Master's Thesis : Cell segmentation in whole-slide cytological images

Auteur : Testouri, Mehdi
Promoteur(s) : Maree, Raphael
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2019-2020
URI/URL : http://hdl.handle.net/2268.2/8979

University of Liège

Faculty of Applied Sciences

Cell segmentation in whole-slide cytological images

A thesis submitted as part of the fulfillment of a Master's degree in Computer Sciences and engineering

Author:
Mehdi Testouri

Supervisor:
Dr. Raphaël Marée

Academic year 2019 - 2020

Abstract

University of Liège - Faculty of Applied Sciences

Cell segmentation in whole-slide cytological images

Master's degree in Computer Sciences and engineering

Author : Mehdi Testouri Supervisor : Dr. Raphaël Marée

The digital pathology is a field of medicine that leverage computer aided techniques to view, manage and analyse microscope acquired images, commonly called whole-slide images. The field has been rapidly evolving in the recent years thanks to novel research around AI based techniques for automated diagnosis. A popular sub-field of the automated diagnosis is the segmentation (or detection) of relevant regions in a whole-slide image.

This thesis deals with the problem of cell detection using segmentation in whole-slide cytological images as part of an automated diagnosis system for the thyroid cancer. The work is conducted within Cytomine R&D project team from ULiège.
Among the challenges is the implementation of a segmentation algorithm using novel deep learning methods while dealing with incomplete training data.

The proposed solution comprises of a U-Net network for the segmentation along with an iterative data improvement method for incomplete data completion. The implementation also achieves the required level of modularity and scalability for the subsequent integration in the ULiège Cytomine instance which was almost complete.

Promising results were obtained thus demonstrating the abilities of the U-Net and the data improvement method. However, inaccuracies remained mainly due to false positives and all available data weren't used because their incompleteness couldn't be fully addressed. Further work should enhance the training set building, for instance using active learning, and especially pay attention to the diversity and completeness of the data used.

Acknowledgement

I would like to thank my supervisor Raphaël Marée for his continuous following and helpful guidance as well as his introduction to me to the interesting field of digital pathology.

I also want to thanks Romain Mormont for his precious help while tackling the challenges of this work and his advises for the writing of this thesis.

I finally want to thanks my family and close friends for their support for this work and throughout the years.

Contents

Chapter 1

Introduction

In today's medical world, the analysis of biomedical images is one of the most important method of assessing a patient health issues and ultimately produce a diagnosis. Thanks to major advances in imagery technologies such IRMs, microscopes, scanners, X-rays, etc. There is now the ability to produce high resolution medical images for various purposes and medical practitioners. However, while the process of producing those images is highly automated, the analysis of them still mainly rely on the assessment of a human expert and this brings problems associated with the human factors. Human can be tired, make mistakes or have biases and while there is no intention to question the medical practitioners who remain highly qualified experts, this is obvious that there is room for improvements.
One area well concerned by this is the cancer diagnosis. It goes without saying that cancer is one of the most important health issue affecting humanity and it is thus a matter of importance to improve diagnosis techniques. Nowadays, thanks to the emergence of new capabilities of AI there is now the opportunity to address this issue by using AI to provide the practitioner with a help for his/her diagnosis to further improve it. A recent study published in *Nature* [12] even showed that the AI can surpass the expert for the diagnosis of breast cancer. Such AI methods should thus be investigated for the diagnosis of other cancers and, in the context of this thesis, the concern is the diagnosis of the thyroid cancer by detection of relevant cells in whole-slide cytological images. This work thus falls into the field of digital pathology whose practice is to manage, view and analyse whole-slide images (WSI) which are microscope acquired images that consequentially have the characteristic of being very huge.

The main contributions of this work are the implementation and assessment of an U-Net based segmenter, the implementation of an iterative data improvement method and finally the integration of the segmenter into a web application for results visualization.

The following paper is structured as follow :

- Chapter 2 will introduce the problem informally and formally and provide an overview of the state of the art methods.
- Chapter 3 will describe the methodology and implementation of the solution.
- Chapter 4 will assess the solution and present the results before the final conclusion and improvement suggestions.

Chapter 2

Machine learning for biomedical image analysis

The problem addressed by this thesis is to develop an efficient method able to segment the cells of interest in whole-slide cytological images of the thyroid. The segmentation algorithm is implemented using deep learning and computer vision techniques.
This problem thus falls into the broader category of machine learning applications where the goal is to learn an algorithm from a training dataset, so that this algorithm can make accurate prediction on unseen data.

2.1 Cell detection in thyroid cytology

2.1.1 Context

The process of automated diagnosis based on medical images is typically based on the detection of relevant tissues or cells, their presence may be indicating underlying problems. For instance, some types of tissues or cells present in a scanner can be a sign of an emerging cancer.

This thesis is conducted within the Cytomine R&D project at ULiège. Cytomine [1] is an open source internet application aimed for collaborative analysis of multi-gigapixel images and the ULiège hosts its own instance of Cytomine.
The ULiège Cytomine team in collaboration with the ULB Erasme Hospi-

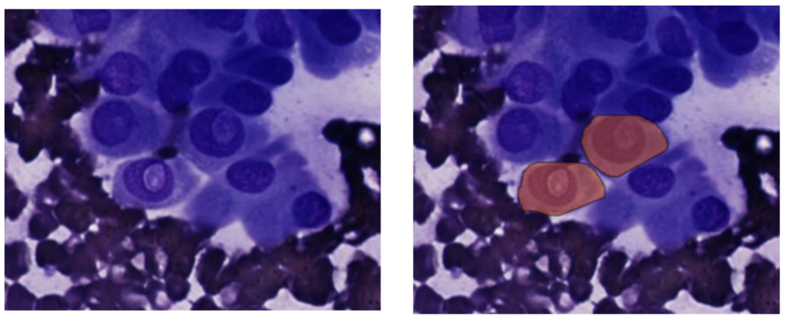

Figure 2.1: Example of papillary cells with inclusion highlighted in orange.

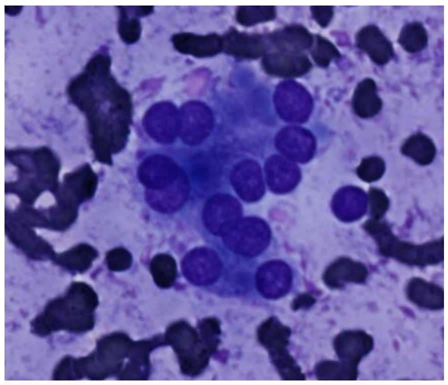

Figure 2.2: Example of a proliferative follicular architectural pattern.

tal (Dr. Isabelle Salmon) is developing an automated diagnosis application working on whole-slide cytological images of the thyroid, obtained by fine needled aspiration biopsy. The **ultimate goal** of the project is to be able to detect 2 different elements in the whole-slide images as they may highlight the presence of an emerging cancer :

- The papillary cells with inclusion (Figure 2.1).
- The proliferative follicular architectural patterns (Figure 2.2).

The detection of these elements is performed in two main steps : the segmentation and classification. The segmentation is used to extract useful areas in the whole-slide image (*e.g* areas with cells) while the classification determines the type of the extracted areas (*e.g* type of cells within the area). The goal of this Master thesis is to develop the segmentation algorithm but

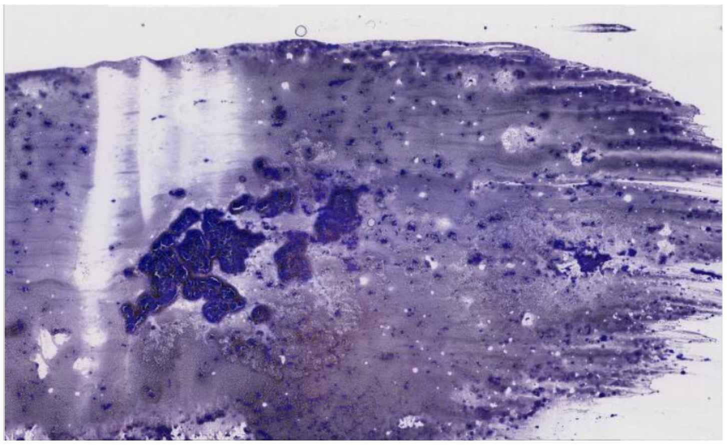

Figure 2.3: Example of a whole-slide image (104704 × 172032 pixels).

also provide an integration of it in the ULiège Cytomine instance (by implementing a Cytomine software) so that it is possible to run the segmentation on a hosted whole-slide image and visualize the results in the Cytomine web interface.

Whole-slide images are very big with hundreds of thousands of pixels in height and width (see Figure 2.3 and 2.4) so that it is inconceivable to perform the segmentation directly on them. Instead, the whole-slide image is cut into tiles (or crops) of reasonable size and the segmentation algorithm is trained and run on them.

The reasons for the approach of first separating the cells from the background and then classifying them are the following :

- While the segmentation works at a *structure* level, the classification algorithm developed by the Cytomine R&D team works at a *cell* level for the detection of *papillary cells with inclusion*. The segmentation can help to remove useless areas in the WSI and thus reduce the area on which the classification is applied. This is analogous to how a practitioner proceeds, he/she first looks for areas with cells and then looks for problematic cells in those areas. If the classification was directly applied on the WSI, it would drastically increase the computation time because of the reduced size of the crops used and the need of processing all areas of the WSI.

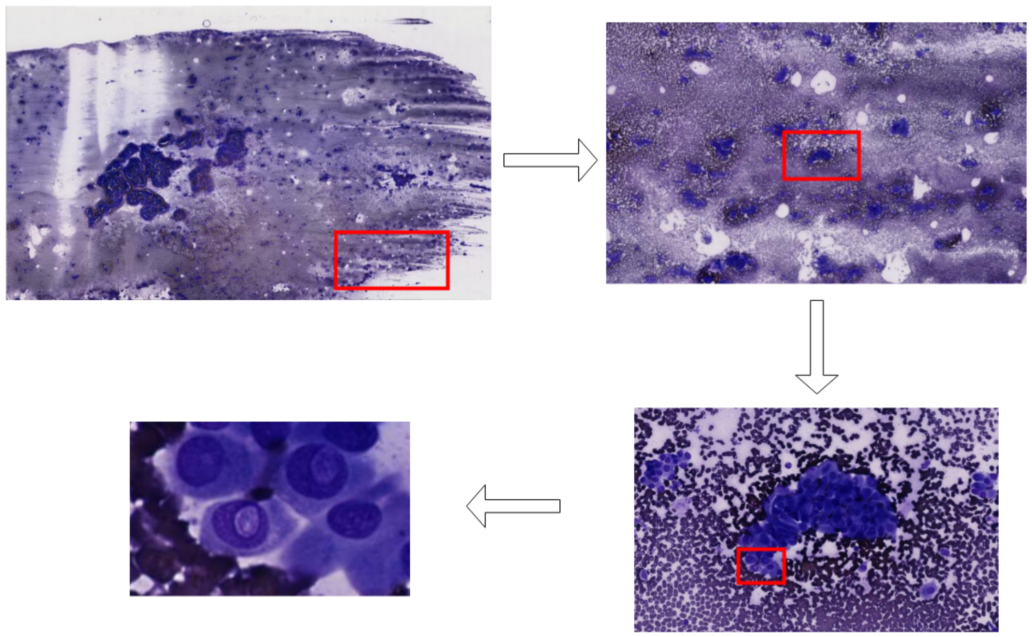

Figure 2.4: From whole-slide perspective to cell perspective.

- Even if the classification isn't very reliable, the segmentation can help the practitioner with the diagnosis by presenting him/her among all cells detected in the images (and thus possibly missing cells in the practitioner diagnosis) on which he/she can decide those with problems. This can help speeding up the diagnosis and provide redundancy.

The different types of cells and patterns that are the target of the segmentation are illustrated in Figures 2.5 and 2.6.

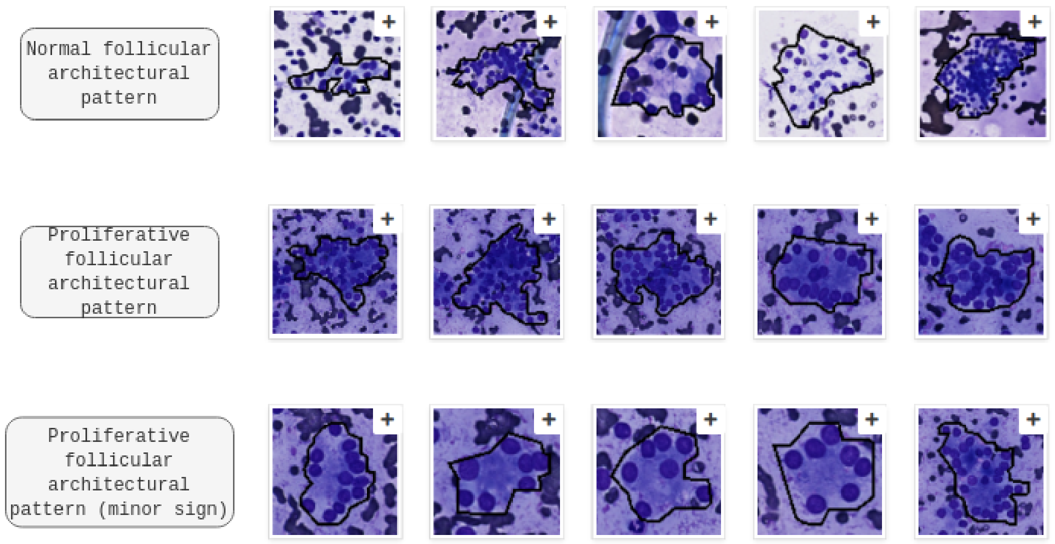
Normal follicular architectural pattern
Proliferative follicular architectural pattern
Proliferative follicular architectural pattern (minor sign)

Figure 2.5: Illustration of the different patterns.

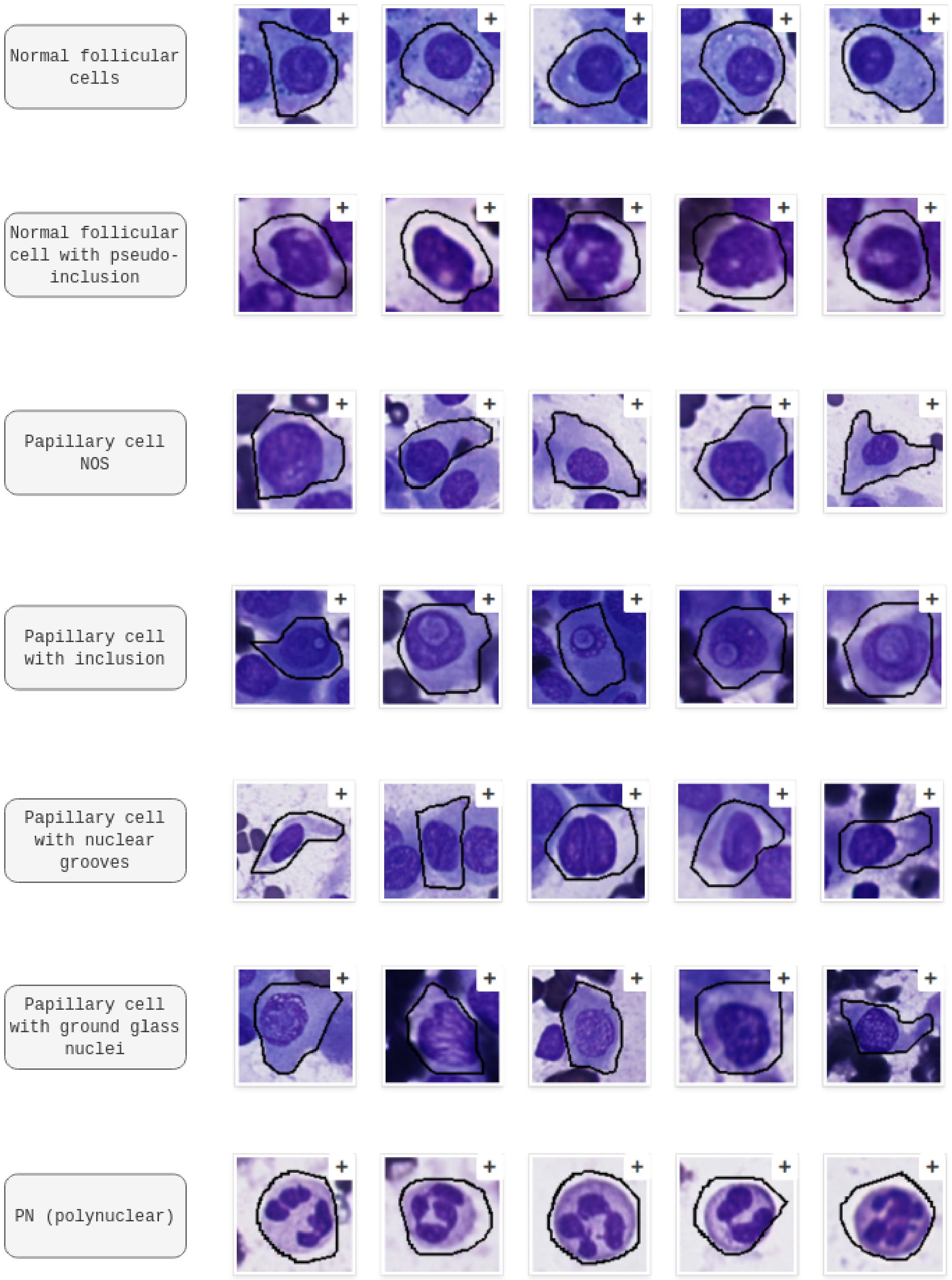
Normal follicular cells
Normal follicular cell with pseudo-inclusion
Papillary cell NOS
Papillary cell with inclusion
Papillary cell with nuclear grooves
Papillary cell with ground glass nuclei
PN (polynuclear)

Figure 2.6: Illustration of the different cells.

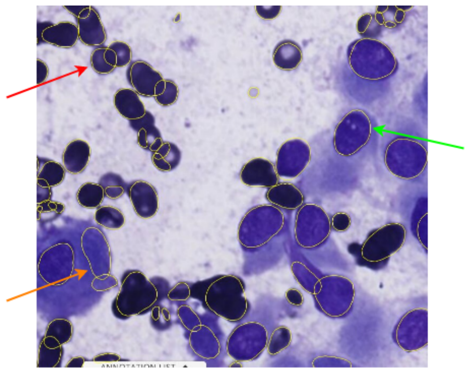

Figure 2.7: Stardist cell detection algorithm applied to a tile. Green arrow : correctly segmented nuclei; Orange arrow : false positive on foreground; Red arrow : false positive on background.

2.1.2 Complete pipeline

In order to delineate individual cells, the Cytomine team has put in place the *StarDist* model [18] [20] which is used to segment cell nucleus. However while this algorithm works well on pre-selected foreground, it generates too much false positives on the background, as seen on Figure 2.7, and thus the foreground selection is required for it to work properly.

The complete pipeline for the diagnosis envisioned by the Cytomine R&D team works as follow (see Figure 2.8):

1. Per-tile foreground segmentation followed by tiles merging in the whole-slide image.
2. Nucleus segmentation using *StarDist* in the foreground areas.
3. Classification of the nucleus and patterns in the foreground using deep learning classification models [14][13].

The complete pipeline is illustrated on Figure 2.8. The goal is to be able to run this pipeline directly from the Cytomine web interface so that the results can be interactively visualized. The implementation of the Cytomine cell segmentation software is the concern of this work.

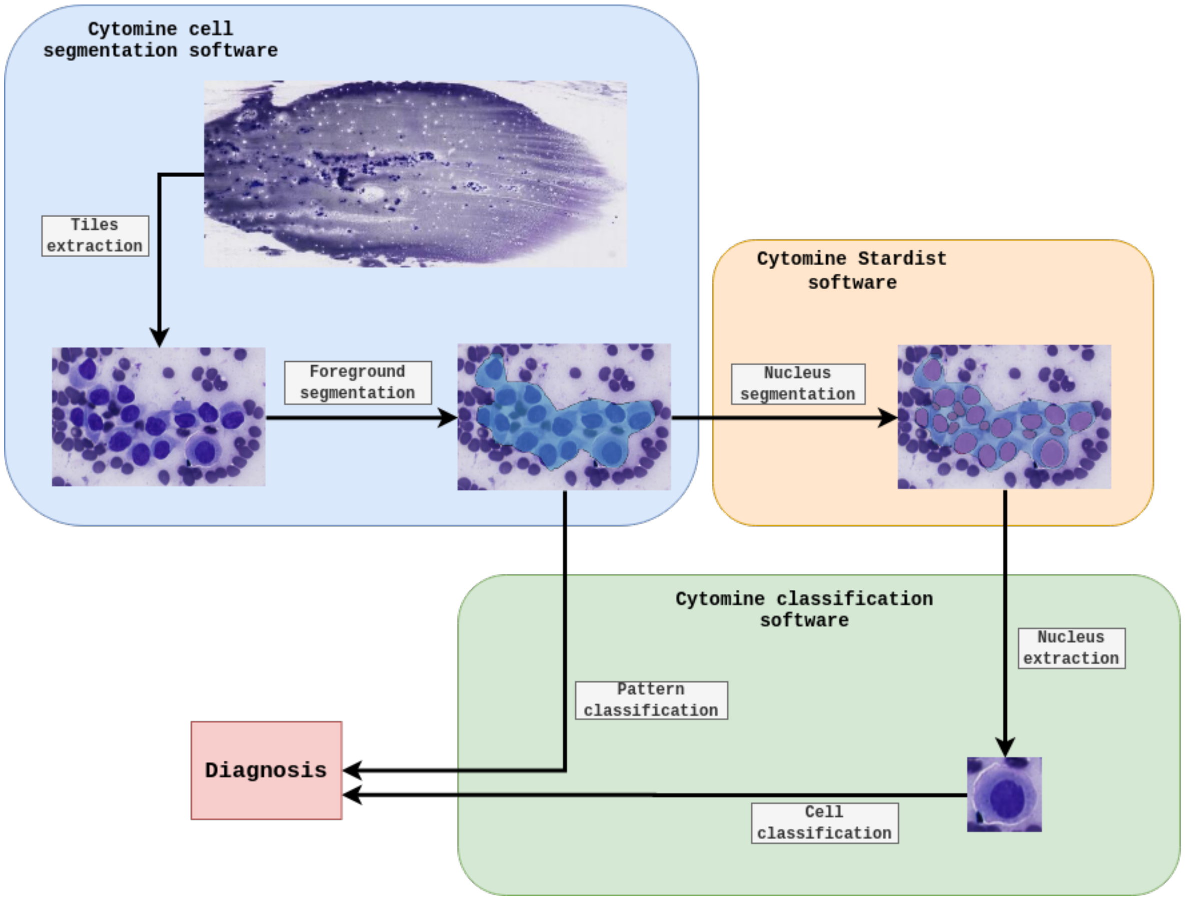

Figure 2.8: Complete diagnosis pipeline.

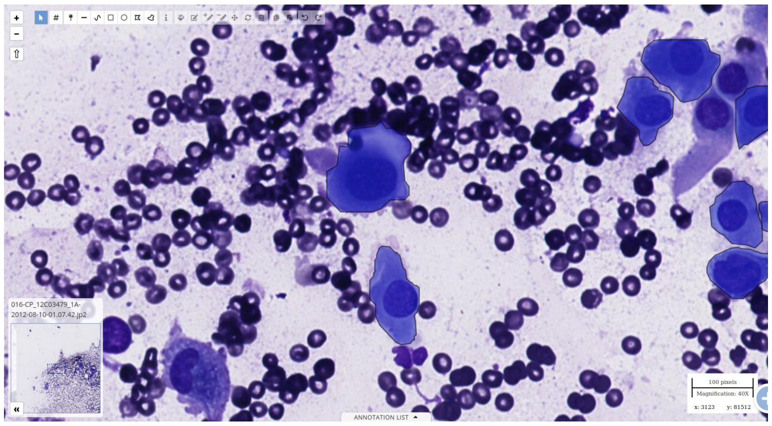

Figure 2.9: Example of expert made annotations (blue) viewed from the Cytomine web interface. It can be seen that the annotation don't highlight all cells and are thus incomplete.

2.1.3 Detection using segmentation

The segmentation consists in producing a segmentation mask (which is another image) from an original image. This mask binds each pixel of the input image to a specific class and as such, the segmentation is also being referred as a "pixel-wise classification" problem. The image crops and associated masks used to train the algorithm are derived from man-made (by expert) annotations on the whole-slide images. Such annotations are illustrated on Figure 2.9.

The mask can be binary by assigning the class 0 to a non-interest pixel (also referred as background) and 1 to a pixel of interest (also referred as foreground) and the segmentation is thus said to be binary. The 0 class is commonly represented by a black color while the class of interest is represented with a white color. A binary segmentation is illustrated in Figure 2.10. The mask can also contain more than 2 classes, in such cases, the segmentation is said to be multi-class. In this work, the addressed problem is a binary segmentation.

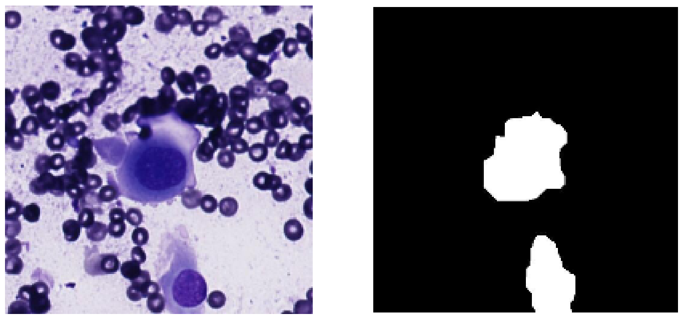
Figure 2.10: Binary segmentation.

2.2 Problem definition

2.2.1 Supervised learning

Definition

The problem of training a segmentation algorithm falls into the realm of supervised learning which is itself a sub-domain of machine learning. Rudiment of supervised learning are introduced in this section.

Let $\mathcal{X}$ and $\mathcal{Y}$ be 2 random variables and let us consider an unknown distribution $P(X, Y)$ and training data (x_i, y_i) drawn from the distribution where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, $i = 1, ..., N$, $\forall N \in \mathbb{N}$. The x_i is the input or simply the data and y_i is the associated output or the *label* hence the name *supervised* learning.
In the scope of this work, x_i represents one crop and y_i the associated mask.

The supervised learning is applied to solve inference problems where the goal is to estimate a function f:

$$f : \mathcal{X} \longrightarrow \mathcal{Y} \tag{2.1}$$

The inference problem can either be a classification or a regression.

In **classification**, the y_i represents classes or discrete categorical values and thus, given the training set, (2.1) estimates for any new x:

$$\arg\max_y P(Y = y | X = x)$$

In **regression**, the y_i represents continuous numerical values and thus, given the training set, (2.1) estimates for any new x:

$$\mathbb{E}[Y|X = x]$$

This project addresses is a classification inference problem as the labels y_i represent a segmentation map containing pixel-wise association with a class.

Empirical risk minimization

In order to evaluate (2.1), the concept of **loss function $\mathcal{L}$** is introduced :

$$\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$$

where $\mathcal{L}(f(x), y) \geq 0$ measures how $f(x)$ is close to y, the lesser $\mathcal{L}$, the closer $f(x)$ is.

Let us assumes $\mathcal{F}$ is the set of function f that can be produced by the learning algorithm, the goal is to find f_* that minimizes the **expected risk** $R(f)$:

$$f_* = \arg\min_{f \in \mathcal{F}} R(f) \tag{2.2}$$

where,

$$R(f) = \mathbb{E}_{(x,y) \sim P(X,Y)}[\mathcal{L}(f(x), y))] \tag{2.3}$$

However, since $P(X, Y)$ is unknown, it is not possible to evaluate (2.3) and thus an approximation has to be made.

Let us assume a training set $d = \{(x_i, y_i)|i = 1, ..., N\}$, then the **empirical risk** $\hat{R}(f, d)$ can be computed as follow:

$$\hat{R}(f, d) = \frac{1}{N} \sum_{(x_i, y_i) \in d} \mathcal{L}(f(x_i), y_i)$$

If d is independent and identically distributed (i.i.d), then $\hat{R}(f, d)$ is **unbiased** and can be used to find an approximation of f_*. This procedure is called **empirical risk minimization** and it is implemented in most machine learning applications to train a model.

Under-fitting and over-fitting

Let $\mathcal{Y}^{\mathcal{X}}$ be the set of **all possible** functions $f : \mathcal{X} \longrightarrow \mathcal{Y}$. The **Bayes model** $f_B \in \mathcal{Y}^{\mathcal{X}}$ is the model that achieves the minimal expected risk R_B:

$$R_B = \min_{f \in \mathcal{Y}^{\mathcal{X}}} R(f)$$

f_B is a (theoretical) **optimal** model for the distribution $P(X, Y)$.

Let f_*^d be a model obtained using a training algorithm on a dataset d and let $R(f_*^d)$ be the expected risk.
If the learning algorithm allows for the production of high complexity models, the latter could be trained such that the empirical risk is arbitrarily small :

$$R(f_*^d) \geq R_B \geq \hat{R}(f_*^d, d) \geq 0 \tag{2.4}$$

In such a case, the model is said to **overfit** since it is too specialised on the training data. The empirical risk $\hat{R}(f_*^d, d)$ is low on d but the expected risk (or generalization error) is expected to be higher.
By contrast, a model is said to **underfit** if its complexity is too low and the generalization error can still be decreased.

The goal of training is thus to find the optimal balance between overfitting and underfitting as illustrated on Figure 2.11.
In order to achieve that balance, the minimization of $\hat{R}(f_*^d, d)$ does not seem suited to minimize the expected risk as illustrated by (2.4).

An unbiased solution to approximate the expected risk is to evaluate the model on a **test** set d_{test} **independent** from the training set d_{train}.
However, this test set should be only used as a final step for the model assessment and the hyper-parameters must not be tuned using it. Indeed, model tuning using the test set will likely lead to overfitting of the model on that set, thus compromising the *neutral* assessment of the generalization error.
Instead, another set called **validation** set d_{valid} is introduced for model tuning, it also has to be independent from the training set.
An overview of the evaluation process is shown on Figure 2.12.

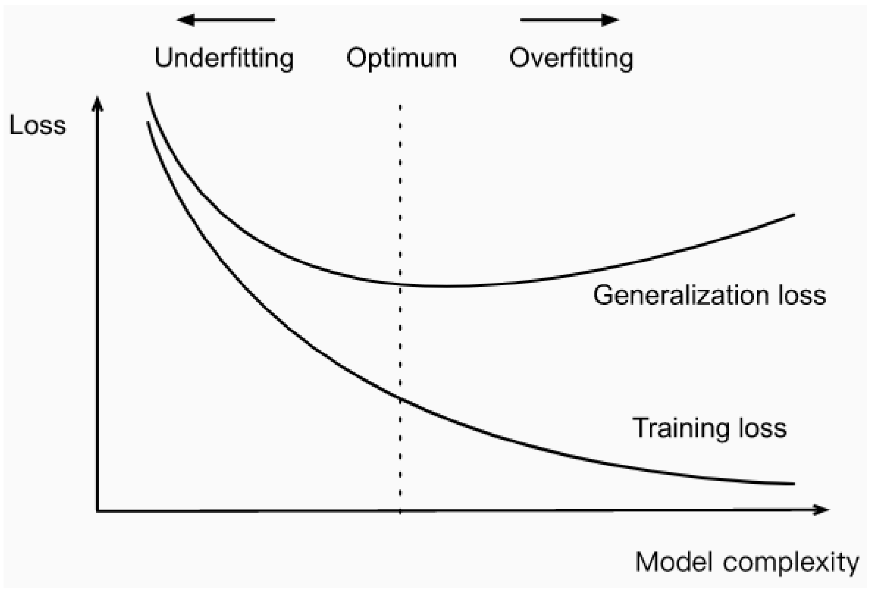

Figure 2.11: Overfitting and underfitting [2].

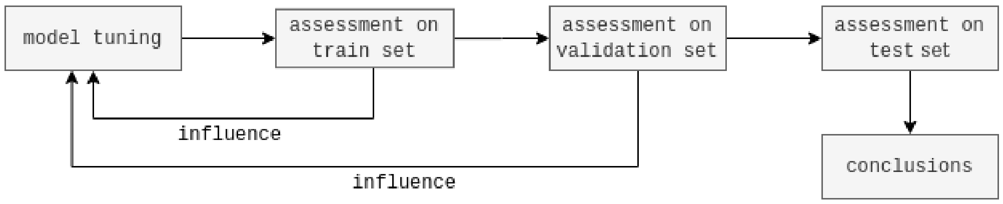

Figure 2.12: Evaluation process.

2.2.2 Segmentation

The segmentation can be expressed as a pixel-wise classification problem :

Let $d = \{(X_k, Y_k)|k = 1, ..., N\}$ be a dataset of N samples.
Each X_k is an RGB crop in the form of a tensor of size $H \times W \times C_x$ where :

$$\begin{gathered} H \text{ is the crop height.} \\ W \text{ is the crop width.} \\ C_x = 3 \text{ is the number of channels.} \end{gathered}$$

Let $p_{k,i,j} \in X_k$ with $i = 1, ..., H$ and $j = 1, ..., W$ be a pixel of X_k.
Each $p_{k,i,j}$ is thus a vector $(r_{k,i,j}, g_{k,i,j}, b_{k,i,j})$ where :

$$(r_{k,i,j}, g_{k,i,j}, b_{k,i,j}) \in ([0, 255], [0, 255], [0, 255]) \quad \forall k, i, j$$

The Y_k is the segmentation mask corresponding to X_k. It is a tensor of size $H \times W \times C_y$ where $C_y = 2$ because of the 2 type of classes for each pixel in X_k. The first class is the **background** and the second class is the **foreground** or cell area.

Each *pixel-wise* element $e_{k,i,j} \in Y_k$ with $i = 1, ..., H$ and $j = 1, ..., W$ is a vector $(b_{k,i,j}, f_{k,i,j})$ such that $\forall k, i, j$:

$$\begin{cases} b_{k,i,j} = 0 \quad \text{or} \quad 1 \\ f_{k,i,j} = 0 \quad \text{or} \quad 1 \\ b_{k,i,j} + f_{k,i,j} = 1 \end{cases} \tag{2.5}$$

and :

$$\begin{gathered} b_{k,i,j} = 1 \longrightarrow p_{k,i,j} \text{ is a background pixel} \\ f_{k,i,j} = 1 \longrightarrow p_{k,i,j} \text{ is a foreground pixel} \end{gathered}$$

We thus have X_k as data and Y_k as the corresponding label. The aim is to learn a function $f : \mathcal{X} \longrightarrow \mathcal{Y}$ using the dataset d such that for any given crop X, $f(X)$ returns a segmentation mask of the cells present in X.

In practice, in order to upload new annotations to the Cytomine server, the predicted masks need to be converted to polygons objects. This operation is done using the `rasterio` framework [3].

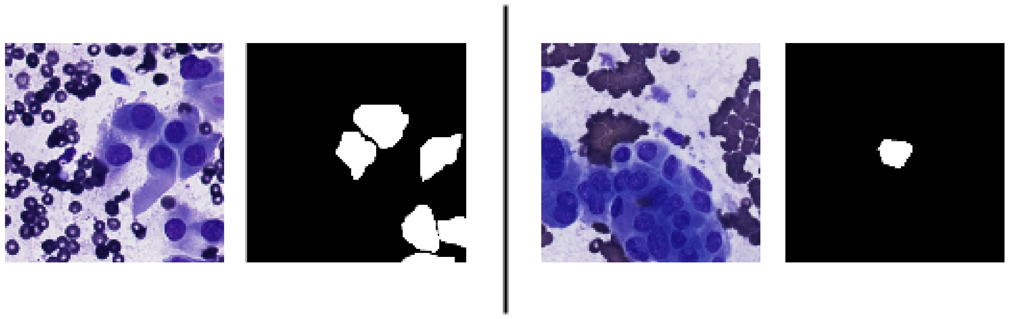

Figure 2.13: 2 pairs of crop and annotation mask. For each pair, the left is the area to annotate and the right the annotated area converted to a mask. The left pair represents an almost complete annotation while the right one represents a sparse annotation.

2.2.3 Main challenges

Weak annotations One of the main issue with the data provided by the ULiège Cytomine instance is that the aforementioned annotations (and thus their derived masks) are not always complete, i.e the segmentation mask does not contain all cells of interest, and thus some pixels belonging to cells are labelled as "background". The annotation can be almost complete or sparse as shown on Figures 2.13 and 2.14.

Given the size of whole-slide images and the fact that the process of manual annotation is time-consuming, experts make the understandable choice of focusing on the most interesting cells. It is thus a common issue in the biological imagery that the annotations are incomplete and those kind of problems are commonly referred as **weakly supervised** learning since supervised learning assumes correctly (completely) labelled data.
This issue affects mainly the annotations of individual cells in a cell group rather than annotations of the whole cell groups, in the latter case, the available data provides more complete annotations. The Figure 2.15 provides examples of *pattern* (or group) annotations.

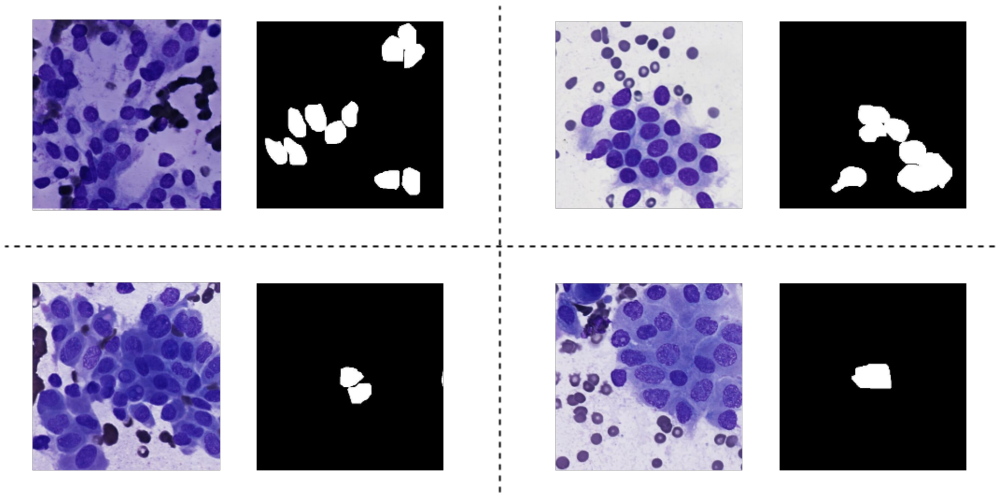

Figure 2.14: More examples of incomplete (top) and sparse (bottom) annotations.

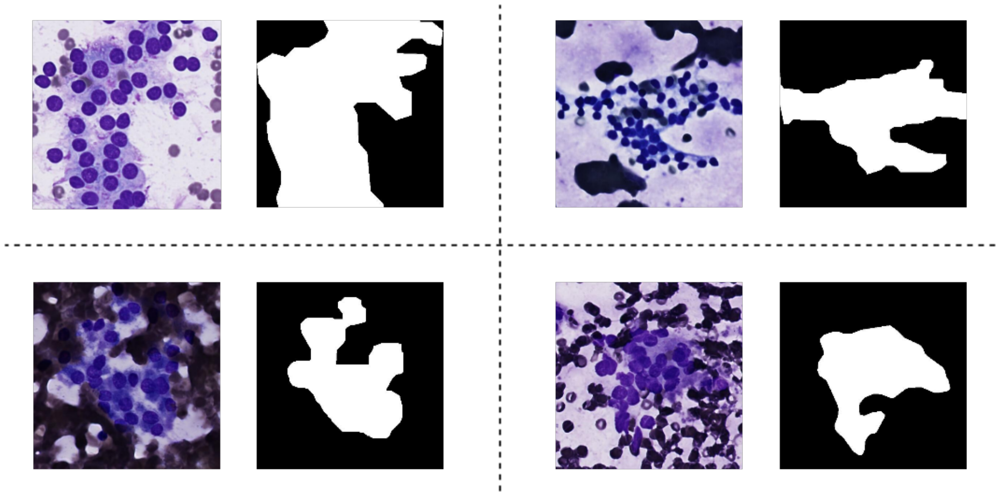

Figure 2.15: Examples of *pattern* (group) annotations.

Whole-slide images variety Another challenge concerns the ability of making a model that generalizes well on new whole-slide images. Indeed, as it can be seen on Figure 2.16, there is a high variety between different whole-slide images at a macroscopic level, often resulting in a different type of background at a microscopic level.

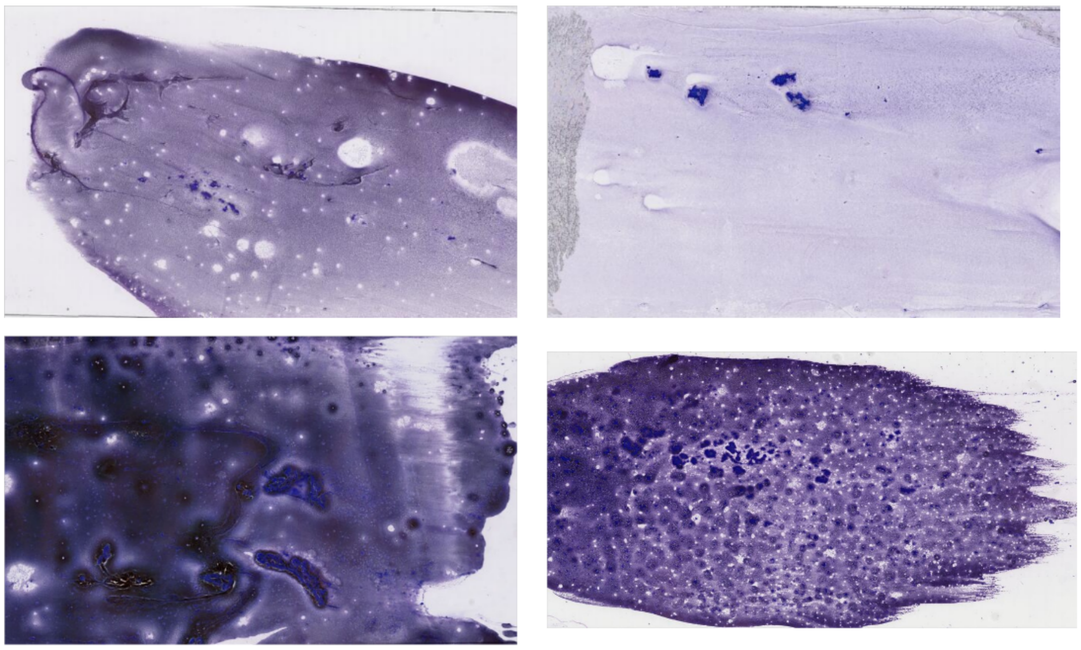

Figure 2.16: Illustration of the variety between whole-slide images.

This variety can be explained by the following causes:

- The WSIs don't necessarily come from the same lab with the same **hardware**. Moreover, different practitioners with different **procedures** (cutting, staining, scanning, ...) to proceed the aspiration biopsy and scanning will impact the production of WSIs.
- Each **patient** will obviously have some specificity associated with his WSIs such as cells types and density, etc.
- The **time** factor on its own can embed the above causes, the closer the time, the greater the likelihood of the WSIs to share common characteristics.

In practice, this variety will make the training of the model harder as unseen whole-slide images may have important variations, compared to the ones used in training and validation sets.

2.3 Related works and state of the art

In this section, several papers with related works are reviewed. This can provide an overview of some of the state of the art methods applied to solve problems that resemble this work

2.3.1 Whole-slide cytological images analysis

In [6], the authors describe a method of automated diagnosis of the thyroid cancer in which the first part is similar to this work, that is to determine areas of interest (relevant cells) in whole-slide cytopathology images. A second algorithm (second part of their paper) is then used to determine the malignancy of the cells.
Their approach differs though as they do not perform cell segmentation. Instead they try to classify portions of the image (sub-images) that are relevant (Figure 2.17) from those who are not, by using a CNN which is trained on labelled sub-images. The relevancy of a sub-image is 1 if it contains follicular groups of cells and 0 otherwise (it is labelled as background) and the goal is thus to separate the parts of the original image that contains follicular groups from the background which is assumed to constitute the majority of the image. Their dataset comprises of 908 WSIs of size $\approx 40000 \times 25000$ pixels. However, their work is not applicable in this project because the segmentation is required as part of the detection of papillary cell with inclusion (which does not constitute cells of interest in the paper) and because of the other advantages it brings as explained in section 2.1.1.

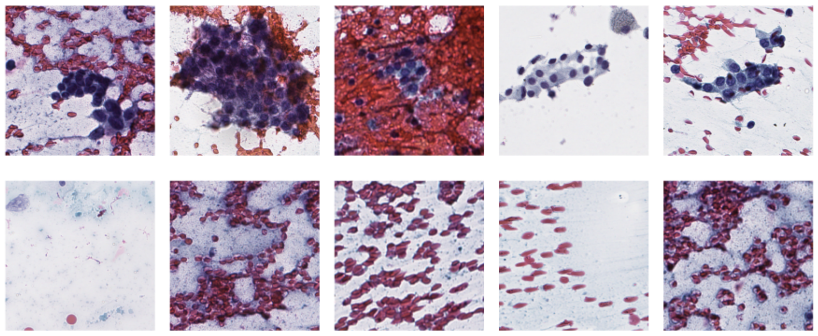

Figure 2.17: Figure from [6]. On the top, instances containing groups of cells of interest and, on the bottom, instances containing the background.

2.3.2 Cell segmentation

The following section presents a series of papers that compare different approaches for the cell segmentation.

Caicedo et al. [5] gives a comparison of the different state of the art methods used for biological segmentation, namely *DeepCell* and *U-Net* and also more conventional machine learning methods with no use of deep learning such as the random forest for nucleus segmentation in fluorescence images (examples on Figure 2.18) . Even though the present work is about cells segmentation, it seems very likely that the efficient methods for nucleus segmentation will also be good candidates as the problem is similar. *DeepCell* is a CNN with the configuration detailed in [19] whereas the *U-Net* follows an architecture similar to an auto-encoder with convolutional layers.
The segmentation problem is formulated as a *boundary detection* problem which consists in identifying three types of pixels: background, interior of nuclei and boundaries of nuclei. This is thus a three-class pixel-wise classification problem and it necessary to have the boundary classification to handle overlapping nuclei.
The test set used to assess the models contains 5,720 nuclei which accounts roughly for 57 images. The results clearly illustrate that the deep learning methods are better in terms of their defined accuracy metrics as they provide more accurate boundaries but are also better at dealing with overlapping nuclei. The two neural networks have different types of errors. *U-Net* is able to detect nuclei of various size better (less false negative errors) (Figure 2.19)

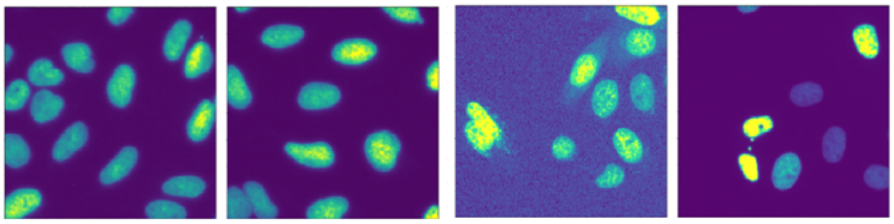
Figure 2.18: Figure from [5]. Example of fluorescence images.

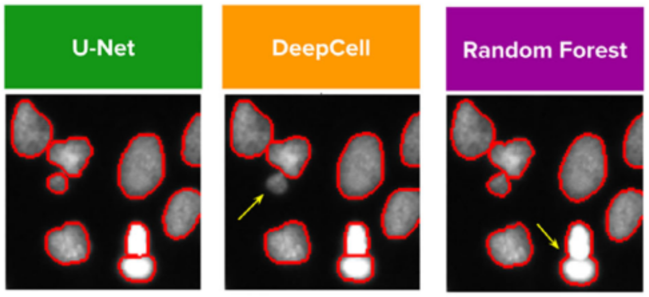

Figure 2.19: Figure from [5]. On the center, a false negative error made by *DeepCell* and, on the right, a split error made by the random forest.

than *DeepCell*, however, it also detects imaginary cells more often (more false positive errors). *U-Net* is also better at detecting various cells edges (reduced merge error) but also see ones where it shouldn't. One should recall that the goal of this work is to produce segmentation of cells of interest so that they can be processed by a second algorithm. In this context, false negative errors are crucial. Indeed, if it is assumed that the second algorithm can deal with irrelevant cells or background, skipping relevant cells might compromise its results. The *U-Net* seems thus more suitable.
Finally, the paper illustrates the benefit of the data augmentation which helps to increase the accuracy even though *DeepCell* and *U-Net* can perform well on small datasets. Basic augmentation techniques are image rotations, symmetries, change of contrast and illumination.

A comparison between the performance of the *U-Net* and the *SegNet* is performed in [9] in the context of cells segmentation of the endothelial layer of the cornea. Again, the problem is similar though the tissue of interest is not from the thyroid. It can thus be expected that the results from this

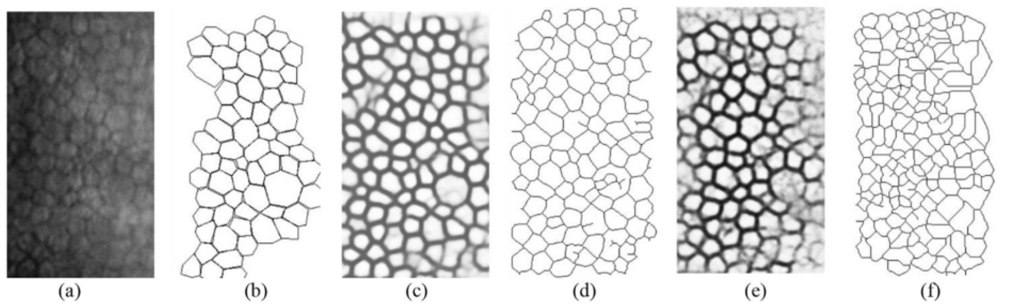

Figure 2.20: Figure from [9]. (a) is an example image from the test set; (b) represents the manual ground truth annotations; (c) & (d) are the probability output and binary segmentation result from *U-Net*; (e) & (f) are the probability output and binary segmentation result from *SegNet*.

study may apply to the present work.
The implemented *U-Net* uses 16 convolutional layers whereas the *SegNet* accounts for 26 and has about 4 times more parameters. The evaluation metrics are the Dice and Jaccard coefficients. The dataset contains 130 images which is split in a training and test set of 100 and 30 images respectively. Despite its inferior complexity, the *U-Net* has been decribed by the authors to be quote : *"far superior to SegNet for segmenting ECs in specular microscopic images"*. The difference in accuracy in illustrated in Figure 2.20 and it can be seen that *U-Net* is clearly superior. The authors think that it could be attributed to the skip connections in the *U-Net* between the encoding and decoding layers.

Another comparison of segmentation methods is performed in [4]. The problem is the segmentation of cells in microscopy image of glioblastoma tissue which is not an easy tissue for this task as it is claimed in the paper (see Figure 2.21).
The paper assesses the performance of supervised and unsupervised method as well as non machine learning methods. Among the supervised learning methods, there are two deep learning ones : a standard *U-Net* method and a novel deep learning one called ASPP which combine *U-Net* with atrous spatial pyramid pooling. Among the performance measures are once again the Dice coefficient and the Jaccard index but also the sensitivity and specificity which is interesting in the context of the present work. The results show a clear advantage for the deep learning based method for Dice, Jac-

card but also in terms of sensitivity and specificity. The *U-Net* performed slightly better on pixel-based performance metrics but ASPP was better at separating merged objects. The dataset used in this study was comprised of 50 fluorescence microscopy tissue images of glioblastoma cells from which 30 were used for training and 20 used for evaluation. The paper concludes that the deep learning methods performed best but their training were time consuming.

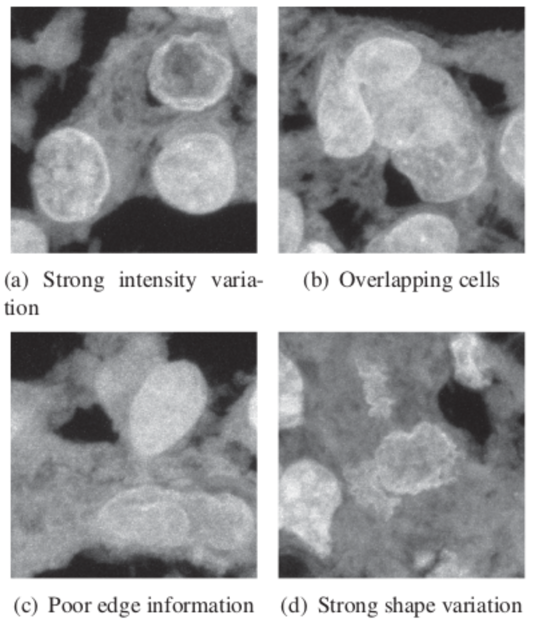
(a) Strong intensity variation
(b) Overlapping cells
(c) Poor edge information
(d) Strong shape variation

Figure 2.21: Figure from [4] which illustrates the challenges associated with the segmentation of glioblastoma cells.

2.3.3 Weakly supervised learning

As explained in section 2.2.3, the presence of incomplete masks in the data means that method applied in weakly supervised learning may need to be investigated. Weakly supervised learning for segmentation can mean that the masks are incomplete or that there is no mask at all. In the latter case, the label is attributed to the complete image. For instance, in binary segmentation, if at least one pixel of the image is foreground, then the image is labelled as foreground otherwise it is labelled as background.

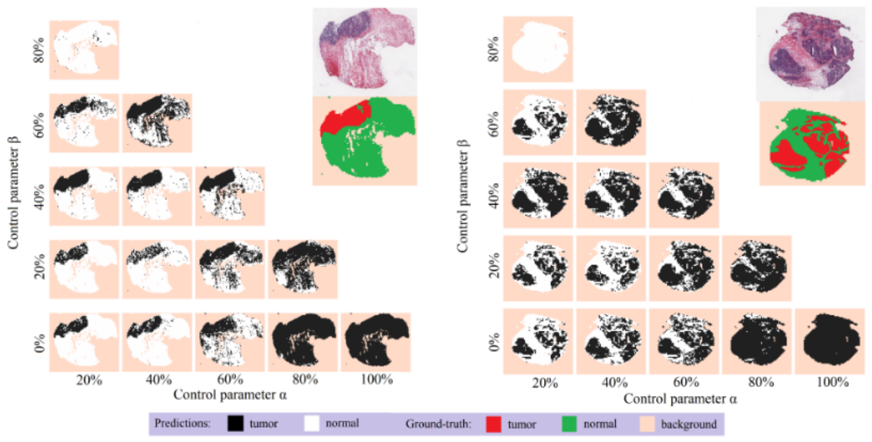

Figure 2.22: Figure from [10]. Unfiltered predicted tumor maps obtained with 15 bench-marked framework configurations.

In [10], the authors describe a method to learn a segmentation model from image level labels (presence of tumor or not in the whole-slide image). The method relies on multiple instance learning (MIL) and uses two parameters (α and β) to assume tumor area and normal tissue area respectively. The framework is assessed using a dataset that contains 6481 whole-slide images obtained from The Cancer Genome Atlas which is divided in training (65%) validation (15%) and testing sets (20%). The assessment show promising results with some configurations of α and β, giving annotations close the expert ones (see Figure 2.22).

In [11], the authors propose a solution to a problem of training a segmentation algorithm with sparse or incomplete annotations in the scope of gastric cancer tissue segmentation. The method works by first extracting a patch (sub-region) of interest from the images for the network training and predicting (patch-based FCN). In order to combined the patches predictions and rebuild the whole segmentation map, an overlapped region forecast algorithm and other post-processing operations are used. The patches are selected by using a a threshold on the annotated area (use the most *complete* patches). To further enhance the results, a reiterative learning scheme is introduced which consists in iterative training on the self-annotated training set (Figure 2.23). The method achieved a mean Intersection over Union coefficient (IoU) of 0.883 and a mean accuracy of 91.09%.

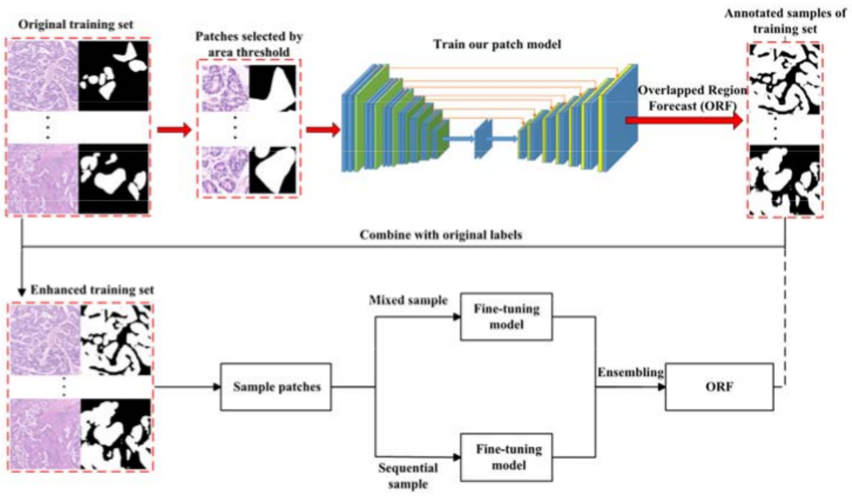

Figure 2.23: Figure from [11]. Overview of the reiterative learning pipeline

2.4 Fully convolutional network

2.4.1 Principle

The implemented segmentation algorithm uses a type of *fully convolutional network* (or FCN), which as the name suggests, is based on convolution. Rudiments about convolution operations and their use in the scope of neural network is given in this section.

Kernel The core of the FCNs is the convolution operation which is an operation between an input tensor and a given kernel. Kernels can be seen as **filters** whose convolution with an input image can highlight specific features in it.

Convolution Let us consider an input image I of size $H \times W \times C$ where C is the number of channels and a kernel u of size $h \times w \times C$. The convolution between I and $\mathbf{u}$ is given by :

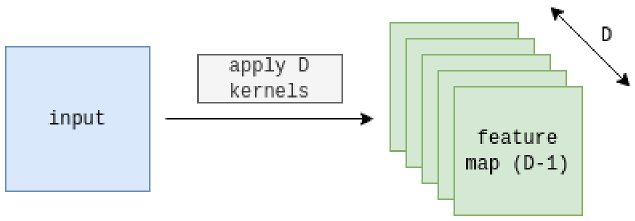

Figure 2.24: Feature maps.

$$I \circledast \mathbf{u} = O$$

where,

$$O_{n,m} = \sum_{c=0}^{C-1} \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} I_{c,n+i,m+j} \cdot \mathbf{u}_{c,i,j} \tag{2.6}$$

and O is of size $(H - h + 1) \times (W - w + 1)$ (padding can be added to preserve the input size).

In the context of FCNs, a bias $\mathbf{b}$ is introduced so that (2.6) becomes:

$$O_{n,m} = \mathbf{b}_{n,m} + \sum_{c=0}^{C-1} \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} I_{c,n+i,m+j} \cdot \mathbf{u}_{c,i,j} \tag{2.7}$$

where $\mathbf{b}_{n,m}$ and $\mathbf{u}$ are parameters to **learn**.

Intuitively, the bigger the value of $O_{n,m}$, the closer the *area* around $O_{n,m}$ matches the kernel feature. For instance, if $\mathbf{u}$ is used to detect edges and $O_{n,m}$ has a high value then the area corresponding to $O_{n,m}$ can be seen as an edge. The output O is called the feature map (for the kernel $\mathbf{u}$) as it *maps* regions of the image matching the kernel feature.

Feature maps Several kernels can be convoluted with the input so that different features can be extracted. The output of those convolutions can be stacked so that the total output is of size $D \times (H - h + 1) \times (W - w + 1)$ where D is called the **depth** and the output is called feature maps (Figure 2.24).

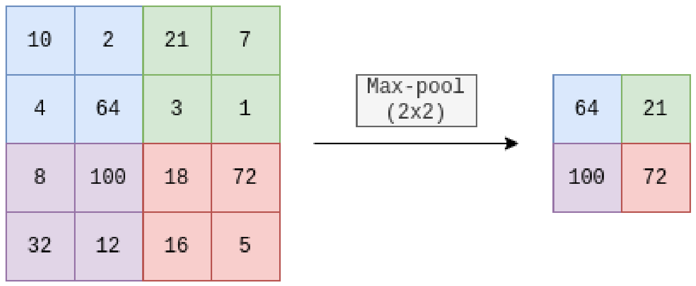

Figure 2.25: Illustration of a (2×2) max-pooling operation.

Max-pooling and up-sampling The max-pooling operation is used to reduce the input dimensions. It can be seen as a special type of kernel used for **down-sampling** by selecting the maximum value within a defined area in the input. The max pooling operation is illustrated on Figure 2.25.
The **up-sampling** is to opposite operation and is used to increase the input dimensions.

ReLU The rectified linear unit (ReLU) is an activation function f defined by:

$$f(x) = \max(0, x)$$

Activation functions are applied at the output of a layer in neural network. In the scope of FCNs, the ReLU is applied after the convolution operation $i.e : x = O_{n,m}$.

Architecture The convolution, max-pooling/up-sampling and ReLU together are the fundamental building blocks of the FCNs and they are generally assembled using the following rules:

- A convolution (`conv`) is followed by a ReLU.
- Several convolution operations can be chained to form a convolution block (`conv-block`).
- A max-pooling/up-sampling can be appended at the end of a convolution block.
- The network is made of two part : the **encoding** part where the input size is decreased from the initial input using max-pooling and an **en-**

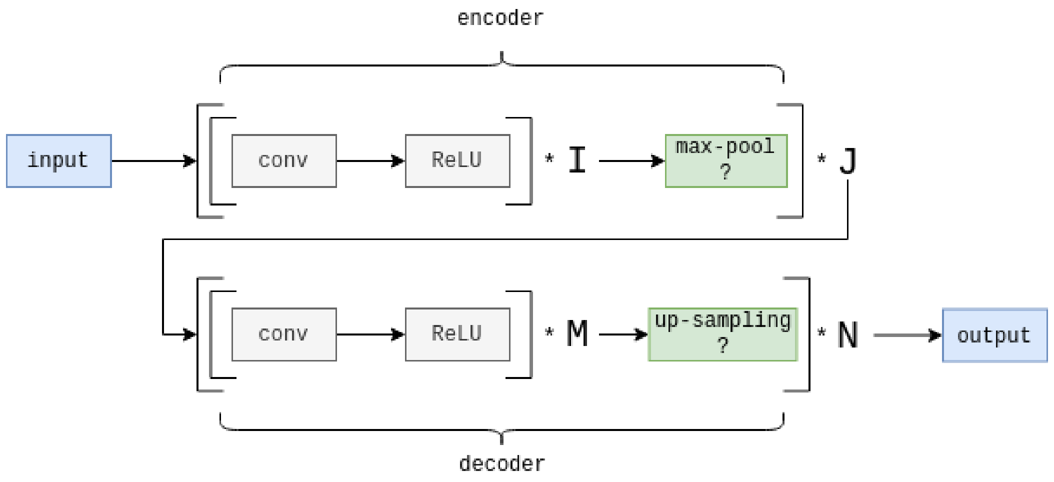

Figure 2.26: Generic FCN architecture with $I, J, N, M \in \mathbb{N}$. '?' marks the element as optional.

coding part increasing again the input size using up-sampling all way to the network output.

Once the building blocks are assembled following those rules, an FCN looks like the generic architecture shown on Figure 2.26.

Training In order to update the model parameters, the training makes use of an optimizer whose role is to tune the model parameters so that they minimize the loss function $\mathcal{L}$ over the training set. The core principle behind the optimizer is the stochastic gradient descent or SGD which is itself a popular optimizer.

Let θ be the model parameters and X an input image with $Y_p = f(X;\theta)$ then θ is updated as follows :

$$\theta_{t+1} = \theta_t - \gamma(\frac{1}{B}\sum_{b=1}^{B} \nabla\theta\mathcal{L}(Y_{n(t,b)}, f(X_{n(t,b)};\theta)))$$

where γ is the learning rate, B the batch size, t the step and $n(t,b)$ is a random order of sample visiting. The principle of the SGD is thus to find a

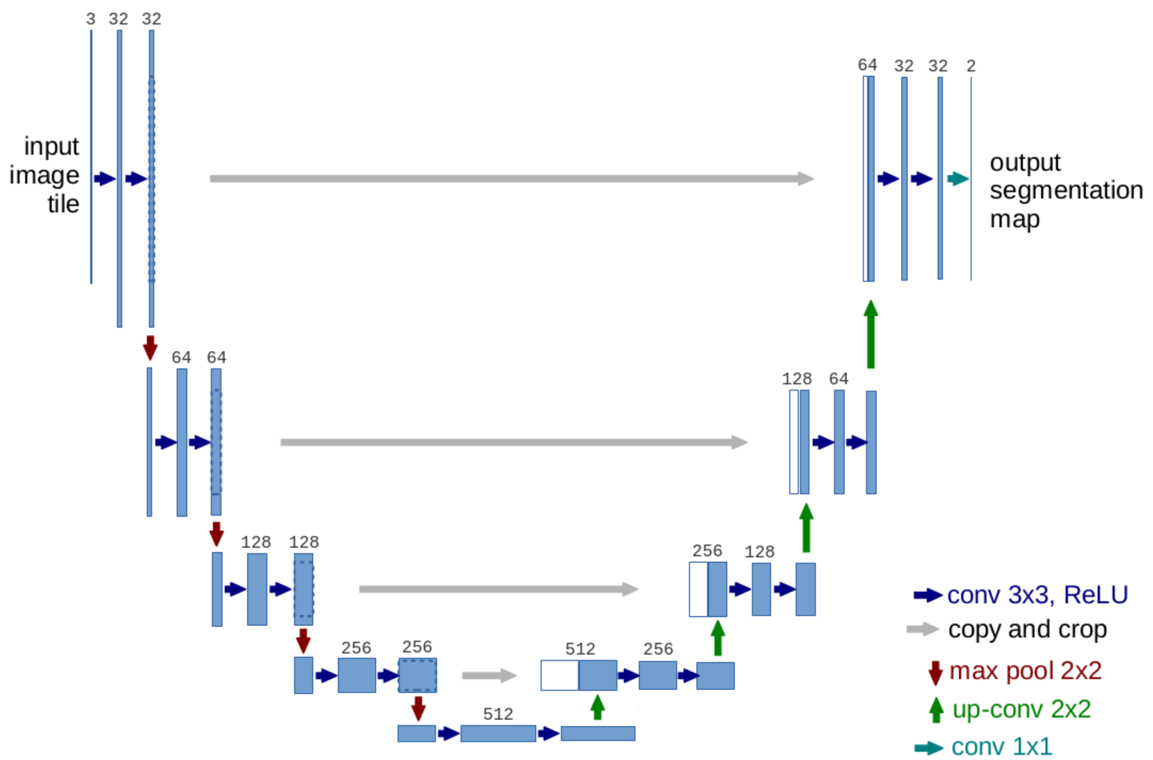

Figure 2.27: U-Net architecture, image derived from [16]. The numbers indicate the channels (or filters) count for each convolution.

minimum of $\mathcal{L}$ by following the slope of the gradient to update the parameters.

2.4.2 U-Net

The core of the segmentation algorithm is based on U-Net[16] which is a FCN with the addition of *skip connections* between `conv-block` in the encoder and the decoder as illustrated on Figure 2.27.

Intuitively, the U-Net works by first extracting the relevant features of the image in the encoding part, the deeper in the network the more complex

the detected features become. For instance, the very first `conv-block` might detect simpler features like edges, color intensity, ... The next `conv-block` will use those *general* features to derive more complex ones and so on.
The decoding part works in reverse and uses encoded image features to rebuild a complete image. The *skip connections* added in the U-Net help this process by giving information relative to the original image at each stage of the reconstruction.

2.5 Evaluation metrics

In order to evaluate the quality of the segmentation in a formal way, it is necessary to make use of metrics. Those that have been used are introduced in this section.
The two metrics used are the **Dice coefficient** also known as F1 score and the **Jaccard index** also known as the intersection over union (IoU). Theses are commonly used metrics in segmentation problems [9][19][4].

2.5.1 Dice coefficient

Let Y be the ground truth mask and Y_P be the predicted mask.
The Dice coefficient d is a similarity measure between Y and Y_P where:

$$d(Y_P, Y) = \frac{2|Y_P \cap Y|}{|Y_P| + |Y|} \tag{2.8}$$

In practice, (2.8) can be computed using a vector operation.
Let $\mathbf{y}$ be the ground truth mask in vector format and $\mathbf{y_P}$ be the predicted mask also in vector format, (2.8) can be rewritten as follow :

$$d(\mathbf{y_P}, \mathbf{y}) = \frac{2|\mathbf{y_P} \cdot \mathbf{y}|}{|\mathbf{y_P}|^2 + |\mathbf{y}|^2}$$

The values of d range from 0 to 1. A value of $d = 0$ indicates that the sets are completely different while $d = 1$ indicates a full similarity and thus the closer d is to 1, the better.

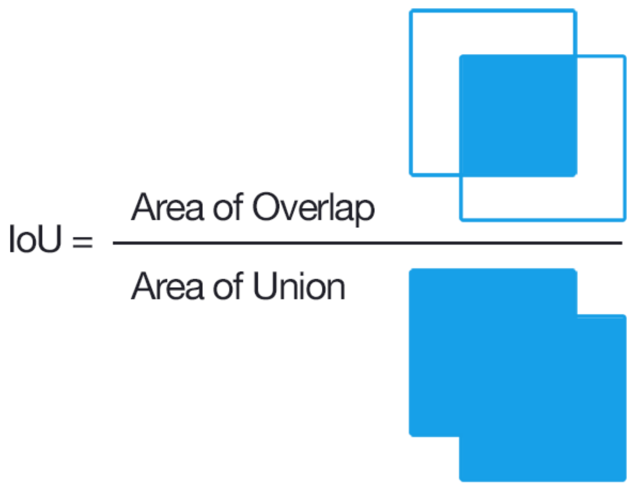

Figure 2.28: Intersection over union [17].

2.5.2 Jaccard index

The Jaccard index coefficient j is also a similarity measure between Y and Y_P where:

$$j(Y_P, Y) = \frac{|Y_P \cap Y|}{|Y_P \cup Y|} \tag{2.9}$$

Informally, it can be considered as a measure of the overlap between Y and Y_P as shown on Figure 2.28.

The Jaccard index and Dice coefficient are related each other and j can be computed from d using :

$$j = \frac{d}{2 - d}$$

The values of j also range from 0 to 1. A value of $j = 0$ indicates no overlap while $j = 1$ indicates a full overlap and thus the closer j is to 1, the better.

Chapter 3

Proposed methodology

This chapter describes the implementation and methodology done to solve the segmentation problem. Among the relevant points is a proposed iterative data improvement solution for the weak annotation and the implementation of a generic and modular segmenter for easy integration in Cytomine.

3.1 Overview of the method

The Figure 3.1 provides a simplified overview of the method along with the different modules used to implement the Cytomine cell segmentation software. This provides a broader view over the key elements implemented in this project and how the model was obtained.

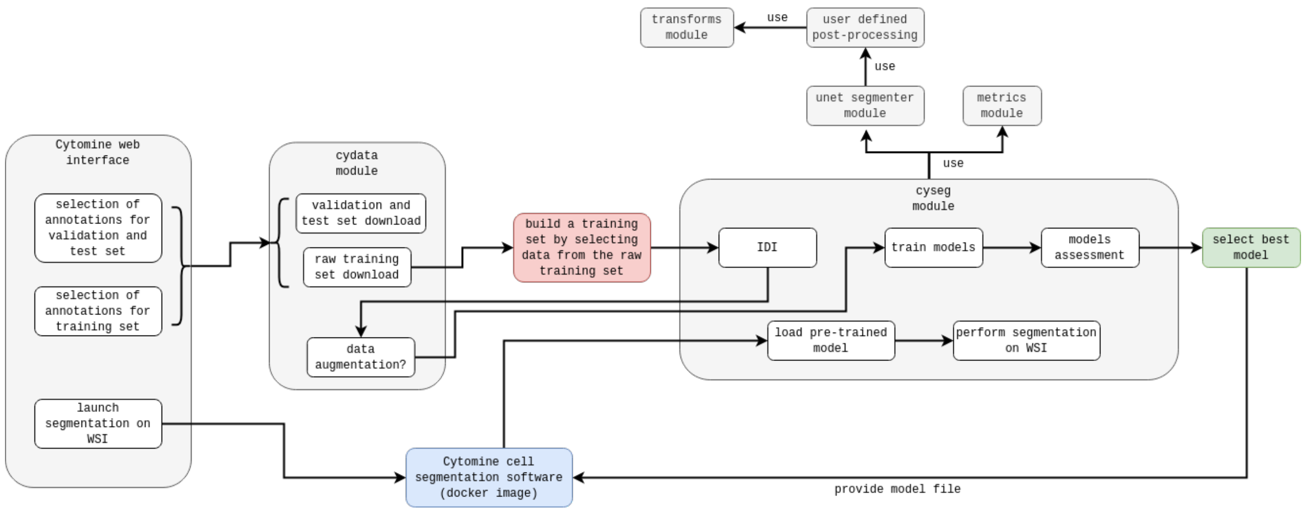
transforms module
use
user defined post-processing
use
unet segmenter module
metrics module
use
Cytomine web interface
selection of annotations for validation and test set
selection of annotations for training set
launch segmentation on WSI
cydata module
validation and test set download
raw training set download
data augmentation?
build a training set by selecting data from the raw training set
cyseg module
IDI
train models
models assessment
select best model
load pre-trained model
perform segmentation on WSI
Cytomine cell segmentation software (docker image)
provide model file

Figure 3.1: Overview of the methodology and modules. The '?' marks the augmentation step as optional because it wasn't used in practice since it hasn't significantly improved the results but improvements could be brought to it. 'IDI' stands for iterative data improvement.

3.2 Implementation architecture

The software developed for this work is built around 6 main modules.

1. `cydata` (CYtomine DATAsets) : This module contains all utilities used to handle the datasets. It implements modes for dataset downloading from the Cytomine server, splitting a dataset and augmenting a dataset. It was also used to add regions of interests in Cytomine WSIs for the building of the validation and test sets.

2. `cyseg` (CYtomine cells SEGmentation) : This is the main module, it can run the various modes associated with the implemented segmenter which are : training the model, segmenting a dataset and/or Cytomine WSIs and improving a dataset. It can serve as a standalone program for testing/prototyping of a segmenter model but it can also be easily run in Cytomine for segmentation of whole-slide images.

3. `segmenter` : This generic class specifies and implements the elements of a segmentation module that can be easily customized and integrated in

an image processing pipeline. This module also contains the integration of the developed algorithm (`UNet` class) within the Cytomine server using the SLDC framework [15] to abstract the Cytomine WSIs to the model. The Figure 3.2 illustrates the segmenter customizations. The classes weights enable to associate different weights to each classes for loss and metric computations.

4. `unet` : This is a subclass of `Segmenter` that implements the segmentation using the U-Net architecture. It also contains the definition of the used training loss and post-processing transforms. This module in fact contains the user defined element shown on Figure 3.2.

5. `transforms` : This module contains a set of implemented transforms that can be applied to an image for pre/post processing. The following transforms were implemented : normalize, smoothing, erode dilate and threshold. More details in the post-processing section.

6. `metrics` : It implements the two metrics used in this project : the Jaccard index and the Dice coefficient. These metrics have been implemented in such a way that they are compatible with the `PyTorch` gradients computation so they can be used as part of a loss function.

The aforementioned modules were implemented in *Python* using mainly the `PyTorch`, `OpenCV` and `Cytomine-Python` frameworks.

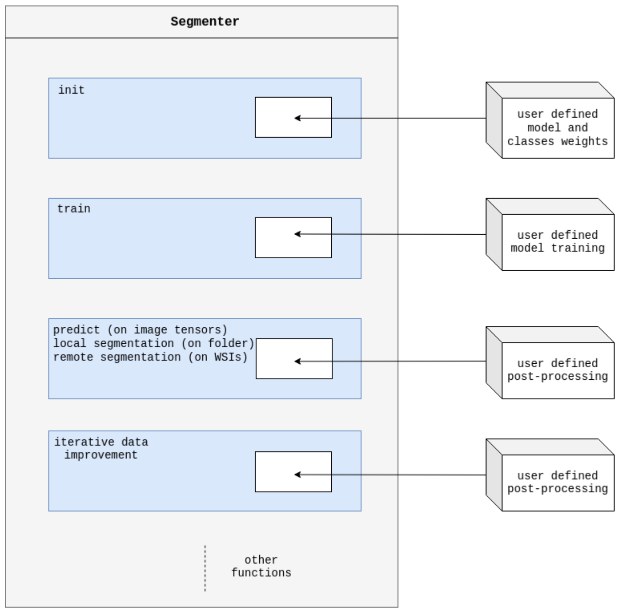
Segmenter
init
train
predict (on image tensors)
local segmentation (on folder)
remote segmentation (on WSIs)
iterative data improvement
other functions
user defined model and classes weights
user defined model training
user defined post-processing
user defined post-processing

Figure 3.2: Segmenter customizations.

3.3 Dataset

3.3.1 Dataset building

As shown on 3.1, the dataset is built using the following steps:

1. **Annotations selection** : The Cytomine web interface has a feature allowing the user to select the annotations he wants and then to download a *csv* file containing all those annotations with their relevant associated features such as: project id, image id, coordinates in the whole-slide image, ... This file can then be used at the second step to download the dataset. The dataset can also be downloaded using JSON files provided by the Cytomine server and containing the aforementioned components.

2. **Dataset download** : For each annotation in the file, a crop as well as its masked version is extracted from the original whole-slide image using the coordinates of the annotation.

3. **Data selection and improvement** : As it will be further explained in the results, the data from the *raw* dataset was needed to be selected to build a smaller dataset with less incomplete and sparse annotations. This new dataset is then improved (completing the weak annotations) using iterative data improvement (see section 3.5).

4. **Data conversion** (not on Figure 3.1) : While the RGB crops don't need data conversion to be used in the training process, their corresponding mask must be converted from an RGB tensor to a mask tensor according to the format specified in (2.5). This conversion is however done when the mask is loaded and not in the actual file so that the crop and mask can be easily visualised. An example of a sample of the dataset is shown in Figure 2.10.

3.3.2 Data augmentation

The data augmentation is used to increase the number of data available by applying various transformations to them. This is a commonly used technique to make up for a lack of data or improve model generalization.

Data augmentation of the training data was implemented in this work in an effort to improve the results and the following augmenters were used :

- Apply a vertical symmetry with a probability of $p = 0.5$.
- Apply a horizontal symmetry with a probability of $p = 0.5$.
- Randomly translate the image along the X axis between -15% to 15% (zero padding added).
- Randomly translate the image along the Y axis between -15% to 15% (zero padding added).
- Rotate the image by 0, 90, 180 or 270 degrees randomly.

The augmenters were applied in a sequential fashion to each crop of the dataset (and its corresponding mask), the sequential order being randomized for each crop. One augmentation step on a dataset will thus double its size. The Figure 3.3 shows an example of a crop being augmented using the augmenters sequentially.

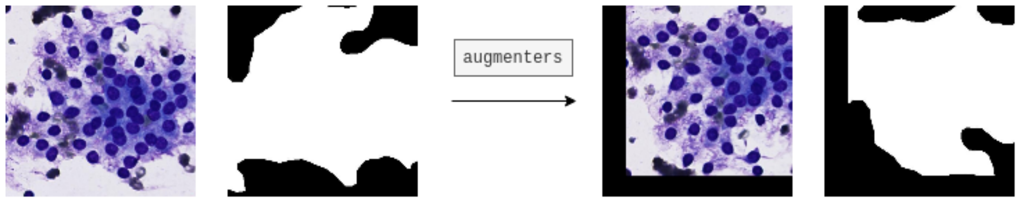

Figure 3.3: Example of an augmented crop along with its corresponding mask.

3.4 Segmentation algorithm

3.4.1 Motivation

The inquiry about the the state of the arts methods in section 2.3 showed the U-Net to be well suited for segmentation in biomedical applications, in addition to surpass other models for the same task. The architecture also has the following advantages:

- Well documented and numerous implementation examples available.
- Easy to modify/adapt for different tasks.
- As stated in [16], the network can be efficiently trained with few images which might mean that there won't be the need to use all the data (a large portion of them containing weak annotations) and that the training might be faster.

The choice for the core algorithm was thus made with U-Net.

3.4.2 Training

As explained in section 2.2.1, the training of the U-Net is performed by adjusting its parameters so that it minimizes a loss function $\mathcal{L}$.

Let Y_P be the predicted segmentation mask and let Y be the ground truth segmentation mask. The loss function $\mathcal{L}$ was define as (similar definition in [11]):

$$\mathcal{L} = \mathcal{L}_{bce} + \mathcal{L}_{dice}$$

$\mathcal{L}_{bce}$ is the binary cross-entropy loss which can be written as follow :

$$\mathcal{L}_{bce}(Y_P, Y) = -\frac{1}{N}\sum_{i=0}^{N-1} Y_i \log(Y_{Pi}) + (1 - Y_i)\log(1 - Y_{Pi})$$

where N is the number of pixels in the mask, Y_i is 1 if the pixel in i is foreground (0 for background) and Y_{Pi} is the predicted probability of the pixel in i to be foreground. The first term of the sum penalizes false negatives while the second term penalizes false positives.

$\mathcal{L}_{dice}$ is the dice loss which is a dissimilarity measure between 2 sets that is commonly used in segmentation problems :

$$\mathcal{L}_{dice}(Y_P, Y) = 1 - d(Y_P, Y)$$

where $d(Y_P, Y)$ is the Dice coefficient introduced in section 2.5.1.

The optimizer used is *Adam*[8] which is an extension of the SGD and batch normalization[7] was also employed in the model to speed up the training.

The model is trained on 512×512 crops with a batch size set to 1 as in [16] to help reduce memory footprint on the GPU. The implemented U-Net has a variable depth and also support multi-class segmentation (though only binary segmentation was used).

3.4.3 Post-processing

The post-processing encompasses all treatments made to the predicted segmentation mask *after* it is produced by the model. The goal is to further refine the results which often include reducing the noise. The following transforms were used.

Erode dilate Erosion can be achieved by sliding a kernel over the image and replacing the pixel value in the image at the center of the kernel by the minimal pixel value in the image overlapped by the kernel.
The dilation is the opposite operation which instead replaces the pixel value in the image by the maximal pixel value in the image overlapped by the kernel.

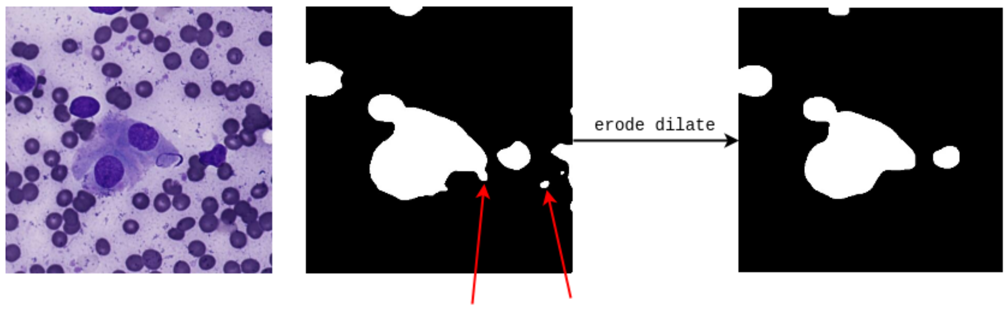

Figure 3.4: Erode dilate transform (the arrows indicate noise examples).

The combination of erosion and dilation can be successful to suppress noise in the image. Indeed, the noise is first removed by the erosion but not restored by the dilation unlike the rest of the image. An application of the erode dilate transform is illustrated on Figure 3.4.

Smoothing The smoothing (or blur) is useful to soften the edges of the mask thus making it more *natural*. A (3×3) smoothing can be achieved by simply using the following kernel:

$$\mathbf{u} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The new pixel value is computed by averaging the value with the one of the neighboring pixels.

Threshold Since the output values given by the model are continuous (the predicted values correspond to the probabilities of the pixels to be foreground), it is necessary to determine a threshold from which the prediction for each pixel will be considered foreground or background, the most common choice being to simply *round* the predicted value. The threshold transform is thus used to convert the output into discrete values :

$$v(i,j)' = \begin{cases} 1 & \text{if} \quad v(i,j) > \mathit{threshold} \\ 0 & \text{otherwise} \end{cases}$$

where $v(i,j)$ is the value of the pixel at position (i,j).

Tiles merging As the segmenter works on tiles and some inaccuracies can occur, especially on the edges (lack of context), this can reduce the quality of the segmentation mask once the tiles have been reassembled. To mitigate this problem, it is necessary to consider methods for tiles merging. Fortunately, for the segmentation on WSIs, the SLDC[15] framework already implements tiles merging techniques so that no further work is required. The segmenter was consequentially developed with a *per-tile* segmentation approach and no reassembling. However, for reasons that will be further explained, a (late) choice was made to assess the models on crops bigger than 512×512 thus requiring tiles reassembling. In order to implement some sort of merging for those local segmentations, the idea was to apply the aforementioned post-processing techniques on the reassembled mask. In practice, as it will be seen on the results, this technique proves to be efficient enough for the models assessment as the tiles related inaccuracies can hardly been seen.

3.5 Iterative data improvement

3.5.1 Motivation

In order to deal with the weak annotations, a method involving iterative data improvement (IDI) is proposed. This section will focus on the principle of the technique and more in depth results will be shown in section 4.2.

3.5.2 Principle

One of the properties of the convolutional neural network is the shift invariance which is the ability to withstand shifting in the feature of the image while still detecting those features. This can be partly explained by the use of the same kernels that are identically applied on the complete image (kernels sharing).
If we take the example of the cells detection, the U-Net will learn to detect cells in the image and, under the good assumption, in such a manner that it does not only depend on the location of those cells.

The phenomenon (kernels sharing) that enables the shift invariance property can also have other repercussions that can be exploited. In the context of weak annotations, it means that the model will struggle to fit exactly the

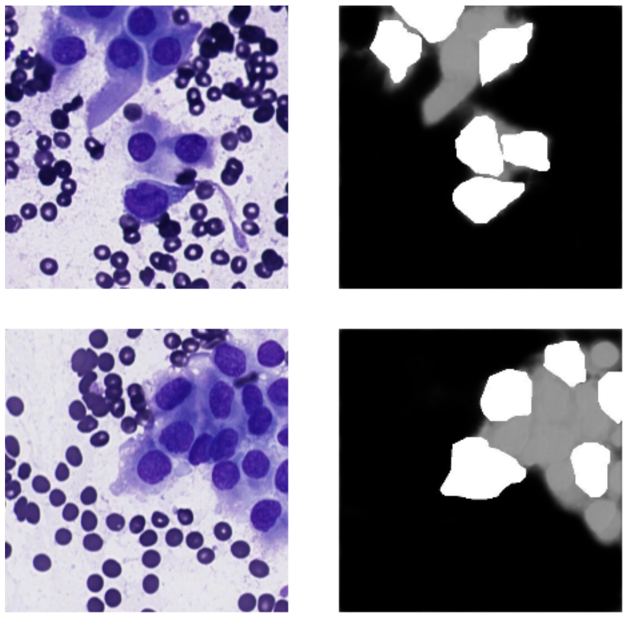

Figure 3.5: Results obtained by training a model on incomplete annotations. The expert annotations are in white while the low tone gray are the model predictions. The low tone indicates a low confidence of the model and thus no threshold transforms were applied to obtain this result.

training data because if it learns to detect cells at some locations, it will likely detect the cells at other locations on the image because they will share the same features. What is likely to happen is that the output probabilities of the foreground class for the pixels belonging to detected cells will be lower as a trade-off for the loss and this will be especially the case if the dataset is large and filled with sparse annotations. This reasoning is obviously simplified as other factors are involved. In practice, this phenomenon was observed and example is illustrated on Figure 3.5 with a model trained on 107 incomplete annotations.

The principle of the iterative data improvement tries to exploit the phenomenon shown on Figure 3.5 and works in the following steps (see Figure 3.6):

1. **Training** : The model is trained on the dataset to improve.
2. **Segmentation** : The model is used to predict the segmentation masks

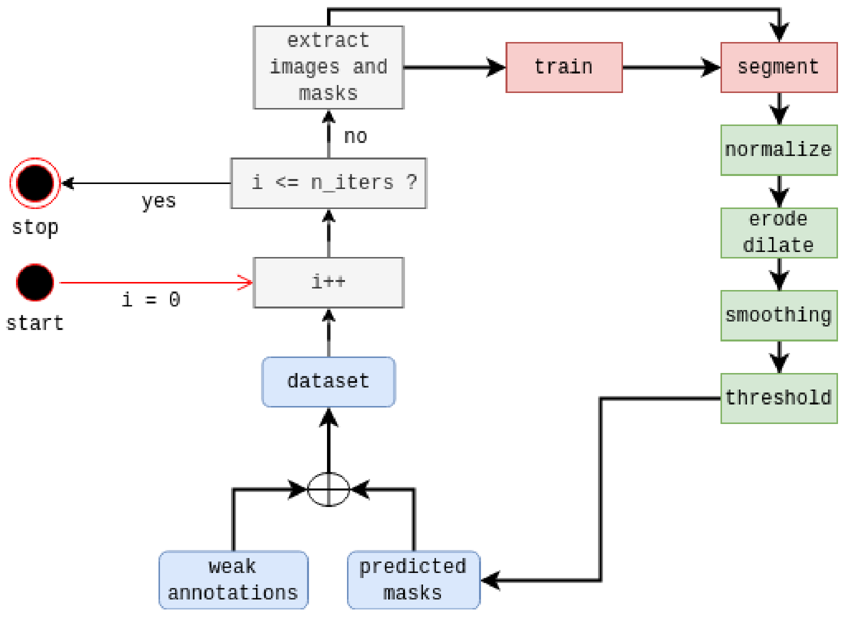

Figure 3.6: Iterative data improvement.

of the dataset to improve.

3. **Post-processing** : The required post-processing is applied to the predicted masks.

4. **Merging** : The predicted masks are merged with the weak annotations using a bit-wise OR operation and the file is saved in place of the old ones.

5. **Repeat** : Repeat the above steps iteratively.

3.5.3 Post-processing

While IDI can work well if the dataset is relatively small and without too much sparse annotations, it is very likely that it won't converge properly on less suitable data as shown on Figure 3.7 because the output probabilities of the foreground class for cell pixels would be under the threshold.

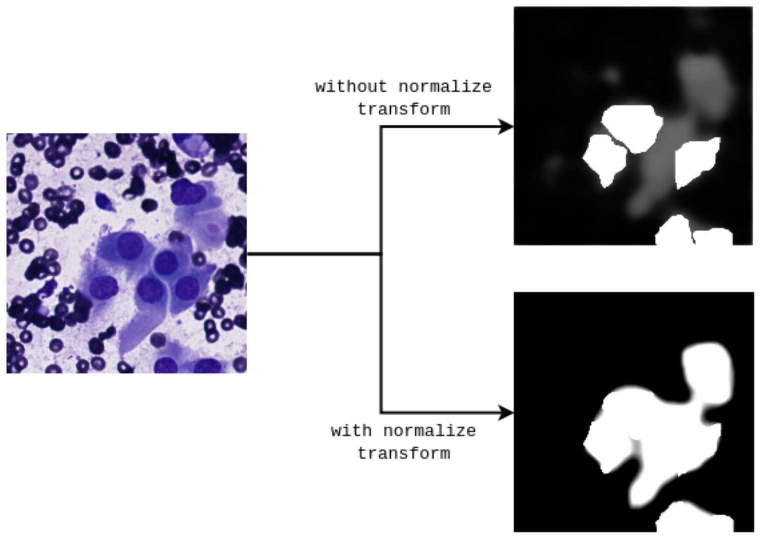

Figure 3.7: IDI with and without the normalize transform (dataset of 236 annotations, half of them being sparse). Without the normalize transform, the mask on top would not contain any predicted annotations after the threshold is applied given the low output probabilities for the foreground class while for mask below, the threshold would not drastically alter it.

To mitigate this problem one might want to adjust the threshold but this solution is complicated in practice as it is difficult to determine the correct value and that value will depend on the dataset. A better solution is to normalize the predicted masks **before** they are merged with the weak annotations (see Figure 3.7).

Let M be the predicted mask with C channels, the normalized mask M' is computed *per channel* as follow:

$$M'_c = \frac{M_c - \mu(M_c)}{\sigma(M_c)} \quad \forall c = 0, ..., C-1$$

where $\mu(M_c)$ and $\sigma(M_c)$ are respectively the mean and standard deviation of the channel M_c.

In practice, the normalization enables the IDI to work on bigger datasets with more sparse annotations as it will be shown in the results.

3.6 Cytomine integration

The integration in Cytomine can be divided into main 2 steps :

1. The first step is to implement the required functionalities in the `cyseg` and `segmenter` modules so that the program is able to be run from the Cytomine server and perform a **remote** segmentation of WSIs using a pre-trained model. This step mainly involves integrating the SLDC[15] framework into the `segmenter` module and adapting the interface of `cyseg`.

2. The second step involves the creation of a Docker image of the application and the insurance of the compliance with the Cytomine software development guidelines. The Cytomine software is run from a Docker image (virtualization technology to run software in a containerized environment) so that there is no problem with the required environment and libraries while also providing easy updating.

The `cyseg` program has 2 functions for remote segmentation of WSIs : segmenting portions (or windows) (Figure 3.8) or complete segmenting of the WSI. Segmenting only portions of the WSI can be useful to assess the model

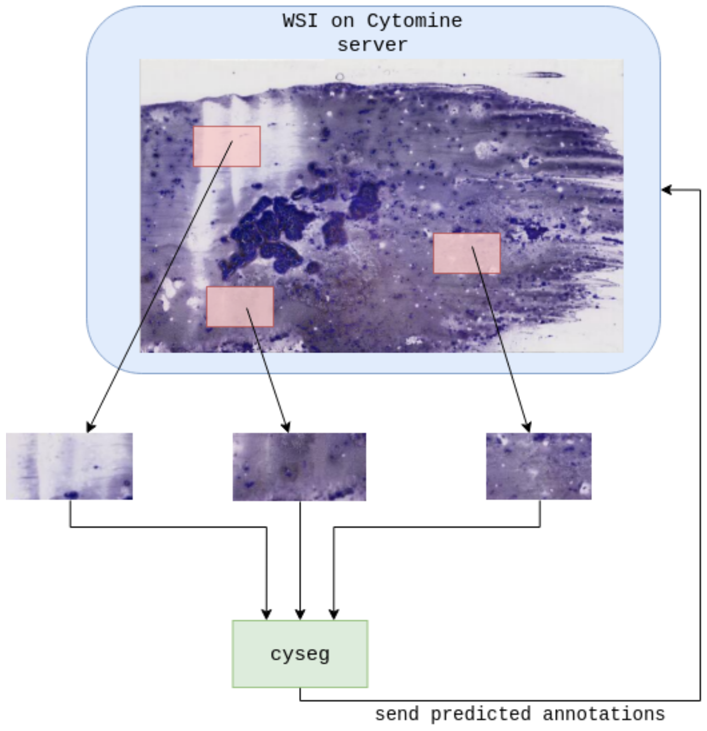

Figure 3.8: Segmentation of portions (windows) of the WSI.

on specific areas (for instance, containing expert annotations) while avoiding the computational overhead of the segmentation on a complete WSI. Segmenting on windows can also open the way of a parallelized segmentation of the WSI and the increased computational speed associated (the merging of windows should be addressed in this case).

In order to upload the predicted annotations, the predicted masks need to be converted to polygon objects that can be efficiently sent to the Cytomine server. The polygons represent the areas covered by foreground in the predicted masks. The relevant steps of the remote segmentation of a Cytomine WSI are illustrated on Figure 3.9.

At the time of writing of this thesis, the integration is almost complete, the program is able to segment portions of the WSI but some technical problems prevent the segmentation on the complete WSI. These problems

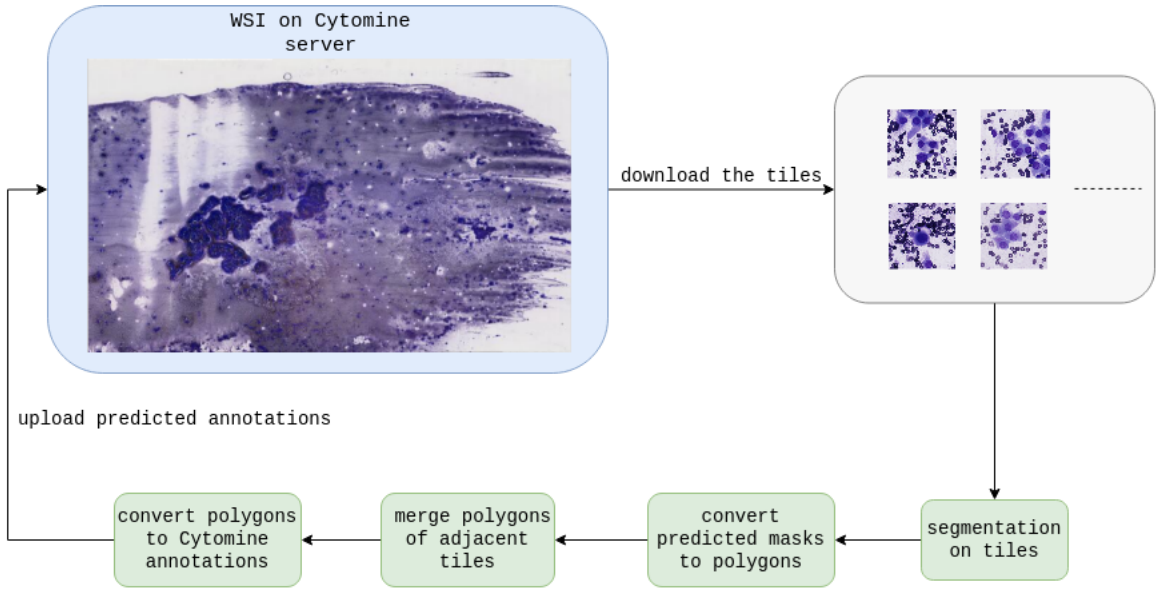
WSI on Cytomine server
download the tiles
segmentation on tiles
convert predicted masks to polygons
merge polygons of adjacent tiles
convert polygons to Cytomine annotations
upload predicted annotations

Figure 3.9: Remote segmentation of a Cytomine WSI. The polygon objects represent the areas covered by foreground in the predicted masks.

will hopefully be solved for the defense.

Chapter 4

Experiments and Results

The following chapter will describe the various experiments that were made along with their associated results while assessing the performance of the solution under various aspect. The impact of the training set and the hyper-parameters of the model will be assessed on a validation set. The computation performance will also be assessed before drawing the conclusion and suggesting improvements.

4.1 Validation and testing sets

According to the introduced evaluation process in Figure 2.12, the datasets must be split into three distinct ones. However, ensuring that the crops are uniquely used in each set is not enough because of the bias resulting from the specificity associated to the whole-slide image from which the crops in the datasets derived, as explained in section 2.2.3.

It is thus crucial that the data used to train the model does not integrate the specifity of the whole-slide images in the validation or testing otherwise it will compromise the relevance of the results regarding the generalization of the model.

The sets were constructed as follows :

1. WSIs were put aside for the testing and validation sets, the remaining were put in the training set and each WSI is unique in each set.

set	WSIs as source	# crops	crop size (pixels)
validation	3	20	2048×2048
test	3	25	2048×2048

Table 4.1: Summary of the validation and test sets.

2. Attention was made by looking at the IDs and times of the WSIs to ensure that they don't share the same patient and time across the different sets.

3. Because of the weak annotations (section 2.2.3) and the need to have complete data for model validation and assessment, manual annotations were added to the selected WSIs. Some were made by completion of the annotations of the expert but since there weren't enough of them, the vast majority were manually added. A total of 307 annotations were manually added.

4. The crops and their corresponding masks were extracted as described in section 3.3.1.

The size of the crops in the validation and testing set is 2048×2048 which is 16 times bigger than the size of the training crops. This choice is motivated by the following reasons :

- The greater the crop, the closer it represents the WSI it belongs to because it contains more and diverse types of cells and background.

- It is a less tedious and faster task of annotating a few bigger regions of interest than creating a lot of smaller annotations. As this step must be done manually, this is a great advantage.

The characteristics of the sets are summarized in Table 4.1. Although the number of crops seems limited, their bigger size means that the actual number of segmentations performed by the model is 16 times bigger.

4.2 Results

The goal of the assessment is to evaluate the **methodology** used to train the models and also to perform model **selection** for the integration in Cytomine. Several tests were made to assess the method regarding various aspects of the problem and the results are presented in this section.

4.2.1 Selection and assessment stages

The process of assessing the proposed methodology was divided in several stages described below.

Stage 0 : Early prototyping This phase is not clearly defined and typically involves the early experiments made while implementing the model. This stage does not give clear results though it is useful to bound the problem and assess the feasibility of a solution.

Stage 1 : Dataset selection The goal of this phase is to determine which dataset to use for model training, namely the raw dataset (`d_raw`) or a selected dataset (`d_selected`) whose elements are selected from the raw dataset. This approach is motivated by the following reasons :

- Experiments made during the early prototyping seem to indicate that the model performs better on reduced datasets made up of selected data than on the raw dataset downloaded from the Cytomine server. The assumption is that the larger presence of incomplete and sparse annotations in the raw dataset hurts the training.
- The *raw* dataset downloaded from the Cytomine server contains erroneous annotations and/or corrupted data. For instance, some crops are abnormally blurred, some annotations are erratic, etc. This means that a first cleaning step (which is hard to automate) is required and, given the size of the *raw* dataset, this can be an extensive task.
- The time required to train the model would be significantly bigger on the raw dataset than on a selected one.
- The validation set is equivalent to 320 (512×512) crops and the test set is equivalent to 400 crops. By using a reduced training set of similar size, the results become more relevant.

Because of the disparity in annotations quality, the datasets are split into 2 parts : pattern annotations (mostly complete) and cell annotations (mostly incomplete or sparse). The latter will be subjected to IDI before being recombined with the pattern annotations. The resulting dataset is used for training (Figures 4.1 and 4.2) .

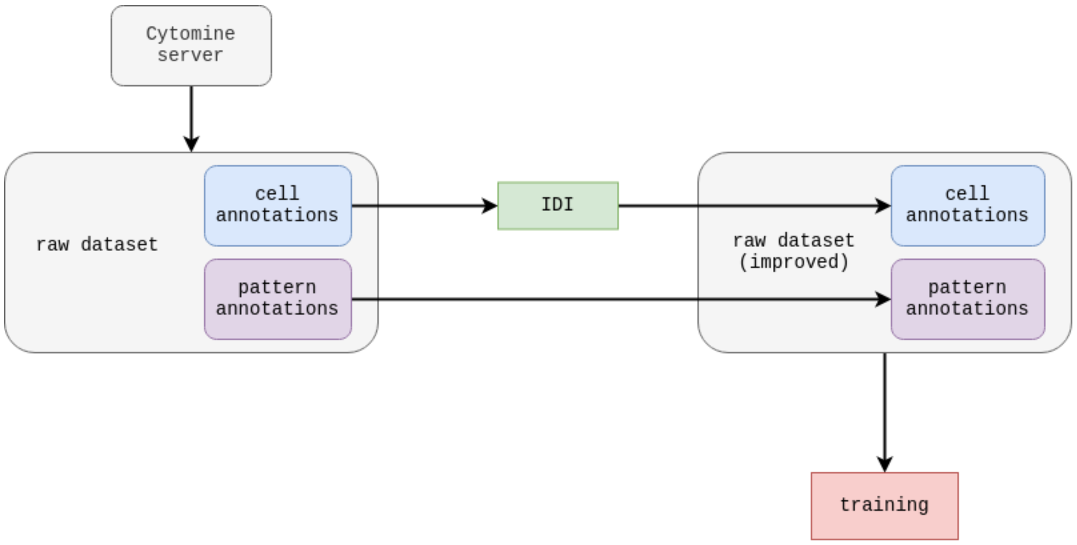

Figure 4.1: Training using the raw dataset.

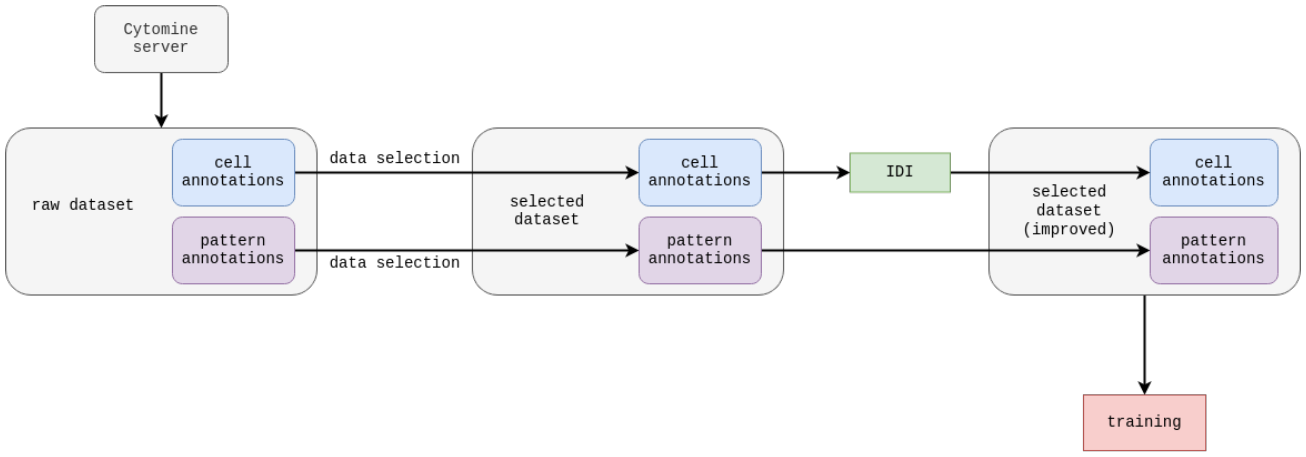

Figure 4.2: Training using the selected dataset.

Data augmentation will also be applied to the chosen dataset (either `d_raw` or `d_selected`) to study its potential benefits.

Stage 2 : Model selection This phase will study the impact of 2 hyper-parameters : the initial number of filters n and the threshold t. The goal is to find the most suitable combination of them. The models are trained using the dataset chosen at stage 1 and validated on the validation set. The performance of the models in terms of computation time and memory usage depending on n will also be assessed.

The hyper-parameter n is the value of the **initial number of filters** of the U-Net, *i.e* 32 on Figure 2.27. Since the number of filters is doubled at each stage, n affects the number of filters at **each** stage of the U-Net. For instance, a value of $n = 8$, would mean a number of filters in the last stage equal to 128. The hyper-parameter n is thus used to control the model **complexity**.
The threshold t is simply the threshold value used by the threshold transform applied after the predictions.

The **goal** of this phase is to be able to derive a model **selection method**.

Stage 3 : Assessment on test set This is the final phase which will assess the selection method on the test set in order to draw the final conclusions.

4.2.2 Data selection

The dataset `d_selected` is built using the 2 following rules:

- Select the most complete annotations ($\approx$ 50% or more of annotated cells in the crop).
- Annotations should be selected from diverse WSIs.

A summary of the datasets is given on Table 4.2.

First, the IDI is applied on the subset containing the cells of both datasets. Then a model is trained on the resulting improved datasets (improved cell annotations along with the pattern annotations) and evaluated on the validation set.

set	# crops	of which patterns	of which cells	crop size (pixels)
`d_raw`	4677	1731	2946	512×512
`d_selected`	407	300	107	512×512

Table 4.2: Summary of the raw and selected datasets.

set	# crops	IoU	Dice
`d_raw`	4677	0.35	0.52
`d_selected`	407	0.67	0.80

Table 4.3: Average IoU and Dice scores obtained by training models on the raw and selected datasets. Evaluation performed on a validation set of 20 crops of size 2048×2048.

Since the optimal hyper-parameters of the model are not known, different values of n (8, 16, 32) were tested (3 models for each) and t is fixed to a standard 0.5 (round). It should be noted that the goal is not to find an optimal model but rather to identify a tendency.
The final results on Table 4.3 report the average IoU and Dice obtained with each dataset.

What can be seen from Table 4.3 is that `d_selected` performs substantially better than `d_raw`. As explained above, it is very likely that the huge presence of weak and erratic/erroneous annotations affect the model performance on `d_raw`. Moreover, as it can be noticed on Table 4.2, the proportion of cell crops (and thus poorer annotations) in `d_raw` is significantly bigger than the proportion of patterns while the situation in `d_selected` is the opposite.

IDI seems more efficient on a smaller dataset and it is confirmed by an inquiry of the results of the IDI applied on both datasets. On shared crops with weak annotations among both datasets, the IDI managed to produce better annotations as illustrated by an example on Figure 4.3. However, IDI on both datasets also produced some false positives as shown on Figure 4.4.

Given the results, the chosen dataset is `d_selected`. The next step was to perform data augmentation on it to see if it managed to improve the results using the same assessment strategy as before. An augmentation step

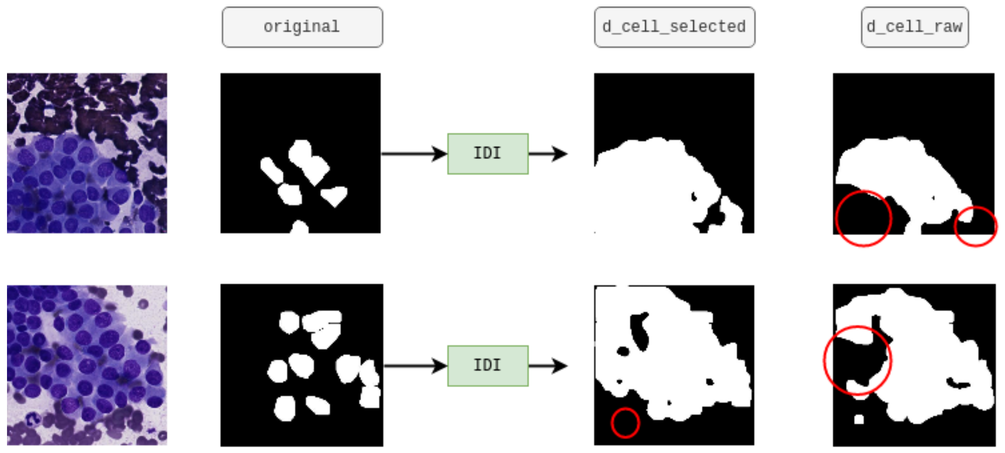

Figure 4.3: IDI using the raw and selected datasets. Missing foreground highlighted in red.

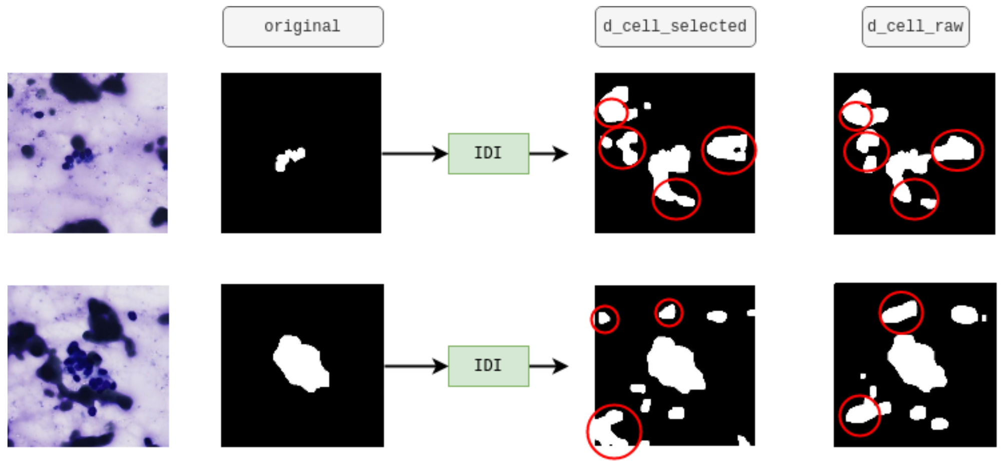

Figure 4.4: IDI using the raw and selected datasets. Examples of false positives are highlighted in red.

set	aug. steps	# crops	IoU	Dice
`d_selected`	0	407	0.67	0.80
`d_selected`	1	814	0.67	0.80
`d_selected`	2	1221	0.66	0.79

Table 4.4: Average IoU and Dice scores with and without data augmentation. The number of steps indicate the number of times data augmentation was applied on the original dataset. Evaluation performed on a validation set of 20 crops of size 2048×2048.

means that data augmentation is applied once on the original dataset and thus each augmentation step will increase the dataset size by its original size. The Table 4.4 shows the average IoU and Dice obtained with and without data augmentation.

As shown on 4.4, there is no significant change on the scores with 1 step of data augmentation. As for 2 augmentation steps, the results are slightly worse but, again, with no significant difference. It thus seems that the proposed data augmentation does not help improving the results and is even slightly decreasing them when significantly more data is added. It should be noted however that the apparent ineffectiveness of the data augmentation may be due to the augmenters employed. Data augmentation should not be considered completely inefficient as other augmenters may be tested.

To conclude with the data selection, the following tests will be made on `d_selected`. The identified tendency is that greater datasets don't seem to produce substantially better results. It seems however that the correctness and diversity of the data are beneficial. The following guidelines for dataset building can thus be derived :

- The data should come from a diverse pool of WSIs so that the model can better generalize on different types of background.
- The annotations effort should focus on lesser but more complete annotations ($\approx 50\%$ or more of annotated cells in the crop) rather than making a large portion of sparse annotations.

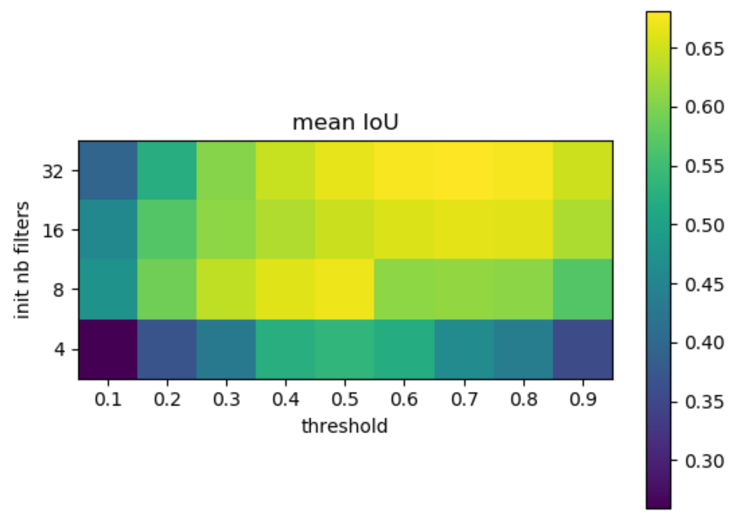

Figure 4.5: Mean IoU on validation set depending on the initial number of filters n and the threshold t.

4.2.3 Hyper-parameters tuning

The goal of this section is to study the influence of the hyper-parameters n and t (defined in section 4.2.1) on the segmentation performance. Four values of n were tested : 4, 8, 16, 32. 10 models were trained with each value of n and for each one of them, 10 values of t were tested and the final results report the IoU scores obtained for each pair (n, t) on Figures 4.5, 4.6 and 4.7.

It can be seen right away from 4.5 that $n = 4$ performs substantially worse than the others. The observed tendency is that as n increases, the average IoU will get better over a wider range of threshold values. For instance, for $n = 4$ the spectrum of *good* values would be t in the close neighborhood of 0.5. For $n = 8$ it can be 0.4 to 0.5, for $n = 16$ it's 0.6 to 0.8 and for $n = 32$ it's 0.5 to 0.8. The best models on average were obtained with $n = 32$ associated with a t of 0.6 and 0.7 and interestingly, with $n = 8$ and $t = 0.5$. The need for a higher threshold may indicate that the model is prone to **false positives** whereas in the other case, it is prone to **false negatives**.

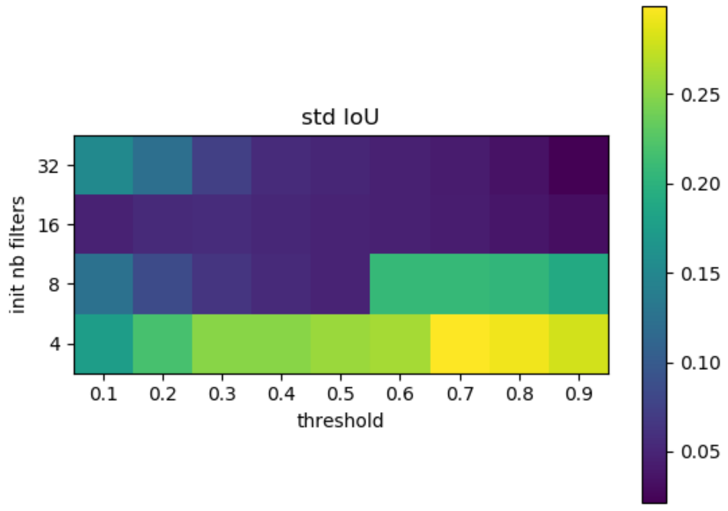

Figure 4.6: Standard deviation for the IoU on validation set depending on the initial number of filters n and the threshold t.

The Figure 4.5 illustrates one of the issue with the training method which is the variation of the IoU score obtained between models trained with exactly the same hyper-parameters and data. This **training instability** is a recurring situation in deep learning due the stochastic nature of the optimizer. However, while small variations can be acceptable, the variation of IoU observed in this project can be greater than 0.1 which is significant.

On Figure 4.6 it can be seen again that $n = 4$ performs worse than the others by having the bigger standard deviations for all thresholds. The model training with $n = 8$ seems reasonably stable for $t = 5$ but unstable beginning with $t = 6$. As for $n = 16$ and $n = 32$, the model training is reasonably stable for $t \geq 0.5$. However, even for those most stable configurations, the standard deviation is still around 0.05 which is too high to be considered truly stable. One should also recall that a *stable* configuration of (n, t) does not mean good accuracy. It means consistency of the results and therefore, the recommended configuration should be a trade-off between stability and accuracy.

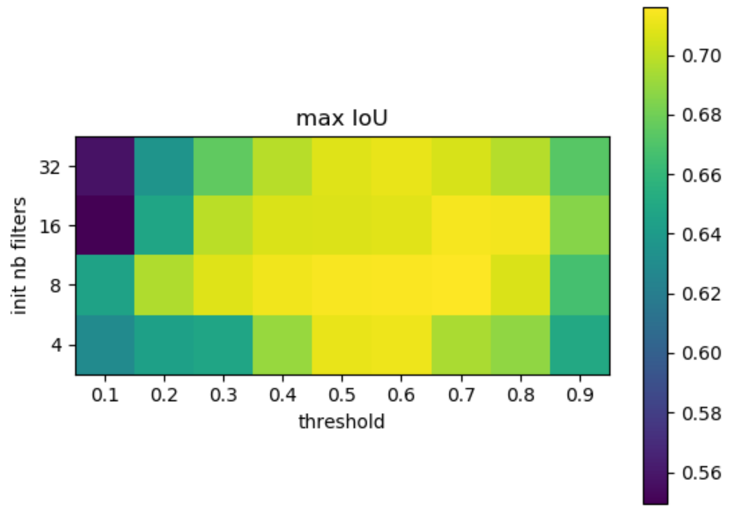

Figure 4.7: Max IoU obtained validation set depending on the initial number of filters n and the threshold t.

A last interesting observation is to look at the best IoU score obtained for each configuration. In this case, it is interesting to see that *good* models (IoU ≥ 0.7) can be obtained with **any** n and a threshold $0.5 \leq t \leq 0.8$.

Aside from $n = 4$ that can be safely put aside given its poor results with nearly every aspects. It is rather difficult to draw definitive conclusions based on those observations. The sample of models may not be large enough or other tests may be required to better distinguish the tendencies. Still, it seems that models trained with higher n associated with a threshold t slightly above 0.5 give more stable configurations with good mean IoU scores. $n = 16$ and $n = 32$ with a threshold of 0.6 or 0.7 can thus be recommended. Good models were also found with $n = 8$ and $t = 0.5$. However, this configuration does not seems stable with higher thresholds.

The Figures 4.8 and 4.9 illustrate segmentations made on the validation set using a model trained with $n = 16$ combined with $t = 0.6$. Some false

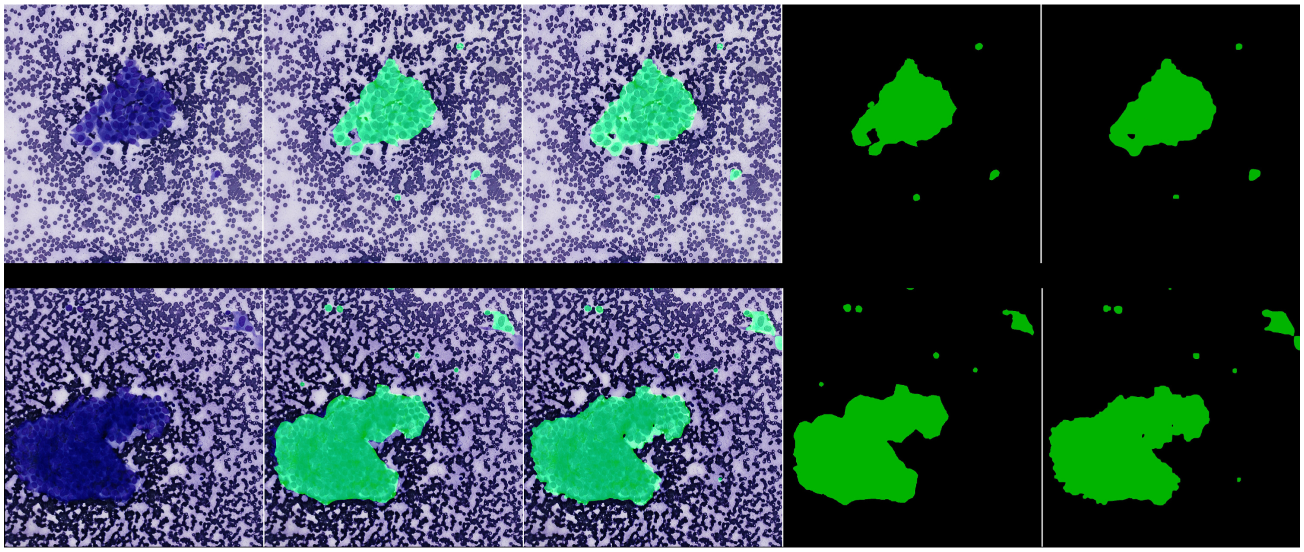

Figure 4.8: Segmentations on the validation set. From left to right : the crop to segment, crop with ground truth mask, crop with predicted mask, ground truth mask alone, predicted mask alone.

positives can be seen on the bottom crop of Figure 4.9 and the background generating it seems similar to the one in Figure 4.4. Those false positives were also observed in other crops no matter the configuration and it seems that they are one of the main cause of reducing the IoU score.

4.2.4 Performance

The goal of this section is to assess the computation performance of the model and the hardware used is the following :

- i7-4710MQ 3.5GHz
- NVIDIA GTX860M with 2GB VRAM
- 12GB RAM

This hardware is not comparable to the capabilities of a GPU cluster which could not be used due to lack of time. The goal of this performance

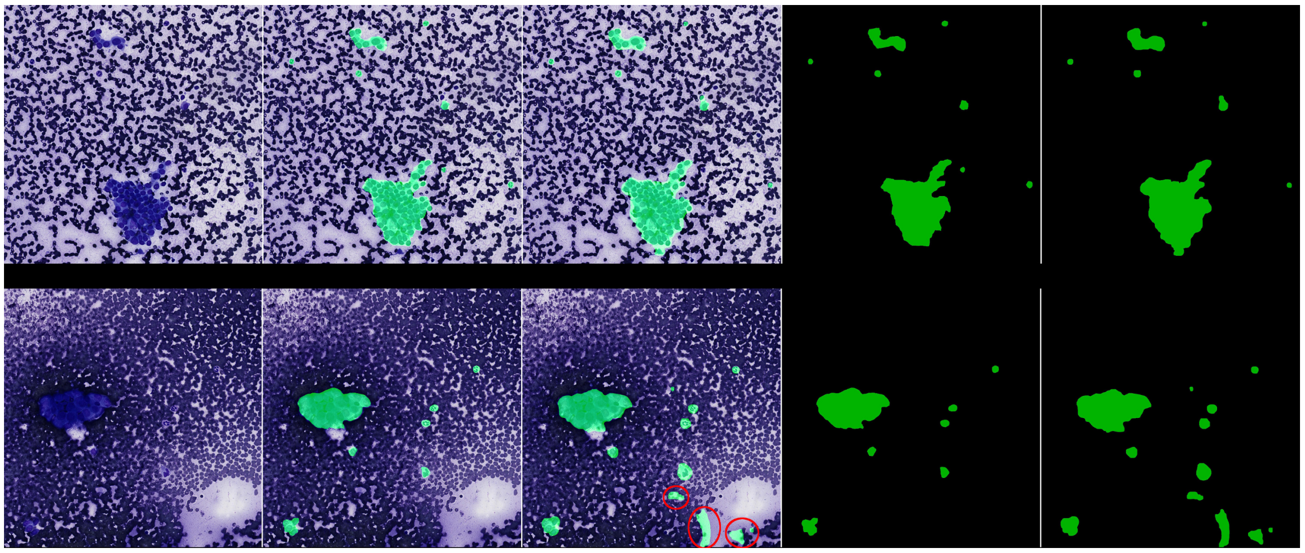

Figure 4.9: Segmentations on the validation set. From left to right : the crop to segment, crop with ground truth mask, crop with predicted mask, ground truth mask alone, predicted mask alone. Some false positive are highlighted in red on the bottom crop.

n	training time	# crops	crop size
8	2m09s	407	512×512
16	3m40s	407	512×512
32	7m51s	407	512×512

Table 4.5: Training times for different values of n which controls the complexity of the network.

n	segmentation time	# crops	crop size
8	0m29s	20	2048×2048
16	0m37s	20	2048×2048
32	0m59s	20	2048×2048

Table 4.6: Segmentation times for different values of n.

assessment is thus to again identify tendencies, as the potential gaps of performance between different configurations could fade away on the GPU cluster. Nevertheless, assessing the performance on modest hardware can potentially drive the model to a more efficient implementation and given the size of the whole-slide images, it can be useful to have a faster model.

Computation time The segmenter can be run and trained using the CPU or the GPU, though the latter is obviously preferred and used to assess the computation time. The different computation times of training and segmentation with respect to n are shown on Tables 4.5 and 4.6 (t does not affect the performance).

The training time is significantly smaller for $n = 8$ which is about 1.7 times faster than $n = 16$ and 3.7 times faster than $n = 32$. This is expected since n controls the complexity of the model and thus the number of parameters in the model to learn. The training time seems reasonable but will sharply increase with bigger datasets and/or models. It should be noted, however, that while a smaller training time can be useful for fast prototyping/testing, it shouldn't be a huge matter of importance for production models as they are generally not often retrained and the accuracy matters the most.

The segmentation time is again smaller for $n = 8$ which is about 1.3 times

n	RAM usage [GB]	VRAM usage [GB]	# crops	crop size
8	1.3	0.64	407	512×512
16	1.3	0.86	407	512×512
32	1.3	1.35	407	512×512

Table 4.7: Memory usage for training.

faster than $n = 16$ and 2 times faster than $n = 32$. A quick estimation can be made to try to determine the segmentation time of a whole-slide image with $n = 16$.

Let n_{wsi} be the total number of 512×512 tiles contained in a WSI, then using the dimensions from Figure 2.3 :

$$n_{wsi} = \frac{104704 \cdot 172032}{512 \cdot 512} = 68712$$

Let t_{wsi} be the WSI segmentation time :

$$t_{wsi} = \frac{37}{20 \cdot 16} \cdot n_{wsi} = 7944\text{s} \approx 2\text{h}12\text{min}$$

It should be noted that it is not considering the communication overhead with the Cytomine server, so the actual time should be bigger.

The segmentation time seems quite substantial with modest hardware but it can be expected to be *fast enough* using a cluster. In this case, the segmentation time is more relevant from a production perspective as it can help to reduce the diagnosis time.

Resource usage Attention was made to reduce the memory footprint so that it is possible to handle huge datasets though the crop size remains the bottleneck because of the limited GPU VRAM. The memory usage (RAM and VRAM) for different values of n is studied and the results for training and segmentation is reported on Table 4.7 and 4.8 respectively.

What can be seen from Table 4.7 is that the parameter n does not affect the RAM usage but the VRAM usage. This is due to the model being stored on the GPU memory while the RAM usage is caused by the crops being buffered before being passed to the model. The memory usage increases of

n	RAM usage [GB]	VRAM usage [GB]	# crops	crop size
8	2.4	0.5	20	2048×2048
16	2.4	0.6	20	2048×2048
32	2.4	0.77	20	2048×2048

Table 4.8: Memory usage for segmentation.

$\approx 34\%$ from $n = 8$ to $n = 16$ and of $\approx 57\%$ from $n = 8$ to $n = 16$. Considering an augmentation of $\approx 100\%$ between $n = 32$ to $n = 64$, a model with $n = 64$ could be easily trained on a GPU with 4GB of VRAM (not all VRAM is accessible to `PyTorch`). The memory usage seems very reasonable though and one can conclude that the model training can be easily scaled on a GPU cluster in terms of memory usage.

For segmentation, one can notice an increase of RAM usage which is due to the bigger size of the crops being buffered. The increase of VRAM usage from $n = 8$ to $n = 32$ is twice as small as for the training which means that the model can also be scaled for segmentation concerning memory usage.

4.2.5 Model selection

The selected configuration is $n = 16$ combined with $t = 0.6$. The reason is that $n = 16$ seems to be a good compromise between accuracy (mean IoU), training stability (std IoU), segmentation speed and resource usage. It is as accurate and stable as $n = 32$ while performing segmentation faster. $n = 8$ was not chosen as the models were shown to be unstable with $t \geq 0.6$ but this threshold can be useful to attenuate false positives.
In *real* conditions, the recommendation would thus be to train models on validation set and select the one with the best IoU score with the additional requirement that the IoU should be ≥ 0.7 knowing that the best models that can be produced are around IoU ≈ 0.7 (Figure 4.7).

4.2.6 Assessment

To assess the model selection, 30 models were trained using the recommended configuration and the best ones were selected on validation set (IoU ≥ 0.7) then they were assessed on test set. The results on Table 4.9 report the IoU

set	mean IoU	std IoU	# crops	crop size
validation	0.7103	0.006	20	2048×2048
test	0.6604	0.0135	25	2048×2048

Table 4.9: Results on test set compared with validation set using model selection.

scores on both validation and test sets using the selected models.

The results obtained on the test set are lower than those obtained on validation by ≈ 0.05. This is a significant difference but after inspection of the segmentations (see Figure 4.10), it seems that once again the IoU scores are driven down by false positives obtained on some types of background (including 4.4). On other types of crops however, the model seems to perform well, as shown on Figure 4.11.

Regarding the stability of the produced models, the standard deviation is reasonably low on test set. Indeed, it is well under the 0.05 originally obtained by training models with $n = 16$ (Figure 4.5) which may imply that the selected models from the validation set indeed share common characteristics and thus they have a similar **generalization**.

To conclude the assessment, the selection method produces models that are consistent and perform reasonably well but a problem of false positive on some types of background is one the main factor affecting the IoU scores.

4.3 Improvements

This section summarizes the various improvements that can be built upon this work to address the main issues with the proposed solution.

Improve model training and accuracy One of the main problem of the model is the generation of false positives which affect IoU score and the non-negligible variability of those models produced by the training is also an issue. The following solutions can be envisioned by order of priority :

1. The first solution to explore would be to improve the training set by adding more complete and diverse annotations, especially the ones con-

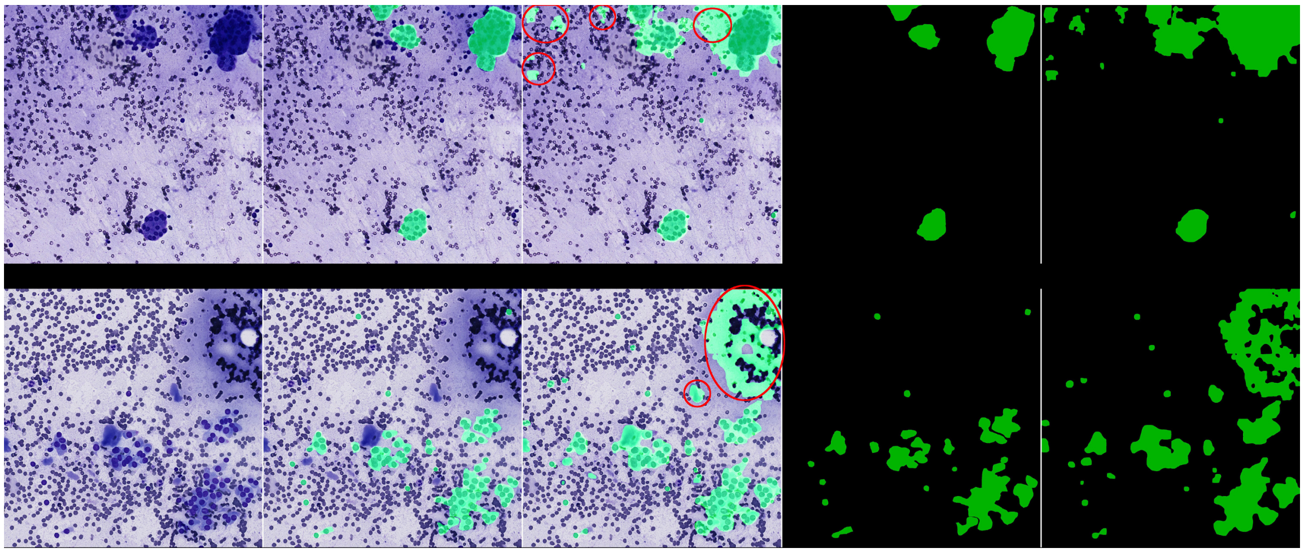

Figure 4.10: Segmentations on the test set. From left to right : the crop to segment, crop with ground truth mask, crop with predicted mask, ground truth mask alone, predicted mask alone. Examples of false positives are highlighted in red.

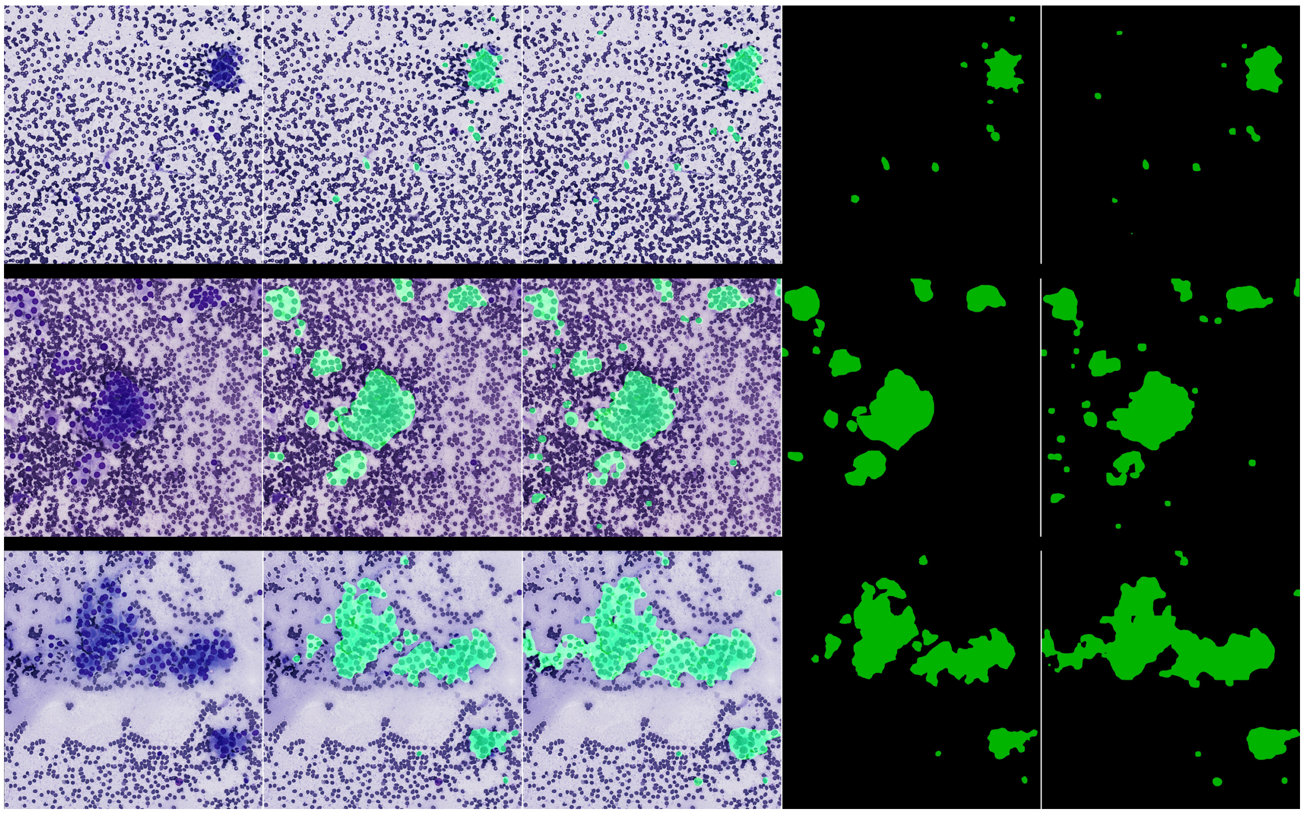

Figure 4.11: Segmentations on the test set. From left to right : the crop to segment, crop with ground truth mask, crop with predicted mask, ground truth mask alone, predicted mask alone.

taining the problematic background. This solution is linked to the need of improvement regarding the data selection process to build the training set.

2. Improve the proposed data improvement and data augmentation methods.

3. Tune/change the optimizer.

4. Try another network architecture or search for improvement of the U-Net.

Theses solutions can, of course, be combined.

Data selection process The main problem with the selection of data is that it is a manual process and hard to automate because a way of automatically selecting *good* data would assume a model able to detect the cells and patterns correctly. Nevertheless, there could be ways of accelerating/enhancing the data selection process by, for instance, using **active learning**.

Increase validation and test set sizes The gap of the results between the validation and test set showed that models performing reasonably well on validation set can be subjected to greater inaccuracies on test set. Increasing the size of validation and test sets can help getting more relevant model selection ans results. The problem being that it can be an extensive task and should be delegated to experts.

Integration Even though a partial integration in Cytomine was achieved, the full integration is held by some technical problems that need to be addressed.

Post-processing on GPU While the computation of the predicted masks is done on the GPU, the post-processing is currently implemented on the CPU, which forms a bottleneck. Smaller segmentation time could be achieved by moving the post-processing onto the GPU.

Chapter 5

Conclusion

This thesis addressed a digital pathology problem of cell segmentation in whole-slide cytological images with the end goal of contributing to an automated diagnosis system of the thyroid cancer. Among the challenges was the presence of incomplete/weak annotations in the dataset which made the training of a supervised learning algorithm harder.

The implemented solution uses a U-Net network in conjunction with noise reduction post-processing for the segmentation. An iterative data improvement method has also been proposed and managed to partly deal with the weak annotations, though it couldn't with all available data. This led to the need of building a training set by selecting the data from the whole set of data. The implementation has also been made modular so that the developed segmenter can be easily integrated in another image processing framework and the Cytomine instance of the ULiège. The integration in the latter being almost complete.

The results are promising and show that the U-Net in conjunction with data improvement can produce satisfying segmentations though inaccuracies due to false positives remain. The solution should also be easily scaled on a GPU cluster for faster segmentations thanks to its reasonable resource usage.

One main recommendation for further work would be to address the building of the training set by, for instance, using active learning and paying attention to the diversity and completeness of the data.

Bibliography

[1] Cytomine. `https://cytomine.be`.

[2] Overfitting and underfitting graph. `https://github.com/d2l-ai/d2l-en/blob/master/img/capacity_vs_error.svg`.

[3] Rasterio. `https://rasterio.readthedocs.io`.

[4] D. Baltissen, T. Wollmann, M. Gunkel, I. Chung, H. Erfle, K. Rippe, and K. Rohr. Comparison of segmentation methods for tissue microscopy images of glioblastoma cells, 2018.

[5] Juan C. Caicedo, Jonathan Roth, Allen Goodman, Tim Becker, Kyle W. Karhohs, Matthieu Broisin, Csaba Molnar, Claire McQuin, Shantanu Singh, Fabian J. Theis, and Anne E. Carpenter. Evaluation of deep learning strategies for nucleus segmentation in fluorescence images, 2019.

[6] David Dov, Shahar Ziv Kovalsky, Jonathan Cohen, Danielle Elliott Range, Ricardo Henao, and Lawrence Carin. A deep-learning algorithm for thyroid malignancy prediction from whole slide cytopathology images, 2019.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[9] Chaitanya Kolluru, Beth A. Benetz, Naomi Joseph, Harry J. Menegay, Jonathan H.Lass, and David Wilson. Machine learning for segmenting cells in corneal endothelium images, 2019.

[10] Marvin Lerousseau, Maria Vakalopoulou, Marion Classe, Julien Adam, Enzo Battistella, Alexandre Carré, Théo Estienne, Thé ophraste Henry, Eric Deutsch, and Nikos Paragios. Weakly supervised multiple instance learning histopathological tumor segmentation, 2020.

[11] Qiaokang Liang, Yang Nan, Gianmarc Coppola, Kunglin Zou, Wei Sun, Dan Zhang, and Yaonan Wang Guanzhen Yu. Weakly supervised biomedical image segmentation by reiterative learning, 2018.

[12] McKinney, S.M., Sieniek, M., Godbole, and V. *et al.* International evaluation of an ai system for breast cancer screening, 2020.

[13] Romain Mormont, Pierre Geurts, and Raphaël Marée. Comparison of deep transfer learning strategies for digital pathology, 2018.

[14] Romain Mormont, Pierre Geurts, and Raphaël Marée. Multi-task pretraining of deep neural networks fordigital pathology, 2020.

[15] Mormont Romain. A workflow for large scale computer-aided cytology and its applications. `https://matheo.uliege.be/handle/2268.2/1314`, 2016.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[17] Adrian Rosebrock. A visual equation for intersection over union (jaccard index). `http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`, 2016. Creative Commons Attribution-Share Alike 4.0 International license.

[18] Uwe Schmidt, Martin Weigert, Coleman Broaddus, and Gene Myers. Cell detection with star-convex polygons. In *Medical Image Computing and Computer Assisted Intervention - MICCAI 2018 - 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part II*, pages 265–273, 2018.

[19] David A. Van Valen, Takamasa Kudo, Keara M. Lane, Derek N. Macklin, Nicolas T. Quach, Mialy M. DeFelice, Inbal Maayan, Yu Tanouchi, Euan A. Ashley, and Markus W. Covert. Deep learning automates the quantitative analysis of individual cells in live-cell imaging experiments, 2016.

[20] Martin Weigert, Uwe Schmidt, Robert Haase, Ko Sugawara, and Gene Myers. Star-convex polyhedra for 3d object detection and segmentation in microscopy. In *The IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2020.