the best of our knowledge, no prior research has been published regarding the image classification of 3D nuclear images. The only solution is to program using a deep learning framework and eventually rely on Google Colab computers to get a free access to a sufficient computer power. Most tools for generic image classification, such as Google AutoML Vision, Roboflow, H2O or KNIME, are unfortunately limited to 2D images and are not free and open source.

Object detection consists in either locating the centroid of each object or the coordinates of a box surrounding it. Among the 31 methods found for object detection, only 5 provide a code and a trained model (StarDist [68], SP-CNN [115], KiNet [116], NucleusDetection [117], and QCANet [118]) and only 2 can handle 3D images (QCANet and StarDist). Removing the trained model availability constraint allows to include nnDetection [89] in this review which automatically handles the configuration of the complex set of hyper-parameters such as in nnU-Net [86] (see Section 2.3.1). Underlying all these methods is the generation of a distant map, a grey-scale image where each nucleus is coloured with a gradient of black to white, black being the edges, and white, the centroid of the nucleus. The deep learning model is generally a 2D or 3D U-Net model, a standard model for image segmentation. With this approach, object detection is thus a subset of image segmentation. Detection-specific methods, such as YOLO [119] or Faster R-CNN [120], have been used in the aforementioned 31 publications but none have provided a code. Nevertheless, a solution for 2D images is to use ZeroCostDL4Mic [70] which includes the YOLO approach, among many other methods, in an online and user-friendly interface powered by Google Colab. Users have yet to provide an annotated dataset and 3D image analysts are limited to use semantic segmentation tools presented later in this paragraph.

Image segmentation is traditionally separated in two categories: semantic segmentation, which consists in classifying each picture element in one class of object, or instance segmentation, which consists in additionally discriminating each object in a certain category into individual instance (Figure 2-10). For 101 compiled publications about nucleus segmentation, 35 provide a code and only 10 of those handle 3D images. As image segmentation has become the main topic of interest of this thesis, more details on this task will be provided. During the course of this literature review project, six main difficulties linked with nuclei segmentation, image analysis and deep learning has emerged:

- **Nuclei-related difficulties**, linked to the variability in cell shape, size and texture in different tissues.
- **Noise-related difficulties**, including background complexity, poor signal-to-noise ratio, uneven colour distribution, heterogeneous capture conditions or sample preparation.
- **Image modality-related difficulties**, associated with the diversity of available devices and their configurations (2D or 3D, confocal or electron microscope, etc.).
- **Manual annotation-related difficulties**, mostly due to the annotator subjective biases or the inter-observer variability when several human annotators are involved.
- **Method-related difficulties**, involving the lack of robustness of classical computer vision techniques, the configuration and training of deep learning methods which can also imply expensive computational costs (especially for 3D applications), or the explainability of deep learning models.
- **Use-related difficulties**, including deep learning tool installation and friendliness for non-programmer users.

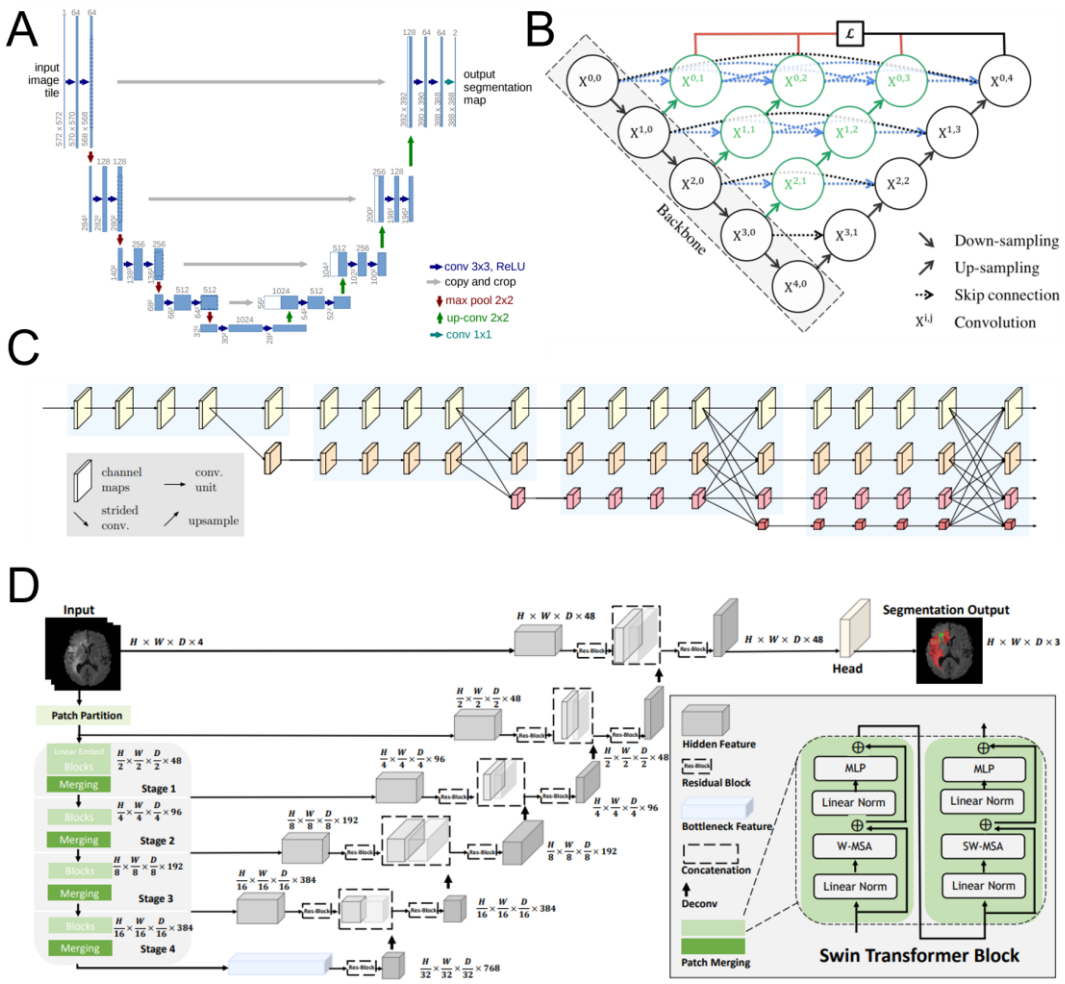


Figure 2-12 – **U-Net model variants**. **(A) Original U-Net model** from [58] with a noticeable the encoder-decoder architecture. **(B) U-Net++ model** from [121] with a dense version of the U-Net model and the encoder is here named *backbone*. **(C) HRNet model** from [122], another dense variant of the U-Net model, has reached the state-of-the-art for 2D semantic segmentation in 2022. **(D) Swin UNETR model** from [123] including a Swin-Transformer backbone [64] for 3D segmentation which reached the state-of-the-art for medical image segmentation in 2022.

From the two types of image segmentation task cited above, the least difficult one is semantic segmentation. Even if easier than instance segmentation, semantic segmentation is an active field of development as many applications still requires improvements, such as filament segmentation, tumour segmentation or even nuclei segmentation. Semantic segmentation methods are mainly based on the U-Net model [58] also known as Feature Pyramid Network [124]. This model has two main parts, an encoder (the descending U-branch) and the decoder (the ascending U-branch)

(Figure 2-12-A). The encoder successively encodes the input image input a series of *feature maps* with progressively lower dimensions. As shown on Figure 2-5, high dimension feature maps contain information about simple shapes such as edges while low resolution feature maps condense complex information such as the shape of a cell nucleus. To improve the performance of the U-Net model the encoder is often replaced by one of the backbone models mentioned in Section 2.2.1 such as the EfficientNet [62]. Other improvements of the U-Net model involve adding a dense network of convolutions between the encoder and the decoder such as in the U-Net++ model [121] (Figure 2-12-B) or the HRNet model [122] (Figure 2-12-C), or using Transformer operations (Figure 2-12-D). The decoder of the U-Net model composes the final segmentation mask from the information extracted by the feature maps of the encoder. The U-Net model has been adapted for 3D images [125] and integrated into ZeroCostDL4Mic.

Due to its dual task (classification of picture elements in both class and instance), instance segmentation is more challenging than semantic segmentation. Moreover, instance segmentation is often related to nucleus segmentation as it allows for the discrimination of objects within clusters. In the literature review conducted during this project, four categories of instance segmentation approaches have been identified:

- **The centre-border approaches** rely on a ruse to transform the instance segmentation task into a semantic segmentation task by asking the model to segment the 'centre' and the 'border' of each instance of object which leaves enough spaces between objects to be classified with a classical connected component algorithm. These approaches won the 2018 Data Science Bowl challenge for 2D nucleus segmentation [126] with an ensemble of 32 U-Net models. Mesmer approach in DeepCell [127] also relies on centre-border segmentation and additionally integrates cell segmentation.
- **The distance map approaches** involve predicting a distance map, akin to those previously introduced with object detection, to concurrently find the centroid of each nucleus and their segmentation. For this type of approach, deep learning methods differ from one another by the way the distance map is computed. StarDist [68] traces a star-convex polyhedron in the centre of each nucleus, Cellpose [72], [73] uses a heat diffusion simulation that starts from the centre and iteratively spreads towards the borders and NISNet3D [128]

directly predicts the gradients along the three dimensions of the image. The three methods handle 3D images though Cellpose works with a slice-by-slice approach along each of the three axes (approach often called '2.5D').

- **The bounding box approaches** combine a detection method with bounding boxes and a semantic segmentation method, among which, probably the most famous one, is Mask R-CNN [129]. Mask R-CNN is a general approach for simultaneous 2D object detection and segmentation which have been successfully applied to 2D nucleus segmentation (see Matterport implementation: https://github.com/matterport/Mask_RCNN). A series of small convolution modules, inspired by the Faster R-CNN method, specially adapts a classification backbone to predict coordinates of object bounding boxes and, for each box, a segmentation mask is computed using another set of modules. At the time of writing, no 3D version of Mask R-CNN appeared to be freely available. A 2D nucleus-specific bounding box approach named NuSeT [130] is publicly available and includes a graphical user interface.
- **The centroid approaches** are very similar to the bounding box approaches yet replacing the bounding box detection by a centroid detection. QCANet approach [118] involves two different models to compute separately the centroid and the segmentation mask of each nucleus. Each centroid seeds then a standard connected component algorithm such as the watershed algorithm to find instances in the segmentation masks.

Probably the best way to start experimenting with the previous segmentation methods, is to use online tools such as Cellpose (https://www.cellpose.org/), DeepCell (https://www.deepcell.org/), NucleAIzer (https://www.nucleaizer.org/), or the recent Segment Anything (https://segment-anything.com/). However, these tools are limited to 2D images. When working with 3D images, one recommendation could be to use the offline version of Cellpose. Even though, DeepCell and QCANet also provide such offline tools, it will be seen in Section 2.2.4 that those are difficult to reuse. If the images to predict are not like the training ones, the pre-trained models of these methods might need some retraining (as shown in Figure 2-7). In this case, a suggestion is to use StarDist in case of round and clustered nuclei or to use ZeroCostDL4Mic for any other cases. An alternative solution called Biom3d is presented in Chapter 3 and represents the main contribution of this thesis.

2.2.4 Benchmarking methods for 3D nucleus segmentation

To push the review further, a comparative study of six of the previous methods for 3D nucleus segmentation was conducted. This work was supported by the help of two interns, Pedro Mezquita and Adama Nana, for the deep learning method implementation and testing, and by the help of Sophie Desset, co-supervising this thesis, for the dataset annotation. The goal of this benchmarking was to identify deep learning methods that solve the limitations of NucleusJ plugin for nucleus and chromocenter segmentation. Ideally, these selected methods should be easy enough to be used by non-programmers. Given the research-oriented nature of this project, a more comprehensive analysis was conducted on one of the selected methods in Section 2.3.1, to identify its potential drawbacks and serve as a foundational basis for further exploration and the development of novel innovations.

A novel 3D nucleus dataset. To compare the methods, a dataset of 93 tridimensional images of *A. thaliana* nuclei captured with structured illumination microscopy was manually annotated by Sophie, being a microscopist and a specialist in the domain. This dataset is a subset of the one published by Tristan Dubos [29] a former PhD student in the team in Clermont-Ferrand (dataset accessible here: https://OMERO.bio.fsu.edu/webclient/?show=project=2801). As plant nuclei do not form clusters, each nucleus can be isolated with the *autocrop* functionality of NucleusJ and stored in individual images. The annotations were made with napari software [6]. The choice of this software was motivated by its integrated 3D viewer. To prevent software-related bias, it was also chosen not to use semi-automated annotation program. Each voxel annotation has thus been carefully manually curated. In the beginning of this benchmarking project, the annotations were done by me, a non-expert in nucleus image, with the help of ilastik [7], a semi-automated software. As will be illustrated further in this Section, these non-expert annotations turned out to be of poor quality after being verified by an expert and were thus discarded.

Methodology. Six deep learning methods were intended to be benchmarked: DeepCell [127], Cellpose [72], QCANet [118], StarDist [68], NuSeT [130] and nnU-Net [86]. Unfortunately, opposed to what was suggested in its associated article, NuSeT did not support 3D images. Among the other methods three provided a trained model for nucleus detection (DeepCell, Cellpose, QCANet) and the two others (nnU-Net and StarDist) a script to train a model on a novel dataset. The three methods

providing a model did not provide a script to retrain their model on 3D images. To get a fair comparison, all methods were evaluated on the same set of 28 images extracted from the 93 images. The trainable methods were trained on the remaining 65 images of the dataset. During training, the choice of the training methodology, including the choice of the validation set, was determined by the default configuration of each implementation. The evaluation metrics were the Dice score and the Hausdorff distance (Figure 2-13). The Dice score can be defined with the following formula:

$$Dice(y, \hat{y}) = \frac{2|y \cap \hat{y}|}{|y| + |\hat{y}|} = \frac{2\sum_i (y_i \cdot \hat{y}_i)}{\sum_i y_i + \sum_i \hat{y}_i}$$

where y is the segmentation ground truth and $\hat{y}$ the segmentation predicted by the model. The Dice score can easily be implemented in Python leveraging the Numpy library.

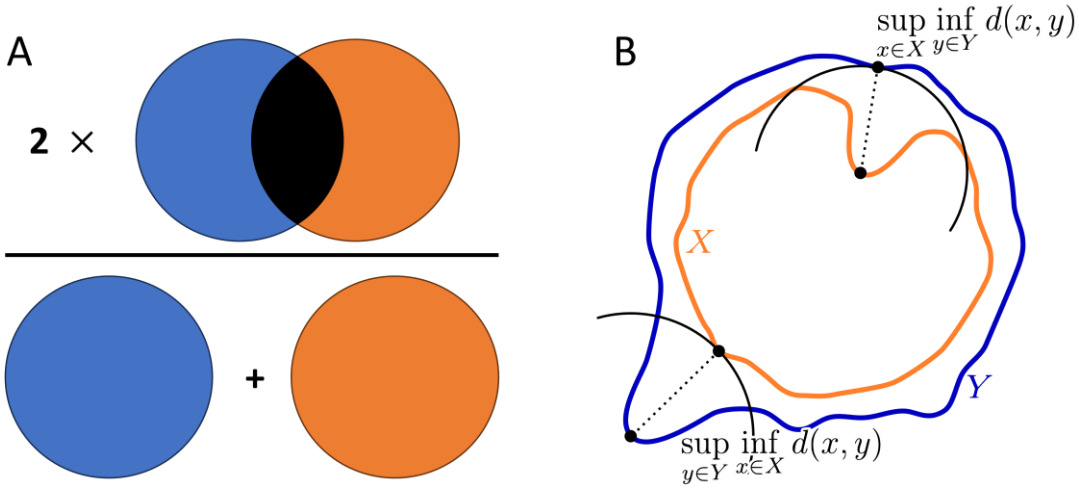


Figure 2-13 – **Metrics for biomedical segmentation. (A) The Dice score** is twice the volume of the intersection (*black*) between the two volumes of the ground truth (*blue*) and the prediction (*orange*) divided by the sum of the two volumes. **(B) The Hausdorff distance** is the maximum between two directed distances between two surfaces, one for the ground truth (*blue*) and one for the prediction (*orange*). A directed distance between two surfaces is the maximum of all the minimum distances obtained when spanning across one of the surfaces. Illustration adapted from https://commons.wikimedia.org/wiki/File:Hausdorff_distance_sample.svg

The Hausdorff distance can be defined by the following equation:

$$Hausdorff(y, \hat{y}) = \max\left(h(y, \hat{y}), h(\hat{y}, y)\right)$$

where $h: (y, \hat{y}) \longmapsto \max_{p_y \in y} \min_{p_{\hat{y}} \in \hat{y}} \left\| p_y - p_{\hat{y}} \right\|$ is the directed Hausdorff distance and $\|\cdot\|$ is the Euclidean norm. The implementation used for this study is integrated in the MONAI library based on the Scipy library [131]. These two metrics have been chosen for their complementarity and their popularity in the biomedical field but some limitations recently highlighted by a large consortium of scientists [132] might change this trend.

Results. The results of this benchmarking (Figure 2-14) showed that the method called nnU-Net clearly overpasses its counterparts. Surprisingly, the classical methods implemented in NucleusJ came second. It could be argued that the trained deep learning models were not ideally adapted to this specific nucleus dataset, such as DeepCell which distinctly predicted the 3D image layer-by-layer instead of considering all three dimensions, or QCANet which omitted to the segment the nucleolus region. Despite this constraint, Cellpose was able to achieve relatively high accuracy by using the model called "cyto2" instead of "nucleus" and by removing some prediction noise due to artifacts in DAPI staining. As no retraining script for 3D images was available, this candidate could not be retrained. StarDist method did not have a pretrained model but had a well-made tutorial on how to train a new one. However, StarDist includes a strong prior knowledge about the objects of interests: they must be star-convex polyhedron which means that each object must have a centre where every other point in the object is accessible by tracing a straight line starting from this centre. This method is well suited to round nuclei grouped in clusters as in animal cell cultures, but not to plant nuclei, which often have curved, elongated shapes. This benchmarking was finally completed with an inference time comparison motivated by the strong differences noted during evaluations. This was done only for the four best methods (QCANet, Cellpose, StarDist, nnU-Net), on the same computer with a Nvidia 2080Ti GPU, using the 28 test images and then averaged to obtain an approximate of the inference time per image. Large gaps between inference time were thus pointed out with, for example, an extreme time ratio of 1800 between the worst candidate (QCANet) and the best one (nnU-Net).

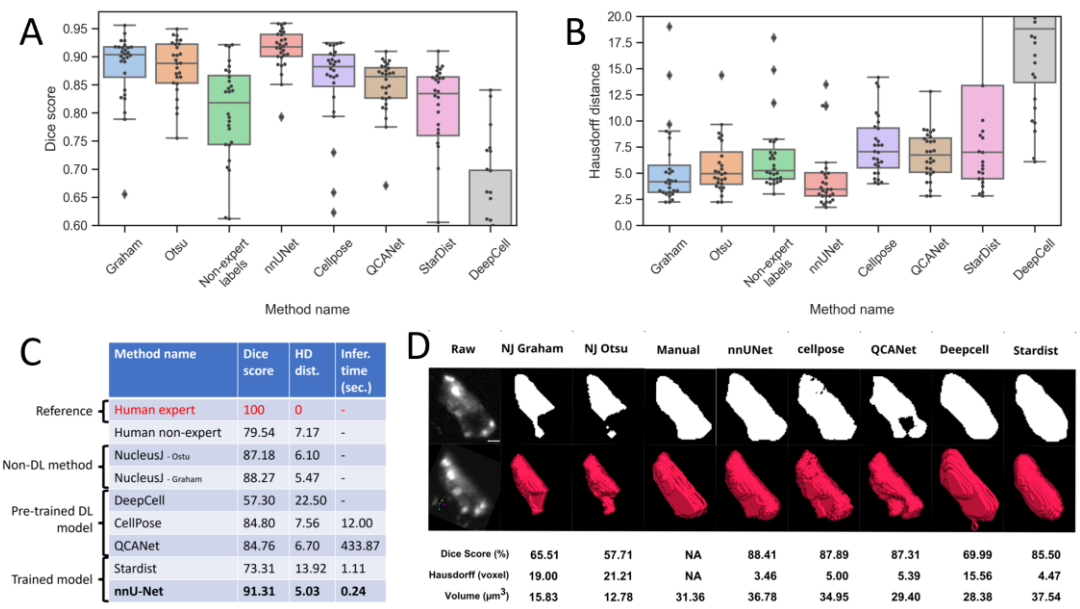


Figure 2-14 – **Benchmarking results on 3D nucleus dataset.** Scores are computed over a test set of 28 images annotated by an expert. Five deep learning methods (nnU-Net, Cellpose, QCANet, StarDist, DeepCell) are compared to two non-deep learning methods (Otsu and Graham) and to one other manual annotation conducted by a non-expert. **(A)** Box plot of the Dice scores. **(B)** Box plot of the Hausdorff distance. **(C)** Mean values of the Dice score and the Hausdorff distance and the inference time per image in seconds. **(D)** Views of the segmentation results for one nucleus both in 2D (*upper row*) and in 3D (*middle row*). For each result, the corresponding Dice score, Hausdorff distance and volume is displayed on the *lower row*.

Containerization. To foster reusability of this work, each method has been packaged into a Docker container by the two interns (code available here: https://github.com/GuillaumeMougeot/nuclei_benchmark). The Docker container contains the operating systems with all drivers and applications specific to each method. A Docker "recipe" was written in a single file called a Dockerfile. This recipe can be used to install and configure all required dependencies and files required for installing the methods, including for instance, the installation of the Python libraries or the downloading of the deep learning model. This Docker methodology (Figure 2-15) allows each method to be isolated on a single computer and thus avoids any conflict, which could be due, for example, to two conflicting Python libraries trying simultaneously to access to the same GPU resources. Another advantage of containerization is to standardize user access to each method. For all methods, the user can now install it by simply running

build.sh shell script and, if possible, one can then run *run_predict.sh* script on a new dataset of raw images to get some predictions. If the model needs training, the script *run_training.sh* can train it on an annotated dataset. Containerizing deep learning methods is not yet the norm in the field (only DeepCell provided a Dockerfile) but it could evolve in the future driven by initiatives such as BIAFLOWS [133] or IFB-Biosphère [134].

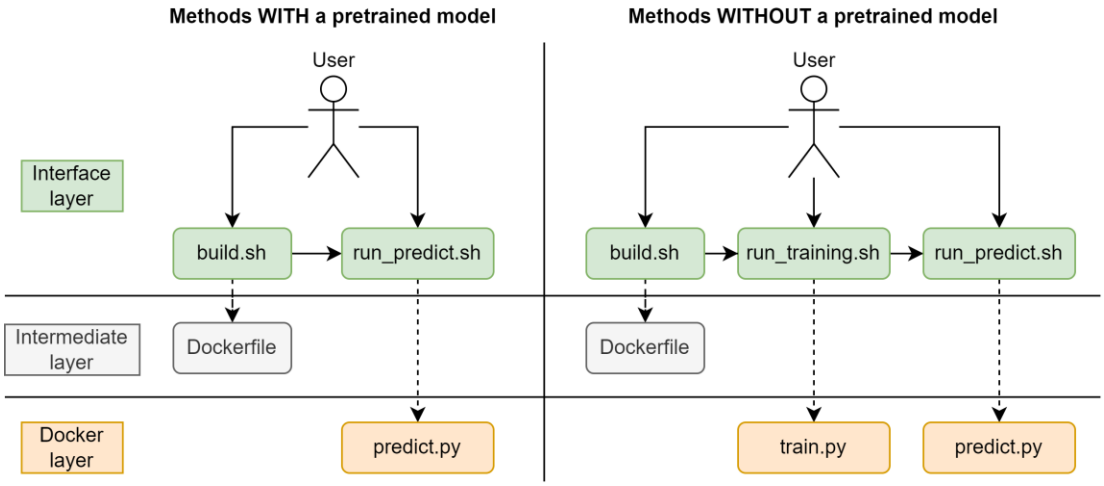


Figure 2-15 – **Containerizing and interfacing the deep learning methods**. Each method can be installed on any computer having Docker via a single script *build.sh*, which relies on a installation "recipe" stored in a Dockerfile, and then reused with *run_predict.sh* on a new dataset. Methods that must be trained have an additional *run_training.sh* script to start training a model on a new dataset. The specificities of each method are thus transparent for the user and packaged inside the Docker image.

Conclusion. nnU-Net method stood out during this benchmarking. Not only because of its accuracy and speed but also for its versatility. While limited to semantic segmentation, Section 2.3.1 will demonstrate that nnU-Net can be applied to a large panel of biological and medical segmentation problems, achieving high accuracy by automating the configuration of all hyper-parameters. This crucial property will significantly influence the direction of this thesis. A successful research project can probably be determined by a good balance between specificity and generality. A project that is too specific risks being limited to a particular group of individuals, location, or time, potentially rendering it less impactful over time. On the other hand, a project that is overly generic might not resonate with any specific audience. In both cases, the project may fade into oblivion. While the plant nucleus problem might be,

to some extent, too specific for deep learning applications, an accessible and adaptive tool for segmenting 3D microscopy images leveraging this breakthrough technology is still lacking. A quick observation in biology laboratories reveals a significant number of image analysis challenges that are still manually addressed but could be rapidly solved with deep learning. Inspired by the adaptability of nnU-Net, the design of the segmentation tool presented in Chapter 3 will thus attempt to fulfil the expectations of plant biologists but also to generalize on a broader spectrum of applications. This novel tool will go beyond the original development ideas of nnU-Net by incorporating a new fundamental aspect: modularity.

2.3 Improving deep learning methods

nnU-Net belongs to a class of methods called *Automated Machine Learning*, abbreviated in *AutoML*. This class of methods attempts to tackle the first difficulty arising when training machine learning methods: configuring of the model and training hyper-parameters, parameters that must be set manually and significantly impact the performance of the model. However, determining the optimal values for hyper-parameters is often a challenging and time-consuming task, relying on empirical approaches based on developer experience. AutoML thus seeks to automate this process and will be the focus of the first Subsection below. The second difficulty with training of deep learning methods is its dependence on large, annotated dataset, often expensive to produce, especially for biological and medical imaging. As a result, the second Subsection will discuss an array of research efforts aimed at addressing this challenge by reducing the number of required manual annotations.

2.3.1 AutoML and nnU-Net

AutoML. Various approaches exist to deal with the configuration of training and model hyper-parameters. Regarding the optimization of training hyper-parameters, the first and probably the most intuitive one is called *grid search*. For each training hyper-parameters, such as the learning rate or the batch size, a range of plausible values is defined by the developer. Each combination of hyper-parameter values will then serve to train a different model. The best model will thus determine the best set of training hyper-parameters. This approach very early demonstrated its performance for deep learning methods [135] but due to its high-computational cost has progressively been replaced by more efficient methods such as *random search* [136]. However, grid and

random search still suffer from being too uniformed methods of exploration, completely omitting to exploit good candidates. Methods such as *Sequential Model-Based Global Optimization* [137], formalization of Bayesian optimization, aim thus at defining the deep learning model and the loss function as a probability distribution, called a *surrogate model*. Common choices of surrogate model can be Gaussian Processes, Random Forests or Tree Parzen Estimators. Based on an history of several pairs of loss values and training hyper-parameters, the surrogate model estimates the loss evolution when varying the hyper-parameters. A selection function, such as the Expected Improvement, can then be used to choose a novel set of hyper-parameters and the whole process can restart until reaching a satisfying loss. Genetic Algorithms [138] are another set of methods improving random search by viewing the set of hyper-parameters as a set of genes. The loss value obtained from training a deep learning model using a certain set of genes represents the ability of the set of genes to "survive". Only the sets of genes with the highest score survive. After several iterations, crossovers and mutations are performed among the set of genes to obtain new sets of genes and the process is reiterated.

Model hyper-parameter configuration includes the selection of the model itself and the design of its inner architecture, such as choosing the number and size of each layer. Bayesian Processes and Genetic Algorithms have been used as well to configure model hyper-parameters, both being thus part of an expanding domain called *Neural Architecture Search*. The original ideas of Neural Architecture Search [139], [140] was to exploit Reinforcement Learning to automatically design the set of interacting operation within a deep learning model. Attempts have been made to apply it to 3D medical imaging [141], but still fall short behind manually design models. A reason that could explain that automatically designed models for image segmentation are still overpassed by human-made ones is that carefully designed, and extremely large models can extract a consequential number of features in images, which means that a significant gain in performance could be achieved by simply focusing on selecting appropriate features and cutting down inappropriate ones instead of developing task-specific models.

nnU-Net. nnU-Net [86] is a trade-off between AutoML and manual methods. It was originally created to win several online competitions of medical segmentation, such as the Medical Segmentation Decathlon [142], without the need to manually reconfigure

the hyper-parameters to win a new challenge. As some of the hyper-parameters are indeed set automatically, nnU-Net could be considered as an AutoML methods. However, the heuristics defining them are deterministic, meaning that the set of hyper-parameters is fully determined by the characteristics of the dataset without any exploration of hyper-parameter space by algorithms such as random search. These heuristics were created empirically based on the intuition of their developers and on a tremendous repetitions of experiments done over many biological and medical datasets. The input of the heuristics is called the *data-fingerprint*, a configuration file obtained by scanning the entire training dataset and extracting relevant information in the images such as their median shape, their median sampling, or the distribution of their voxel intensity (Figure 2-16). This information is then used for:

- **Data pre-processing**, which includes image intensity normalization, and image and annotation resampling. Resampling is usually required when the spatial samplings of the two images in the dataset are dissimilar (see Section 1.2.3 for more details about image spatial sampling), which often happens in medical dataset and can also occur in biological datasets if, for example, the microscopist decides to change the sampling size along z-axis while capturing a set of 3D images.
- **Data loading**, which includes the batch size and the patch size. Due to computer memory limitations, patching (*cropping*) is a mandatory step when dealing with 3D images and deep learning methods. Indeed, when training a deep learning model, it is required to temporally store every intermediate output (*feature map*) to compute the gradients for backpropagation and model parameters update. The size of each feature map is determined by the input patch size. An appropriate patch size is also determinant for the final performance of the model. nnU-Net patch size strategy considers the GPUs memory limits, the image anisotropy, and the network topology. The final patch size will thus tend to be as big as the GPU memory allows it, to have a similar anisotropy as the median image size and to have dimensions that can be divided by a power of two.
- **Network topology** (*model architecture*), which includes the dimension of each pooling layers and of the kernel of each convolution layers. Both are mainly determined by the greatest power of two dividing the patch size. As pooling

layers successively reduce feature maps by a factor of two, reducing them too much along anisotropic dimensions might lead to non-integer dimensions. nnU-Net model is thus a standard U-Net model but with flexible pooling sizes which follows patch size anisotropy. The size of convolution kernels follows the same rule.

- **Cascade trigger,** which includes the creation of two U-Net models instead of one. Cascade models are used only for extremely large images. With such images, the maximum patch size allowed by computer memory might be too small for the deep learning model to get a global understanding of the data. nnU-Net follows here a coarse-to-fine approach: the first of these U-Net model will work with a downscaled version of the images, performing a pre-segmentation which will then be input to the following U-Net alongside full-size images.

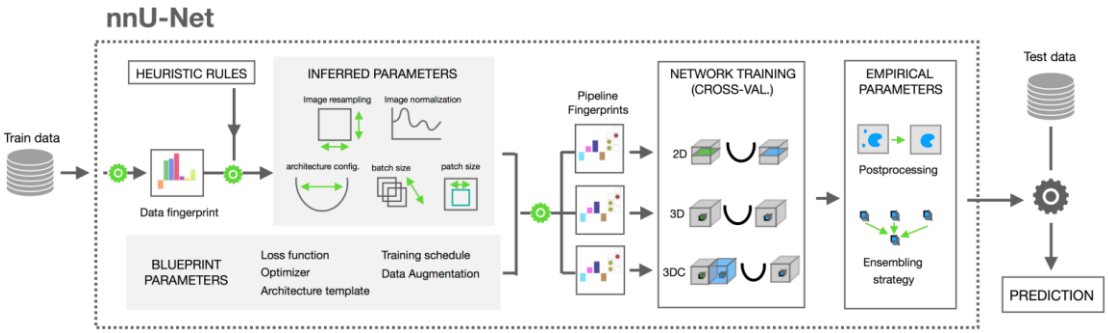


Figure 2-16 – **nnU-Net automated method for hyper-parameter configuration.** A data-fingerprint is extracted by scanning the dataset. It serves to determine the values of rule-based parameters (*inferred parameters*) such as the patch size of the network topology. Other hyper-parameters are fixed (*blueprint parameters*) such as the data-augmentation or the optimizer configuration. After training completion, trained models are selected with an ensemble selection process (*empirical parameters*). Illustration adapted from [86].

These rule-base parameters are completed by a large set of fixed hyper-parameters also determined by the extensive experiments. This set includes:

- The rest of the model hyper-parameters such as the depth of each convolution kernel which follows the original U-Net publication [58].
- The optimizer hyper-parameters, imposed to be Stochastic Gradient Descent with a learning rate reducing during training with a polynomial decay.

- The rest of data-loading hyper-parameters, with data-augmentation and foreground forcing. Foreground forcing imposes one third of each image patch to include a foreground region (on the object of interest) in its centre.
- The training procedure, with the number of epochs and the frequency of validation.
- The loss function, imposed to be the sum of the Dice score and the cross-entropy between the ground truth and the prediction.
- The inference procedure, with test-time augmentation, inference patching strategy, and ensemble prediction. During inference, each image is patched following a grid sampling strategy: predictions are successively computed for patches starting from the top-front-left corner then sliding toward the bottom-back-right corner. For information redundancy, each inference patch overlaps half the dimensions of the previous one and, for each patch, a 3D Gaussian mask is used to filter prediction inaccuracy on patch edges. The ensemble prediction is based on the cross-validation strategy: five different U-Net models are trained on five non-overlapping subsets of the training set and each model then computes a prediction. The five resulting predictions are averaged to obtain the final output.

This list is an incomplete overview of the entire nnU-Net configuration process (see their code on GitHub for more details: https://github.com/MIC-DKFZ/nnUNet/tree/master).

To cite a recent post from nnU-Net author (GitHub website, March 2023, https://github.com/MIC-DKFZ/nnUNet/discussions/1189), this method was originally created to prove a point: *It really took a while to notice that […] maybe, maybe, a significant proportion of the research out there [MICCAI] suggested something that was just not there. […] So, I started trying to prove a point by overengineering the U-net to the max. […] It is quite interesting that even to date [March 2023], and to the best of my knowledge, there is no segmentation method that really, convincingly outperforms nnU-Net in the medical domain.* In his discussion, Fabian Isensee, made three assumptions for why this is the case:

- *Things other than the network architecture matter more in medical imaging* such as the patch size.

- *Segmentation problems in the medical domain are different [than natural image problems]*. Medical images are obtained in a much more controlled environment, meaning that often deep learning models reach the upper limit accuracy which is the inter-rater variability.
- *Small dataset sizes are disadvantageous for complex architectures* meaning that spending time collecting more data will increase model performance significantly more than defining novel model architectures.

Should it be concluded that any improvements are hopeless? Not quite. Independently from nnU-Net author, these thoughts have been concluded in the beginning of this thesis and reinforced by the previous work on the review. Two important conclusions have been drawn:

- Incorporating the high performances of nnU-Net heuristics within an extremely flexible framework could, first, rapidly give a good baseline method for any new datasets and, second, be quickly reshaped to fit the dataset specificities. This is the main guideline of Chapter 3 developments.
- As data-annotation remains the bottleneck of deep learning model training, methods that address the reduction of the amount of work involved in data annotation should be the focus of attention. This will be the subject of the following Subsection and the main contribution of Chapter 4.

2.3.2 Reducing the number of manual annotations

Annotating biological or medical images with expert eyes is a costly process. Computer scientists have thus created entire field of research aiming at reducing the need for manual annotations. In this Subsection, a brief overview of different categories of methods that can help achieve this goal will be provided. Some of these methods, such as self-supervised methods, have originally been designed to enhance the performance of deep learning models with a given amount of data, have the potential to perform as well as classical supervised deep learning methods with less data, making them valuable in this context. Probably against certain schools of thought, this extensive research field will be divided here into four categories: synthetic data generation, weak supervision, active learning, and transfer learning. Each category will also be exemplified with relevant literature examples related to the biological focus of this work: nucleus image analysis.

Synthetic data generation involves artificially creating new instances to augment the training set. There are two main approaches: data-augmentation and generative methods. Data augmentation applies small transformations to the input image and annotation, such as spatial, colour, or noise variations, to provide the model with different perspectives without altering the nature of the objects. Data-augmentation is almost a mandatory step in any deep learning methods. While data augmentation frameworks like Albumentations (https://github.com/albumentations-team/albumentations) [143] have been well-established for 2D images, their adaptation for 3D images is a relatively recent development with frameworks such as batchgenerator (https://github.com/MIC-DKFZ/batchgenerators) [144], rising (https://rising.readthedocs.io/en/stable/), pymia (https://pymia.readthedocs.io/en/latest/) [145] or TorchIO (https://torchio.readthedocs.io/index.html) [146]. Generative methods aim to augment small datasets by generating artificial images and corresponding annotations. A first example for 3D nucleus segmentation is CytoPacq [100] a non-deep learning and online easy-to-use software. Generative Adversarial Networks (GANs) [147] are deep learning methods designed to generate realistic images from limited data [148]. They have successfully been applied to 3D nucleus image generation with both their original version [149]–[151] and a derivatives such as CycleGAN [152].

Weak supervision tackles the challenge by training a network using noisy or partial annotations. Instead of requiring detailed annotations for each instance, the annotator may only provide minimal or incomplete annotations. For instance, instead of fully delineating the boundaries of objects, the annotator may only mark a small spot in the centre of each object to be segmented [101], [153], [154]. These methods use the last feature maps of the model to retrieve what it “looks at” in an image when classifying it. Weak supervision may also rely on partially annotated data, such as in [155] where only bounding boxes of nuclei and limited manual segmentation are annotated in each training image.

Active learning (or *human-in-the-loop*) involves initiating the training process of an artificial neural network with only a limited number of annotations, while iteratively requesting the human annotator to provide annotations for selected images. These images are strategically chosen to maximize their relevance and information content for effective training. For example, images may get selected from rarely occurring

classes or from the edge of different classes. Active learning have been applied to medical image analysis [156] while for nucleus results can be found on classification [157] and on whole slide image segmentation [158]. DeepCell [127] and Cellpose [73] developers have both released an active learning tool they used to quickly annotate large datasets of cells and nuclei for 2D image segmentation. However, no results have been found for 3D bioimages. In the future, recent developments in fundamental research, such as Segment Anything [101], could yet greatly benefit the domain if adapted to 3D.

Transfer learning gives the model a general knowledge before being specialized. Generally, a base model, called *backbone*, is pretrained with a large annotated generic dataset such as ImageNet [42] and is then specialised by replacing or adding trainable layers [99], [159] and by being retrained on the specific task. Self-supervised learning, a similar two-step process, can be consider a sub-category of transfer learning. It is a novel solution involving pretraining the model on a vast collection of unannotated images using a pretext task. The pretext task is an artificial task, such as predicting image rotations or solving a jigsaw puzzle, that helps the network acquire general knowledge about the underlying objects, such as edge detection, shape understanding, and composition. The pretrained model is then fine-tuned on the desired downstream task, such as segmentation, leveraging the acquired prior knowledge to reduce the reliance on extensive annotations. Self-supervised learning methods have been successfully applied to nucleus segmentation in 2D [77], [160]. Recent developments have demonstrated very promising results in computer vision such as premises of unsupervised instance segmentation [161] based on Vision Transformers [63]. These last improvements must still find their way to 3D bioimaging. Furthermore, it is worth noting that the term *semi-supervised learning* is deliberately excluded here, as it encompasses a simultaneous approach where both the pretext task and the downstream task are performed concurrently. Consequently, in the frame of this thesis semi-supervised learning is considered a subset of self-supervised learning.

2.4 Conclusion

Deep learning methods have revolutionized image analysis by achieving unprecedented levels of accuracy and automating feature selection. These methods now only require human intervention in dataset creation and model design. This shift in focus has also been extended to the code-sharing process. If a method publication

and its code are the DNA of deep learning, then the datasets, trained model, documentation, and development environment are the epigenetic signals required for its proper functioning. Without these additional components, the solution becomes impractical to reuse.

Bridging the gap between biologists and computer scientists can be accomplished in several ways. Biologists can contribute by sharing their datasets publicly, providing valuable resources for deep learning engineers. On the other hand, computer scientists can enhance collaboration by sharing their methods, complete with all necessary materials and ideally with user-friendly interfaces. Ultimately, probably the most effective way to bridge this gap is through interdisciplinary work combining expertise. Encouraging good practices in specialized method development and sharing can lead to more accessible and adaptable deep learning solutions for biological image analysis.

Authors' contributions – cf. Appendices.

Chapter 3

Biom3d, an easy-to-use and modular framework for 3D segmentation methods

Well-designed components are easy to replace.
Eventually, they will be replaced by ones that are not so easy to replace.

-"Sustrik's Law"-

Chapter plan – First, nnU-Net limitations (3.1), then code philosophy (3.2) and then hierarchical description of the software functionalities (3.3-3.5)

3.1 Biom3d philosophy

To counteract the "publish and perish" tendency in the field of deep learning [162], a new development philosophy will be defined in this Chapter. A novel framework following this philosophy, named Biom3d, is *code sustainability*. As deep learning is a fast-moving domain, maintaining a sustainable code is a challenging aim requiring a highly adaptable code architecture. It means that the code must be organized in easily replaceable blocks organized along a clear backbone. Even though this Chapter focuses on Biom3d, the proposed principles for code sustainability presented here might be applicable to a broader scale. Therefore, this Section will only consider the general code philosophy behind Biom3d, voluntarily and momentarily setting aside the specific image segmentation goal.

Figure 3-1 – **Logo of Biom3d.**

One of the starting points for code sustainability is to broaden the spectrum of potential users, notably those addressed by methods such as nnU-Net (Figure 3-2). Not only because this can increase the user community, but also because it will involve both end-users and developers. This is particularly true for software inherited from fundamental research, which attempts to reach applied research. Involving end-users will ensure that the software meets the needs of an external community and has concrete applications. This gives it both meaning and feedback for improvements. On

the other hand, involving external developers will attract novel ideas while strengthening a community of maintainers.

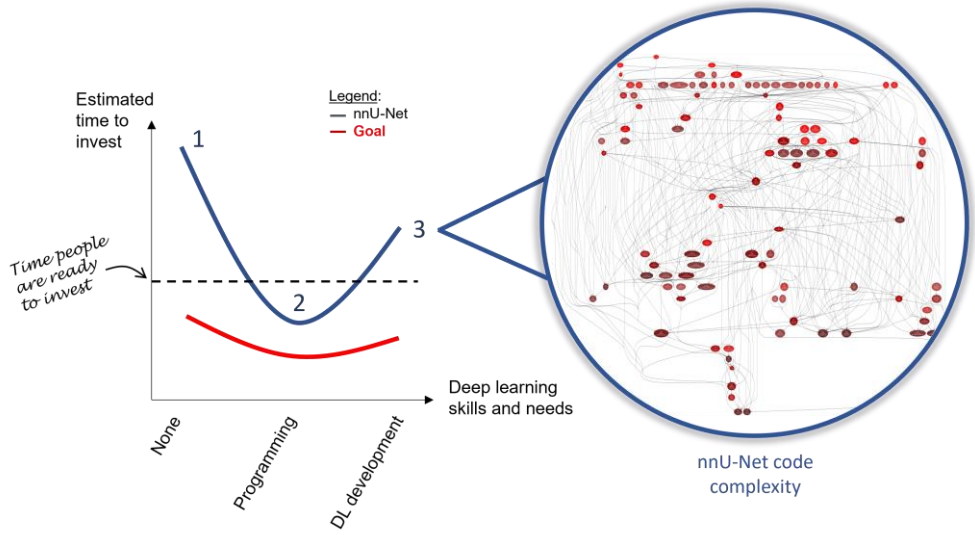


Figure 3-2 – **Schematic representation of nnU-Net limitations.** In the abscissa are conjointly represented the skills and needs of a deep learning user while in the ordinate is represented the estimated time required to use the method. nnU-Net (*blue*) is accessible for only a limited spectrum of users due to: **(1)** the lack of a graphical user interface, **(2)** its adaptation to Nifti format only and **(3)** its poor code readability, with a lack of developer documentation, and modularity, which weakened code resiliency and limit code resuse. The ideal method (*red*) should thus be accessible to the whole sprectrum of users. On the right is represented the internal dependencies of nnU-Net, each red node is a Python script of the method, and each connection represents a mutual dependency between two scripts. This network, obtained with *pydeps* package (https://github.com/thebjorn/pydeps), is an example of highly coupled code.

For each of these user profiles, the objective was to substantially reduce the time required to invest in using a deep learning method, while simultaneously expanding the scope of accessible functions. The intention was to prioritize usability to the same extent as functionality. The interpretation of software usage varies significantly based on the user's profile. Therefore, throughout the development process, the subsequent user profiles were established:

- **Non-Programmers** typically lack programming experience but possess basic IT knowledge. They anticipate a straightforward installation process facilitated

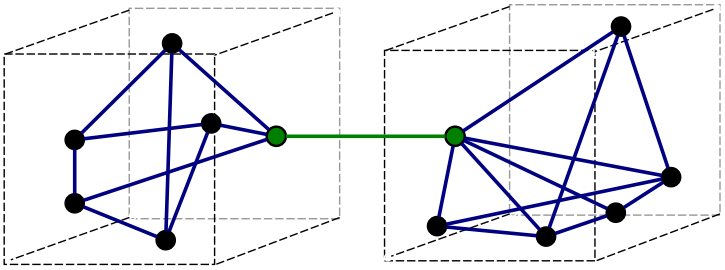
by downloading software from a website and expect an intuitive Graphical User Interface.

- **Python Programmers** encompass users who possess a foundational understanding of Python programming and are familiar with installing and employing Python libraries. They expect either a Command Line Interface (CLI) or an Application Programming Interface (API) featuring straightforward arguments. They anticipate having the capability to fine-tune parameter settings more precisely than is feasible through a GUI alone.
- **Deep Learning Programmers** possess an advanced understanding of deep learning theory and are well-versed in working with libraries like PyTorch or TensorFlow. They desire a well-documented, coherent, and modular deep learning framework. They may also seek to comprehend the codebase and easily undertake tasks such as adding, removing, or modifying sections of code.

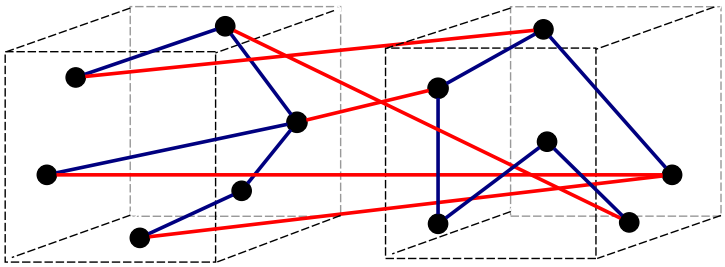
Biom3d development philosophy consists in fulfilling these user expectations while providing as many functionalities as possible. Current state-of-the-art methods usually target only one of these user profiles: nnU-Net, Cellpose or Stardist target Python Programmers, while ZeroCostDL4Mic or DeepImageJ are mainly addressed to Non-Programmers. In Biom3d, in addition to clear installation and interfaces, the development philosophy encompasses two fundamental aspects: code clarity and code modularity.

- **Code Clarity**: The code should remain comprehensible from its overall architecture to its individual lines. To aid new users in understanding the codebase, it can be presented in a hierarchical, tree-like structure. This approach starts with high-level, general functions and progressively delves into more granular details and functionalities, facilitating a gradual and systematic comprehension of the code.
- **Code Modularity**: Code components should be as self-contained and independent as possible (Figure 3-3). In computer science, this concept is also known as *code cohesion* and is opposed to *code coupling* [163]. This approach allows programmers to seamlessly incorporate or remove specific code components without causing undue disruptions to the overall system. Achieving code modularity often involves a delicate balance. A highly interconnected code (with high *code coupling* but low *code cohesion*) can be

efficient by using shared internal functions across different components. This can lead to improved performance and reduced code redundancy. However, this level of interconnection can make the codebase more sensitive to changes. Alterations to one section of the code might unintentionally impact other areas, complicating maintenance, and updates. Even though highly cohesive codes tend to be more durable, finding the right balance between modularity and interconnectedness can be essential to ensure both code efficiency and resilience.



(A) Good (loose coupling, high modularity/cohesion)



(B) Bad (high coupling, low modularity/cohesion)

Figure 3-3 – **Code modularity. (A) Good code modularity**. Code components are clearly isolated and have a clearly defined link. **(B) Bad code modularity**. Code components have a low internal cohesion and are strongly intertwined. Illustration adapted from https://commons.wikimedia.org/wiki/File:CouplingVsCohesion.svg.

3.2 Biom3d performances on image segmentation

Biom3d in its default configuration is designed to segment a large variety of 3D objects, in images potentially having multiple channels and simultaneously presenting objects from different classes. These volumetric images can originate from CT-scan, MRI, confocal microscope, or electron microscope. Biom3d matches and sometimes exceeds nnU-Net performance on wide range of applications (Table 3-1).

Medical applications. The datasets used to benchmark Biom3d were extracted from well-known online challenges, namely the Medical Segmentation Decathlon (MSD)

[142] and the Multi-organ Abdominal challenge (often called BTCV challenge) [164]. Both include sets of pairs of image and annotation for training and sets of images only for testing. All files are in Nifti format. The MSD challenge is a set of ten segmentation challenges of various human organs captured with CT-scan and MRI. BTCV challenge contains 30 CT-scans of abdomens where 13 organs of various sizes were annotated. As commonly done with online challenges, challengers are expected to submit their predictions of the test dataset on the online platform. However, the evaluation process can take up to two days to be released which may significantly slow down the development process of Biom3d. Moreover, the MSD only accepts submission done overall of its ten datasets which also requires a long training time. This training time is estimated at 10 days for a single model training on all ten datasets with a single GPU Nvidia RTX 3090 and may reach 50 days for a proper fivefold cross-validation. With the computational resources available for this project, this was soon found to be impractical and alternative solutions were found.

First, all the medical segmentation datasets were split into two equal parts, one half to constitute a new training set and another half for the testing set. This way, the testing set has now annotations that can be used to rapidly evaluate a new method. It is worth noting that the testing set did not serve as a validation set during training but only for final evaluation when the training was done.

Second, for the MSD, which contains ten datasets of different organs capture with CT-scan and MRI, only three representative datasets were selected: the Lung dataset (Task06 of the MSD), the Pancreas dataset (Task07), and the BrainTumour dataset (Task01). Judging by the accuracy of the best state-of-the-art methods, the Lung dataset (and the Colon dataset) is one of the hardest single-class tasks from the MSD. The Pancreas dataset is a challenging two-class dataset of CT-scans used, for instance, to assess self-supervised methods in [165] and it will be reused for the same purpose in Chapter 4. The BrainTumour dataset is both a multi-class and multi-channel set of MRI which originates from another well-known medical segmentation challenge, the BraTS challenge [37].

On all these datasets, the Dice scores of Biom3d exceed those of nnU-Net (Table 3-1). The main reason are probably better choices in the data-augmentation pipeline. nnU-Net scores were obtained by training it on a single subset (*fold*) on these datasets using its publicly available implementation (https://github.com/MIC-DKFZ/nnUNet).

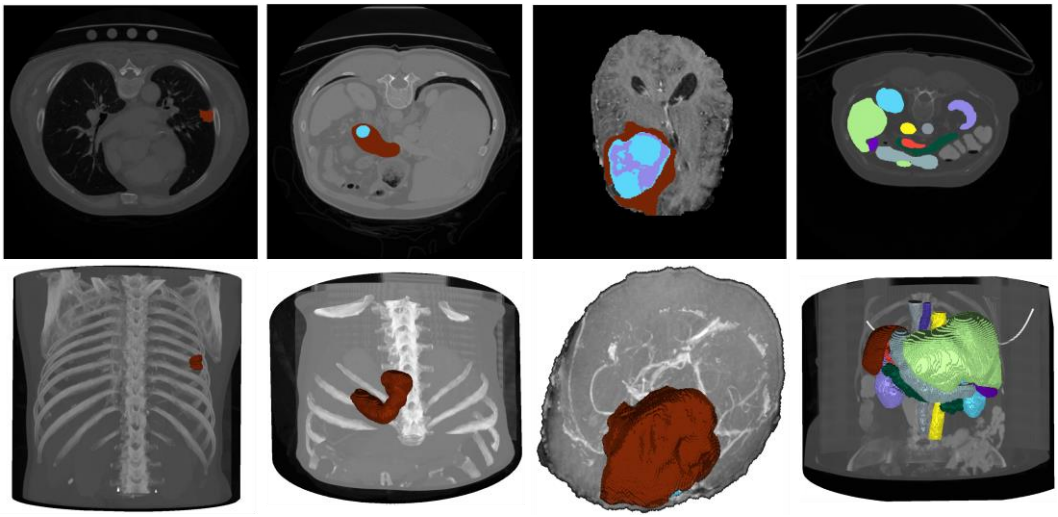


Figure 3-4 – **Samples of Biom3d predictions over medical datasets.** By column, from left to right: the Lung dataset, the Pancreas dataset, the BrainTumour dataset, the BTCV dataset. These views were captured using napari software [6].

Microscope applications. Biom3d was also evaluated on the nucleus dataset presented in Chapter 2 and reached similar Dice score as nnU-Net (Table 3-1). As the team in Oxford involved in the project was interested in electron microscope images of nuclei, evaluation of Biom3d was also conducted with this modality. For instance, Nadine Field, PhD student in the Oxford's team, captured 3D electron microscope images of *A. thaliana* roots to extract its nucleus characteristics (Figure 3-5-B). She thus annotated a section of a wide image of plant root to pass it to Biom3d. Unfortunately, her dataset as well as the only publicly available dataset of 3D electron microscope images (Table 2-3) [166] is composed of one image in the training set and one image in the testing set. nnU-Net requires at least 5 images in the training set due to cross-validation settings (it can be reduced to 2 images by changing the default settings but not to 1). Biom3d was thus upgraded with a novel pre-processing step to allow single image training: the image and its annotation mask are split in two volumes using a plane orthogonal to the largest axis (Figure 3-5-A). The new validation volume was arbitrarily chosen to be 20% of the original volume while the new training volume contains the remaining 80%.

An important limiting factor in predicted segmentation is noise (Figure 3-5-B). An existing strategy [86] is to detect in the training images, for each class of object, if there is a single connected component. If it is the case, then all but the largest connected component is kept for this class. This could be limiting in both cases presented in

Figure 3-5 because several objects are displayed per class (mitochondria on the top row and nuclei on the bottom row). To counteract this limitation, an additional post-processing strategy has been included in Biom3d removing volumes smaller than a threshold defined by the Otsu's method, originally applied here on volume distribution instead of intensity distribution (Figure 3-5-B). Both pre-processing and post-processing allows thus Biom3d to be well-adapted to electron microscope images. Finally, Biom3d was evaluated on the light microscope dataset of nuclei presented in Chapter 2 and used for benchmarking. As can be seen in Table 3-1 and on Figure 3-6-A, Biom3d achieved a mean and standard deviation of accuracy similar to those of nnU-Net.

Table 3-1 – **Biom3d overpasses nnU-Net on a variety of datasets.** The first four rows represent medical datasets while the last two rows represent biological datasets.

Dataset name	Modalities	Number of channels	Number of classes	Biom3d Dice score	nnU-Net Dice score
Lung	CT	1	1	**0.651**	**0.651**
Pancreas	CT	1	2	**0.566**	0.403
BrainTumour	MRI	4	3	**0.723**	0.719
AbdomenOrg	CT	1	13	**0.846**	0.824
MitoEM	EM	1	1	**0.907**	-
NucleusConf	Confocal	1	1	**0.914**	0.912

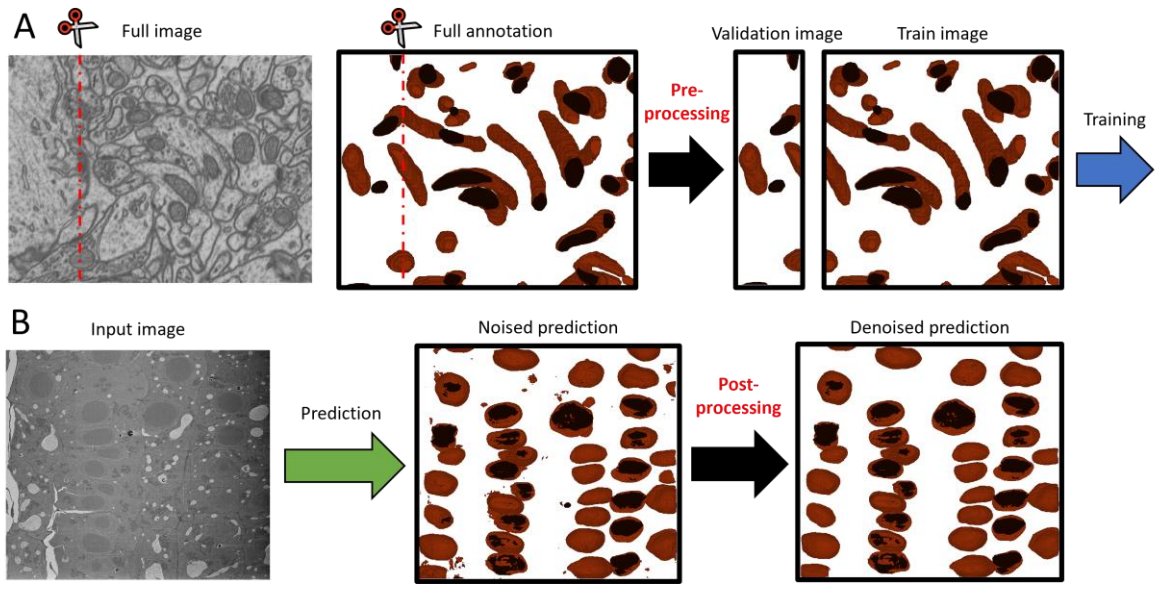


Figure 3-5 – Additional pre- and post-processing of Biom3d to cope with biological data. (A) Additional pre-processing to deal with single image dataset: the training image and its corresponding annotation are split in two new portions, one for training and one for validation. The images and annotations come from [166]. (B) Additional post-processing to remove prediction noises: only the biggest connected components are kept. The Otsu's algorithm is applied on the volume distribution to find the minimal valid volume. The image has been captured by Nadine Field in Oxford Brookes.

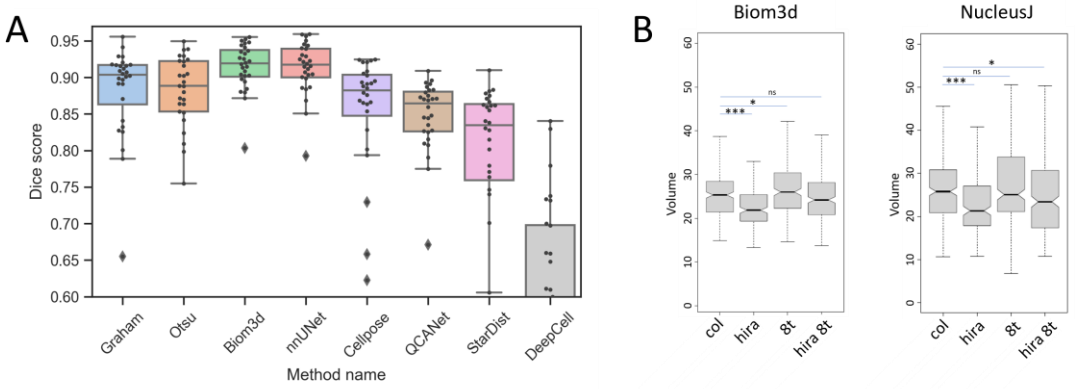


Figure 3-6 – Biom3d performances on the nucleus datasets. (A) Completed benchmarking of Chapter 2 with Biom3d performances: Biom3d reaches similar accuracy as nnU-Net. (B) Comparison of Biom3d and NucleusJ on 4 nucleus datasets: the higher accuracy of Biom3d predictions demonstrates a lower variability in nuclei volumes. "Col" are wild-type nuclei while "hira" and "8t" represent mutant nuclei. "Hira 8t" are double mutant nuclei.

3.3 Biom3d, an easy-to-use tool

The first type of users of Biom3d are Non-Programmers, end-users such as biologists or radiologists. This profile expects an easy-to-use interface and a minimum amount of manual configuration to achieve their goal. Biom3d integrates thus several Graphical User Interfaces depending on their means and needs.

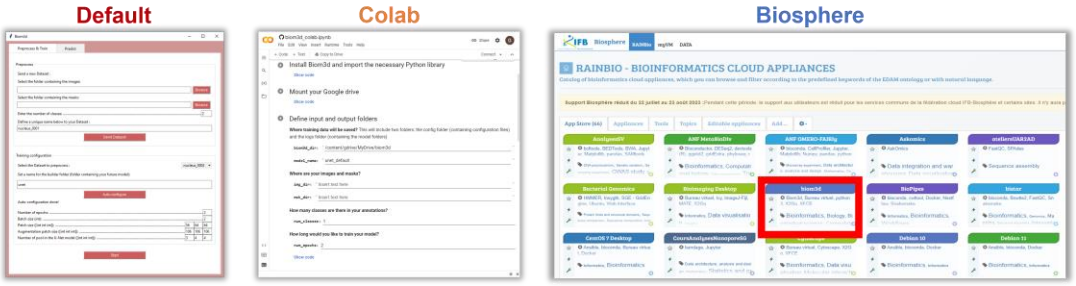
3.3.1 Graphical User Interfaces

Biom3d has three Graphical User Interfaces (Figure 3-7-A):

- **The Default interface**, integrated in the software
- **The Google Colab interface** (https://colab.research.google.com/github/GuillaumeMougeot/biom3d/blob/master/docs/biom3d_colab.ipynb)
- **The Biosphere interface** (https://biosphere.france-bioinformatique.fr/catalogue/appliance/216/)

Each interface has been designed to respond to different needs. An overview of the advantages and disadvantages of each interface is presented in Figure 3-7-B. The Default interface is a local installation and boasts the quickest usability. As everything remains offline, users retain complete control over data transfers while upholding data ownership. However, utilizing this interface necessitates access to a GPU equipped with at least 10GB of VRAM for training and 4GB of VRAM for prediction, entailing the installation of deep learning computation libraries and drivers (such as CUDA and CuDNN for Nvidia GPUs). To address these limitations, Biom3d comes with two additional online interfaces bypassing the need for specific computing resources or installations. The Google Colab interface harnesses Google's servers which are freely accessible for a finite period. Since data is processed directly on these servers, it requires uploading the data. However, transferring substantial datasets to private servers might be challenging due to space limitations or ownership concerns. A final solution, yet limited to French public research groups, involves utilizing the Biosphere interface. Biosphere is a public initiative to mutualized access to the national computing centres and open-source software developed within the research community. While in this case data ownership remains intact, the current version of Biosphere is restricted to CPU usage, resulting in slower processing speeds. Consequently, this alternative is best suited for predictions or demonstrations only.

A



B

Interface	Fast	Unlimited access	Fast data transfer	Control over data ownership	No GPU requirements	Training parameter editing	No Linux knowledge for installing	No Linux knowledge for starting
Default	✓	✓	✓	✓	✗	✓	≈ (required for remote)	✓
Colab	✓	✗ (6 free hours)	✗	✗	✓	≈ (with programming)	✓	✓
Biosphere	✗ (no GPU)	≈ (French community only)	✗	✓	✓	✓	✓	✗

Figure 3-7 – **Three interfaces of Biom3d. (A) Views of the three interfaces. (B) Comparison of the three interfaces.** In green and red circles are highlighted the main advantages and disadvantages of each interface.

3.3.2 Workflows: Training and prediction

This Subsection will describe in more details some of the hidden mechanisms behind Biom3d Interfaces. Explanations will be illustrated with the Default Interface, but these mechanisms are independent from the chosen Graphical Interface. Two main workflows can be described: one for training (Figure 3-8) and one for prediction (Figure 3-9).

Training workflow. The goal of this workflow is to output a trained model and a pre-processing methodology adapted to a specific dataset. To do so, the user inputs a set of 3D images, annotated with a software such as napari. The number of classes of objects and the name of the output folder containing the future deep learning model must be filled. Once the "Auto-configuration" button is pressed, a cascade of operations starts (Figure 3-8).

First, the dataset is scanned, and a set of characteristics, called a *data-fingerprint*, is extracted, such as the median image size or the median spatial sampling.

Second, the data-fingerprint is used to preprocess the raw images: image intensities are z-normalized (using mean and standard deviation), and images are resized to align with the median spatial sampling.

Third, the data-fingerprint is then used to compute some of the training hyper-parameters such as the batch size, the patch size, or the number of successive pooling layers in the U-Net model. The data-fingerprint and the training configurations are both stored in a single *config.yaml* file. To maintain user oversight of training configurations, these settings are displayed in the interface and can be edited. This feature proves beneficial, especially for variables like the number of epochs. In the default nnU-Net implementation, this is arbitrarily set at 1000 epochs, which may require adjustment for scenarios like confocal images where fewer epochs suffice. Similarly, manual modification of hyperparameters might be necessary when users opt to annotate objects solely within a specific image region, leaving other areas unannotated. For instance, suppose objects in only the first z-slices of the image are annotated. Currently, patch size is determined solely by the median image shape. Consequently, in anisotropic images with substantial dimensions along the x- and y-axes but modest dimensions along the z-axis, the patch size retains a similar aspect ratio, yet with reduced dimensions compared to the original image. During training, patches are positioned randomly within the input image. If the patch dimensions do not align with the original image's z-axis, the patch might be placed randomly in an annotated object region or in a region with analogous properties but without annotations. This inconsistency can lead to erratic behaviour in the training curve. The z-dimension of the patch might thus be adapted manually. Such changes must still consider the dimensions of the GPUs memory not to overload it.

Fourth, once configured the training can start by pressing “Start” button. The training curves, the best deep learning model and sample of predictions are automatically saved every epoch in the output folder.

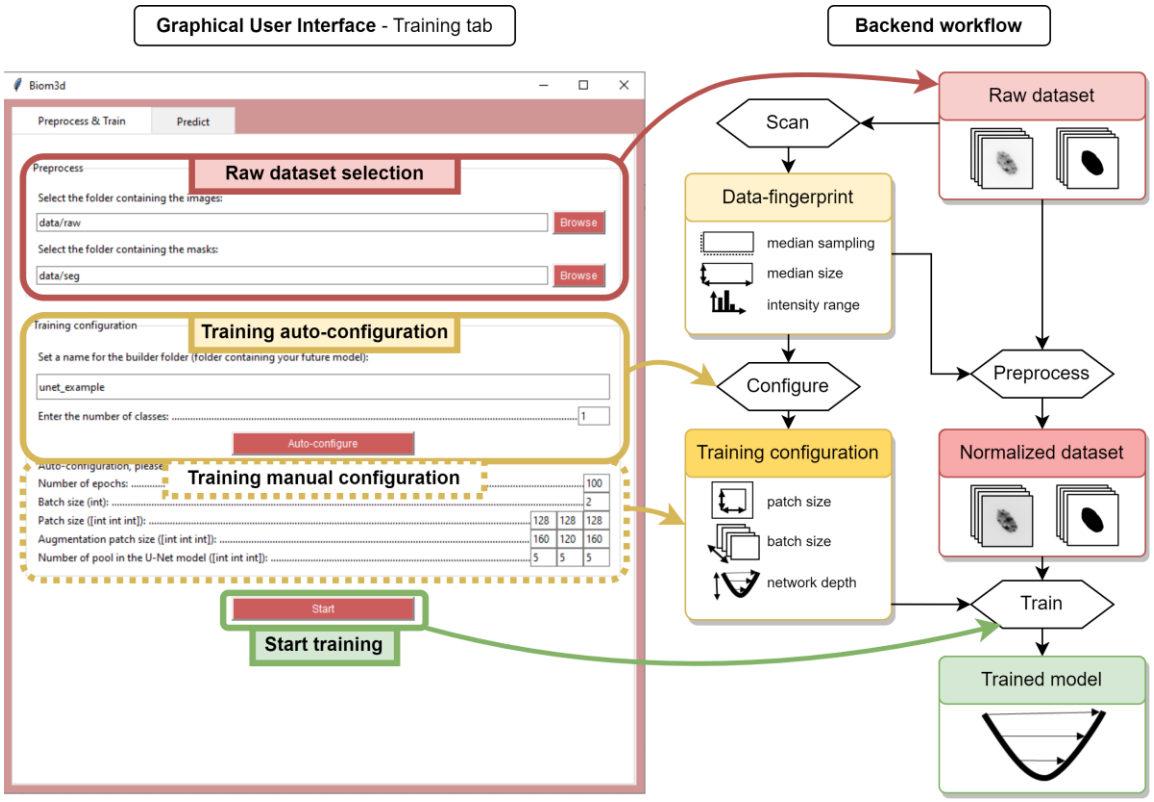


Figure 3-8 – **Training workflow of Biom3d. (*Left*) Training tab of the graphical interface.** The user specifies the path to the folders containing training images and annotations, then defines a name for the configuration file and the future trained model, and the auto-configuration can start. Automatically defined parameters can be adjusted manually if needed before starting the training. **(*Right*) Backend workflow.** Once the "Auto-configuration" button is pressed, the data-preprocessing starts. The dataset key elements (median shape, etc.) are extracted and used to normalize all the images and to define training configuration (patch size, etc.). If the "Start" button is pressed, a deep learning model will be trained and saved along with the pre-processing methodology (data-fingerprint).

Prediction workflow. Once trained, a model can be used on novel images similar to the ones of the training set (Figure 3-9). To match the training pre-processing, the pre-processing of the prediction images is configured using the data-fingerprint stored in the training output folder. The images are then passed to the deep learning model. Predictions are eventually post-processed before being saved. The prediction workflow is much simpler and can be used to rapidly predict thousands of images.

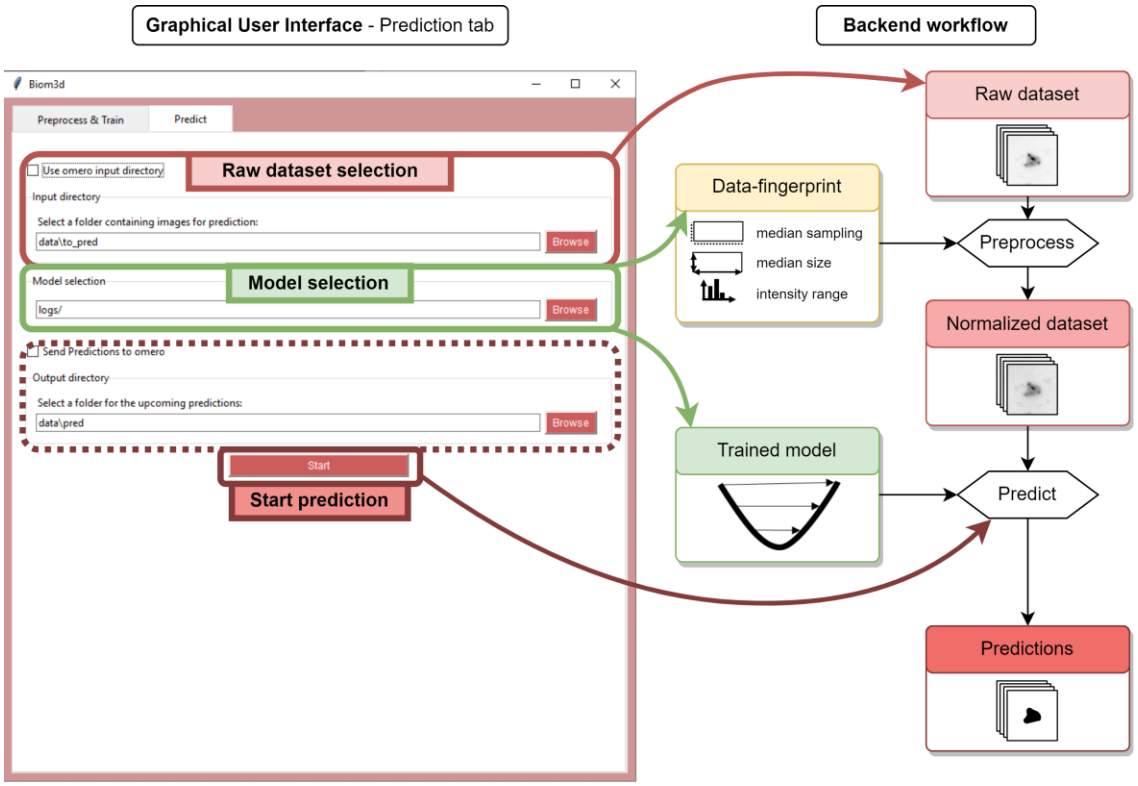


Figure 3-9 – **Prediction workflow. (*Left*) Prediction tab of the graphical interface.** The user chooses a folder containing raw images, the path to a trained model and a folder for the future predictions. The predictions start when the "Start" button is pressed. **(*Right*) Backend workflow.** The raw images are normalized using the data-fingerprint of the training dataset. The trained model is then loaded and used to compute predictions.

3.3.3 Pooling resources: Remote access and OMERO access

As previously mentioned, having access to a GPU large enough to train a deep learning model is not always affordable to a single research team. On the other hand, if a single team has access to a GPU, this resource might be underexploited, as training a deep learning model is generally not required daily. Pooling computing resources might therefore be beneficial economically and environmentally. Furthermore, biological images generally tend to require large storage capacities. Pooling storage resources as well could yield similar advantages, in addition to contributing to the preservation of data integrity.

Remote access. Remote access is integrated as a core functionality of Biom3d (Figure 3-10). When opening the Default interface, users can choose between the local version or the remote version. The Remote version requires the existence of a remote

server where the Python package of Biom3d is installed. This server must have a compatible GPU and must be accessible via Secure Shell (ssh). Two independent installations are thus required: one for the client computer with the graphical interface and one for the server computer with the command line interface (see Section 3.4.1 for more details about the command line interface). In the graphical interface, the user indicates server address, username, password and eventually the name of the Python or Anaconda environment where Biom3d has been installed. Optionally, Biom3d can be accessible via a proxy server or a Virtual Private Network (VPN) so to be accessible via the Internet. Once connected, the remote version of the graphical interface looks like the local version, except for the fact that the dataset must now be sent to the server before pre-processing. The remote prediction process is like the local one except that data must be first sent to the computing server before prediction and the results must be retrieved after prediction.

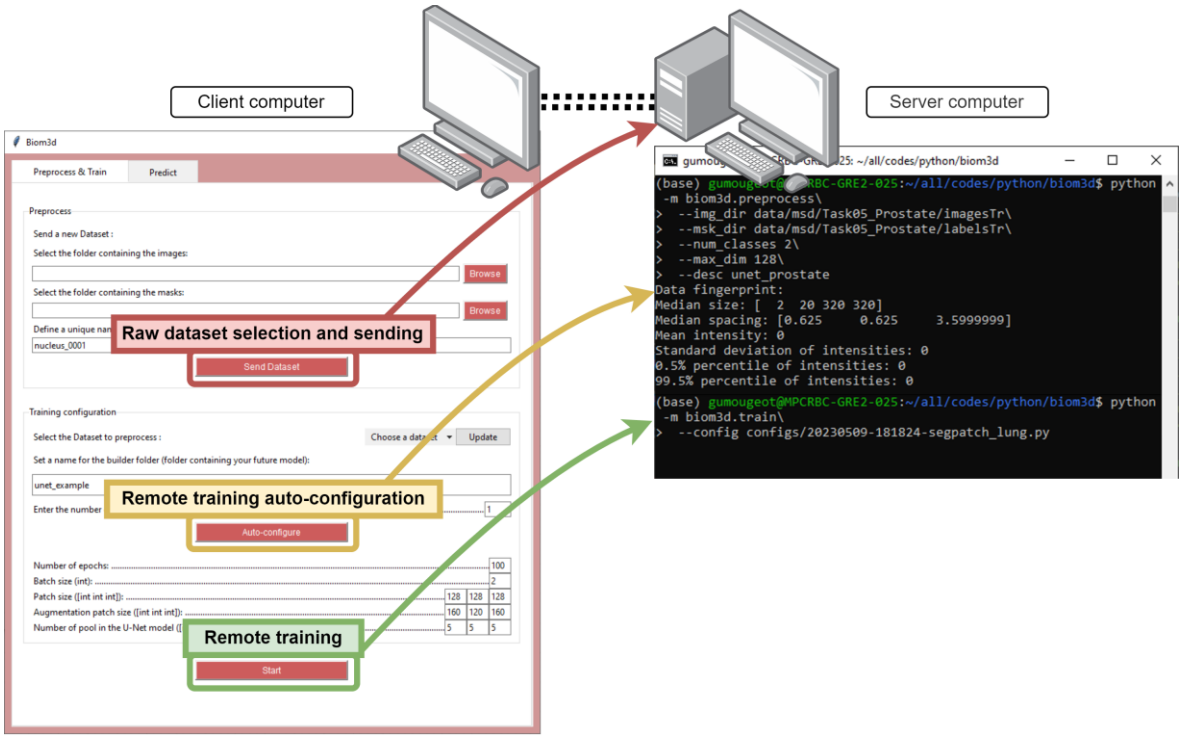


Figure 3-10 – **Remote access to Biom3d. (*Left*) The Default Interface in remote version** looks like the local version, actions being executed on the remote server. **(*Right*) The server computer** is equipped with a GPU and the Python package of Biom3d and run the computations ordered by the graphical interface.

OMERO access. To pool storage resources in biology, one of the tools in use is OMERO. OMERO is a software designed to store multi-channel 3D images in a single

repository designed to preserve data integrity. The client interface and web interface of OMERO both allow organization of images and viewing without the need to download them locally. OMERO does not handle complex image analysis but provides ways to download them on a computing server. Doing so in Python is not straightforward but possible (it was necessary to directly contact OMERO's authors to understand and write the appropriate commands) and has been integrated in Biom3d code and interface. Biom3d can download raw images from the storage server to the computing server once the OMERO server address, user details and dataset identification number have been uploaded to the interface. Once downloaded, the prediction process can start on the computing server. Once finished, the prediction can either be downloaded locally or sent back to the OMERO server. Uploading back a set of predictions on an OMERO server is a bit more complicated than downloading it due to image metadata. As presented in the previous Chapter, the metadata is the set of information surrounding the captured image among which the most notable one is the image spatial sampling as it allows spatial measurements to be made. The original metadata of the input image must thus be integrated into the prediction results. Microscopy images are usually stored in TIFF format which have very diverse ways to store metadata. This diversity has been the subject of extensive research during this project and will be discussed in the following paragraph.

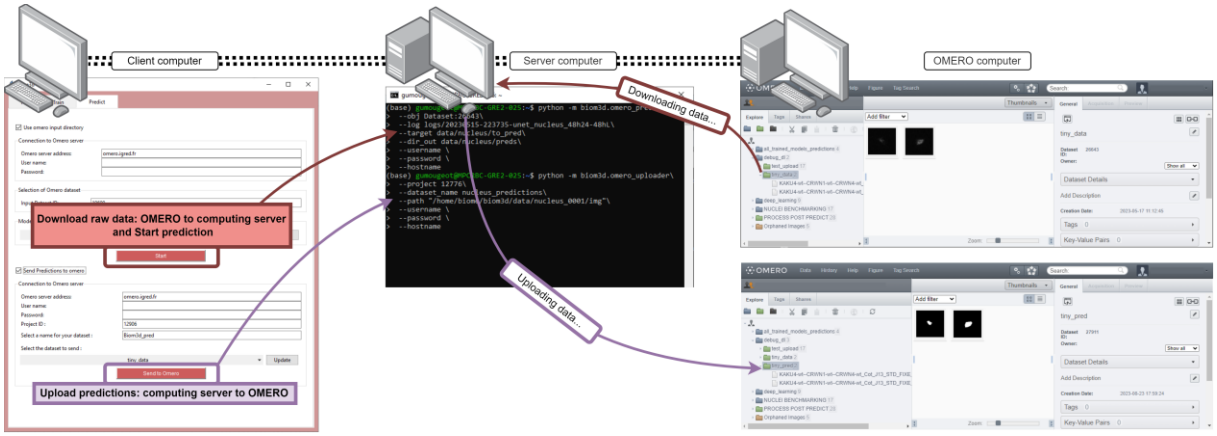


Figure 3-11 – OMERO access. (Left) The remote version of the Default interface allows to download an OMERO dataset.

TIFF formatting. TIFF stands for Tag Image File Format meaning that image metadata is stored in *tags*. A tag consists of a small set of various pieces of information among which the most important ones are the tag code, name, and value (see Table 3-2

for an example of image metadata). There is a finite set of possible tag name (a complete list can be found here: https://www.loc.gov/preservation/digital/formats/content/tiff_tags.shtml). For example, "XResolution" tag contains the number of pixels per spatial resolution unit along the x-axis. Despite extensive effort of TIFF format creators to make the list of tag exhaustive, this cannot match the diversity of possible configurations in all imaging devices (especially when knowing that TIFF format was originally designed for fax communication and that it is now used to store multi-channel and multi-dimensional images). Microscope constructors have found a stratagem to counteract this limitation: exploiting the "ImageDescription" tag. This TIFF tag has quickly become a catch-all tag, used to store, if not all, most of the information related to image capture. For example, Leica MM AF microscope uses this tag to store a list of information in an XML-like format, where can be found, for instance, the x-axis resolution (see Table 3-2 "spatial-calibration-x") instead of using the official "XResolution" tag. Every imaging constructor has developed their own TIFF metadata formatting technique. Even software developers have decided to follow their own interpretation of the TIFF tag philosophy. Even open-source software, such as FIJI/ImageJ, have fallen into this pitfall. This phenomenon has caused the community to dub TIFF files as "Thousands of Incompatible File Format". Yet, some efforts have emerged attempting to unify TIFF derivatives, at least for biological data, among which the most noticeable one is probably Bio-Formats. Bio-Formats is an initiative started by the Open Microscopy Environment (OME) group. This consortium of public and private institutions creates open-sources software and formats for microscopy, such as the OMERO software. Bio-Formats is a tool allowing to read most of the industry TIFF formats and to convert them into a unique universal format called OME-TIFF. However, this initiative is limited to Java programs and could not be integrated into Biom3d without requiring starting the "Java Virtual Machine" which was considered an excessive use of resource for a relative small problem. The solution currently integrated into Biom3d is the result of tedious research strewn with many frustrating failures resulting in a compromise. It is based on the *tifffile* Python package developed by Christoph Gohlke. This package is well-adapted for FIJI/ImageJ TIFF and OME-TIFF but currently fails to retrieve relevant information (spatial resolution, etc.) in proprietary formats such as the one presented in Table 3-2. Most of the code integrated in Biom3d has been inspired by contributions of tifffile's author (among which this post has been determinant: https://forum.image.sc/t/python-copy-all-

metadata-from-one-multipage-tif-to-another/26597/8). This method is certainly a major point of improvement in Biom3d and more generally to Python image reader and writer. It might require extensive effort to understand the structure of TIFF files to preserve metadata integrity when passing them to segmentation prediction.

Table 3-2 – **Example of TIFF metadata obtained with a Leica MM AF microscope.** The "ImageDescription" tag departs from the official tag system and contains most of the information related to the microscope configuration. Such formatting is microscope-dependant. This metadata was extracted using tifffile Python package.

Tag code	Tag name	Tag value
254	NewSubfileType	FILETYPE.PAGE
256	ImageWidth	2048
257	ImageLength	2048
258	BitsPerSample	16
259	Compression	COMPRESSION.NONE
262	Photometric Interpretation	PHOTOMETRIC.MINISBLACK
270	ImageDescription	<prop id="MetaDataVersion" type="float" value="1"/> <prop id="ApplicationName" type="string" value="MetaMorph"/> <prop id="ApplicationVersion" type="string" value="7.8.12.0"/> <PlaneInfo> <prop id="plane-type" type="string" value="plane"/> <prop id="pixel-size-x" type="int" value="2048"/> <prop id="pixel-size-y" type="int" value="2048"/> <prop id="bits-per-pixel" type="int" value="16"/> <prop id="autoscale-state" type="bool" value="on"/> <prop id="autoscale-min-percent" type="float" value="0.05"/> <prop id="autoscale-max-percent" type="float" value="0.05"/> <prop id="scale-min" type="int" value="14"/> <prop id="scale-max" type="int" value="7011"/> <prop id="spatial-calibration-state" type="bool" value="on"/> <prop id="spatial-calibration-x" type="float" value="0.1032"/> <prop id="spatial-calibration-y" type="float" value="0.1032"/> <prop id="spatial-calibration-units" type="string" value="um"/> <prop id="image-name" type="string" value="k4c1c4_cot1&4&8__w11 DAPI SIM variable_s1"/> <prop id="threshold-state" type="string" value="ThresholdOff"/> <prop id="threshold-low" type="int" value="0"/> <prop id="threshold-high" type="int" value="65535"/> <prop id="threshold-color" type="colorref" value="4080ff"/> <prop id="zoom-percent" type="int" value="50"/> <prop id="gamma" type="float" value="1"/> <prop id="look-up-table-type" type="string" value="by-wavelength"/> <prop id="look-up-table-name" type="string" value="Set By Wavelength"/> <prop id="photonegative-mode" type="bool" value="off"/> <prop id="gray-calibration-curve-fit-algorithm" type="int" value="4"/> <prop id="gray-calibration-values" type="float-array" value=""/> <prop id="gray-calibration-min" type="float" value="-1"/> <prop id="gray-calibration-max" type="float" value="-1"/> <prop id="gray-calibration-units" type="string" value=""/> <prop id="plane-guid" type="guid" value="{168AEE33-9914-4B20-A4F2-B9085DC714B9}"/> <prop id="acquisition-time-local" type="time" value="20180523 14:22:20.702"/> <prop id="modification-time-local" type="time" value="20180523 14:27:41.885"/> <prop id="camera-binning-x" type="int" value="1"/> <prop id="camera-binning-y" type="int" value="1"/> <prop id="camera-chip-offset-x" type="float" value="0"/> <prop id="camera-chip-offset-y" type="float" value="0"/> <prop id="_IllumSetting_" type="string" value="1 DAPI SIM variable"/> <prop id="_MagNA_" type="float" value="1.4"/> <prop id="_MagRI_" type="float" value="1.515"/> <prop id="_MagSetting_" type="string" value="63x Oil"/> <custom-prop id="Camera Bit Depth" type="float" value="16"/> <custom-prop id="Electron Count Conversion Factor" type="float" value="0.49"/> <custom-prop id="Electron Count Conversion Offset" type="float" value="100"/> <custom-prop id="Exposure Time" type="string" value="60 ms"/> <custom-prop id="Leica Condenser Top" type="string" value="Condenser Top Out"/> <custom-prop id="Leica Condenser Turret" type="string" value="BF"/> <custom-prop id="Leica Contrast Method" type="string" value="FLUO"/> <custom-prop id="Leica DIC Turret" type="string" value="-"/> <custom-prop id="Leica Filter Changer" type="string" value="405"/> <custom-prop id="Leica Fluor Intensity Manager" type="string" value="30%"/> <custom-prop id="Leica IL Aperture Diaphragm" type="float" value="7"/> <custom-prop id="Leica IL Shutter" type="string" value="Closed"/> <custom-prop id="Leica Immersion Mode" type="string" value="Immersion"/> <custom-prop id="Leica Lamp" type="float" value="18"/> <custom-prop id="Leica Lamp Switch" type="string" value="Incident"/> <custom-prop id="Leica Objective Turret" type="string" value="63x"/> <custom-prop id="Leica SI Diaphragm" type="float" value="6"/> <custom-prop id="Leica Stage X" type="float" value="44343"/> <custom-prop id="Leica Stage Y" type="float" value="4345.46"/> <custom-prop id="Leica TL Aperture Diaphragm" type="float" value="18"/> <custom-prop id="Leica TL Field Diaphragm" type="float" value="5"/> <custom-prop id="Leica TL Polarizer" type="string" value="TL Polarizer

		<prop id="stage-position-x" type="float" value="44343"/> <prop id="stage-position-y" type="float" value="4345.46"/> <prop id="stage-label" type="string" value="Position10"/> <prop id="z-position" type="float" value="106.859"/> <prop id="wavelength" type="float" value="450"/> Out"/> <custom-prop id="Leica TL Shutter" type="string" value="Open"/> <custom-prop id="Leica Z Motor" type="float" value="106.859"/> <custom-prop id="OptiGrid Paddle Z" type="float" value="5128"/> <custom-prop id="OptiGrid Voltage 1" type="float" value="37.216"/> <custom-prop id="OptiGrid Voltage 2" type="float" value="55.7118"/> <custom-prop id="OptiGrid Voltage 3" type="float" value="75.1461"/> <custom-prop id="OptiGrid Voltage 4" type="float" value="95.4526"/> <custom-prop id="Shutter Fluo" type="string" value="Open"/> <custom-prop id="Shutter Trans" type="string" value="Closed"/> <prop id="number-of-planes" type="int" value="75"/>
273	StripOffsets	(8, 8200, 16392, 24584, 32776, 40968, 49160, 57352, 65544, 73736, 819282…
274	Orientation	ORIENTATION.TOPLEFT
277	SamplesPerPixel	1
278	RowsPerStrip	2
279	StripByteCounts	(8192, 8192, 8192, 8192, 8192, 8192, 8192, 8192, 8192, 8192, 8192…
305	Software	MetaSeries
306	DateTime	20180523 14:22:20.702

3.4 Biom3d, a toolbox for bioimage analysts

As previously mentioned, the Default Interface of Biom3d (both remote and local) comes as a Python package which requires some basic Python understanding that will be presented in the first Subsection below. Additionally, Python Programmers needs might not be fulfilled by the Graphical Interfaces only. This type of user might be interested in understanding, exploiting, and controlling more complex aspects of Biom3d. For instance, they may be interested in accessing to a Command Line Interface to be able to execute Biom3d on a High-Performance Computing (HPC) resources in “batch mode” on a large variety of datasets. They may look also to fine-tune existing models or to make predictions using a set of models. All these problems can be solved with the Command Line Interface of Biom3d and will be addressed in the second Subsection below. Furthermore, Python Programmers might be interested in experimenting with a broader degree of freedom in the deep learning hyper-parameters. They might as well be interested in changing the optimizer parameters or the type of deep learning model. These challenges are tackled with the modular structure of Biom3d and presented in the third Subsection.

3.4.1 Installing Biom3d

More details about installation instructions can be found on Biom3d documentation website: https://biom3d.readthedocs.io/en/latest/installation.html

The Command Line Interface of Biom3d is accessible using a Windows command prompt or a Linux Terminal. Installing the Default interface of Biom3d requires the installation of the Biom3d Python package, locally for the local version, and on a Linux server for the remote version. The installation of the Biom3d package requires a computer with a GPU of at least 10 Gb of VRAM and with CUDA library, CuDNN library, Python3 and Pytorch library installed preferably in a virtual environment such as pip environment or Anaconda. A single line of code is then required to complete the installation:

```
(base) gumougeot@MPCRBC-GRE2-025:~$ pip install biom3d[all]
```

Once the installation is completed Biom3d is ready-to-use both for the Default interface and the Command Line Interface.

Creating a Python package. To facilitate its installation, Biom3d has been referenced in PyPI (Python Package Index, https://pypi.org/), instead of being integrated in a Docker or Anaconda environment. This technique allows to publicly and internationally reference Biom3d while making transparent all requirements (installed when running the install command). This setup was facilitated by the newest version of pip which only requires writing a single *pyproject.toml* file, preventing the previous need of using backend tools such as Setuptools. Python package creation has thus recently been simplified and documented by the PyPA community (more information can be found here: https://packaging.python.org/en/latest/tutorials/packaging-projects/). Python packaging is well adapted to seamlessly share and install Python tools. Yet, Biom3d is dependent on some external libraries (CUDA, CuDNN, Pytorch) and integrating it along these libraries in a single virtual environment might represent another important future improvement. To do so, Singularity is currently the candidate of choice. Like Docker, it isolates all dependencies and the appropriate operating system. Opposed to Docker, Singularity does it within a single file, which can then ease its sharing and use on computer clusters.

3.4.2 Command Line Interface

More details about the Command Line Interface can be found on Biom3d documentation website: https://biom3d.readthedocs.io/en/latest/tuto_cli.html

The code of Biom3d, as for most Python packages, is organized in Python scripts. Biom3d has two types of Python scripts: module scripts and executable scripts. Module scripts store lists of functions or classes used by the executable scripts. Details about module scripts will be given further in this thesis. To each executable script corresponds one Command Line Interface with the same name. For instance, the executable script *preprocess.py* oversees the preprocessing. It can be executed, from any location in the computer, once Biom3d Python package is installed, with the following command:

```
python -m biom3d.preprocess --img_dir data/images --msk_dir data/masks --
num_classes 1 --desc unet_example
```

Similarly, any of Biom3d command listed in Table 3-3 can be executed with the syntax:

```
python -m biom3d.script_name --arg1 arg1_value --arg1 arg2_value
```

Where `script_name` is the name of the Python script, `--arg1` is the name of an argument and `arg1_value` the value of this argument.

Table 3-3 – **Biom3d commands.** Each command corresponds to one executable script of Biom3d. Only the main arguments are listed. The complete list of arguments for each command can be found on Biom3d documentation website or displayed using the following: `python -m biom3d.script_name --help`.

Type	Script name	Description	Typical arguments	
			Name	**Role**

Main	`preprocess`	Image and mask preprocessing.	`--img_dir`
			`--msk_dir`
			`--num_classes`
			`--desc`
				Image directory.
				Mask directory.
				Number of classes of objects.
				Future name of the configuration directory and model.

	`train`	Model training or fine-tuning.	`--config` `--log`	Path to the configuration file created during preprocessing. Path to a log directory.

	`pred`	Prediction with one or several models on a dataset.	`--log`
			`--dir_in`
			`--dir_out`
				Path to one or several log directories.
				Path to the input image directory.
				Path to the output mask directory.

	`eval`	For test purpose: evaluate Biom3d predictions against annotations using Dice metric.	`--dir_pred`
			`--dir_lab`
			`--num_classes`
				Path to the prediction directory.
				Path to the annotation directory.
				Number of classes of objects.

	`preprocess_train`		`--img_dir`	Image directory.

Combined		Combination of preprocessing and training.	`--msk_dir` `--num_classes`	Mask directory. Number of classes of objects.
OMERO	`omero_pred`	Prediction from an OMERO dataset.	`--obj`	Either “Dataset:ID” or “Project:ID” where ID is the OMERO identification of the Dataset or the Project.
			`--log`	Path to a log directory.
			`--target`	Path to the directory where OMERO images will be downloaded.
			`--dir_out`	Path to the output mask directory.
			`--username`	OMERO username.
			`--password`	OMERO password.
			`--hostname`	OMERO server address.
			`--upload_id`	(Optional, work only for Dataset) ID of the OMERO Project to upload the predictions directory into.
	`omero_uploader`	Upload a dataset to OMERO	`--project`	ID of the OMERO Project to upload the predictions directory into.
			`--path`	Path to the directory to upload.
			`--dataset_name`	Future name of the OMERO Dataset containing the predictions.
			`--username`	OMERO username.
			`--password`	OMERO password.
			`--hostname`	OMERO server address.
	`omero_downloader`	Download an OMERO Dataset or Project.	`--obj`	Either “Dataset:ID” or “Project:ID” where ID is the OMERO identification of the Dataset or the Project.
			`--target`	Path to the directory where OMERO images will be downloaded.
			`--username`	OMERO username.
			`--password`	OMERO password.
			`--hostname`	OMERO server address.
GUI	`gui`	Launch the Graphical User Interface.	-	-

The command line interface has the same capabilities as the Graphical User Interface yet with three significant differences:

- The configuration file generated after the preprocessing is fully editable and will be the focus of the next Subsection.

- Command lines have many advantages notably in Linux-based Operating Systems such as to be executed in batch or on a computer cluster, or to be integrated in a bash script.
- A Command Line Interface user can fine-tune a model or execute a prediction with a set of models instead of one only.

Fine-tuning. Fine-tuning consists in retraining a pretrained model on a new dataset, particularly useful for transfer learning. To do fine-tuning with Biom3d, the following can be used:

```
python -m biom3d.train\
 --log logs/20230522-182916-unet_default\
 --config configs/20230522-182916-config_default.py
```

The two main arguments above encapsulate two important and original concepts of Biom3d:

- `--log` defines the location of the directory storing the trained model parameters (stored in a *.pth* file) along with all relative information such as training curves and snapshots or the configuration files. If the `--log` argument is employed alone, the `script_name` script will consider that the user wants to restart an existing training that might have been interrupted.
- `--config` is the location of the configuration file defining the training configuration (data preprocessing, optimizer, model architecture etc.). If the `--config` argument is used alone, Biom3d will create a new log folder and start a new training.

If `--log` and `--config` are used simultaneously, Biom3d will use the configuration file to configure a new training and the trained model parameters to initialize the novel model weights. Model architectures must be compatible.

Multi-model prediction. Another originality of Biom3d Command Line Interface is the possibility to execute predictions from a set of different models (*ensemble prediction*). Ensemble prediction has been proven to be particularly effective especially when working in teams such as for Kaggle competitions [126]. nnU-Net is limited to perform ensemble prediction within the frame of its cross-validation configuration. For example, with five folds cross-validation, this means that the

predictions of a set of five identical model architectures trained on the same dataset but with different training/validation splits will be merged to obtain the final segmentation result. In Biom3d, with its modular architecture, a user can decide to run predictions with an arbitrarily large set of model architectures and each trained on a different dataset by simply using the following command:

```
python -m biom3d.pred\
 --log logs/20230522-182916-unet_default logs/20230425-162133-unet_btcv\
 --dir_in data/btcv/Testing/img\
 --dir_out data/btcv/Testing/preds
```

In this example, some predictions are computed on the BTCV test set using two different models stored in two different *log* folders.

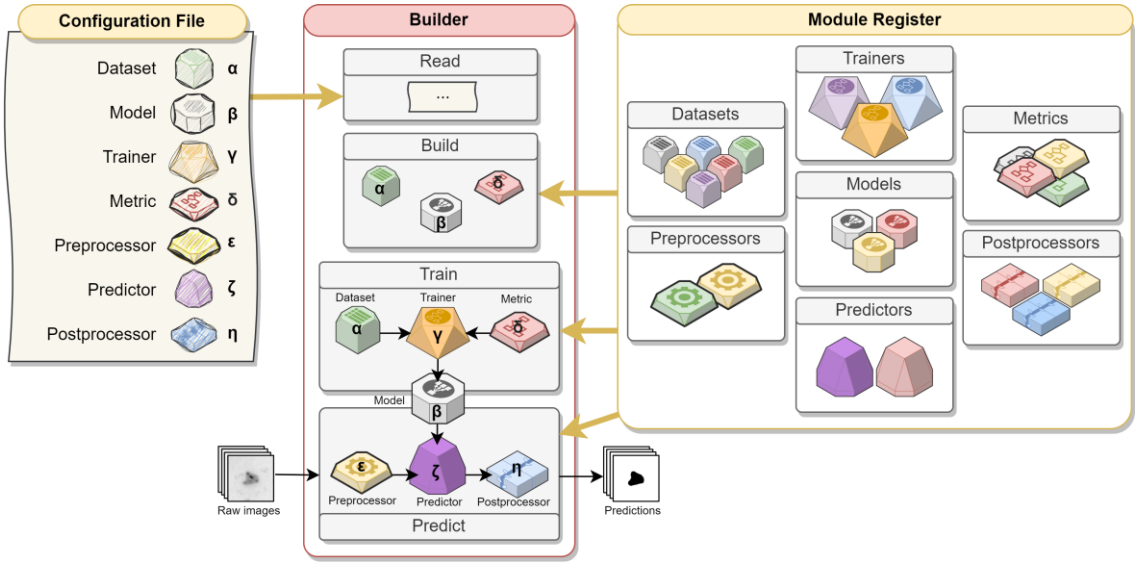


Figure 3-12 – **Configuration File, Builder and Module Register. (*Left*) The Configuration File** lists the names of existing Modules appearing in the Module Register and defines their parameters (*Greek letters)*. **(*Middle*) The Builder** reads parameters and Module names in the configuration file. It then retrieves the corresponding modules from the Module Register and builds them with their parameters. The Builder can then be used to train a Model with a Dataset, a Trainer, and a Metric. Once trained, the Model can be used by the Builder to predict the segmentation masks of some raw images using a Preprocessor, a Predictor, and a Postprocessor. **(*Right*) The Module Register** lists all existing modules of Biom3d. There are currently seven different types of modules, and each type has different variants (*colour shades*).

3.4.3 Configuration File, Builder, and Module Register

More details about the Biom3d Modules can be found on Biom3d documentation website: https://biom3d.readthedocs.io/en/latest/index.html

Biom3d modularity is not limited to the Command Line Interface but also allows to access and edit finer parameters such as the type of deep learning model or the metric used for training. This is made possible without having to go through the inner codebase of Biom3d thanks to three novelties: the Configuration File, the Module Register, and the Builder.

Configuration File. The configuration file is one of the core elements of Biom3d modularity and is an entry point editable by Programmer Users (Figure 3-12). This concept departs from nnU-Net implementation and was inspired by OpenMMLab (https://github.com/open-mmlab), a large group of computer scientists aiming at rapidly and regularly integrating the forefront state-of-the-art contributions in deep learning for image processing within several highly modular and open-source frameworks (MMSegmentation for 2D segmentation algorithms, MMDetection for 2D detection, etc.). In OpenMMLab frameworks, each training is configured via a single file, the configuration file, describing the "recipe" to assemble a deep learning model, a data-pipeline, an optimizer, a visualizer to observe the learning curves etc. (an example of their configuration file can be found here: https://mmsegmentation.readthedocs.io/en/latest/user_guides/1_config.html). The configuration files of Biom3d significantly depart from those of OpenMMLab being both simplified and completed (an example of Biom3d's configuration file can be found here: https://github.com/GuillaumeMougeot/biom3d/blob/main/src/biom3d/config_default.py).

The configuration file of Biom3d includes the definition of two types of hyper-parameters:

- **Stand-alone hyper-parameters** includes parameters defined in the Graphical User Interface (patch size, number of epochs, etc.) as well as finer hyper-parameters such as the initial learning rate or whether to use half-precision float-point format or not. Here follows an extract from Biom3d configuration file defining stand-alone hyper-parameters:

```
[…]
# number of classes of objects
# the background does not count, so the minimum is 1 (the max is 255)
NUM_CLASSES = 1
# number of channels in the input images
NUM_CHANNELS = 1

[…]
# batch size
BATCH_SIZE = 2
# patch size passed to the model
PATCH_SIZE = [128,128,128]
# larger patch size used prior rotation augmentation to avoid "empty" corners.
AUG_PATCH_SIZE = [160,160,160]
# number of pooling done in the U-Net
NUM_POOLS = [5,5,5]

[…]
# number of epochs
NB_EPOCHS = 1000
# optimizer parameters
LR_START = 1e-2
WEIGHT_DECAY = 3e-5
```

- **Module hyper-parameters** are key-value dictionaries. The first key-value pair precises the name of the module being used and defined in the Module Register. The second key-value pair defines the parameters of this module. In the following example the Model module named `"UNet3DVGGDeep"` designed for image segmentation is configured using some of the stand-alone parameters (*dark blue*) and one custom parameter (*factor*):

```
# model configs
MODEL = Dict(
    fct="UNet3DVGGDeep", # module name from the module register
    kwargs = Dict(
        num_pools=NUM_POOLS,
        num_classes=NUM_CLASSES if not USE_SOFTMAX else NUM_CLASSES+1,
        factor = 32, # multiplication factor for the depth of each convolution
        use_deep=USE_DEEP_SUPERVISION,
        in_planes=NUM_CHANNELS,
    )
)
```

The Module Register. The Module Register lists all available modules. Modules are classified into seven categories (Figure 3-12):

- **Datasets**, modules loading a pre-processed dataset to pass it to the deep learning model.
- **Models**, modules defining the deep learning model.
- **Metrics**, modules used to define the training loss, the validation loss and other training metrics.
- **Trainers**, modules defining how the Dataset module, the Model module, the Metric modules, and the optimizer are working together to train the deep learning model.
- **Pre-processors**, modules used to pre-process the training dataset and to pre-process raw data before prediction.
- **Post-processors**, modules defining how the deep learning model output are processed before being saved.
- **Predictors**, modules defining how the Pre-processor module, the Model module, and the Post-processor module are working together to compute predictions on novel images.

A Module Register entry is organized as follows:

```
from biom3d import ModuleClass

module_type = Dict(
    ModuleName = Dict(fct=ModuleClass, kwargs=Dict(parameter='value'))
)
```

For each module, the Module Register imports the Python function or the Python class from Biom3d code (`from biom3d import ModuleClass`) and stores it into a key-value dictionary named by type of module (`module_type`). In this dictionary, the key (`ModuleName`) can be arbitrarily defined but should be both explicit and short as it corresponds to the one used in the Configuration File. The value is also a key-value dictionary (`Dict(fct=ModuleClass, kwargs=…)`) including Biom3d function or class and, eventually, some of its parameters. Here is an example of a section of Biom3d Register listing available Model modules:

```
# model register
from biom3d.models.unet3d_vgg_deep import UNet
from biom3d.models.encoder_vgg import VGGEncoder, EncoderBlock

models = Dict(
    UNet3DVGGDeep   =Dict(fct=UNet, kwargs=Dict()),
    VGG3D           =Dict(fct=VGGEncoder, kwargs=Dict(block=EncoderBlock,
use_head=True)),
)
```

In this example, two Model modules have been registered, `UNet3DVGGDeep` a segmentation model and `VGG3D` a classification model. Model modules are Python classes like Dataset modules and Metric modules. The other type of modules are Python functions. Python classes will have to be instantiated by the Builder. Adding a new module to Biom3d consists thus simply in importing a Python class or a Python function and adding a novel entry to the corresponding dictionary (see next Section for more details).

The Builder. To metaphorize, if the Module Register lists all existing gene alleles of all possible genes and the Configuration File lists the genes of one specific individual, the Builder is the nucleus and its skeleton which organizes the DNA of this individual. The Builder is a Python class that makes all separated modules be and act together in four different steps (Figure 3-12):

- **Configuration File Reading.** The Configuration File can be provided in a Python file format or YAML format. The Configuration File Reading can also be accompanied by Model weights loading if fine-tuning, training restarting or predicting are intended.
- **Building.** The Python class modules listed in the Configuration File are then built. This Building process consists in instantiating this type of modules by calling their class constructors with the parameters defined in the configuration file. Module building is accompanied by the creation of a new *log* folder or the loading of an existing one and by the building of the optimizer function and the Callbacks (see next Section). Building automatically starts after Configuration File Reading.
- **Training.** Once built, training can be started by calling the Builder's Train class function. This function calls the selected Trainer module, along with the

Dataset module and the Metrics modules, a number of times equal to the number of epochs configured.

- **Predicting.** Once trained, a Model can be applied on novel data. The same Preprocessor module used to prepare the training Dataset processes the data, before passing it to the Predictor module, which applies the deep learning model. The Predictor can, for instance, patch the input images to a specific size. The Postprocessor finally processes the data by, for example, applying a threshold, removing noise, resizing images, etc.

With the Command Line Interface and the Module-oriented architecture of Biom3d, Python Programmer have access to a large panel of functionalities without the need for reading or editing the inner code of Biom3d. More than just changing fine module hyper-parameters, Biom3d Configuration File allows them to change entirely the module definition by selecting another one in the Module Register. Last, once the training is started, all training metadata (Configuration File in Python and YAML formats, training curves, terminal logs, training and validation splits of the dataset, training image snapshots, etc.) will be saved in a unique folder (the "logs" folder) along the model parameters.

3.5 Biom3d, a framework for deep learning developers

Biom3d has been meticulously crafted at a granular code level, affording Deep Learning Programmers to easily understand, edit, or add code components. Biom3d has also been structured as both a fully usable Python package and an experimental platform, providing a flexible arena for novel advancements, thus allowing it to stay aligned with cutting-edge deep learning methods. This Section, aimed at Deep Learning Programmers, dives into the deep layers of Biom3d's conceptualization: its code use and its code architecture. First, it provides a top-down explanation encompassing the overall code structure and the Builder module. Afterwards, it delves into each module and its characteristics, including the hitherto unexposed Callback module. Finally, it considers the myriad future potentials that Biom3d holds, including, for instance, those leveraged by its compatibility with MONAI and TorchIO libraries.

3.5.1 Code and Builder design

As mentioned in the previous Subsection, the structure of Biom3d is articulated around the Builder module which serves as a backbone for other Modules. Similarly, nnU-Net

code strategy is to define a single class called nnUNetTrainer encompassing all code core functionalities such as deep learning model initialization, model training or predictions. One major issue with nnU-Net implementation is that this class is deeply linked to the definitions of the other parts of the code and includes a very large number of tasks, making it a "all-knowing object". This "God Object" issue is one of the standard software engineering anti-pattern, an inefficient coding strategy [167] that increases code coupling and decreases code readability. Biom3d Builder is more of an independent and constituent block linking other independent objects together rather than an agglomerate of functions. The independency of Biom3d Module means that they can be easily extracted and reused out of Biom3d code context.

How to use the Builder. The Builder class use has been simplified as much as possible while keeping most of the complex functionalities required for deep learning model training. Understanding the panel of applications of the Builder starts by looking inside some of the frontend scripts designed for the Command Line Interface:

- **train.py** script demonstrates how to use the Builder to train a new deep learning model, to restart an interrupted training, or to fine-tune an existing model. All of this can be done with only three short lines:

```
from biom3d.builder import Builder
builder = Builder(config_path, log_path, training=True)
builder.run_training()
```

The first line instantiates the Builder class with two arguments: `config_path`, path toward a configuration file obtained during Biom3d preprocessing, and `log_path` path toward an existing log directory. Both can eventually be affected with a Python string or set to `None`. This choice will influence the behaviour of the Builder class:

	`log_path=None`	`log_path='value'`
`config_path=None`	Not allowed.	Load an existing model, to restart a training or to make predictions.
`config_path='value'`	Define a new model to start a new training.	Load an existing model to fine-tune with a new configuration.

The second line (`builder.run_training()`) starts the training process.

- **pred.py** script shows how to use the Builder to start predicting on novel images:

```
from biom3d.builder import Builder
builder = Builder(config_path=None, log_path, training=False)
img = builder.run_prediction_single(img_path, return_logit=False)
builder.run_prediction_folder(dir_in, dir_out, return_logit=False)
```

  In this case `config_path` is set to `None`. The log path is either a Python string or a list of Python strings. If the `log_path` is a Python list of string, then the Builder will load them all to perform ensemble prediction. The penultimate and last lines illustrate how to start a prediction with one image or with a folder of images. `return_logit` argument can be useful to return the probability map instead of the segmented image. The probability map is used, for instance, to indicate the certainty level of the model in each region of the prediction.

As shown in the previous Section, both scripts can also be used using the Command Line Interface with the arguments shown in Table 3-3. One of their hitherto additional elements is the `--name` argument which allows access in a flexible manner to more pre-implemented use cases of Biom3d. For example, here is a Command Line with **pred.py** script:

```
python -m biom3d.pred\
 --name seg_eval\
 --log logs/20230522-182916-unet_default\
 --dir_in data/btcv/Testing/img\
 --dir_out data/btcv/Testing/preds\
 --dir_lab data/btcv/Testing/msk
```

Where the `--name seg_eval` argument indicates that **pred.py** script will first segment a set of test images and then evaluate each prediction with a set of labels stored in `--dir_lab` directory. A complete list of valid names can be displayed using `--help` argument. The `--name` argument thus allows to easily add many experimental applications to Biom3d and will extensively be exploited in the following Chapter when experimenting with self-supervision.

Internal originalities of the Builder. The main concepts of the Builder process have already been exposed in Figure 3-12. Yet, inside **builder.py** script several originalities deserve more detailed explanations. Let us delve into each of them going from configuration reading to Optimizer and Callbacks definition.

First, the configuration reading and writing can be done both in YAML format and Python format. This might sound simple but is technically challenging for Python file format. YAML format, easy-to-read and easy-to-write, has been specifically designed to store configuration files. Its strict and simple syntax allows to preserve configuration integrity. Yet, this simplicity makes it less flexible and less practical when experimenting. On the other hand, Python configuration file allows to add comments and explanations about parameters and modules. All parameters are Python variables that can be reused several times in the configuration file which avoids copy-pasting problems. Both format advantages have been exploited in Biom3d: Python file format for defining or editing novel configuration files and YAML file format for storing configurations to preserve their integrity. A single function called `adaptive_load_config` and defined in **utils.py** script adaptively read the two file formats. In **utils.py** script is also defined `save_python_config` and `save_yaml_config`, used during preprocessing and output folder creation respectively to save configuration files.

Second, the Builder oversees the creation of the output folder, necessary for FAIR data sharing, containing:

- **Two trained deep learning models**, one storing the model parameters at the best validation loss and one at training end.
- **Image snapshots**, illustrating in a visual representation the performance of the model in the end of each epoch by applying it on a validation image.
- **The training curves**, stored in Tensorboard format and in a CSV file.
- **The terminal prints** (stdout), stored in a text file to conserve all important messages displayed on screen during training.
- **The data folds**, stored in a CSV file. This file is used for cross-validation. It contains the list of all images in the training set associated with a number between 0 and the number of folds. When training with fold 0, for instance, images associated with 0 will be used for validation while others for training.

Third, the core function allowing the Builder to cement other modules together and granting Biom3d of its flexible code architecture is called `read_config`. Here is an example applying the Predictor defined in the Configuration File:

```
out = read_config(
    self.config.PREDICTOR,  # Module name in the Configuration File
    register.predictors,    # Register category
    img = img,              # Module parameter n°1
    model = self.model,     # Module parameter n°2
    **img_meta)             # Module parameter n°3 and more
```

It receives as argument a Module name, a Register category and all eventual Module parameters coming either from the Builder or from the Configuration File. `read_config` finds the appropriate Module in the Register and executes it (or builds it if it is a Python class) with the appropriate parameters. Thanks to `read_config`, defining a novel module is as easy as defining a novel Python function or a Python class. The only expectations are:

- To add the new module in the register.
- To be careful with the interactions with other modules. In the previous example, a newly defined Predictor module expects at least one image (`img`) and one model (`model`) as input and one predicted image as output (`out`).

This new function/class thus does not even have to be part of Biom3d source code and can come from another Python package. As will be shown later, Biom3d models are, for instance, compatible with MONAI models.

Fourth, the Builder currently creates internally two types of modules which are then only editable through Biom3d source code:

- **The Optimizer**, which currently is Stochastic Gradient Descent (SGD). This choice follows the one of nnU-Net. A comparison with the more recent Adam optimizer was made in the beginning of Biom3d development and showed SGD to be better for 3D image segmentation.
- **The Callbacks**, which are Python classes holding functions that will be called during training at different frequencies (only once, every epoch, or every batch). Callbacks will be detailed in the next Subsection.

The Optimizer could theoretically be a Biom3d Module but all attempts to prove that a better optimizer than SGD have currently failed. Future developments could yet make this change necessary (as will be shown in following Chapter, some self-supervised learning techniques require large batch training and are optimally trained with LARS optimizer [168]). Even though, Callbacks definitions are coded in a

separated Python script (**callbacks.py**), they are not considered to be proper Modules, as they require to be called explicitly in the Builder (thus not using `read_config`). This recommendation is because each Callback has very different input expectations, making them impossible to define generically.

Developer documentation. All these coding originalities, and much more, are documented within Biom3d documentation and code. Biom3d documentation is divided into three levels:

- **A general documentation**, available online (https://biom3d.readthedocs.io/en/latest/), including tutorials for the Interfaces and giving a general overview explanation. It has been made available online using https://readthedocs.io/ website and Sphinx and Markdown formatting.
- **A function and class documentation**, following NumPy style formatting (https://numpydoc.readthedocs.io/en/latest/format.html) and giving details about each section of the code. ReadTheDocs includes an automatic reader for this type of documentation which avoid the need to duplicate it.
- **An in-line documentation**, explaining each line of code and providing a granular level of detail.

To conclude, Biom3d is more than just a Python package. Contributors are welcomed to add new documented pieces of code or modules to Biom3d. The powerful base mode of Biom3d and its flexible architecture allows thus to respond to a broad variety of applications, such as the recurring novel challenges posted on https://grand-challenge.org/ website.

3.5.2 Modules' description

After looking at the Builder, the "skeleton" of Biom3d, this Subsection will detail some of the various contributions regarding the Modules, the "organs" of Biom3d. Light will mostly be shed on default Biom3d Modules developed for 3D image segmentation. Biom3d is initially created to solve this task, even if it could go much further. This Subsection starts by presenting the data preprocessing and data loading methodologies which took a significant part of this thesis project. It will then detail the principles and mechanisms of the novel Callbacks Modules and how they articulate with the Trainer Module. Last, details will be given about the Predictor Module and its associated Postprocessor Module.

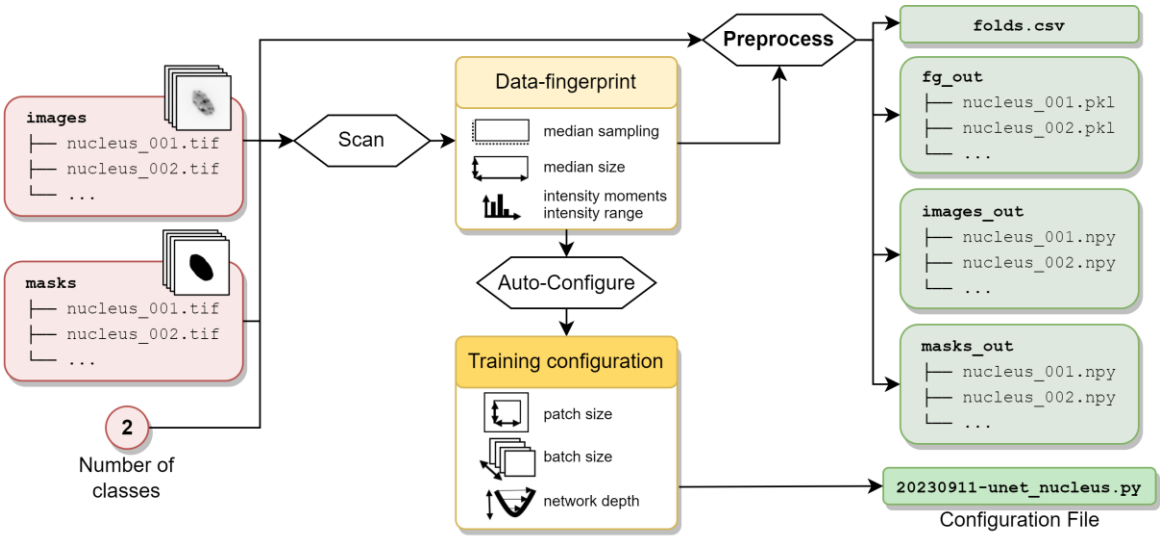


Figure 3-13 – **Auto-configuration and Preprocessing of training dataset for 3D segmentation.** Image and mask folders (*red, left*) are scanned to extract their data-fingerprint. The data-fingerprint is used to preprocess the images and masks and to automatically configure the future training (*yellow, centre*). The outputs of these two steps (*green, right*) are: a CSV file (*folds.csv*) describing which files will be used for training or validation, a Configuration File, the pre-processed images (*images_out*) and masks (*masks_out*), and the location of the foreground voxels (*fg_out*).

Auto-configuration and Preprocessing. This Paragraph looks under the hood of the overviewed preprocessing workflow shown in Section 3.3.2 and on Figure 3-8. This workflow, which only a part is a Biom3d Module, prepares training dataset for 3D segmentation (Figure 3-13) in three main steps:

- **Data Scanning** extracts the so-called **Data-Fingerprint** of the dataset.
- **Training Auto-configuration** outputs a **Configuration File** for training configuration.
- **Preprocessing** outputs a **Normalized Dataset** to start training with Biom3d.

The Data-Fingerprint is the median image size, the median spacing (if a spacing appears in image meta-data) and the intensity moments (mean and the standard deviation) and range (0.5% and 99.5% percentiles) of voxel values of the image located in the foreground region of the mask.

With this information, the Auto-Configuration can start with a series of heuristics. Biom3d heuristics mimics the behaviour of nnU-Net heuristics while being simplified.

In Biom3d, the patch size follows median size proportions and is dividable by a power of 2. For instance, if the median image size is (95,512,512) then the patch size will be (40,224,224). To simplify the explanations, this is done by dividing the median size multiple times by 2 until reaching a value below 7, rounding this value, and then multiplying it back by powers of 2. The number of powers of 2 gives the number of pooling layers in the future U-Net model. The number of pooling layers and the patch size are limited to (5,5,5) and (128,128,128) respectively to minimize memory footprint. The batch size is set to 2 and eventually increased if the patch size is lower than the maximum value. Choosing an appropriate patch size had proven to be determinant in the performance of the model.

The Prediction Preprocessing Module is a part of the Training Preprocessing operations. The Training Preprocessing is first initialised (*purple box* in Figure 3-14) (1) by creating three output folders for the future preprocessed dataset, (2) by, optionally, splitting single images, and (3) by distributing, in a CSV file, training images and validation images in K-folds for cross-validation. Then starts the Preprocessing routine (*blue box* in Figure 3-14) which will be applied individually to each pair of image and mask, and which does five tasks:

- **Adaptive image reading** automatically loads 3D images stored in NumPy format (.npy), in TIFF format (.tif or .tiff) or in any other medical formats that can be read by SimpleITK library (list of available format: https://simpleitk.readthedocs.io/en/master/IO.html) such as Nifti format. This function, called `adaptive_imread`, and its pendant, called `adaptive_imsave` (automatically saving an image or a mask in any format along its meta-data), are important additions to Biom3d (nnU-Net for instance is limited to Nifti format) and essential to load microscopy images (generally stored in TIFF format). While compressed formats are appropriate for long term storage (such as Nifti or TIFF format), fast reading formats are preferred for deep learning applications (such as NumPy format). After preprocessing, images and masks are stored in NumPy format.
- **Adaptive Reshaping** consists in uniformizing the shape of images and masks, so they all have four dimensions in the correct order (***channel****, depth, heigh, width*). Among these four dimensions, the size and position of the ***channel*** dimension are the only determinant factors. Biom3d has been extensively

tested and adapted to the myriad of shapes that can appear in manually created datasets. For instance, in the same dataset, it can sometimes be found, images with only three dimensions (*depth, heigh, width*), with shape (*depth, heigh, width, **channel***), or even with shape (*depth, **channel**, heigh, width*)! These fluctuations are even more diverse for masks, where users can decide to write their annotations in different colours in the same dimension (the mask then has only 3 dimensions) or to add one channel for each novel annotation. If using coloured annotations, users could mistakenly decide to use values between 0 and 255 arbitrarily, instead of following consistent choices, or to use more colours than existing classes. Among all custom datasets that have been treated through Biom3d Preprocessing not even one was following consistent annotating choices. nnU-Net has chosen to let the responsibility fall upon the user annotators, eventually throwing errors. Biom3d Preprocessing is equipped with a large panel of safeguards to automatically correct most (if not all) user annotator mistakes. This has proven to be decisive in avoiding the "first-click abandonment" phenomenon, mainly for Non-Programmers.

- **Normalizing** mainly concerns images (not masks). The objective of normalization is to limit voxel intensity range to accelerate deep learning model convergence. Two normalizing strategies exists in Biom3d (mimicking nnU-Net ones): **Z-Normalization** (default normalization) and **Median Intensity Z-Normalization** (based on intensity moment and range obtained during Auto-Configuration, and originally designed for CT-scans).
- **Resizing** consists in normalizing spatial sampling of all images and masks to the median spatial sampling: images with higher spacing than median one must be enlarged and inversely. Two resizing strategies have been implemented in Biom3d, one for anisotropic images (resized only along isotropic axis) and one for the others. Masks are resized using spline interpolation of order 0 (nearest neighbours) and images of order 3 (trilinear interpolation).
- **Foreground location extraction** is a data-loading computation optimization strategy developed by nnU-Net whose utility will be detailed in the following Paragraph.

Once preprocessed, images and masks are stored in their corresponding output folders. The location of these folders is indicated in the Configuration File and can thus be

anywhere on the computer (nnU-Net forced the user to add specific folders to the Operating System path and to follow a strict folder hierarchy). During Prediction the five Preprocessing steps described above are applied to images only, using the same parameters as for training dataset preparation which are also stored in the Configuration File.

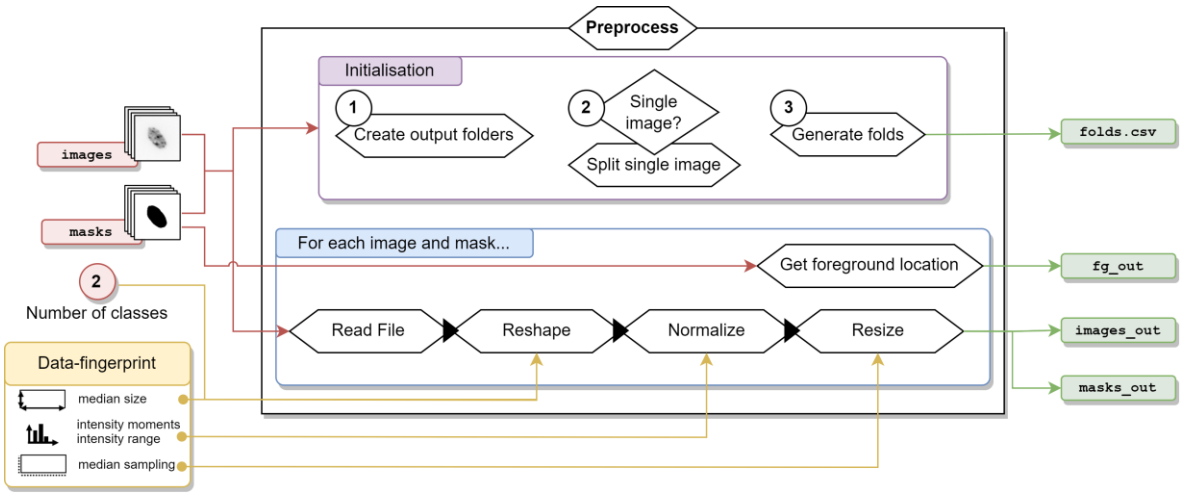


Figure 3-14 – **Training Preprocessing details for 3D segmentation.** The preprocessing starts (*purple box*, *initialisation*) by (1) creating the three output folders (*fg_out*, *images_out*, *masks_out*), (2) optionally splitting single image/mask and (3) splitting the dataset into training and validation folds (*folds.csv*). Afterwards, each image and mask (*blue box*) are read (*Read File*), independently of their format, reshaped (*Reshape*) so to have exactly 4 dimensions in (*channel, depth, heigh, width*) format, z-normalized for images and uniformized for masks (*Normalize*), and resized (*Resize*) so all images and masks have the same sampling. Finally, the locations of foreground voxels are extracted and stored in the appropriate output folder (*fg_out*).

Dataset Module. With Preprocessing, Data-loading strategy development was one of the most (if not the most) time consuming part of Biom3d overall development. As demonstrated by nnU-Net developers, contributions brought by deep learning model improvements to the overall method performance are minimal when compared to data-pipeline improvements. A counter-intuitively large number of levers exist in the data-pipeline that can significantly improve or worsen the model final accuracy and must be properly taken care of before experimenting with the model architecture. This observation is probably far to be limited to bio-medical domain as underlined by Andrej Karpathy in his blog (https://karpathy.github.io/2019/04/25/recipe/): “The first

step to training a neural net is to not touch any neural net code at all and instead begin by thoroughly inspecting your data. This step is critical."

The development journey of the Dataset Module is paved with too many mistakes for all of them to be listed here. Only the final version will be described, with the few points to pay attention to if intending to reproduce it. It was first noticed that nnU-Net data-pipeline has been thoroughly optimized as any experiments attempting to simplify it too much had failed. Yet, while intimately following nnU-Net principles, the default Dataset Module of Biom3d has been drastically optimized, especially by using TorchIO library. These optimizations explain the notable differences between nnU-Net and Biom3d in Table 3-1.

Biom3d methodology exploits the distinction between Pytorch Dataset and Pytorch DataLoader (which nnU-Net does not). A Pytorch Dataset is a Python class having the Python special methods `__getitem__` and `__len__`. These special methods let a user having access to the dataset elements and the dataset length using Python accessors: `dataset[index]` and `len(dataset)`. Biom3d Dataset Modules are Pytorch Datasets (such as presented on Figure 3-12). A Pytorch Dataset can then be called by a Pytorch DataLoader which is a Python generator, shuffling the dataset and grouping dataset elements into batches. Pytorch DataLoader instances can thus be called in Python loops such as in a Biom3d Trainer Module: `for batch, (images, masks) in enumerate(dataloader)`. As Biom3d utilizes Pytorch default DataLoader, the main contribution is the Dataset Module presented on Figure 3-15.

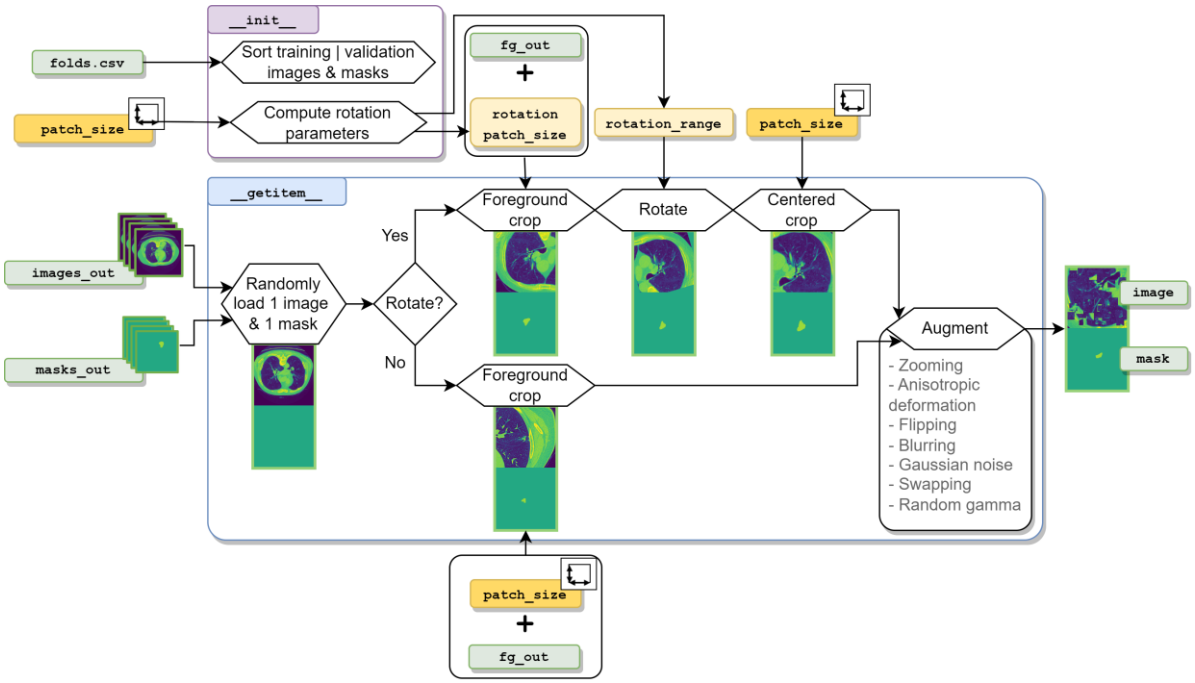


Figure 3-15 – **Biom3d default Dataset Module for 3D segmentation.** In the `__init__` class function (*purple*), the CSV file is used to sort training images from validation images and the patch size is used to determine the parameters of the rotation transformation (rotation angle and rotation patch size, *light yellow*). In the `__getitem__` class function (*blue*), one image and one mask are loaded into computer memory from their local folder. Those are then cropped in regions where foreground objects are located. If rotation augmentation is active, then the foreground crop is performed on a larger patch before being cropped a second time to discard unwanted empty regions in the four image corners. Another series of augmentations is finally applied to obtain a ready to use pair of image and mask patches.

As data loading can be extremely slow for 3D images, Dataset Modules for 3D segmentation are often the bottleneck in training speed and must thus be thoroughly optimized. The `__getitem__` class function of Biom3d is composed of three main steps:

- **Random selection and loading of a pair of image and mask.** The candidate pair is loaded using `adaptive_imread` function and chosen among the preprocessed images and masks in either the training set or the validation set (the CSV file stores this information).
- **Foreground cropping** is, with data-augmentation, the most important part of the data-loading process regarding model performance. It consists in centring

in the 3D patch one of the annotated objects (*foreground objects*). As in volumetric image the foreground-to-background volume ratio is small, the goal of foreground cropping is to "force" the model to see regions of interest. For computation speed improvement, the locations of the foreground objects are pre-computed and stored during preprocessing. During data-loading, one of the foreground locations is randomly selected and used to crop the whole image and mask. For a significant boost in accuracy, it has been noticed that the foreground region should be located exactly in the middle of the patch (not randomly located in the patch). To present other image regions to the model, foreground cropping alternates with random cropping (randomly located patches). For another boost in performance, it has been noticed that there should be a fixed proportion between foreground and random cropping. For instance, with a batch size of 2, one of the pair of image and mask in the batch must consistently be a foreground crop while the other pair must be a random crop.

- **Data augmentation** in the default Dataset Module of Biom3d is a fixed set of transformations, except for the rotation transformation for performance reasons (see Figure 3-15). Biom3d augmentations rely on TorchIO package, specifically designed for volumetric data-pipelines. Biom3d development started with a reproduction of nnU-Net augmentations, which rely on a custom library a little less intuitive to use than TorchIO (yet a little faster). After many attempts, mainly from random searches, Biom3d augmentations were both simplified and made more efficient than those of nnU-Net (see difference in Table 3-1).

While developing Biom3d Dataset Module another important advantage of Biom3d modularity has been discovered and exploited. When trying to reproduce and/or optimize an existing piece of code, a parallel methodology is often employed: on one side, the existing piece of code is deconstructed and tested piece-by-piece to be understood and on the other side, novel and more optimized pieces of code are constructed. The difficulty of this process relies in making end meet: entirely deconstructing the code is often only partially possible and thus construction is limited to isolate only small pieces of code and not larger code sections. As Biom3d easily accepts alien code incorporation, the entire and original "Dataset Module" of nnU-Net

was also integrated as a usable Biom3d Module. Following this integration, a series of intermediate and hybrid Dataset Modules were developed and tested by simply changing a single line in the configuration file. The weak points of the prototype Dataset Modules were thus rapidly spotted and improved.

Model Module. The default Model Module of Biom3d is, as for nnU-Net, a dynamic 3D U-Net which architecture is adapted automatically depending on the number of pooling determined during pre-processing. The number of pooling in the model is adapted to each dimension of the anisotropic patch size to avoid reducing the intermediate feature maps excessively. Biom3d also contains several Model Modules which have been adapted to anisotropic pooling: a 3D EfficientU-Net and a 3D HRNet. The later was also non-existent for 3D images and was thus created specifically as a Biom3d Module. Table 3-4 presents a comparison of these architectures. An additional Swin-UNETR [123] was recently added using MONAI implementation and is thus non-adaptative to the patch size.

Table 3-4 – **Comparison of model architectures on the Pancreas dataset.** Trainings were all performed with 1000 epochs of 250 steps on a Nvidia A100 GPU.

Model	Encoder	Dice score ↑	Training speed ↓
nnU-Net (official)	3D VGG-like	0.6620	**10 hours**
nnU-Net (Biom3d)	3D VGG-like	0.6683	**10 hours**
EfficientU-Net (Biom3d)	3D EfficientNet-b4	**0.6755**	16 hours
HRNet (Biom3d)	3D HRNet	0.6581	38 hours

Callback Module. One of the other originalities of Biom3d are Callback Modules which are inspired by best practices inherent in deep learning competitions. During training, several events occur at certain frequencies and at certain time points. For instance, in the end of each epoch the learning rate is decreased, to improve final accuracy, using a certain function, a polynomial decay used in nnU-Net:

$$\eta_{cur} = (\eta_{max} - \eta_{min}) \left(\frac{1 - epoch_{cur}}{epoch_{max}} \right)^{\gamma} + \eta_{min}$$

where η_{cur}, η_{max} and η_{min} are the new, the maximum and the minimum learning rate, $epoch_{cur}$ and $epoch_{max}$ are the index of current and maximum epoch and γ is the

reduction rate between 0 and 1. Instead of defining this formula into the training loop directly, it is within a so-called Callback Module. Callback Modules are Python classes inheriting from an abstract class which has 6 overloadable class functions, each corresponding to a different time point in the training loop (Figure 3-16). This practice limits editions of the training loop and preserves strong modularity of the code. Many different learning rate functions were rapidly compared using this Callback methodology. Biom3d currently has 7 different types of Callback Modules (see Figure 3-16). Aside from the learning rate functions, there are 4 Saver Callback Modules, saving training information in the end of each epoch, 1 Log Printer, displaying information in the user terminal, and 1 Metric Updater (see next Paragraph).

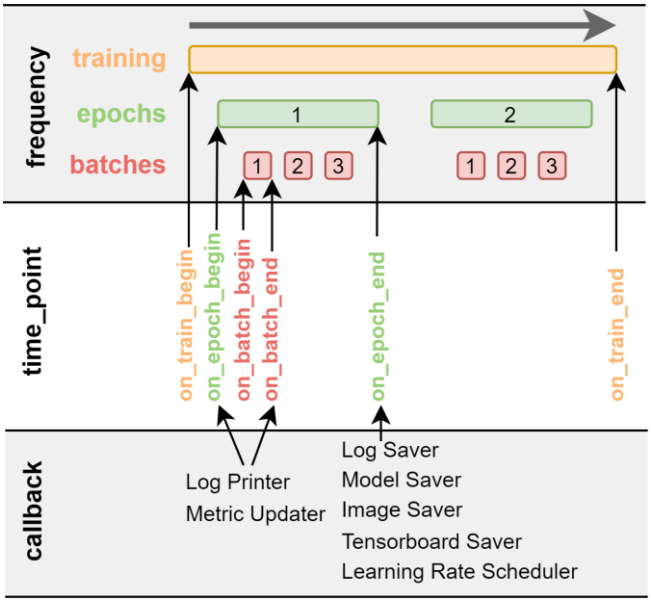


Figure 3-16 – **Callback Module principle.** The whole training (*top row, orange*) is divided into epochs (*green*) themselves divided into batches (*red*). Callbacks are Python classes that can have one or more class-functions, each representing one of 6 different time points (*middle row*). Biom3d currently has 7 types of Callback Modules (*bottom row*).

Metric Module. Metrics in Biom3d inherit from a parent class which inherits from Pytorch Module class and can thus be used as loss functions or as training and validation metrics. Biom3d Metrics have a name, a value and an average argument which allows to access to this argument any time during training. The Log Saver and Log Printer Callbacks for instance regularly interrogate these arguments to save or

display important information. The Metric Updater Callback regularly updates the metric average.

Trainer Module. Trainer Modules are loops that call the Dataset Module to pass training data to the Model Module. Model outputs and ground truths are then passed to the loss Metric Module to update the Model parameters using an optimizer. Callbacks are automatically called at different time points.

Predictor Module and Postprocessor Module. To use a trained model on a novel image, the Builder successfully calls 3 Modules: the Prediction Preprocessor (cf. first Paragraph), the Predictor and the Postprocessor. For 3D segmentation, the Predictor Module decomposes the preprocessed image into a grid of overlapping patches (using TorchIO Grid Sampler), then applies the model to each patch and finally compiles the resulting predictions into a single output. To improve model performance, this process is repeated 8 times on the 8 possible flipped version of the input image (*test time augmentation*). The 8 final outputs are then flipped back and averaged. The Predictor outputs the prediction *logit* which corresponds to the raw model output. To obtain the final mask prediction, the Postprocessor first resizes the output, if the input image was resized during preprocessing, then applies the activation function (*sigmoid* or *softmax*), and, eventually, applies the noise removal function (see Section 3.2). The Builder is finally in charge of saving the final mask along with the input image metadata using the `adaptive_imsave` function. If the user intends to use multi-model prediction, the current version of Biom3d applies the Predictor Module associated with each saved model (Biom3d modularity incorporates thus more than just the Model Module) before averaging the models' predictions, post-processing and saving.

By offering a detailed description of Biom3d Modules, this Subsection not only aimed at opening boxes of Biom3d inner mechanisms but also at opening doors to invite and guide deep learning contributors to refactor and keep improving Biom3d.

3.5.3 Modules' potential

Each of Biom3d Module is not only made to be exploited but also adapted, removed, renewed, broken, combined, etc. Biom3d is designed to be a platform for deep learning developers. To leverage existing works, Biom3d is also compatible with state-of-the-art Python libraries to exploit existing implementations such as TorchIO or MONAI or the recent Segment Anything for Microscopy [169].

The default version of Biom3d is made for 3D Segmentation but the core concepts of the seven types of Modules is applicable to almost all deep learning algorithms and not limited to image analysis. Biom3d could be adapted to 2D images, to instance segmentation, to image generation, to object detection, or to image classification. With a bit more modification, Biom3d could also be applied to structured data, sounds, sequences, etc. It will be seen in next Chapter that Biom3d can for instance perform self-supervision, which involves image classification and image segmentation, on 3D images with only very little editions.

Last, Biom3d is not limited to its predefined modular architecture. New Module types could also be defined. For example, in the course of this thesis, a script computing morphological quantities for 3D nucleus was developed to improve NucleusJ parameter computations, using complex computational geometry algorithms (Marching cube, Delaunay transformation, etc.). This post-post-processing, currently separated from Biom3d code, could easily be added to Biom3d as an optional plugin for interested users in the form of a new type of Module.

3.6 Conclusion

Reproducibility issues have already been pinpointed several times in the literature [40], [133], [162], [170], [171]. The review and benchmarking presented in Chapter 2 have promoted properly documented and open-access codes. Even if crucial, these requirements are yet minimalistic regarding code sustainability. To guarantee a software tool to be reused, two targeted audiences must be satisfied: end users and developers, the former to guarantee the software to be tested, used, and having feedback, and the later to maintain and improve it. Biom3d goes beyond this dichotomy by satisfying a continuum of users, from Non-Programmer to Deep Learning Developers. This was made possible by integrating Graphical User Interfaces and formatting the whole code architecture with modularity in mind on every level of detail.

Authors' Contribution – cf. Appendices.

Chapter 4

Self-supervision of 3D segmentation methods

4.1 First experiments

Self-supervision has created a major buzz in the deep learning community, in the aim to leverage information contained in large-scale unannotated datasets to train large-scale models. In the scope of this project, self-supervision could thus reduce the expensive need of manual annotations for the training of supervised deep learning models.

For images, the main idea of current self-supervised learning methods is to build a coherent embedding space (and more generally a feature extractor). In deep learning theory, if the first layers of a deep learning model extract simple and local shapes and objects, the last layers extract complex and global concepts and abstractions. These last layers output vectors of short dimensions, the embedding vectors. The set of all possible embedding vectors is the embedding space. A "coherent" embedding space means that images looking "similar" (to a human point of view) must be associated in this space, by, for instance, having close embedding vectors, and inversely for images looking different (Figure 4-1).

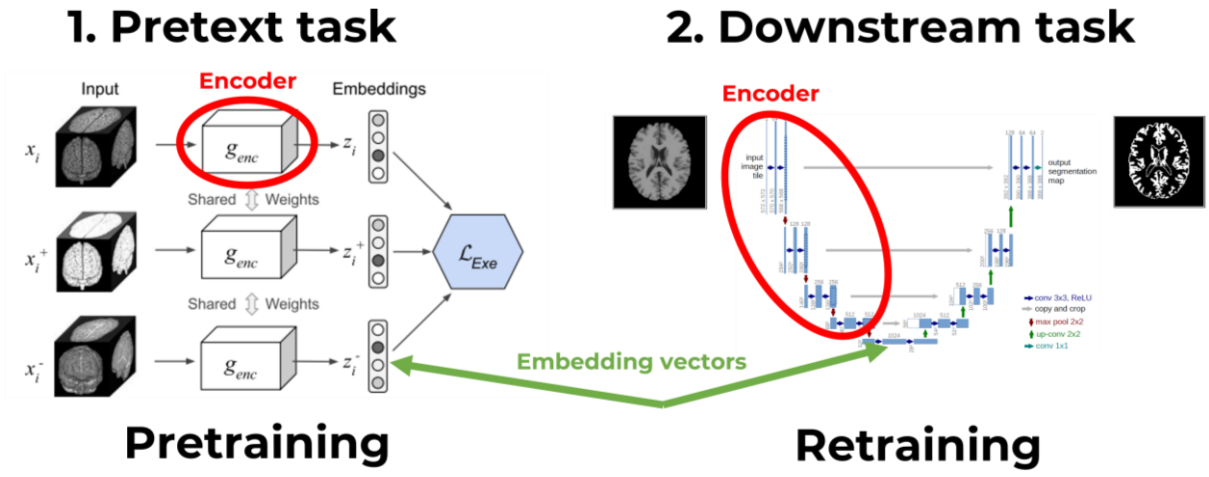


Figure 4-1 – **Self-supervision applied to 3D segmentation.** Self-supervision for images is a two-step process: (1) pretraining on the pretext task (*left*), a triplet task in this case, and (2) retraining on the downstream task (*right*), the segmentation task. The pretraining is only done on the encoder (*left*, *red ellipse*) which will be then integrated

into the U-Net model (*right, red ellipse*). For the triplet pretext task, the embedding vectors of two augmentations of one image (*left, first two rows*) are intended to be identical while different from any other image (*left, last row*). Left illustration adapted from [165] and right illustration adapted from [58].

Self-supervision for images is currently achieved by two main categories of pretext tasks [172]:

- For **encoder-decoder approaches**, the model must output an image identical to the input image. The embedding space is a bottleneck in the model intermediate outputs (*feature maps*) and is constructed automatically during training. The historically well-known auto-encoder method is based on this principle [173], as well as the more recent masked auto-encoder (MAE) methods [174] exploiting Vision Transformers, or distillations methods also having demonstrated impressive results in image generation such as Stable Diffusion [175]. Other examples of encoder-decoder approaches are filling artificial holes in input images or, inversely, removing artificial artefacts. Once pretrained with encoder-decoder approaches, only the model encoder is kept performing retraining, the decoder being discarded.
- For **encoder-only approaches**, the model must output identical embedding vectors given several augmented views of an input image. This is the base principle of methods based on Siamese approaches such as SimCLRv2 [176] and methods involving a "student" and a "teacher" model, the former distilling knowledge into the later via exponential moving average updates, such as BYOL [177] or DINO [161]. Other examples of encoder-only pretext tasks are solving jigsaw puzzles or determining the angles of an image rotation.

In 2020, at the beginning of this PhD project, only preliminary work existed on self-supervised methods for 3D segmentation, with applications mostly in medical fields. The pioneer work of Taleb *et al.* [165] for instance, explored the application of 5 different encoder-only pretext tasks in order to improve performance on their downstream task, the 3D segmentation of pancreas images (see next Section). More recently Model Genesis [178] and CLIP [88] encoder-decoder approaches, combined several medical dataset and also, for the CLIP approach, several data modalities (image and text). Finally, some methods leverage both encoder-decoder and encoder-only

approaches simultaneously, such as Swin-UNETR [179] or DiRA [180]. A continuously updated review of existing self-supervised methods for medical imaging can be found here: https://github.com/HiLab-git/SSL4MIS.

To the extent of the bibliographical research done during this project, no publications have extensively focused on 3D biological data. For this reason, and as biologists can now easily and quickly produce large amounts of unannotated data, self-supervised learning was a promising direction to pursue.

Two interns, Adama Nana in 2022 and Abderrahime Tizi in 2023, have started the exploration of Model Genesis and Swin-UNETR approaches respectively. Adama had to install Model Genesis in a Docker container. On the Nucleus, Pancreas and Lung datasets, pretraining with Model Genesis methodology resulted in a gain in performance on the Lung dataset only (Table 4-1). It is worth noting that Model Genesis also requires a long pretraining (five times longer than the retraining). As it only brought inconsistent results, this method was set aside.

Table 4-1 – **Performance of Model Genesis.** The first two rows give training and testing dataset statistics. The last three rows give the Dice scores of each method on the testing set after being trained (or retrained for Model Genesis) on the training set.

Method	**Lung dataset**	**Pancreas dataset**
Training set (20%)	12 images	53 images
Testing set (25%)	16 images	71 images
nnU-Net	0.498593	0.662048
Biom3d	0.542248	**0.668316**
Model Genesis	**0.561012**	0.660318

Subsequently, Swin-UNETR methodology was integrated into the modular architecture of Biom3d. Abderrahime directly tried to pretrain Swin-UNETR on a custom unannotated dataset of 2000 3D nuclei before retraining it on the benchmark dataset of nuclei which resulted in an insignificant improvement when compared to the supervised baseline. He unsuccessfully tried to use the official implementation of Swin-UNETR. He then tried to progressively reduce the number of annotated images in the nuclei dataset to check if the pretraining was not saturating as self-supervised

pretraining was proven to be interesting only in data-scarce situations. Abderrahime had only the time to draw a supervised baseline comparing Biom3d and nnU-Net (Figure 4-2). In the future, this work should be completed with self-supervised methods. It can be observed that with only 2 images the training already performs well (>82%) and that the training accuracy rapidly saturates with only 16 annotations.

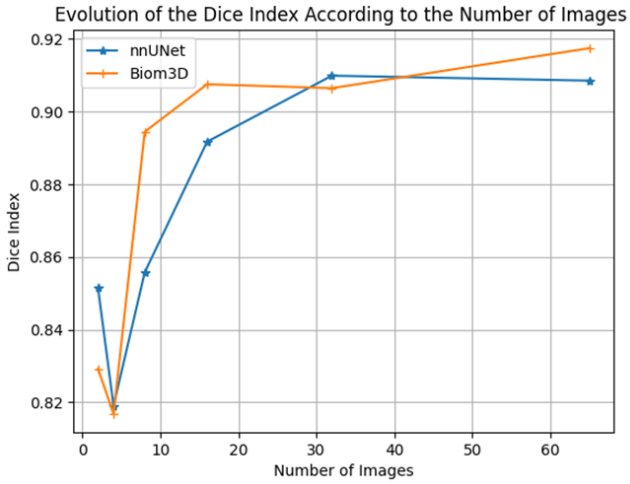


Figure 4-2 – **Preliminary comparison between Biom3d (*orange*) and nnU-Net (*blue*).** The Dice scores are obtained using the same testing set while training on set training sets of increasing sizes (2, 4, 8, 16, 32 and 65 images). The curves drop down between 2 and 4 images probably due to bad annotations.

4.2 Adventures in adapting existing work

In parallel to the previous experiments, thorough experiments were undertaken on two of the previously mentioned publications: [165] and [161]. As one potential weak point of the method discussed above is the Pancreas dataset, the methods discussed below will be evaluated on this dataset, and, eventually, on the Nucleus dataset to also have a biological dataset.

4.2.1 Quest to improve the work of Taleb *et al.*

The work of Taleb *et al.* demonstrated an impressive increase of more than 30% of their Dice score on their Pancreas dataset, passing from 42% to 75%, using a self-supervised model pretrained on the whole dataset and then retraining it on 10% of the annotated images. Comparatively, their supervised model required 6 times more annotation to reach a Dice score of 75%, meaning that self-supervised learning could potentially divide the number of required handmade annotations by 6.

Setting aside the numerous bugs appearing during method installation (bug fixes required Docker) and testing, the pretext task called “Relative 3D patch location” (RPL) delivered promising results among the 5 proposed pretext tasks. Training and testing in this reutilization were performed directly on the Nucleus dataset. As their method only works with cubic images of size (128,128,128), every image was resized using 100% of the unannotated images for pretraining and, successively 10% and 25% of the segmentation annotations for retraining. The RPL method gave an impressive gain in accuracy of 20% and 10% respectively on the testing set (Figure 4-3).

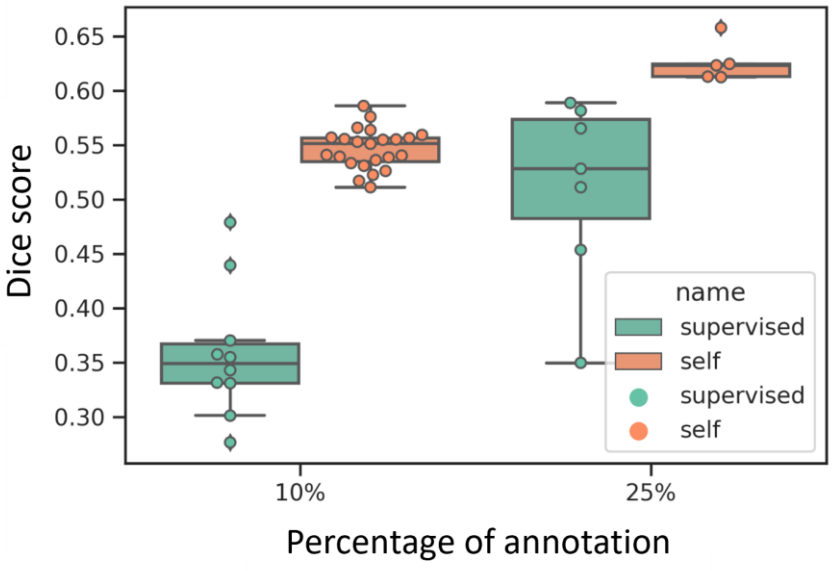


Figure 4-3 – **Region 3D patch location method of Taleb *et al.* applied on the testing set of the Nucleus dataset.** Supervised training consists in training on the 10% (or 25%) of the training set and then evaluating on the testing set. Self-supervised training consists in pretraining on 100% of the unannotated training set, retraining on 10% (or 25%) and testing on the testing set. Each dot represents one reproduction of the experiment and the average Dice score obtained on the testing set for this experiment.

These experiments were all performed using Taleb *et al.* implementation. Afterwards, their methodology was integrated into Biom3d framework, and the same experiments were performed. Unfortunately, this only revealed important gaps in Taleb *et al.* methodology when training a supervised model. Indeed, and very surprisingly, the performances of the supervised model of Biom3d (in a prototype version at that time) matches the ones of the self-supervised model (55% for 10% annotation and 63% for 25%). After exploring the Taleb *et al.* code, it was noticed that they used less epochs

to train their supervised model than to retrain their self-supervised one. Additionally, their data-pipeline is probably under-optimized, specifically their data augmentation procedure. Thus, in their code configuration, self-supervising probably simply replaced the role of data-augmentation. Finally, the limitation to small cubic images of fixed size is another important limiting factor of Taleb *et al.*'s work.

To conclude, even though they provided almost all required elements (except the dataset) to guarantee that their results to be reproduced, the weaknesses of their implementation only appear when trying to recode it. Taleb *et al.* should be thus more viewed as a proposition of novel ideas than a concretely reusable method.

4.2.2 Defeating the DINO

Another promising candidate in the burgeoning world of self-supervision emerged in 2021: DINO [161]. Even more than Taleb *et al.* methods, the DINO method belongs to fundamental research in computer vision, as, in its raw version, it is only applicable to 2D images of daily life objects. This method, very similar to BYOL [177], consists in training simultaneously two classification models: a "student" model and a "teacher" model. For one input image, the student model is trained using both large (*global*) and small (*local*) image patches that are resized to the same size and randomly augmented, while only global patches are passed to the teacher model. The student model parameters are then updated with a cross-entropy loss between its embedding vector outputs and the teacher outputs. For crops coming from identical images, the cross-entropy loss pushes the student model to output embedding vectors like those of the teacher model. The teacher model parameters t_i at iteration i are updated using the student model parameters s_i and the exponential moving average formula: $t_{i+1} = \alpha t_i + (1-\alpha)s_i$. The originality of DINO is that the teacher output is centred (subtracted with a moving average centre value) and sharpened (divided by a user defined value called *temperature*). This originality supposedly guarantees a better training stability. Indeed, when training with such a self-supervised methodology two undesirable events, called *collapses*, can happen:

- **Centring collapse** appears when all embedding vectors are uniform, independently from the input. Sharpening should counteract it.
- **Sharpening collapse** appears when only a single dimension of the embedding vectors is dominant. Centring should prevent it.

A balance should thus be found, unfortunately manually, to guarantee training proper convergence. A way to check if one of the two collapses occurs is to monitor the teacher entropy and Kullback-Leibler (KL) divergence, the teacher cross-entropy being the sum of these two terms. The KL-divergence being zero represents a collapse, a centring collapse if the teacher entropy is a non-zero constant and a sharpening collapse if it is zero.

Application to 3D images. For biological and medical application, DINO also stood out because it should require smaller batch size than other self-supervised methods. SwAV [181], for instance, its predecessor, required a batch size of 65000 images, which is almost impossible to handle with 3D images.

DINO was integrated in Biom3d framework. DINO originally relies on 2D Transformer Models but was supposed to work also with CNN models. The Biom3d base-encoder model was thus upgraded with the DINO "head", a few dense layers in the end of the model allowing embedding vector output. The "teacher" and "student" model paradigm was compatible with Biom3d as it originally accepts lists of models. The DINO loss function and trainer was integrated among Biom3d Metric and Trainer Modules. The LARS optimizer was added to the Builder Module. Finally, a new Dataset Module, derived from the base one, was created. Most of the development was spent adapting the Dataset Module to 3D images. Indeed, DINO required local patches to intersect with global patches. In 2D images, selecting randomly a global patch with a size ranging between 0.4 and 1.0 time the image size is sufficient to be sure to capture enough information so when randomly selecting a local patch the model will be able to understand that these two come from the same image. In 3D biomedical images, due to the nature of the observed objects, following such a strategy will not work. It would be very hard (if not impossible) to tell if two non-intersecting patches originate from the same image. A new patching methodology called *SmartPatch* guaranteeing patch intersection was thus integrated into the DINO Dataset Module of Biom3d (Figure 4-4). With the help of new Biom3d Callbacks, SmartPatch also offers the possibility to finely and dynamically control the intersection level and the two patch sizes during training.

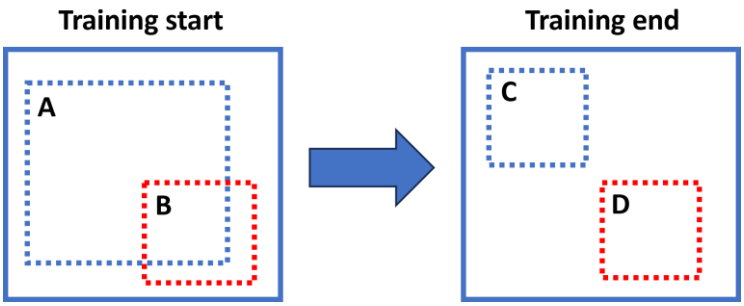
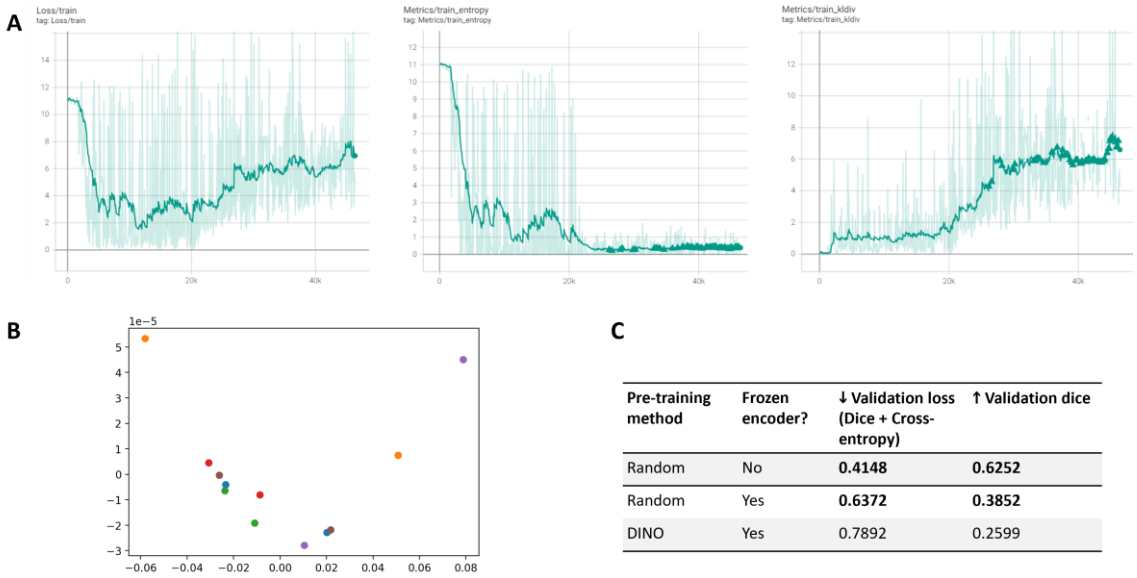


Figure 4-4 – **SmartPatch methodology.** The plain squares represent the image, the dashed blue squares represent the global patches, and the dashed red squares represent the local patches. When the training starts (*left*) the global patch encompasses almost the entire image volume and intersect with the local patch. During training the global patch size and the intersection constraint are slowly reduced. In the training end (*right*), the global and local patch have the exact same size and are selected randomly in the image.

So began a long list of experiments to find an appropriate set of hyper-parameters allowing training convergence and avoiding collapse. This journey started with the default hyper-parameters of DINO applied on the Nucleus dataset and on the Pancreas dataset, and was carried on with many adjustments involving all the aforementioned hyper-parameters (embedding vector dimension, loss temperatures, patch size, model head, optimizer parameters, etc.) and more (training “warmups”, etc.). It ended with the results displayed on Figure 4-5.



Pre-training method	Frozen encoder?	↓ Validation loss (Dice + Cross-entropy)	↑ Validation dice
Random	No	**0.4148**	**0.6252**
Random	Yes	**0.6372**	**0.3852**
DINO	Yes	0.7892	0.2599

Figure 4-5 – **DINO results on the Pancreas dataset. (A) Example of training loss (*left*), entropy (*middle*), and KL-divergence (*right*).** DINO loss first converges before diverging (without collapsing as the KL-divergence raises). **(B) Principal Component Analysis of the embedding vectors of a model trained with DINO.** Dots with identical colours represents different patches of the same image and should normally form clusters. This is an example of sharpening collapse as only one dimension is dominant (x-axis). **(C) Comparison of a DINO pretrained encoder and a randomly initialized encoder.** The first row corresponds to a standard segmentation training with a full U-Net model. For the last two rows, the encoder is frozen (untrained), meaning that only the decoder is trained. A randomly initialized encoder performed better than a pretrained encoder with DINO.

To conclude, the DINO training method was very difficult to handle to make the loss converge. Self-supervised pretraining is also complicated as, during pretraining (which can be very long or computationally expensive), only the training curves can be monitored. It is necessary to wait for the completion of the retraining to note an eventual improvement. Additionally, even when the training curves looked correct, the embedding vectors were not arranged properly (cf. PCA on Figure 4-5-B). Finally, it was noticed that a randomly initialized U-Net with a frozen encoder (left untouched during training, Figure 4-5-C) already performed well on the Pancreas Dataset and that a frozen encoder pretrained with DINO performed worse. DINO, yet promising, was abandoned for the rest of this thesis. For further investigations going beyond the scope of this work, DINO is left integrated as part of Biom3d Modules.

4.3 Self-supervising a full 3D U-Net with triplet and angular losses

Not to completely give up on self-supervision, it was decided to carry on the exploration based on two important observations obtained from previous explorations:

- The decoder plays a significant role in feature extraction in the U-Net model. Further investigations departing from Table C in Figure 4-5 showed that a U-Net model with a frozen encoder can reach a Dice score of 0.5784 on the Pancreas Dataset against 0.7483 for a fully trained U-Net. In this configuration, it means that training the encoder, only brings an improvement of 17% on the final score. It implies first that the current encoder is limited. Indeed, as shown in Table 3-4, changing the encoder to a more complex architecture can improve the final accuracy by 1%. Second, it also implies that incorporating the decoder in the pretraining task could be a good direction of research.
- The complex DINO methodology (or others originating from fundamental research in computer vision) is very difficult to handle. Simpler, more understandable pretraining methods might be better starting points for 3D imaging.

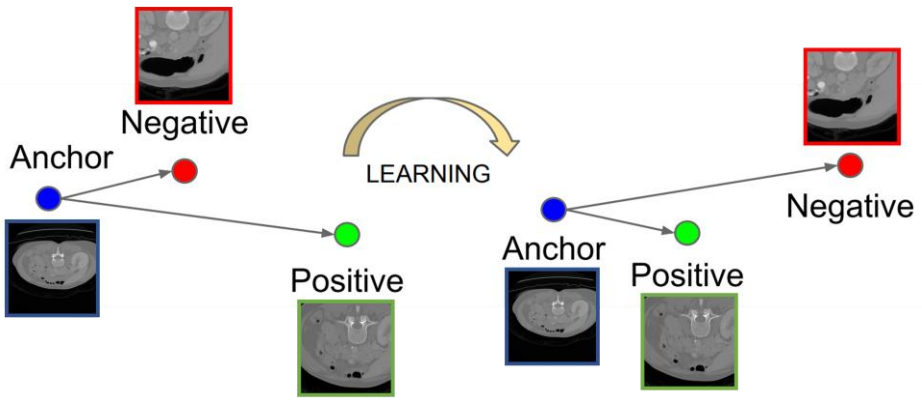


Figure 4-6 – **Triplet loss applied to 3D images.** This loss minimizes the distance between embedding vectors of two augmented views of the same image (*anchor*, *positive*) and increases the distance from other images (*negative*). Illustration adapted from [182].

Triplet loss. The first goal was thus to create a simple pretraining loss and to carefully check if this loss can build a coherent embedding space. Probably the simplest possible

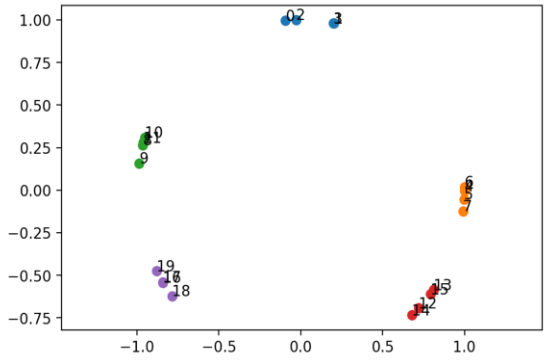
pretraining loss is the Triplet loss [182] (Figure 4-1-C and Figure 4-6) which is defined by the following formula:

$$L(v_{anc}, v_{pos}, v_{neg}) = \max\left(0, \left\|v_{anc} - v_{pos}\right\|^2 + \alpha - \left\|v_{anc} - v_{neg}\right\|^2\right)$$

Where v_{anc} and v_{pos} are the embedding vectors of two views of the same image (or the same individual) and v_{neg} is the embedding vector of a view of another image (or another individual). The parameter α, generally equal to 0.2, controls the minimal margin that the model must generate between the two distances $\left\|v_{anc} - v_{pos}\right\|^2$ and $\left\|v_{anc} - v_{neg}\right\|^2$. This loss thus brings together v_{anc} and v_{pos} while pushing away v_{neg}.

For this thesis project, applying the Triplet loss to 3D images was made possible by the SmartPatch method which was added to a new Triplet Dataset Module in Biom3d. After several adjustments, two successful experiments, departing from those made with DINO on Figure 4-5, finally confirmed that the Triplet loss was working: the Principal Component Analysis done on the embedding space (Figure 4-7-A) and the retraining of a U-Net model with a frozen encoder (Figure 4-7-B). However, when retraining the Triplet pretrained model on the Pancreas segmentation dataset, the test set accuracy stayed at the supervised level (Table 4-2-Exp2).

A



B

Pre-training method	Frozen encoder?	↓ Validation loss (Dice + Cross-entropy)	↑ Validation dice
Random	No	**0.4148**	**0.6252**
Random	Yes	0.6372	0.3852
DINO	Yes	0.7892	0.2599
Triplet	Yes	**0.5874**	**0.5449**

Figure 4-7 – Triplet loss preliminary results on the Pancreas dataset. (A) Principal Component Analysis of the embedding vectors of a pretrained encoder. Each colour represents embedding vectors of patches coming from the same image. The Triplet loss trained the model to form appropriate clusters. (B) Evaluation of the Triplet pretraining performance on segmentation with a frozen (untrained) encoder. The Triplet method is better than the random baseline.

Pretraining the full encoder-decoder. As discussed earlier, the decoder is a good feature extractor and including it in the pretraining task could potentially improve the model accuracy on the segmentation task. This idea is another originality of this thesis: to use the entire U-Net model (an encoder-decoder) with an encoder-only pretraining approach. This contribution also clearly departs from the encoder-decoder methods described earlier because the decoder will not be discarded in the end of the pretraining and will be reused as is during retraining. Figure 4-8-A illustrates the encoder-decoder pretraining with the Triplet loss and 3D images. To limit the size of the output only the antepenultimate layer of the decoder is considered. On the Pancreas dataset, this new methodology finally overpassed the supervised baseline by a significant margin (Table 4-2-Exp3).

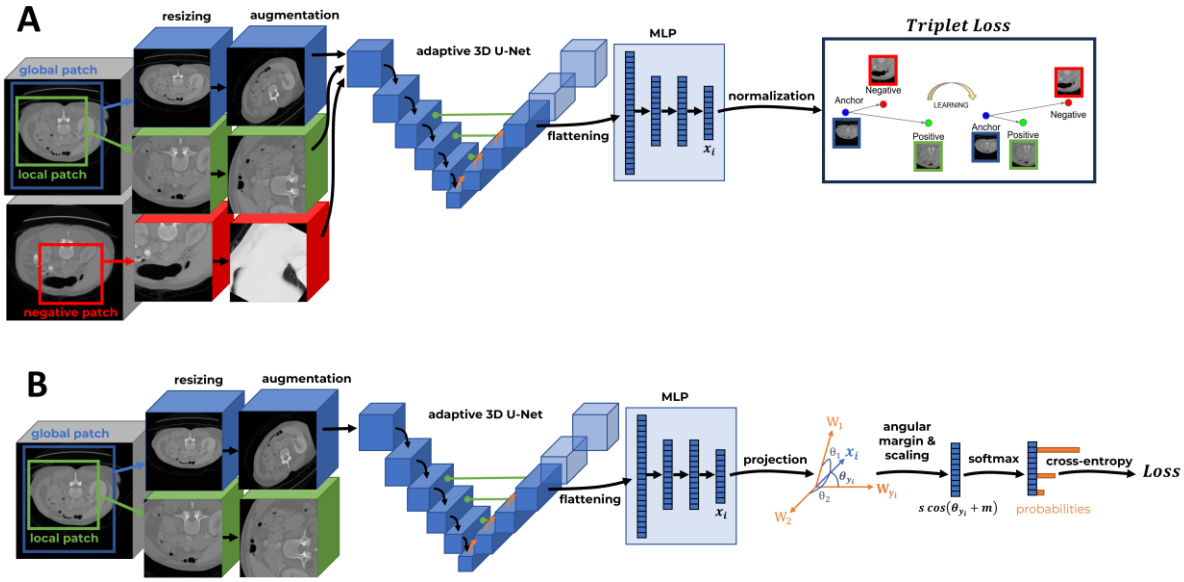


Figure 4-8 – **Pretraining an encoder-decoder on 3D images with (A) the Triplet loss and (B) the Arcface loss.** From left to right: the original images are first patched and resized using SmartPatch and augmented. All patches are then passed to the encoder-decoder. The encoder-decoder output is flattened and passed through a Multi-Layer Perceptron (MLP). For the Triplet loss (A), the embedding vector (x_i) is normalized and given to the loss function. For the Arcface loss (B), the embedding vector is projected in the space spanned by the last layer parameters to retrieve its angle, an angular penalty is added, and the cross-entropy function is applied.

Table 4-2 – **Summary of the main results of the Triplet and Arcface pretraining methods on the Pancreas dataset.** The Dice score (*right column*) are segmentation

results obtained on the test set of the Pancreas dataset. The first line (*Exp1*) is the supervised baseline (no pretraining). The last three lines (*Exp2-4*) are pretrained, retrained and tested on the the Pancreas dataset. The second line (*Exp2*) is a pretraining of the encoder only with the Triplet loss. The third line (*Exp3*) is a pretraining of the U-Net encoder and decoder with the Triplet loss and the fourth line (*Exp4*) with Arcface loss.

Experience index	**Pretraining model**	**Pretraining loss**	**Pretraining time GPU Nvidia A100 (in minutes)**	**Dice score**
Exp1	None	None	0	0.6683
Exp2	3D VGG-like	Triplet	415	0.6633
Exp3	3D U-Net	Triplet	1230	**0.6867**
Exp4	3D U-Net	Arcface	**553**	0.6843

Arcface loss. As for the Triplet loss, the Arcface loss [183] originates from face recognition problems. It is an improvement of the cross-entropy loss frequently used in classification problems. In this self-supervised context, each training image is assigned with a different class. The normalized embedding vector is the penultimate output (Figure 4-8-B). The last layer parameters are normalized and thus serve as a projection space and an intermediate before classification (*softmax*). This will let the model generalize properly and be able to construct a coherent embedding space for both seen (training) and unseen (testing) images. The Arcface loss can be written as follows:

$$L(\theta) = -\log\left(\frac{e^{s\cos(\theta_y+m)}}{e^{s\cos(\theta_y+m)} + \sum_{j=1,j\neq y}^{n} e^{s\cos(\theta_j)}}\right)$$

where $\theta = \text{acos}\,(v)$, v being the output of the last layer, θ_y is the element of index $y \in [\![1, n]\!]$ of the vector θ and n is the number of classes (typically the number of images in the pretraining dataset). The hyper-parameters s and m improve the inter-class and intra-class distribution of the embedding vectors and must be carefully chosen. On the Pancreas dataset, the Arcface pretraining of the encoder-decoder model did not improve the Triplet loss pretraining results but divided by more than 2 the pretraining time (Table 4-2-Exp4).

4.4 Conclusion

Arcface pretraining could thus be the best self-supervised candidate among all tested methods for 3D image segmentation. Indeed, in the scope of this thesis, existing self-supervised works such as Model Genesis, Taleb's methods and DINO were unfortunately not able to overpass supervised methods. While being a good candidate, Arcface has yet several constraints to pay attention to:

- **Loss diverges to "NaN"** (Not a Number). This issue is generally caused by the cross-entropy loss which can encourage last layers to raise their parameters beyond the bounds of half-precision floating-point format (float16, ±65504). This can be solved by either switching to single-precision (float32), but training may slow down, or normalizing each intermediate outputs, but the training may collapse to constant.
- **Constant collapse**, like DINO, happens when all embedding vectors are equal. The loss value then collapses to $-\log\left(\frac{1}{1+n\ e^{s(1-cos(m))}}\right)$. Adjusting s and m value can solve this issue but might be time-consuming. Changing the type of layer normalization (from layer normalization to batch normalization) can also be a solution.
- **Training divergences** happens when the loss progressively goes up to infinity (instead of going down to zero). If augmentation is used, deactivating it could solve this issue. Complexifying the model architecture (switching to an EfficientU-Net) did not show any improvement. This issue is still an open problem.
- **Over-pretraining** occurs when the pretraining is performed for too long. Training with the Arcface method (or the Triplet method) for 2000 epochs instead of 1000 on the Pancreas dataset surprisingly gives back the supervised accuracy (Dice score of 0.66). This problem is another open problem, and no existing publication was found tackling it.

Due to all these constraints, the research results done during this thesis would encourage to first try to use the Triplet loss before switching to the more optimized Arcface loss.

The modular architecture of Biom3d has strongly facilitated the experiments of this Chapter with self-supervised methods. However, partly due to the very performant

supervised baseline of Biom3d, it showed that the self-supervised methods have an existing but only limited improvement ability for 3D segmentation. These experiments thus raised lots of novel questions:

- Does the Triplet/Arcface self-supervision of a full U-Net work with other 3D datasets?
- Why does SmartPatch work so well? And what is the best SmartPatch strategy?
- If the decoder is so good at extracting features from a random encoder, is it possible to imagine a decoder-only model?
- Could a better architecture (Vision Transformer, etc.) perform as well with Triplet/Arcface self-supervision?
- The testing set is currently part of the pretraining set. Does the final accuracy change if it was not the case?
- Could it be good to pretrain on a dataset full of a large variety of 3D images and objects?
- Retraining partially erases the pretrained parameters, how to handle this oblivion phenomenon? By progressively unfreezing model weights? By using knowledge distillation? Continual learning is the deep learning subdomain studying this issue. Or by using semi-supervised learning?

Authors' contributions – cf. Appendices.

Chapter 5
Discussion

This Chapter will first return to the conclusions of Chapters 2 to 4 and discusses the numerous potential research directions opened during this thesis. The discussion will then look at the bigger implications and choices behind this research project.

5.1 About this work

This work originated from several biological questions on nuclear and chromatin morphologies and their relationship with gene expression. These questions led biologists to use microscopy to capture tridimensional images. To retrieve quantitative and statistical information, they exploited classical computer vision software, such as Fiji/ImageJ, to segment their images. However, this type of software reached its limit when facing the complexity and variability of nucleus and chromocentre shapes in *A. thaliana*.

The numerous promises brought by the novelty of deep learning methods were yet dulled by an important lack of reproducibility, as illustrated in the review generated as part of the work in this thesis [40] and the benchmarking presented in Chapter 2 where only nnU-Net stood out as sufficiently well designed to be useable.

With easy-of-use, modularity and sustainability in mind, a novel deep learning framework called Biom3d was developed during this project and exposed in Chapter 3. Being easy-to-use, Biom3d was successfully used by biologists and solved the initial nucleus and the chromocentre problems. Being flexible and optimized, Biom3d also overpassed the cutting-edge performance of nnU-Net on a wide variety of biological and medical problems. With its modular code architecture, Biom3d is not just another methods but a platform, such as Fiji/ImageJ, that aims to be sustainable and compatible with all recent innovations in deep learning.

Standard deep learning methods are unfortunately limited by their need of manually annotated dataset. Self-supervision aimed at tackling this issue by pretraining models on large unannotated dataset to extract information from the data before being retrained on the task of interest. Thanks to Biom3d modularity, a set of self-supervised

method were rapidly tested in Chapter 4. Among those, the newly created Triplet and Arcface methodologies trained, in an original way, on a full U-Net model showed significant improvements.

Deeply tinged with an interdisciplinary colour, the tools developed during the last three years (Biom3d framework and modules as well as self-supervised experiments) serve and will serve the two teams of biologists involved in this project. They will help them rapidly achieve the previously unreachable goal of precisely analysing large scale dataset of 3D image in a very short amount of time. With words of mouth and publications, these tools may even spread out and serve a larger audience. Ultimately, future developers may be interested in using pieces of code, be inspired by pieces of coding philosophy, or hopefully, be motivated by interdisciplinary research projects.

5.1.1 Reusing deep learning methods

In addition to the work done in Chapter 2, the lack of reproducibility of AI methods has been pinpointed by many and is an annual subject of publication. *Nature methods* journal, for instance, publishes an article on the topic every two years [162], [170], [171], the latest also stressing the fundamental importance of interdisciplinary work.

To bring the discussion further than code reproducibility, this thesis has introduced the notion of *code sustainability*. Too little effort and too little value is devoted to ensuring that code lasts over time, which is directly linked to code reusability by the largest possible community, including both end users and developers. *Nature methods* critics are mainly directed toward the scientific communities that do not publish their code or their data. In parallel, it, and other high ranked journals, put on a pedestal the novelty and the theoretical performance of published tools, as well as their illustrative biological and medical applications. However, they tend to ignore indispensable engineering contributions, which are neither a revolutionary “deep learning” method nor a novel graphical user interface, but the way the software is intelligently developed and organized on every scale level. The value of fundamental software engineering concepts, as was already conceptualized in the 90’s with books such as *Design Pattern* [184] or *Code Complete* [185] , should be further promoted. More generally, and as already understood by industries and companies, researchers and journals could give more credit to the smart and simple design of a tools rather than how (complicated) they look or how they efficiently solve one very specific problem. To give a biological

parallel, sequencing companies will probably give as much (if not more) credit to a contribution significantly optimizing an RNASeq protocol as the invention of a novel sequencing technique.

5.1.2 Biom3d perspectives

Chapter 3 presented Biom3d, a deep learning implementation of a 3D segmentation method, furnished with numerous software engineering contributions. It was developed with the hope of fostering good code practices and easing the use and the enhancement of deep learning methods for 3D images.

This code, as many others, is currently in the hand of too few people to be safe from oblivion. Yet, its development has been constantly and strongly supported by its small user community. It could thus be a significant plus to integrate it in well-known tools such as ZeroCostDL4Mic framework, Bioimage.io website, or as a new napari plugin. Finally, Biom3d modularity could also offer a large panel of possible modules: 2D U-Net, instance segmentation (with Cellpose or Stardist for instance), object classification and detection, active learning etc.

5.1.3 Future of methods for insufficiently annotated datasets

By progressively enlarging the scope of research of Chapter 4, it is easy to imagine plenty of potential research directions:

- Is it possible to generalize the Triplet/Arcface losses with an “Augmentation” loss where the pretraining task consists in guessing the random augmentation applied to input image?
- Can adversarial learning (from generative method) be a good semi-supervised approach? (Figure 5-1)

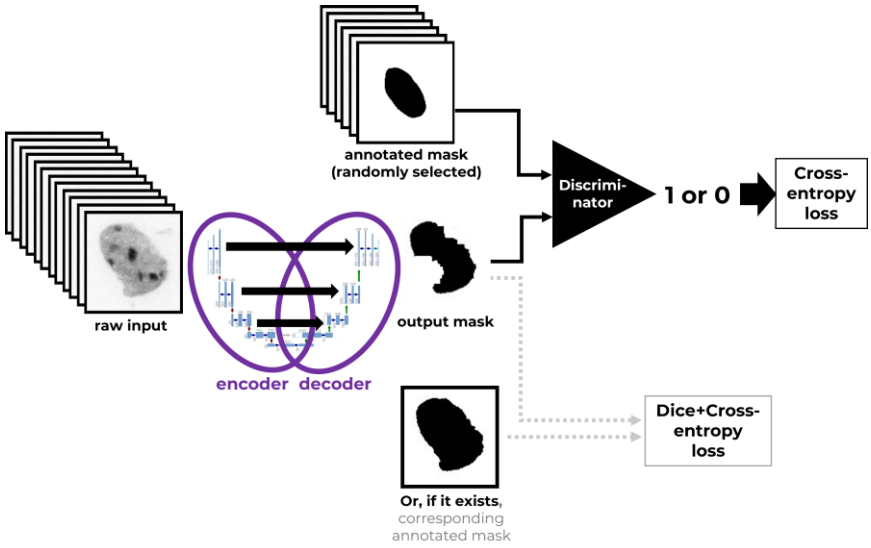


Figure 5-1 – **Idea of adversarial semi-supervised learning for 3D images.** A large set of unannotated images (*left*) is passed to a U-Net model. The output (*middle*) is then compared to a set of annotations (*top*) using a discriminator network and the Generative Adversarial [147] training paradigms (*right*). If an annotation corresponds to a raw image, it is used to guide the U-Net training (*bottom*).

- Are diffusion methods, performant methods combining U-Net with Attention layers, promising methods? Using a similar approach, "Universal Model" [88], for instance, managed to incorporate text in the U-Net training to guide the model on very large segmentation datasets. A single model could thus be very performant on all Medical Segmentation Decathlon tasks.
- Now that deep learning models can generalize better while getting bigger, they are more difficult to train, requiring to store Petabytes of data and Teraflops of computation power. Is mutualization a possibility? Is it better than a set of mutualized models? Can large model be fused (instead of sharing dataset) or partially trained by different teams (each team training only one section of the model)? Is an online equivalent of "ChatGPT" possible for bioimage analysts? Meta group has recently released Segment Anything [101] a "foundation" model for segmentation (they did not release their training code yet) which was soon adapted to 2D and 3D microscopy [169]. Does this type of active learning method represent the future of AI and image analysis?

5.2 A bigger and bigger picture

Maybe unconventionally, as I believe that such a PhD project was inevitably influenced by human minds and beliefs, I would like to end this thesis with a more personal-philosophical note and to step back to look at the even bigger picture (which is quite ironic for an image analysis thesis) in which is framed this work.

5.2.1 An interdisciplinary experience

On a local scale, I would like to come back to the choice that was offered to me when I started. Either I followed the path of what seemed to be the path of hundreds of engineers before me and played the role of an external actor, looking at biology and microscopy from a distant and critical point of view. Or I tried to dive headfirst into the marvelous but chaotic world of biology and be part of it. I would thus take the risk to be alone, and less advised on the technical side, but free and in adequacy with my belief that interdisciplinarity has the potential to bridge gaps between two, or more, giant domains in fundamental research. I followed the second path, and it determined many, if not all, subsequent choices made during this work.

I had to face a lot of challenges along the way. I had to constantly find a balance between three worlds: one full of mathematical abstraction, one full of theoretical and practical physics, and the last full of complex chemistry and mechanics. I had to exploit existing tools as much as I had to explore new horizons. I had to split my mind and my schedule between directing the project and executing it, while (too) often questioning myself on the way I should think.

From my current point of view, microscopy images are not intangible objects anymore. I know now, to a certain extent, the degree of flexibility that can have biologists and microscopists to generate their images and so I can participate in a debate to guide them to make the best of their images. More obviously, I am also not considering software and computer science tools as immovable, as I know existing tools, and to what extent I can use, adapt, and create them to generate exploitable results. This thesis was punctuated with countless scientific exchanges and my biologist advisors might, hopefully, now have a deeper point of view on the realistic possibilities given by computer science. These challenges and experiences of constant mutual learning and teaching were well worth the risks of my initial choice.

5.2.2 AI and its environment, a paradox?

On a more global scale, the notion of “code sustainability”, previously introduced, inevitably set this code and, more generally, this project in the human society network, its current environmental situation, and consequently in its political frame.

In matter of figures, this whole thesis project had a very small energetical impact. Its entire development required 7000 GPU hours distributed in three years. These GPUs needed 350W to work which means that 2.45MWh of energy was required in total, approximately as much as the energy necessary to build them (http://ecoinfo.cnrs.fr/), or the energy generated by an average wind turbine for one hour. In comparison, the single GPT-3 model requires 1297MWh to train, using 3.5 million GPU hours.

“Sustainable” is obviously relative to a political point of view: for some, it means 20 years, for others 1000 years. Biom3d in its current form was created to last as long as possible but will, very probably, not last 20 years. Yet, with its modular design, Biom3d is a platform with a great potential of evolution. Like ImageJ, originally designed to measure Southern blot gels, Biom3d may one day falls into the hands of many innovative developers and evolve in a direction that was not initially intended to be pursued. Additionally, as some other programs such as the Framasoft suite (https://framasoft.org/en/), it might also distil in the mind of some the ideas and the hope of building meaningful, sustainable, and ethical tools.

This is especially true for AI tools which are part of a highly competitive world where most of the main actors are motivated by fame, money, or power. The GAFAM are almost exclusively making profit using their finely adjusted (and private) recommendation algorithms (In 2022, the Facebook app generated a net revenue of 20$ per active user [186], by only using targeted advertising). Banks and private companies are also extensively exploiting AI bots to automate their transactions (trading, etc.) or their workforce (labour replacement, etc.). Finally, human moral can also be deeply endangered when AI is used uncontrollably by domains such as health insurances (search history, connected watch, etc.), surveillance systems (camera, text messages, etc.), justice or governments (recommendation algorithms during decisions or elections, etc.). Autonomous vehicles, face recognition, biological analysis, medical

diagnosis, robotics or even ChatGPT are probably just one small visible tip of the AIceberg.

AI tools have a short life cycle which seems to be shrinking every year, at a rate proportional to the speed of innovation growth. It might be symptomatic of a strong rebound effect. Should science, the very domain of human knowledge that raised red flags regarding exponential growth in a finite world, keep moving faster and faster? Is it to find innovative solutions more quickly to the problems caused by the too rapid evolution of human society? Isn't there a paradox?

Probably, yet, I believe that the destiny of AI is certainly not doomed and can be part of a degrowing and more ethical society. And I am not alone. Shaken by many rebelling movements (Scientist Rebellion, Les Soulèvement de la Terre, etc.), a constantly growing number of computer scientists have recently decided to act through "think tanks" (The Shift Project, Institut du Numérique Responsable, etc.), through associations (Institut Momentum, La Fabrique des Questions Simples, etc.), through novel media (GreenIT etc.), through initiatives (Planet Tech'Care, etc.), through companies (Carbone4, Convention des Entreprises pour le Climat), or through their own research (UMR-EspaceDEV, GDR-Labo1point5, etc.). Pushed, national governances are now reacting with new research grants (ANR-MITI-Frugalité) and ministerial missions (MiNumEco, Ademe). Last but not least, the UNESCO has made AI ethics and AI sustainability one of its main objectives (https://www.unesco.org/en/artificial-intelligence).

Degrowth is more than a need, it is inevitable, but it does not mean that human society should give up on its innovations and notably on some of its largest networks or industries, as often scale effect creates very optimized systems. Wise choices will have to be made, and I believe that researchers can and already have an important role to play as both advisors and actors. They will soon be confronted with the world of handymen who must innovate with tools and materials unsuited to their project. These unavoidable constraints will bring scientists closer and closer to the ground of society realities and political actions. The very ontology of research, its *Dasein*, must be rebuilt. And I believe in human mind to achieve it.

References

[1] M. Minsky, “Microscopy Apparatus,” US3013467A, 1957 [Online]. Available: http://www.freepatentsonline.com/3013467.html

[2] M. Chalfie *et al.*, “Green fluorescent protein as a marker for gene expression,” *Science (80-.).*, vol. 10, no. 5, p. 151, 1994, doi: 10.1016/0168-9525(94)90088-4.

[3] S. Suzuki *et al.*, “Topological structural analysis of digitized binary images by border following,” *Comput. Vision, Graph. Image Process.*, vol. 30, no. 1, pp. 32–46, 1985, doi: 10.1016/0734-189X(85)90016-7.

[4] W. E. Lorensen *et al.*, “Marching cubes: A high resolution 3D surface construction algorithm,” in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987*, in SIGGRAPH ’87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 163–169. doi: 10.1145/37401.37422.

[5] S. Fourey *et al.*, “Normals estimation for digital surfaces based on convolutions,” *Comput. Graph.*, vol. 33, no. 1, pp. 2–10, 2009, doi: 10.1016/j.cag.2008.11.003.

[6] N. Sofroniew *et al.*, “napari.” [Online]. Available: https://doi.org/10.5281/zenodo.5848842

[7] S. Berg *et al.*, “Ilastik: Interactive Machine Learning for (Bio)Image Analysis,” *Nat. Methods*, vol. 16, no. 12, pp. 1226–1232, 2019, doi: 10.1038/s41592-019-0582-9.

[8] F. Eibe *et al.*, “The WEKA workbench,” *Data Min. Pract. Mach. Learn. Tools Tech.*, pp. 553–571, 2016, doi: 10.1016/b978-0-12-804291-5.00024-6.

[9] S. Lang *et al.*, “WekaDeeplearning4j: A deep learning package for Weka based on Deeplearning4j,” *Knowledge-Based Syst.*, vol. 178, pp. 48–50, 2019, doi: 10.1016/j.knosys.2019.04.013.

[10] L. Breiman, “Random Forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[11] Institut Pasteur, “DIVA.” [Online]. Available: https://research.pasteur.fr/en/project/data-integration-and-visualisation-in-augmented-and-virtual-environments/

[12] S. Pieper *et al.*, “3D Slicer,” in *2004 2nd IEEE International Symposium on Biomedical Imaging: Nano to Macro (IEEE Cat No. 04EX821)*, 2004, p. 632–635 Vol. 1. doi: 10.1109/ISBI.2004.1398617.

[13] J. Ahrens *et al.*, “ParaView: An End-User Tool for Large Data Visualization ParaViewWeb View project,” *Vis. Handbook, Elsevier*, vol. 836, 2005, [Online]. Available: https://www.researchgate.net/publication/247111133

[14] D. Mastronarde *et al.*, “3dmod.” [Online]. Available: https://bio3d.colorado.edu/imod/

[15] K. M. Boergens *et al.*, “WebKnossos: Efficient online 3D data annotation for connectomics,” *Nat. Methods*, vol. 14, no. 7, pp. 691–694, 2017, doi: 10.1038/nmeth.4331.

[16] M. Rocklin, “Dask: Parallel Computation with Blocked algorithms and Task Scheduling,” in *Proceedings of the 14th Python in Science Conference*, 2015, pp. 126–132. doi: 10.25080/majora-7b98e3ed-013.

[17] J. Schindelin *et al.*, “Fiji: An open-source platform for biological-image analysis,” *Nat. Methods*, vol. 9, no. 7, pp. 676–682, 2012, doi: 10.1038/nmeth.2019.

[18] D. Legland *et al.*, “MorphoLibJ: integrated library and plugins for mathematical morphology with ImageJ,” *Bioinformatics*, vol. 32, no. 22, pp. 3532–3534, Nov. 2016, doi: 10.1093/bioinformatics/btw413.

[19] D. Zink *et al.*, “Nuclear structure in cancer cells,” *Nat. Rev. Cancer*, vol. 4, no. 9, pp. 677–687, 2004, doi: 10.1038/nrc1430.

[20] D. E. Monica Pradillo *et al.*, “The nuclear envelope in higher plant mitosis and meiosis,” *Nucleus*, vol. 10, no. 1, pp. 55–66, 2019, doi: 10.1080/19491034.2019.1587277.

[21] E. D. M. S. T. C, “Advancing knowledge of the plant nuclear periphery and its application for crop science.” 2020.

[22] D. E. Monica Pradillo *et al.*, “The nuclear envelope in higher plant mitosis and meiosis,” *Nucleus*, vol. 10, no. 1, pp. 55–66, 2019, doi: 10.1080/19491034.2019.1587277.

[23] C. Tatout *et al.*, “The INDEPTH (Impact of Nuclear Domains On Gene Expression and Plant Traits) Academy – a community resource for plant science,” *J. Exp. Bot.*, p. erac005, Jan. 2022, doi: 10.1093/jxb/erac005.

[24] A. V. Probst *et al.*, “Epigenetic inheritance during the cell cycle,” *Nat. Rev. Mol. Cell Biol.*, vol. 10, no. 3, pp. 192–206, 2009, doi: 10.1038/nrm2640.

[25] A. Pecinka *et al.*, “Chromosome territory arrangement and homologous pairing in nuclei of Arabidopsis thaliana are predominantly random except for NOR-bearing chromosomes.,” *Chromosoma*, vol. 113, no. 5, pp. 258–269, Nov. 2004, doi: 10.1007/s00412-004-0316-2.

[26] T. Cremer *et al.*, “Chromosome territories.,” *Cold Spring Harb. Perspect. Biol.*, vol. 2, no. 3, 2010, doi: 10.1101/cshperspect.a003889.

[27] K. Graumann, “Evidence for LINC1-SUN associations at the plant nuclear periphery,” *PLoS One*, vol. 9, no. 3, 2014, doi: 10.1371/journal.pone.0093406.

[28] A. Poulet *et al.*, “Exploring the evolution of the proteins of the plant nuclear envelope,” *Nucleus*, vol. 8, no. 1, pp. 46–59, 2017, doi: 10.1080/19491034.2016.1236166.

[29] T. Dubos *et al.*, “Automated 3D bio-imaging analysis of nuclear organization by NucleusJ 2.0,” *Nucleus*, vol. 11, no. 1, pp. 315–329, 2020, doi: 10.1080/19491034.2020.1845012.

[30] S. Mermet *et al.*, “Evolutionarily conserved protein motifs drive interactions

between the plant nucleoskeleton and nuclear pores," *Plant Cell*, p. koad236, Sep. 2023, doi: 10.1093/plcell/koad236.

[31] S. Beucher *et al.*, "Use of Watersheds in Contour Detection," *Int. Work. Image Process. Real-time Edge Motion Detect.*, pp. 12–21, 1979, [Online]. Available: http://www.citeulike.org/group/7252/article/4083187

[32] A. D. Elliott, "Confocal Microscopy: Principles and Modern Practices," *Curr. Protoc. Cytom.*, vol. 92, no. 1, p. e68, Mar. 2020, doi: 10.1002/cpcy.68.

[33] C. J. Peddie *et al.*, "Volume electron microscopy," *Nat. Rev. Methods Prim.*, vol. 2, no. 1, p. 51, 2022, doi: 10.1038/s43586-022-00131-9.

[34] S. Aryal, "microbiologyinfo.com." [Online]. Available: https://microbiologyinfo.com/differences-between-light-microscope-and-electron-microscope/

[35] K. Coyne, "nationalmaglab.org." [Online]. Available: https://nationalmaglab.org/magnet-academy/read-science-stories/science-simplified/mri-a-guided-tour/

[36] M. Maqbool, "Computed Tomography BT - An Introduction to Medical Physics," M. Maqbool, Ed., Cham: Springer International Publishing, 2017, pp. 221–262. doi: 10.1007/978-3-319-61540-0_8.

[37] B. H. Menze *et al.*, "The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)," *IEEE Trans. Med. Imaging*, vol. 34, no. 10, pp. 1993–2024, 2015, doi: 10.1109/TMI.2014.2377694.

[38] N. Otsu, "A threshold selection method from grey-level histograms.Title," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 9, pp. 62–66, 1979.

[39] R. Jarvis, "On the identification of the convex hull of a finite set of points in the plane.," *Inf Process Lett*, vol. 2, no. 18–21, 1973.

[40] G. Mougeot *et al.*, "Deep learning -– promises for 3D nuclear imaging: a guide for biologists," *J. Cell Sci.*, vol. 135, no. 7, p. jcs258986, Apr. 2022, doi: 10.1242/jcs.258986.

[41] D. C. Cireşan *et al.*, *Flexible, high performance convolutional neural networks for image classification*. 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-210.

[42] A. Krizhevsky *et al.*, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[43] T. Y. Lin *et al.*, "Microsoft COCO: Common objects in context," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8693 LNCS, no. PART 5. pp. 740–755, 2014. doi: 10.1007/978-3-319-10602-1_48.

[44] C. R. Brice *et al.*, "Scene analysis using regions," *Artif. Intell.*, vol. 1, no. 3, pp. 205–226, 1970, doi: https://doi.org/10.1016/0004-3702(70)90008-1.

[45] T. Pavlidis, “Segmentation of pictures and maps through functional approximation,” *Comput. Graph. Image Process.*, vol. 1, no. 4, pp. 360–372, 1972, doi: https://doi.org/10.1016/0146-664X(72)90021-4.

[46] A. Rosenfeld *et al.*, *Digital Picture Processing, Volume 1*, 2nd ed., vol. 53, no. 9. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1982.

[47] H. Steinhaus, “Sur la division des corps matériels en parties.,” *Bull. Acad. Pol. Sci. Cl.*, vol. III, no. 4, pp. 801–804, 1956.

[48] L. Barghout *et al.*, “Real-world scene perception and perceptual organization: Lessons from Computer Vision,” *J. Vis.*, vol. 13, no. 9, pp. 709–709, 2013, doi: 10.1167/13.9.709.

[49] S. Geman *et al.*, “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-6, no. 6, pp. 721–741, 1984, doi: 10.1109/TPAMI.1984.4767596.

[50] M. Kass *et al.*, “Snakes: Active contour models,” *Int. J. Comput. Vis.*, vol. 1, no. 4, pp. 321–331, 1988, doi: 10.1007/BF00133570.

[51] J. M. Morel *et al.*, *Variational Methods in Image Segmentation*. 1995. doi: 10.1007/978-1-4684-0567-5.

[52] T. Dubos *et al.*, “NODeJ: an ImageJ plugin for 3D segmentation of nuclear objects,” *BMC Bioinformatics*, vol. 23, no. 1, pp. 1–11, 2022, doi: 10.1186/s12859-022-04743-6.

[53] D. J. Barry *et al.*, “GIANI: open-source software for automated analysis of 3D microscopy images,” *bioRxiv*, 2021, doi: 10.1101/2020.10.15.340810.

[54] K. Fukushima, “位置ずれに影響されないパターン認識機構の神経回路モデル—ネオコグニトロン—,” *Trans. IECE*, vol. J62–A, no. 10, pp. 658–665, 1979.

[55] D. H. Hubel *et al.*, “Receptive fields of single neurones in the cat’s striate cortex,” *J. Physiol.*, vol. 148, no. 3, pp. 574–591, Oct. 1959, doi: 10.1113/jphysiol.1959.sp006308.

[56] D. C. Cireşan *et al.*, “Mitosis detection in breast cancer histology images with deep neural networks,” in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, K. Mori, I. Sakuma, Y. Sato, C. Barillot, and N. Navab, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 411–418. doi: 10.1007/978-3-642-40763-5_51.

[57] E. Moen *et al.*, “Deep learning for cellular image analysis,” *Nat. Methods*, vol. 16, no. 12, pp. 1233–1246, 2019, doi: 10.1038/s41592-019-0403-1.

[58] O. Ronneberger *et al.*, “U-net: Convolutional networks for biomedical image segmentation,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9351, pp. 234–241, 2015, doi: 10.1007/978-3-319-24574-4_28.

[59] ParallelR, “ImageNet result evolution.” [Online]. Available: https://parallelr.com/2016/02/13/r-deep-neural-network-from-scratch/

[60] K. Simonyan *et al.*, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *CoRR*, vol. abs/1409.1, 2014, [Online]. Available: http://arxiv.org/abs/1409.1556

[61] K. He *et al.*, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016–Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[62] M. Tan *et al.*, “EfficientNet: Rethinking model scaling for convolutional neural networks,” *36th Int. Conf. Mach. Learn. ICML 2019*, vol. 2019–June, pp. 10691–10700, 2019, [Online]. Available: http://arxiv.org/abs/1905.11946

[63] A. Dosovitskiy *et al.*, “an Image Is Worth 16X16 Words: Transformers for Image Recognition At Scale,” *ICLR 2021 - 9th International Conference on Learning Representations*. 2021.

[64] Z. Liu *et al.*, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows,” *Proceedings of the IEEE International Conference on Computer Vision*. pp. 9992–10002, 2021. doi: 10.1109/ICCV48922.2021.00986.

[65] M. D. Zeiler *et al.*, “Visualizing and understanding convolutional networks,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8689 LNCS, no. PART 1. pp. 818–833, 2014. doi: 10.1007/978-3-319-10590-1_53.

[66] E. Ben-Baruch *et al.*, “Attention Is All You Need,” *Advances in Neural Information Processing Systems*, vol. 16, no. D1. pp. 687--694, 2019. [Online]. Available: https://academic.oup.com/nar/article/47/D1/D427/5144153

[67] K. Islam, “Recent Advances in Vision Transformer: A Survey and Outlook of Recent Work.” 2022. [Online]. Available: http://arxiv.org/abs/2203.01536

[68] M. Weigert *et al.*, “Star-convex polyhedra for 3D object detection and segmentation in microscopy,” *Proc. - 2020 IEEE Winter Conf. Appl. Comput. Vision, WACV 2020*, pp. 3655–3662, 2020, doi: 10.1109/WACV45572.2020.9093435.

[69] E. Gómez-de-Mariscal *et al.*, “DeepImageJ: A user-friendly environment to run deep learning models in ImageJ,” *Nat. Methods*, vol. 18, no. 10, pp. 1192–1195, 2021, doi: 10.1038/s41592-021-01262-9.

[70] L. von Chamier *et al.*, “ZeroCostDL4Mic: An open platform to use deep-learning in microscopy,” *bioRxiv*, p. 2020.03.20.000133, 2020, doi: 10.1101/2020.03.20.000133.

[71] L. Heinrich *et al.*, “Whole-cell organelle segmentation in volume electron microscopy,” *Nature*, vol. 599, no. 7883, pp. 141–146, 2021, doi: 10.1038/s41586-021-03977-3.

[72] C. Stringer *et al.*, “Cellpose: a generalist algorithm for cellular segmentation,” *Nat. Methods*, vol. 18, no. 1, pp. 100–106, 2021, doi: 10.1038/s41592-020-01018-x.

[73] M. Pachitariu *et al.*, “Cellpose 2.0: how to train your own model,” *Nat.*

Methods, vol. 19, no. 12, pp. 1634–1641, 2022, doi: 10.1038/s41592-022-01663-4.

[74] K. Sugawara *et al.*, "Tracking cell lineages in 3D by incremental deep learning," *Elife*, vol. 11, p. e69380, 2022, doi: 10.7554/eLife.69380.

[75] C. Wen *et al.*, "3DeeCellTracker, a deep learning-based pipeline for segmenting and tracking cells in 3D time lapse images," *Elife*, vol. 10, p. e59187, 2021, doi: 10.7554/eLife.59187.

[76] T. O. Buchholz *et al.*, "Content-aware image restoration for electron microscopy," *Methods Cell Biol.*, vol. 152, pp. 277–289, 2019, doi: 10.1016/bs.mcb.2019.05.001.

[77] T. O. Buchholz *et al.*, "DenoiSeg: Joint Denoising and Segmentation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12535 LNCS. pp. 324–337, 2020. doi: 10.1007/978-3-030-66415-2_21.

[78] A. Krull *et al.*, "Noise2void-Learning denoising from single noisy images," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2124–2132. doi: 10.1109/CVPR.2019.00223.

[79] F. De Chaumont *et al.*, "Icy: An open bioimage informatics platform for extended reproducible research," *Nat. Methods*, vol. 9, no. 7, pp. 690–696, 2012, doi: 10.1038/nmeth.2075.

[80] A. E. Carpenter *et al.*, "CellProfiler: Image analysis software for identifying and quantifying cell phenotypes," *Genome Biol.*, vol. 7, no. 10, p. R100, 2006, doi: 10.1186/gb-2006-7-10-r100.

[81] W. Ouyang *et al.*, "ImJoy: an open-source computational platform for the deep learning era," *Nat. Methods*, vol. 16, no. 12, pp. 1199–1200, 2019, doi: 10.1038/s41592-019-0627-0.

[82] T. Pietzsch *et al.*, "Mastodon – a large-scale tracking and track-editing framework for large, multi-view images." [Online]. Available: https://github.com/mastodon-sc/mastodon

[83] A. Virzì *et al.*, "Comprehensive Review of 3D Segmentation Software Tools for MRI Usable for Pelvic Surgery Planning," *J. Digit. Imaging*, vol. 33, no. 1, pp. 99–110, 2020, doi: 10.1007/s10278-019-00239-7.

[84] P. A. Yushkevich *et al.*, "User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability," *Neuroimage*, vol. 31, no. 3, pp. 1116–1128, 2006, doi: 10.1016/j.neuroimage.2006.01.015.

[85] I. Wolf *et al.*, "The medical imaging interaction toolkit (MITK): a toolkit facilitating the creation of interactive software by extending VTK and ITK," *Med. Imaging 2004 Vis. Image-Guided Proced. Disp.*, vol. 5367, p. 16, 2004, doi: 10.1117/12.535112.

[86] F. Isensee *et al.*, "nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation," *Nat. Methods*, vol. 18, no. 2, pp. 203–211,

2021, doi: 10.1038/s41592-020-01008-z.

[87] J. Wasserthal *et al.*, "TotalSegmentator: robust segmentation of 104 anatomical structures in CT images," 2022, [Online]. Available: http://arxiv.org/abs/2208.05868

[88] J. Liu *et al.*, "CLIP-Driven Universal Model for Organ Segmentation and Tumor Detection." arXiv, 2023. doi: 10.48550/ARXIV.2301.00785.

[89] M. Baumgartner *et al.*, "nnDetection: A Self-configuring Method for Medical Object Detection," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12905 LNCS, pp. 530–539, 2021, doi: 10.1007/978-3-030-87240-3_51.

[90] M. J. McAuliffe *et al.*, "Medical Image Processing, Analysis and Visualization in clinical research," in *Proceedings 14th IEEE Symposium on Computer-Based Medical Systems. CBMS 2001*, 2001, pp. 381–386. doi: 10.1109/CBMS.2001.941749.

[91] N. Toussaint *et al.*, "MedINRIA: Medical Image Navigation and Research Tool by INRIA," 2007.

[92] D. Rivière *et al.*, "Anatomist: a python framework for interactive 3D visualization of neuroimaging data," *Python Neurosci. Work.*, pp. 3–4, 2011, [Online]. Available: http://brainvisa.info/doc/pyanatomist/sphinx/

[93] B. Fischl, "FreeSurfer," *Neuroimage*, vol. 62, no. 2, pp. 774–781, Aug. 2012, doi: 10.1016/j.neuroimage.2012.01.021.

[94] M. Jenkinson *et al.*, "FSL - Review," *Neuroimage*, vol. 62, no. 2, pp. 782–90, Aug. 2012, doi: 10.1016/j.neuroimage.2011.09.015.

[95] CIBC, "Seg3D: Volumetric Image Segmentation and Visualization. Scientific Computing and Imaging Institute (SCI)." 2015.

[96] J. Schwartz *et al.*, "Real-time 3D analysis during electron tomography using tomviz," *Nat. Commun.*, vol. 13, no. 1, p. 4458, 2022, doi: 10.1038/s41467-022-32046-0.

[97] J. Liang *et al.*, "Vaa3D-x for cross-platform teravoxel-scale immersive exploration of multidimensional image data," *Bioinformatics*, vol. 39, no. 1, p. btac794, Jan. 2023, doi: 10.1093/bioinformatics/btac794.

[98] A. Kensert *et al.*, "Transfer Learning with Deep Convolutional Neural Networks for Classifying Cellular Morphological Changes," *SLAS Discov.*, vol. 24, no. 4, pp. 466–475, 2019, doi: 10.1177/2472555218818756.

[99] M. Raghu *et al.*, "Transfusion: Understanding transfer learning for medical imaging," *Advances in Neural Information Processing Systems*, vol. 32. 2019.

[100] D. Wiesner *et al.*, "CytoPacq: a web-interface for simulating multi-dimensional cell imaging," *Bioinformatics*, vol. 35, no. 21, pp. 4531–4533, Nov. 2019, doi: 10.1093/bioinformatics/btz417.

[101] A. Kirillov *et al.*, "Segment Anything." 2023.

[102] V. Ljosa *et al.*, "Annotated high-throughput microscopy image sets for

validation," *Nat. Methods*, vol. 9, no. 7, p. 637, 2012, doi: 10.1038/nmeth.2083.

[103] E. Williams *et al.*, "Image Data Resource: A bioimage data integration and publication platform," *Nat. Methods*, vol. 14, no. 8, pp. 775–781, 2017, doi: 10.1038/nmeth.4326.

[104] C. Allan *et al.*, "OMERO: Flexible, model-driven data management for experimental biology," *Nat. Methods*, vol. 9, no. 3, pp. 245–253, Feb. 2012, doi: 10.1038/nmeth.1896.

[105] L. S. V Thomas *et al.*, "Fiji plugins for qualitative image annotations: routine analysis and application to image classification," *F1000Research*, vol. 9, p. 1248, Feb. 2021, doi: 10.12688/f1000research.26872.2.

[106] W. Zimmer *et al.*, "3D BAT: A Semi-Automatic, Web-based 3D Annotation Toolbox for Full-Surround, Multi-Modal Data Streams." 2019.

[107] A. Dutta *et al.*, "The VIA annotation software for images, audio and video," *MM 2019 - Proc. 27th ACM Int. Conf. Multimed.*, pp. 2276–2279, 2019, doi: 10.1145/3343031.3350535.

[108] Tzutalin, "Labellmg." [Online]. Available: https://github.com/tzutalin/labelImg

[109] P. Bankhead *et al.*, "QuPath: Open source software for digital pathology image analysis," *Sci. Rep.*, vol. 7, no. 1, 2017, doi: 10.1038/s41598-017-17204-5.

[110] "Paintera." [Online]. Available: https://github.com/saalfeldlab/paintera

[111] M. Prakash *et al.*, "Leveraging Self-supervised Denoising for Image Segmentation," in *Proc. - Int. Symp. Biomed. Imaging*, 2020, pp. 428–432. doi: 10.1109/ISBI45749.2020.9098559.

[112] A. S. Goncharova *et al.*, "Improving Blind Spot Denoising for Microscopy BT - Computer Vision – ECCV 2020 Workshops," A. Bartoli and A. Fusiello, Eds., Cham: Springer International Publishing, 2020, pp. 380–393.

[113] J. Chen *et al.*, "Three-dimensional residual channel attention networks denoise and sharpen fluorescence microscopy image volumes," *Nat. Methods*, vol. 18, no. 6, pp. 678–687, 2021, doi: 10.1038/s41592-021-01155-x.

[114] H. Qu *et al.*, "Joint segmentation and fine-grained classification of nuclei in histopathology images," in *Proc. - Int. Symp. Biomed. Imaging*, 2019, pp. 900–904. doi: 10.1109/ISBI.2019.8759457.

[115] M. Tofighi *et al.*, "Prior Information Guided Regularized Deep Learning for Cell Nucleus Detection," *IEEE Trans. Med. Imaging*, vol. 38, no. 9, pp. 2047–2058, 2019, doi: 10.1109/TMI.2019.2895318.

[116] F. Xing *et al.*, "Pixel-to-Pixel Learning with Weak Supervision for Single-Stage Nucleus Recognition in Ki67 Images," *IEEE Trans. Biomed. Eng.*, vol. 66, no. 11, pp. 3088–3097, Nov. 2019, doi: 10.1109/TBME.2019.2900378.

[117] M. Valkonen *et al.*, "Generalized fixation invariant nuclei detection through domain adaptation based deep learning," *IEEE J. Biomed. Heal. Informatics*,

p. 1, 2020, doi: 10.1109/JBHI.2020.3039414.

[118] Y. Tokuoka *et al.*, "3D convolutional neural networks-based segmentation to acquire quantitative criteria of the nucleus during mouse embryogenesis," *npj Syst. Biol. Appl.*, vol. 6, no. 1, pp. 1–12, 2020, doi: 10.1038/s41540-020-00152-8.

[119] J. Redmon *et al.*, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016–Decem. pp. 779–788, 2016. doi: 10.1109/CVPR.2016.91.

[120] S. Ren *et al.*, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[121] Z. Zhou *et al.*, "Unet++: A nested u-net architecture for medical image segmentation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11045 LNCS. pp. 3–11, 2018. doi: 10.1007/978-3-030-00889-5_1.

[122] J. Wang *et al.*, "Deep High-Resolution Representation Learning for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 10. pp. 3349–3364, 2021. doi: 10.1109/TPAMI.2020.2983686.

[123] A. Hatamizadeh *et al.*, "Swin UNETR: Swin Transformers for Semantic Segmentation of Brain Tumors in MRI Images," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12962 LNCS. pp. 272–284, 2022. doi: 10.1007/978-3-031-08999-2_22.

[124] T.-Y. Lin *et al.*, "Feature Pyramid Networks for Object Detection," *CoRR*, vol. abs/1612.0, 2016, [Online]. Available: http://arxiv.org/abs/1612.03144

[125] Ö. Çiçek *et al.*, "3D U-net: Learning dense volumetric segmentation from sparse annotation," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9901 LNCS, pp. 424–432, 2016, doi: 10.1007/978-3-319-46723-8_49.

[126] J. C. Caicedo *et al.*, "Nucleus segmentation across imaging experiments: the 2018 Data Science Bowl," *Nat. Methods*, vol. 16, no. 12, pp. 1247–1253, 2019, doi: 10.1038/s41592-019-0612-7.

[127] N. F. Greenwald *et al.*, "Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning," *Nat. Biotechnol.*, p. 2021.03.01.431313, 2021, doi: 10.1038/s41587-021-01094-0.

[128] L. Wu *et al.*, "NISNet3D: three-dimensional nuclear synthesis and instance segmentation for fluorescence microscopy images," *Sci. Rep.*, vol. 13, no. 1, p. 9533, 2023, doi: 10.1038/s41598-023-36243-9.

[129] K. He *et al.*, "Mask R-CNN," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, Mar. 2020, doi: 10.1109/TPAMI.2018.2844175.

[130] L. Yang *et al.*, “NuSeT: A deep learning tool for reliably separating and analyzing crowded cells,” *PLoS Comput. Biol.*, vol. 16, no. 9, pp. 1–20, 2020, doi: 10.1371/journal.pcbi.1008193.

[131] A. A. Taha *et al.*, “Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool,” *BMC Med. Imaging*, vol. 15, no. 1, p. 29, 2015, doi: 10.1186/s12880-015-0068-x.

[132] A. Reinke *et al.*, “Common Limitations of Image Processing Metrics: A Picture Story.” 2022.

[133] U. Rubens *et al.*, “BIAFLOWS: A collaborative framework to reproducibly deploy and benchmark bioimage analysis workflows,” *bioRxiv*, pp. 1–13, 2019, doi: 10.1101/707489.

[134] C. Blanchet *et al.*, “IFB-Biosphère: Services cloud pour l’analyse des données des sciences de la vie,” *Journées RESeaux-JRES 2019*, 2019.

[135] H. Larochelle *et al.*, “An empirical evaluation of deep architectures on problems with many factors of variation,” *ACM Int. Conf. Proceeding Ser.*, vol. 227, pp. 473–480, 2007, doi: 10.1145/1273496.1273556.

[136] Bergstra J *et al.*, “Random search for hyper-parameter optimization,” *J. Mach. Learn. Res.*, vol. 13, no. 2, pp. 281–305, 2012.

[137] J. Bergstra *et al.*, “Algorithms for hyper-parameter optimization,” *Adv. Neural Inf. Process. Syst. 24 25th Annu. Conf. Neural Inf. Process. Syst. 2011, NIPS 2011*, 2011.

[138] Poli, Ricardo *et al.*, “A Field Guide to Genetic Programming ,” pp. 160–164, 2005.

[139] B. Zoph *et al.*, “Neural architecture search with reinforcement learning,” *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, 2017.

[140] B. Zoph *et al.*, “Learning Transferable Architectures for Scalable Image Recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 8697–8710, 2018, doi: 10.1109/CVPR.2018.00907.

[141] S. Kim *et al.*, “Scalable Neural Architecture Search for 3D Medical Image Segmentation,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11766 LNCS, pp. 220–228, 2019, doi: 10.1007/978-3-030-32248-9_25.

[142] M. Antonelli *et al.*, “The Medical Segmentation Decathlon,” *Nat. Commun.*, vol. 13, no. 1, p. 4128, 2022, doi: 10.1038/s41467-022-30695-9.

[143] A. Buslaev *et al.*, “Albumentations: Fast and Flexible Image Augmentations,” *Information*, vol. 11, no. 2, 2020, doi: 10.3390/info11020125.

[144] F. Isensee *et al.*, “Batchgenerators - a Python Framework for Data Augmentation,” 2020, [Online]. Available: https://zenodo.org/record/3632567#.YxpIRmxBzvI

[145] A. Jungo *et al.*, “pymia: A Python package for data handling and evaluation in deep learning-based medical image analysis,” *Comput. Methods Programs Biomed.*, vol. 198, p. 105796, 2021, doi: 10.1016/j.cmpb.2020.105796.

[146] F. Pérez-García *et al.*, “TorchIO: A Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning,” *Comput. Methods Programs Biomed.*, vol. 208, p. 106236, 2021, doi: https://doi.org/10.1016/j.cmpb.2021.106236.

[147] I. J. Goodfellow *et al.*, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems*, vol. 3, no. January, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 2672–2680. doi: 10.3156/jsoft.29.5_177_2.

[148] Y. Zhang *et al.*, “DatasetGAN: Efficient Labeled Data Factory with Minimal Human Effort,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. pp. 10140–10150, 2021. doi: 10.1109/CVPR46437.2021.01001.

[149] K. W. Dunn *et al.*, “DeepSynth: Three-dimensional nuclear segmentation of biological images using neural networks trained with synthetic data,” *Sci. Rep.*, vol. 9, no. 1, 2019, doi: 10.1038/s41598-019-54244-5.

[150] C. Fu *et al.*, “Three dimensional fluorescence microscopy image synthesis and segmentation,” *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2018–June, pp. 2302–2310, Jun. 2018, doi: 10.1109/CVPRW.2018.00298.

[151] L. Wu *et al.*, “RCNN-SliceNet: A slice and cluster approach for nuclei centroid detection in three-dimensional fluorescence microscopy images,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. pp. 3750–3760, 2021. doi: 10.1109/CVPRW53098.2021.00416.

[152] J. Y. Zhu *et al.*, “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks,” *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017–Octob. pp. 2242–2251, 2017. doi: 10.1109/ICCV.2017.244.

[153] H. Qu *et al.*, “Nuclei Segmentation Using Mixed Points and Masks Selected from Uncertainty,” in *Proc. - Int. Symp. Biomed. Imaging*, 2020, pp. 973–976. doi: 10.1109/ISBI45749.2020.9098474.

[154] J. Rony *et al.*, “Deep Weakly-Supervised Learning Methods for Classification and Localization in Histology Images : A Survey,” *Arxiv*, 2020.

[155] Z. Zhao *et al.*, “Deep Learning Based Instance Segmentation in 3D Biomedical Images Using Weak Annotation BT - Medical Image Computing and Computer Assisted Intervention – MICCAI 2018,” A. F. Frangi, J. A. Schnabel, C. Davatzikos, C. Alberola-López, and G. Fichtinger, Eds., Cham: Springer International Publishing, 2018, pp. 352–360.

[156] S. Budd *et al.*, “A survey on active learning and human-in-the-loop deep learning for medical image analysis,” *Medical Image Analysis*, vol. 71. Elsevier B.V., p. 102062, Jul. 01, 2021. doi: 10.1016/j.media.2021.102062.

[157] W. Shao *et al.*, “Deep active learning for nucleus classification in pathology images,” in *Proc. - Int. Symp. Biomed. Imaging*, 2018, pp. 199–202. doi: 10.1109/ISBI.2018.8363554.

[158] S. Wen *et al.*, “Comparison of Different Classifiers with Active Learning to Support Quality Control in Nucleus Segmentation in Pathology Images.,” *AMIA Jt. Summits Transl. Sci. proceedings. AMIA Jt. Summits Transl. Sci.*, vol. 2017, pp. 227–236, 2018.

[159] A. Kolesnikov *et al.*, “Big Transfer (BiT): General Visual Representation Learning,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12350 LNCS. pp. 491–507, 2020. doi: 10.1007/978-3-030-58558-7_29.

[160] M. Sahasrabudhe *et al.*, “Self-supervised Nuclei Segmentation in Histopathological Images Using Attention BT - Medical Image Computing and Computer Assisted Intervention – MICCAI 2020,” A. L. Martel, P. Abolmaesumi, D. Stoyanov, D. Mateus, M. A. Zuluaga, S. K. Zhou, D. Racoceanu, and L. Joskowicz, Eds., Cham: Springer International Publishing, 2020, pp. 393–402.

[161] M. Caron *et al.*, “Emerging Properties in Self-Supervised Vision Transformers,” *Proceedings of the IEEE International Conference on Computer Vision*. pp. 9630–9640, 2021. doi: 10.1109/ICCV48922.2021.00951.

[162] R. F. Laine *et al.*, “Avoiding a replication crisis in deep-learning-based bioimage analysis,” *Nat. Methods*, vol. 18, no. 10, pp. 1136–1144, 2021, doi: 10.1038/s41592-021-01284-3.

[163] E. Yourdon *et al.*, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, 1st ed. USA: Prentice-Hall, Inc., 1979.

[164] E. Gibson *et al.*, “Multi-organ Abdominal CT Reference Standard Segmentations,” *Zenodo, 22-Feb-2018*, vol. 26, no. 6. Zenodo, pp. 1–7, 2018. doi: 10.5281/zenodo.1169361.

[165] A. Taleb *et al.*, “3D self-supervised methods for medical imaging,” *Adv. Neural Inf. Process. Syst.*, vol. 2020–Decem, no. NeurIPS, pp. 1–19, 2020.

[166] A. Lucchi *et al.*, “Learning for structured prediction using approximate subgradient descent with working sets,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 1987–1994, 2013, doi: 10.1109/CVPR.2013.259.

[167] E. Gamma *et al.*, “Design Patterns: Elements of Reusable Object-Oriented Software,” p. 416, 1994.

[168] Y. You *et al.*, “Large Batch Training of Convolutional Networks.” 2017. [Online]. Available: http://arxiv.org/abs/1708.03888

[169] A. Archit *et al.*, “Segment Anything for Microscopy,” *bioRxiv*, p. 2023.08.21.554208, Jan. 2023, doi: 10.1101/2023.08.21.554208.

[170] D. D. Nogare *et al.*, “Using AI in bioimage analysis to elevate the rate of scientific discovery as a community,” *Nat. Methods*, vol. 20, no. 7, pp. 973–975, 2023, doi: 10.1038/s41592-023-01929-5.

[171] L. Maier-Hein *et al.*, “Why rankings of biomedical image analysis

competitions should be interpreted with care," *Nat. Commun.*, vol. 9, no. 1, p. 5217, 2018, doi: 10.1038/s41467-018-07619-7.

[172] W. Su *et al.*, "Towards All-in-one Pre-training via Maximizing Multi-modal Mutual Information." arXiv, 2022. doi: 10.48550/ARXIV.2211.09807.

[173] G. E. Hinton *et al.*, "Reducing the Dimensionality of Data with Neural Networks," *Science (80-.).*, pp. 504–507, 2006.

[174] K. He *et al.*, "Masked Autoencoders Are Scalable Vision Learners," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2022–June. arXiv, pp. 15979–15988, 2022. doi: 10.1109/CVPR52688.2022.01553.

[175] R. Rombach *et al.*, "High-Resolution Image Synthesis with Latent Diffusion Models," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2022–June, pp. 10674–10685, 2022, doi: 10.1109/CVPR52688.2022.01042.

[176] T. Chen *et al.*, "Big self-supervised models are strong semi-supervised learners," *Adv. Neural Inf. Process. Syst.*, vol. 2020–Decem, 2020.

[177] J. B. Grill *et al.*, "Bootstrap your own latent a new approach to self-supervised learning," *Advances in Neural Information Processing Systems*, vol. 2020–Decem. arXiv, 2020. doi: 10.48550/ARXIV.2006.07733.

[178] Z. Zhou *et al.*, "Models Genesis," *Med. Image Anal.*, vol. 67, p. 101840, 2021, doi: 10.1016/j.media.2020.101840.

[179] Y. Tang *et al.*, "Self-Supervised Pre-Training of Swin Transformers for 3D Medical Image Analysis," pp. 20698–20708, 2022, doi: 10.1109/cvpr52688.2022.02007.

[180] F. Haghighi *et al.*, "DiRA: Discriminative, Restorative, and Adversarial Learning for Self-supervised Medical Image Analysis," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2022–June. arXiv, pp. 20792–20802, 2022. doi: 10.1109/CVPR52688.2022.02016.

[181] M. Caron *et al.*, "Unsupervised learning of visual features by contrasting cluster assignments," *Advances in Neural Information Processing Systems*, vol. 2020–Decem. 2020.

[182] F. Schroff *et al.*, "FaceNet: A unified embedding for face recognition and clustering," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07–12–June, pp. 815–823, 2015, doi: 10.1109/CVPR.2015.7298682.

[183] J. Deng *et al.*, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," *CoRR*, vol. abs/1801.0, 2018, [Online]. Available: http://arxiv.org/abs/1801.07698

[184] E. Gamma *et al.*, *Design Patterns: Elements of Reusable Software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1996.

[185] S. McConnell, *Code Complete, Second Edition*. USA: Microsoft Press, 2004.

[186] "Social App Report 2023: Revenue, User and Benchmark Data", [Online]. Available: https://www.businessofapps.com/data/social-app-

report/?utm_source=social&utm_medium=click&utm_campaign=featured-data-ad

Appendices

Availability of datasets and tools

- The benchmarking dataset is available on the public OMERO server hosted by Florida State University and created during the INDEPTH project funded by the COST Actions program. Server address: https://omero.bio.fsu.edu/webclient/?show=project-2801
- The benchmarking code is available on GitHub: https://github.com/GuillaumeMougeot/nuclei_benchmark
- Biom3d is available on GitHub: https://github.com/GuillaumeMougeot/biom3d
- Biom3d documentation is hosted by ReadTheDocs website: https://biom3d.readthedocs.io/en/latest/
- Biom3d Python package is hosted on PyPi official server: https://pypi.org/project/biom3d/
- Biom3d will also be archived on Software Heritage to guarantee its long-term accessibility: https://www.softwareheritage.org/

Authors' contributions

Chapter 2. I undertook the comprehensive review of deep learning methods and wrote the manuscript for the publication, including the supplementary table listing the available components for each method. I chose the methods to be benchmarked and both participated and led the work of Pedro Mezquita and Adama Nana regarding method installation, Docker packaging, application to nucleus segmentation, evaluation on the test set and result interpretation. Sophie Desset chose and annotated the nucleus dataset. Sophie Desset, Christophe Tatout, David E. Evans, Frederic Chausse, Emily Pery, Katja Graumann and Tristan Dubos participated in the revision of the review manuscript.

Chapter 3. I undertook the philosophy design, the coding, the development, the packaging and sharing, the debugging, maintenance, and testing, the valorisation through demos, the documentation and tutorials of Biom3d. I led the work of Sami Safarbati who participated in the development of the default graphical user interface

of Biom3d, in particular by integrating OMERO compatibilities. All my supervisors and advisors participated in giving me feedback either on the applicative side when Biom3d was applied to biological images or on the technical side. They also greatly guided me in the valorisation and publication process of Biom3d, a process still in progress.

Chapter 4. I chose to explore the self-supervised research direction and the methods that were studied. I led the course of the first experiments helped by Adama Nana and Abderrahime Tizi. I made all experiments on Taleb *et al.* work and on the DINO method. I designed, coded, and integrated in Biom3d, the Triplet and Arcface losses as well as the full U-Net pretraining workflow. I wrote the publication manuscript and Frédéric Chausse, Sophie Desset, Katja Graumann and Emilie Pery helped me in the publication process.

Publications, conferences, and workshops

Publications by chapter

Publication – Chapter 2. The review has been published in the *Journal of Cell Science* in 2022 under the title "Deep learning – promises for 3D nuclear imaging: a guide for biologists" [40].

Publication – Chapter 3. Biom3d publication is still in process. Currently two main publications are planned: one addressed to biologists (in a journal such as *Nature methods*) and one addressed to computer scientists (in a journal such as *Computer Methods and Programs In Biomedicine*).

Publication – Chapter 4. A publication encompassing the results obtained with the Arcface loss and the full U-Net pretraining was submitted to MICCAI but was rejected. A new version is currently planned to be submitted to *Transactions on Computational Biology and Bioinformatics.*

List of all publications, conferences and workshops made during this thesis

Publications

- Dubos, T., Poulet, A., Gonthier-Gueret, C., **Mougeot, G.**, Vanrobays, E., Li, Y., Tutois, S., Pery, E., Chausse, F., Probst, A. V., Tatout, C., Desset, S., 2020. Automated 3D bio-imaging analysis of nuclear organization by NucleusJ 2.0. Nucleus 11, 315–329. https://doi.org/10.1080/19491034.2020.1845012
- Tatout, C., **Mougeot, G.**, Parry, G., Baroux, C., Pradillo, M., Evans, D., 2022. The INDEPTH (Impact of Nuclear Domains on Gene Expression and Plant Traits) Academy: a community resource for plant science. J. Exp. Bot. 73, 1926–1933. https://doi.org/10.1093/jxb/erac005
- **Mougeot, G.**, Dubos, T., Chausse, F., Pery, E., Graumann, K., Tatout, C., Evans, D.E., Desset, S., 2022. Deep learning - promises for 3D nuclear imaging: a guide for biologists. J. Cell Sci. 135, jcs258986. https://doi.org/10.1242/jcs.258986

Conferences

- **Mougeot, G.**, Chausse, F., Graumann, K., Desset, S., 2021. Deep Learning for Nuclear Image Analysis and Applications to 3D Plant Nuclei. SEB Annual Meeting 2021. Poster.
- **Mougeot, G.**, Chausse, F., Desset, S., Graumann, K., 2021. Deep learning for nuclear image analysis and application to 3D plant nuclei. Conference article and Poster.
- **Mougeot, G.**, Chausse, F., Graumann, K., Desset, S., 2022. Segmentation of 3D Plant Nuclei using Deep Learning - A (Small) Guide for Biologist. SEB Annual Meeting 2022. Oral presentation.
- **Mougeot, G.**, Desset, S., Chausse, F., Graumann, K., 2023. Making deep learning easy for 3D bio-image segmentation. Oxford Brookes Symposium. Department of Health and Life Science. Oral presentation.
- **Mougeot, G.**, Chausse, F., Graumann, K., Tatout, C., Desset, S., 2023. A modular deep learning framework for 3D bioimage segmentation. IABM 2023. Poster.
- **Mougeot, G.**, Biom3d, a modular deep learning framework for 3D bio-image segmentation. RTMFM 2023. https://www.youtube.com/watch?v=fJopxW5vOhc Oral presentation.

- **Mougeot, G.**, Biom3d: tools for 3D segmentation. GT-MAIIA 2023. Oral presentation.
- **Mougeot, G.**, Chausse, F., Graumann, K., Desset, S., Biom3d, a modular deep learning framework for 3D bio-image segmentation. SEB Annual Meeting 2023. Oral presentation.

Workshops

- **Mougeot, G.,** 2021. Deep learning made easy for microscopy: an introduction to ZeroCostDL4Mic and DeepImageJ. MIFOBIO Workshop 2021.
- **Part of a large group of trainers.** *DeepScopie*: Application of deep learning tools for microscopy image analysis. ANF-DeepScopie Angers 2022.
- (Upcoming) **Mougeot, G.,** Biom3d, an easy-to-use and modular deep learning framework for 3D semantic segmentation. I2K Workshop Virtual Event 2023.
- (Upcoming) **Mougeot, G.,** Desset, S., An easy-to-use deep learning method for 3D semantic segmentation. MIFOBIO 2023.

Copyrights

- Figures 1-1 to Figure 1-4 are homemade.
- Figure 1-5 respects the copyright terms of Cold Spring Harbor Perspectives in Biology: “Authorized users of Cold Spring Harbor Perspectives in Biology may view, reproduce, or store copies of articles for the purposes of scholarly, research, educational, and individual use only.” (source: https://cshperspectives.cshlp.org/site/misc/terms.xhtml)
- Figure 1-6 belongs to public domain.
- Figure 2-1 and Figure 2-2 are homemade.
- Figure 2-3 belongs to public domain.
- Figure 2-4 is homemade.
- Figure 2-5 originates from ArXiv (Creative Common License): https://arxiv.org/abs/1311.2901
- Figure 2-6 originates from ArXiv (Creative Common License): (A) https://arxiv.org/abs/1706.03762, (B) https://arxiv.org/abs/2010.11929, (C) https://arxiv.org/abs/2203.01536

- Figure 2-7 to Figure 2-11 are homemade.
- Figure 2-12 originates from ArXiv (Creative Common License): (A) https://arxiv.org/abs/1505.04597, (B) https://arxiv.org/abs/1807.10165, (C) https://arxiv.org/abs/1908.07919, (D) https://arxiv.org/abs/2201.01266
- Figure 2-13-A is homemade and Figure 2-13-B is under GNU Free Documentation License.
- Figure 2-14 and Figure 2-15 are homemade.
- Figure 2-16 originates from ArXiv (Creative Common License): https://arxiv.org/abs/1904.08128
- Figure 3-1 and Figure 3-2 are homemade.
- Figure 3-3 belongs to public domain.
- Figure 3-4 to Figure 3-16 are homemade. Computer icons and other icons in Figure 3-10 to Figure 3-12 were made by JGraph and are under CC BY 4.0 License: https://github.com/jgraph/drawio)
- Figure 4-1 is adapted from two paper from ArXiv (Creative Common License): https://arxiv.org/abs/2006.03829 and https://arxiv.org/abs/1505.04597
- Figure 4-2 to Figure 4-5 are homemade.
- Figure 4-6 is adapted from ArXiv (Creative Common License): https://arxiv.org/abs/1503.03832
- Figure 4-7 and Figure 4-8 are homemade.
- Figure 5-1 is homemade.
- All Tables are homemade.