Master thesis : Development of cell counting algorithms within pathological tissues

Auteur : Rubens, Ulysse
Promoteur(s) : Geurts, Pierre; Maree, Raphael
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "computer systems and networks"
Année académique : 2016-2017
URI/URL : http://hdl.handle.net/2268.2/3237

University of Liège
Faculty of Applied Sciences

Master Thesis

Development of cell counting algorithms within pathological tissues

Author : Ulysse Rubens

Advisor : Prof. Pierre Geurts

Supervisor : Dr. Raphaël Marée

Graduation Studies conducted for obtaining
the Master's degree in Computer science and Engineering

Academic year 2016-2017

Abstract

Development of cell counting algorithms within pathological tissues.

Author: Ulysse Rubens
Supervisor: Dr. Raphaël Marée, *Advisor*: Prof. Pierre Geurts
Academic year 2016-2017

Counting objects in images is within the reach of everyone. When the number of objects to count becomes large, the task is more difficult and we tend to make errors because of some oversights. For a computer, counting or detecting objects in images is hard. However, the repetitive characteristic of this task does not worry it.

In histopathology, the study of pathology combined to histology (the study of cells and tissues), scientists frequently have to count cells from images of tissues to address hypotheses about pathological or developmental processes. The rise of digital imaging services and the power of computers and networks led to the emergence of digital histopathology, a new field where histopathologists analyze cells and tissues on computer screens. Therefore, it would be interesting to have automatic counting methods available to lighten the work of life scientists.

It this thesis, the problem of cell counting and the closely related problem of cell detection are studied. Supervised machine learning workflows are proposed in that purpose. Five methods are presented, (re-)implemented and tried on four databases of histopathological images, whose two are newly introduced for this kind of problem. We set up the software modules allowing to use these algorithms directly on databases from Cytomine, a web-based application for collaborative analysis of multi-gigapixel images.

Basically, two ways of counting are studied. Firstly, counting can be done by cell detection, the counting trivially being the number of detections. Secondly, counting can be estimated by density estimation, knowing the density for each image's pixel.

An important aspect of this work is the assessment and the comparison of these approaches. Performance results in machine learning literature are sometimes too optimistic due to weak model validation. Robust metrics are thus defined in order to make comparisons possible. Each machine learning algorithm has a set of parameters that can be tuned. They have a large influence on the performance. In order to choose the best combination of parameters, a systematic model selection protocol is employed. It is the first time that such a set of methods has been evaluated on several datasets of such a large size. Our study has helped to better understand the functioning of these methods and to establish their limits.

Results are encouraging. In general, at least one algorithm is able to reach a counting error less than 10% for each dataset.

Acknowledgement

First, I would like to thanks Dr. Raphaël Marée who followed me closely and helped me to realize this work thanks to his precious advice that led to the final result of this thesis.

Then, I would like to thank to my academic advisor, Prof. Pierre Geurts, who has contributed among others to my computer science engineering education and has believed in me to confide to me this subject while I had no prior knowledge in image processing and machine learning.

I would also like to thank the Cytomine team for its availability, but also the administrators of the GIGA supercomputer and Prof. Marc Van Droogenbroeck and his team for the availability of the Nvidia DGX-1 supercomputer.

Finally, I would like to express my special thanks to my parents, and especially my girlfriend, Florine, who have provided me an unwavering support during my five years of studies, and particularly during the making of this master thesis.

Contents

1 Introduction **1**

2 Object counting for computer-aided histology **3**
2.1 Task overview . . . 3
2.2 Applications . . . 5
2.3 Cytomine . . . 6
2.4 Datasets . . . 7
2.4.1 BMGRAZ dataset . . . 8
2.4.2 CRC dataset . . . 9
2.4.3 GANGLIONS dataset . . . 10
2.4.4 ANAPATH dataset . . . 11
2.4.5 Summary . . . 13

3 Survey of existing approaches **14**
3.1 Supervised learning . . . 14
3.1.1 Randomized trees . . . 15
3.1.2 Convolutional neural networks . . . 17
3.1.2.1 Convolution . . . 18
3.1.2.2 Spatial pooling . . . 19
3.1.2.3 Non linearity . . . 19
3.1.2.4 Fully connected layers . . . 20
3.2 Problem formulation . . . 20
3.2.1 Counting by detection . . . 21
3.2.2 Counting by density estimation . . . 22
3.3 Subwindow-based samples . . . 22
3.3.1 Subwindow extraction strategy . . . 23
3.3.1.1 Random strategy . . . 23
3.3.1.2 Exhaustive strategy . . . 23
3.3.1.3 Scoremap constrained way . . . 24
3.3.2 Feature extraction . . . 25
3.4 Randomized trees-based methods . . . 27
3.4.1 Standard Classification (SRTC) . . . 28
3.4.1.1 Training stage . . . 28
3.4.1.2 Prediction stage . . . 29
3.4.2 Regression by proximity score estimation (PRTR) . . . 30
3.4.2.1 Training stage . . . 31

3.4.2.2 Prediction stage . . . 32
3.4.3 Regression by density estimation (DRTR) . . . 34
3.4.3.1 Training stage . . . 34
3.4.3.2 Prediction stage . . . 34
3.5 Convolutional neural networks-based methods . . . 35
3.5.1 Regression by density estimation (FCRN) . . . 35
3.5.1.1 Training stage . . . 36
3.5.1.2 Prediction stage . . . 37
3.5.2 Regression by proximity score estimation (SC-CNN) . . . 38
3.5.2.1 Training stage . . . 40
3.5.2.2 Prediction stage . . . 40

4 Experiments and comparison **42**
4.1 Evaluation methodology . . . 42
4.1.1 Model selection and assessment . . . 42
4.1.1.1 Model assessment and test set . . . 42
4.1.1.2 Model selection and cross validation . . . 43
4.1.1.3 Full procedure . . . 43
4.1.2 Metrics . . . 44
4.1.2.1 Optimization criteria . . . 46
4.2 Design of experiments . . . 47
4.2.1 Pre-processing . . . 48
4.2.2 Randomized trees . . . 48
4.2.3 Convolutional neural networks . . . 50
4.2.4 Post-processing . . . 51
4.3 Tests and results . . . 51
4.3.1 BMGRAZ dataset . . . 53
4.3.1.1 Experiments with SRTC . . . 53
4.3.1.2 Experiments with PRTR . . . 54
4.3.1.3 Experiments with DRTR . . . 56
4.3.1.4 Experiments with FCRN-A . . . 57
4.3.1.5 Experiments with FCRN-B . . . 58
4.3.1.6 Comparison . . . 58
4.3.2 GANGLIONS dataset . . . 60
4.3.3 ANAPATH dataset . . . 62
4.3.4 CRC dataset . . . 63
4.4 Overall comparison . . . 65
4.5 Implementation . . . 66
4.5.1 Language and libraries . . . 66
4.5.2 Other details . . . 67

5 Conclusion and perspectives **69**

A Detailed results for GANGLIONS dataset **71**
A.1 Experiments with SRTC . . . 71
A.2 Experiments with PRTR . . . 72
A.3 Experiments with DRTR . . . 72

A.4 Experiments with FCRN-A 73
A.5 Experiments with FCRN-B 74

B Detailed results for ANAPATH dataset **75**
B.1 Experiments with SRTC 75
B.2 Experiments with PRTR 76
B.3 Experiments with FCRN-A 76
B.4 Experiments with FCRN-B 77

List of Tables **78**

List of Figures **79**

List of Algorithms **81**

References **82**

Chapter 1

Introduction

Counting objects in images is within the reach of everyone. When the number of objects to count becomes large, the task is more difficult and we tend to make errors because of some oversights. For a computer, counting or detecting objects in images is hard. However, the repetitive characteristic of this task does not worry it.

In histopathology, the study of pathology combined to histology, the study of cells and tissues, histopathologists frequently have to count cells from images of tissues to investigate hypotheses about developmental or pathological processes [XNZ15]. This tedious and repetitive task they have to carry out is a waste of time whose they do well without. Very often, a small region is analyzed and counts are inferred for the whole tissue, with little certainty about the representativeness of the statistical sample. Because of the risk yielded by the previous method and because manual analysis of full images is long and tedious, computer programs could be used to assist experts. Nowadays, the usage of digital scanners and the power of computers and networks lead to the emergence of digital histopathology, a new field where histopathologists work more on their computer screen instead of their microscope.

It the framework of this thesis, the problem of cell counting and the closely related problem of cell detection have been studied. The goal is to implement computer programs able to successfully carry out these tasks, by learning which is a cell on tissue images and which is not, using a database of examples established by experts. Image processing and supervised machine learning are used in this purpose.

For the past ten years, several methods of cell - and more generally object - detection and/or counting have been proposed in the scientific literature, with varying degrees of success. These approaches can be classified into two categories in terms of machine learning: some methods employ standard machine learning algorithms such as randomized trees while others belong to the trending field of deep learning. We will analyze the theoretical aspects of these methods but also the practical aspects from a end-user point of view. As far as machine learning algorithms are involved, the methods have to be assessed and compared. The quality of the mentioned

approaches is hard to compare as the authors usually use different datasets[1] with varying cell types and different quality metrics. We will perform a systematic evaluation and comparison of different methods. Moreover, a recurring issue is the lack of availability of datasets. This work will compare the studied methods among four datasets, whose two are newly introduced for the cell counting/detection problem. A second issue is to find the best combination of method's parameters to improve the performance. It is not always possible due to the long training time of machine learning approaches, especially for deep learning. The usage of two supercomputers, whose one is tailored for deep learning algorithms made easier the study related to the influence of these parameters.

In Chapter 2, the problem of cell counting and detection is more deeply introduced and examined in a end-user perspective. In Chapter 3, we perform a survey of available approaches and develop related algorithms. In Chapter 4, an evaluation protocol is designed and a set of experiments are conducted to assess and compare these methods on different datasets. Finally, we conclude in Chapter 5.

[1]In our case, a dataset is a set of microscopy histological images, annotated by experts to indicate presence of cells.

Chapter 2

Object counting for computer-aided histology

Object counting is a common task in histology and histopathology where objects are cells. In Section 2.1, we define these terms and explain why the problem of cell counting is of interest. A bunch of applications is given in Section 2.2. In Section 2.3, we present the Cytomine application for which the algorithms that we will study have been implemented. Finally, the datasets that we will use for the evaluation and comparison of the different approaches are described in Section 2.4.

2.1 Task overview

Every organism is made up of a cell or a collection of cells. A cell is defined as the smallest structural and functional unit of an organism. A tissue is a group of cells that perform a common function and have a similar origin. *Histology* is the study of the structure and the function of microscopic anatomy of human beings which includes the cells, tissues, organs (a collection of tissues) and organ system (a collection of organs) of an organism. These studies are performed especially by the examination of cells and tissues. This field of life science is of great interest in biology and medicine. We can also highlight *histopathology*, the microscopic study of diseased tissues, which is an significant tool in anatomical pathology. For example, an histopathologist can provide accurate diagnosis of cancer and other diseases by histopathological examination.

In practice, histology is typically performed by examining cells and tissues under a light microscope or an electron microscope. In that purpose, tissues have first to be sectioned, stained, and mounted on a microscope slide. The staining operation enhances the ability to visualize or identify microscopic structures. One of the most used stain in histology and histopathology is the Hematoxylin and Eosin one (H&E stain).

Among the tasks of histopathologists, a common one is the count of cells in a (possibly pathological) tissue. There exist several techniques devoted to cell counting and/or localization, with varying degrees of sophistication:

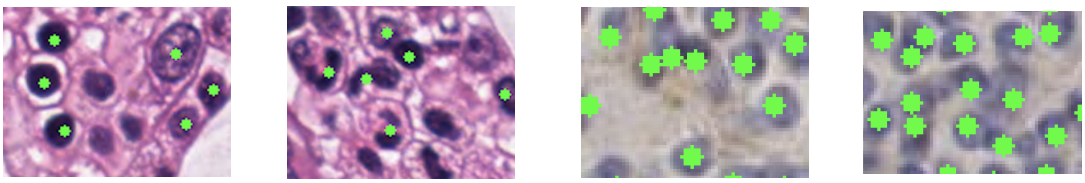
1. **Manual cell counting**: the simplest technique where the experimenter use a counting chamber, *i.e.* a microscope slide with a grid especially devoted to cell counting. This approach suffers from several drawbacks. Firstly, it is a repetitive and tedious task, which increases the risk of making mistakes. Secondly, the experiment is very subjective due to a considerable inter-observer variability (two different observers can count differently) and is possibly not reproducible because of intra-observer variability (a same observer can count differently at different times) [Kai+15], [Kos+17].

2. **Automated cell counting with dedicated machines**: these devices, such as flow cytometers, do not suffer from the issue of manual counting but are often very expensive and they also do not preserve tissue structure.

3. **Automated cell counting by image analysis**: the technique by which we are interested in this thesis. By combining the power of computers and the major breakthroughs in image processing and machine learning, new algorithms can see the day. They fall into the category of computer-aided histology, where histological slides are digitized before to analyze them on a computer screen.

This last kind of technique inherits from all benefits of computer-aided histology such as scientific research speed-up and ease of result sharing with softwares such as Cytomine, an application for collaborative analysis of multi-gigapixel images (see Section 2.3).

Object counting with machine learning is a challenging task, especially when objects to count are cells. Indeed, we have to deal with cells that can take various shapes and sizes (Figure 2.1). Moreover, cell clumping is very frequent. In some methods, non uniform stain on objects of interest can be an issue. Finally, in some applications, visually similar structures and different cell types can occur on images and the developed algorithms have to only detect cells of interest [Art+12].



(a) (b) (c) (d)

Figure 2.1: Variety of cells with visually similar structures that are not cells (a), with various shapes (b) and cell clumping (c) & (d). Green dots are ground truth cells whose size has been exaggerated.

In the supervised approach of machine learning that we will follow in this work, a set of correctly labelled images indicating where are cells is required. This procedure is usually done manually and thus implies the same issues as manual cell counting technique presented before. However, this procedure

1. has to be done only once to train the algorithm before to use it on new and unlabelled images

2. can be done through the Cytomine interface [Mar+16], behind a computer screen, and several experts can independently label images before to review these annotations in order to keep only the perfectly correct ones, in order to minimize inter-observer variability.

3. is done only in some regions of interest extracted from the whole slide. Figure 2.2 shows a slide with an enlarged version of one of its regions of interest.

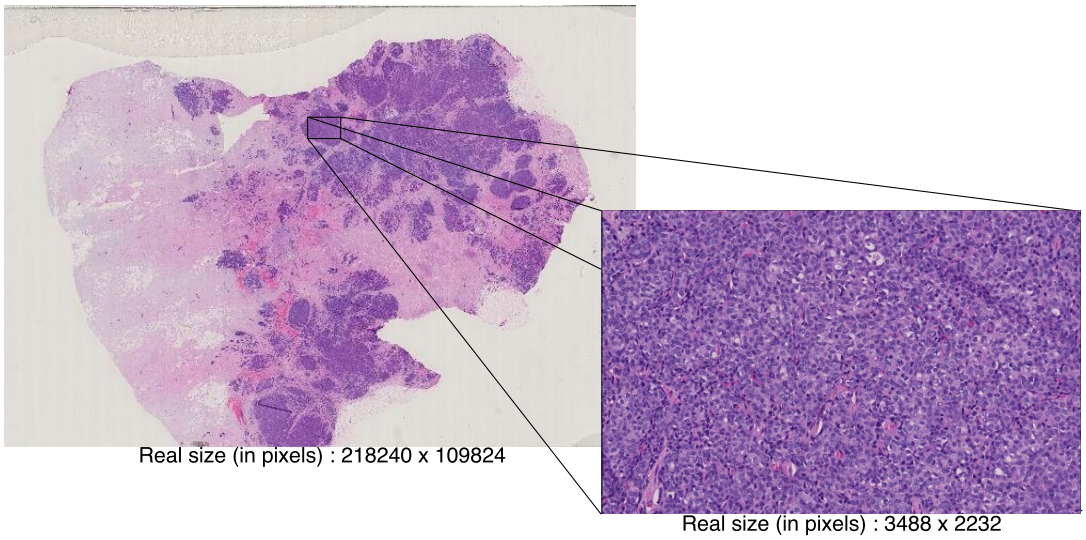


Figure 2.2: A slide from ANAPATH dataset (24 gigapixels) and one region of interest (7.8 megapixels)

It remains to choose how to label a cell on an image. According to [Kos+17], we opted for a simple annotation system. Each cell is marked with a dot in its center (or on the nucleus center, the important thing is to be coherent on a same dataset). Beyond the fact of being the natural way of counting objects, the task is by far less tedious and time-consuming than a segmentation of all cells in a region of interest, which is traditionally used in only image-processing-based approaches. Information provided by dot markers is lower than by segmentation but allow to label larger regions with the same effort. An other but related problem is to produce segmentation for cells from dot-marked images, notably tackled in [Art+12].

2.2 Applications

Cell counting and/or localization is of great interest in numerous applications in life sciences ([Kos+17], [XNZ15]):

- In biology and medicine, the complete blood count is a technique to infer a patient's health from the number of red and white blood cells.

- In molecular biology, the cell concentration can be used to adjust the amount of chemicals to apply in an experiment.
- In microbiology, cell counting is employed to quantify the growth and the behavior of microorganisms such as viruses and bacteria, by counting at constant time intervals.
- In histopathology, cell counts can be used to investigate hypotheses about pathological processes. Also, it can be employed to determine the rate at which the immune system of a patient is dealing with an infection.
- In cancer diagnosis, the quantification of cells having a particular protein in their nucleus helps to determine the proliferation rate of a tumor.

The algorithms that we will exploit have also experienced promising results in other domains such as in surveillance systems of crowd, in product inspection for industrial companies, in wildlife census or in inventories of trees in an aerial photo of a forest [LZ10], [Fia+12].

2.3 Cytomine

Cytomine [Mar+16] is a web-based application for collaborative analysis of multi-gigapixel images. This platform has been designed to promote multidisciplinary collaboration between life scientists and computer scientists. Also, Cytomine aims to accelerate scientific progress and foster image data accessibility and reusability.

It allows experts to navigate through large images as they would do in map applications like Google Maps as shown in Figure 2.3. An annotation system allows users to bring out regions of interest and associate them with some properties.

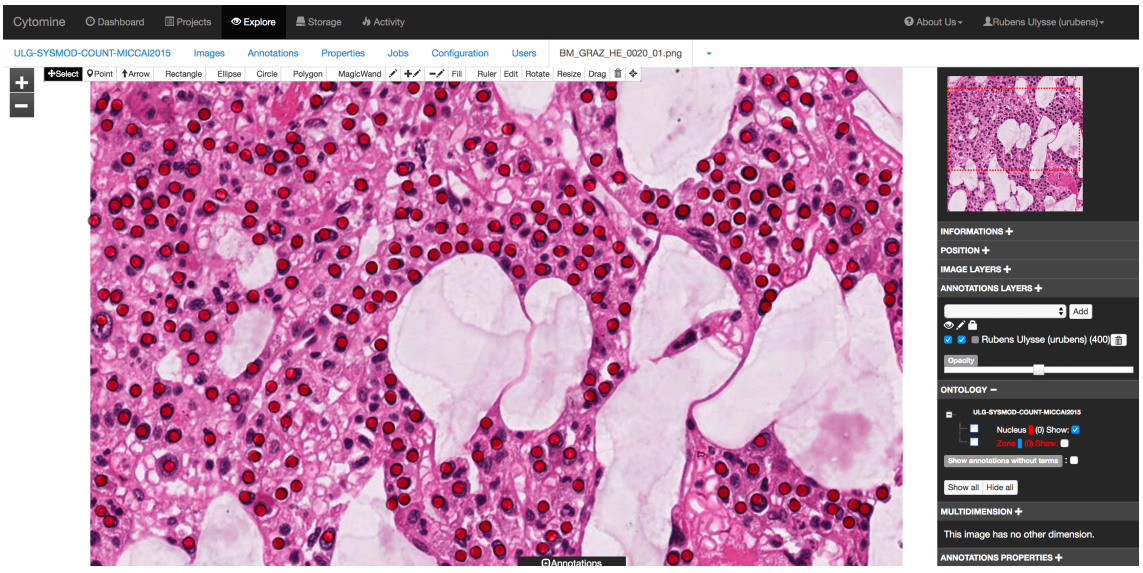


Figure 2.3: Cytomine image analysis module

The web-based characteristic of the platform allows distant experts to exchange annotations, directly from their web browser. In our case, after the digitization

of slides and their upload on Cytomine, histopathologists can annotate regions of interests and mark cells with the dot annotation tool. It constitutes ground truth data for our machine learning algorithms. An other component of Cytomine is its reviewing system which permit to proofread automatically generated annotations, that could be used to improve the performance of the implemented algorithms.

The *software* module (Figure 2.4) allows to link our algorithms to the platform. By this way, results can be directly observed by the scientist on the platform. New executions of softwares can be directly launched from the interface. The application gives thus all that is required from a end-user point of view so that we can focus our work on the development of new methods.

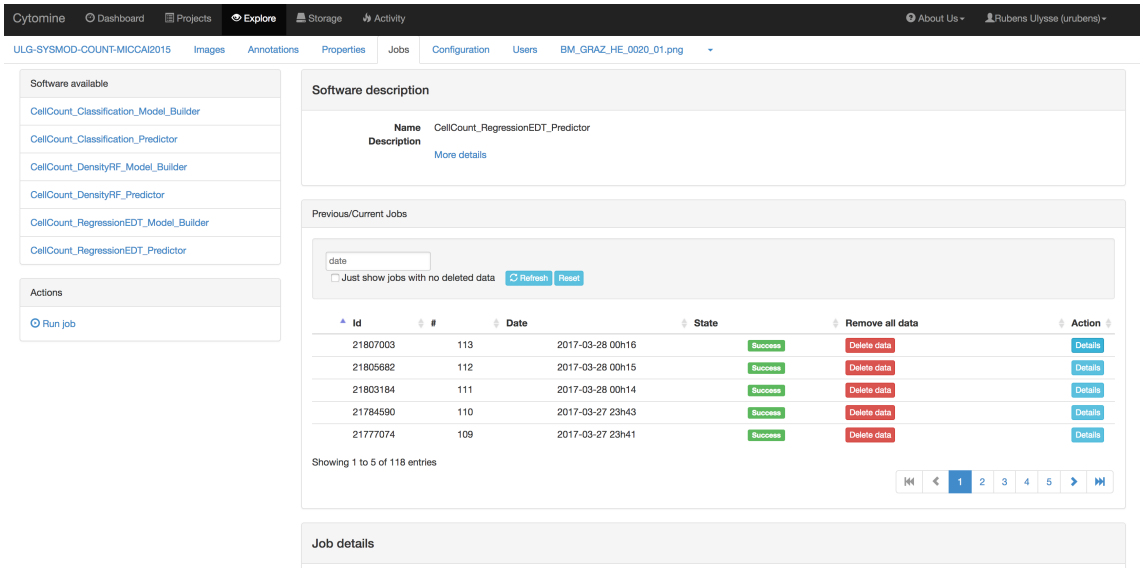


Figure 2.4: Cytomine software module

2.4 Datasets

Four datasets are used to evaluate the methods that we will study through this work. The source of these is diverse:

- Two datasets (BMGRAZ and CRC) have already been employed respectively by [Kai+15] and [Sir+16] to assess their respective methods.
- Two other datasets (GANGLIONS and ANAPATH) are introduced in the context of this thesis for the problem of cell counting. They comes from projects available on the Cytomine platform where labelled annotations have been marked by biomedical researchers from the GIGA research center at the University of Liège[1].

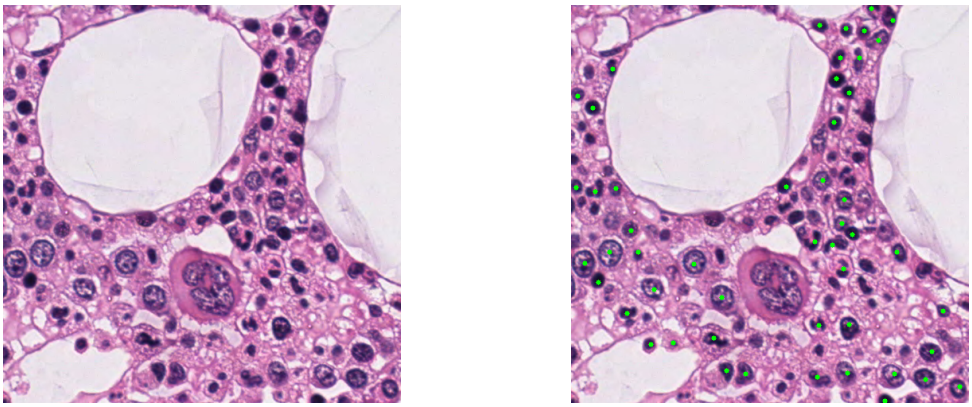
Each dataset consists in a set of microscopy digitized *whole-slide images* of human tissues. We will refer those whole-slide images as *slides* for the remainder of

[1] http://www.giga.ulg.ac.be/

this text. Each slide contains several *regions of interest (ROI)*. They are slide's image crops where cell centers have been annotated with dots by experts and that will be denoted as *images*.

2.4.1 BMGRAZ dataset

The BMGRAZ dataset has been introduced by [Kai+15]. It consists of 11 1200×1200 pixels regions of interest of healthy human bone marrow from 8 different patients where tissues had been stained with Hematoxylin and Eosin. Affiliations between regions of interest and slides are unknown. We thus consider each 1200×1200 image as a slide from which a single region of interest of the same size has been extracted. Figure 2.5 shows a crop of an image from this dataset as an example.



(a) without dot annotations (b) with dot annotations

Figure 2.5: A sample of a BM GRAZ dataset region of interest.

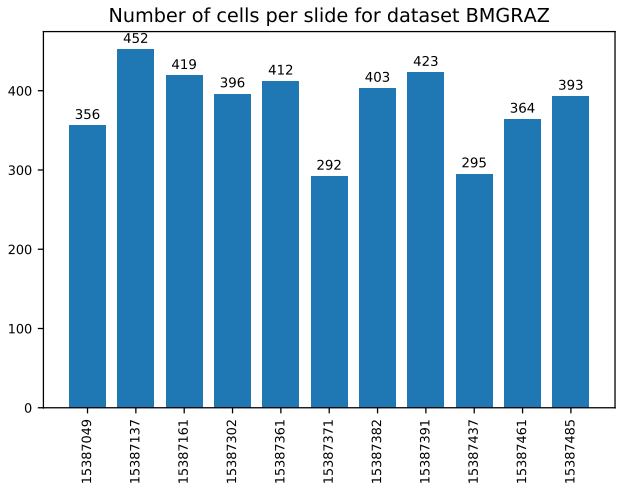


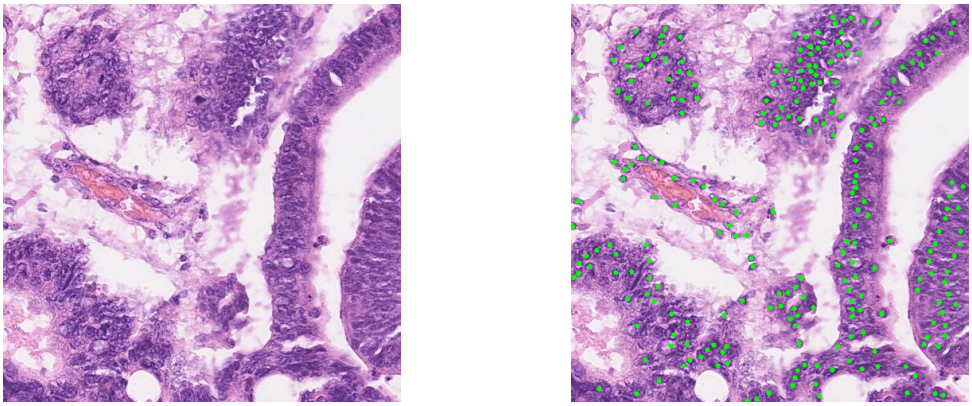
Figure 2.6: Number of cells per slide in dataset BM GRAZ.

A total of 4205 dot annotations corresponding to cell centers were marked, with

382 ± 49 dots per region of interest on average, which can be deduced from Figure 2.6.

2.4.2 CRC dataset

The CRC dataset has been introduced by [Sir+16]. It consists of 100 500×500 pixels regions of interest of colorectal adenocarcinomas that were cropped from non-overlapping areas of 10 whole-slide images at a pixel resolution of $0.55\mu m$/pixel (magnitude 20x), coming from 9 different patients. However, affiliations between regions of interest and whole-slide images are no longer provided by [Sir+16]. Regions of interest are thus considered as coming from all different slides of 500×500. Tissue had been stained with Hematoxylin and Eosin. An example of region of interest is given in Figure 2.7.



(a) without dot annotations (b) with annotations

Figure 2.7: A CRC dataset region of interest.

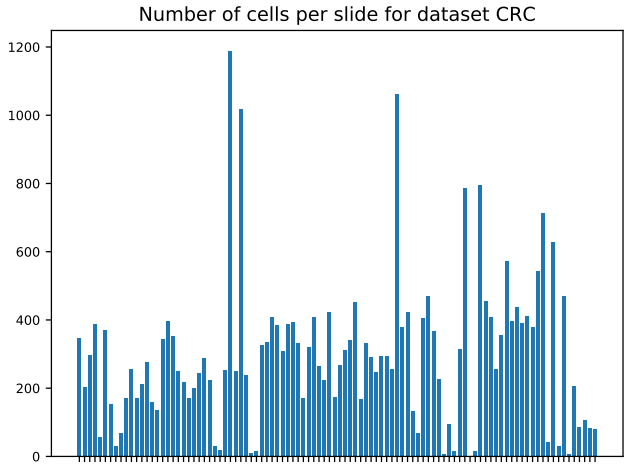
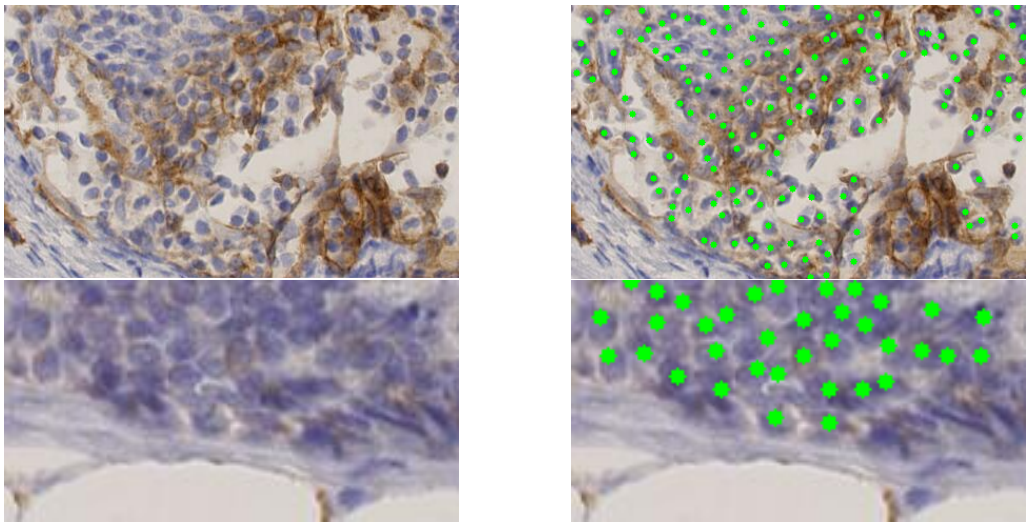


Figure 2.8: Number of cells per slide in dataset CRC.

A total of 29 756 dot annotations corresponding to cell nuclei center are provided with an average of 297 nuclei per ROI with a rather high standard deviation of 217, as presented in Figure 2.8.

2.4.3 GANGLIONS dataset

This dataset is composed of 66 rectangular regions of interest extracted from 11 whole slides, scanned at a pixel resolution of $0.455\mu m$/pixel (magnitude 20x). Tissues on these slides come from sentinel lymph nodes of patients with early stage of cervix cancer and have been stained by immunohistochemistry (IHC). The extraction of regions of interest has been made in such a way that the number of region per slide is more or less balanced as seen in Figure 2.10. Those regions of interest contain 6396 dot annotations depicting cell centers that have been performed by Cédric Balsat[2] on the Cytomine platform. There are 581 ± 177 cells per whole slide (Figure 2.11) and 97 ± 61 per ROI on average. Examples of region of interest are given in Figure 2.9 illustrating the variety of cell appearance in this dataset.



(a) without dot annotations (b) with dot annotations

Figure 2.9: Examples of GANGLIONS dataset region of interest (at different scales).

[2]LBTD laboratory (biologie des tumeurs et du développement), GIGA-Cancer, University of Liège

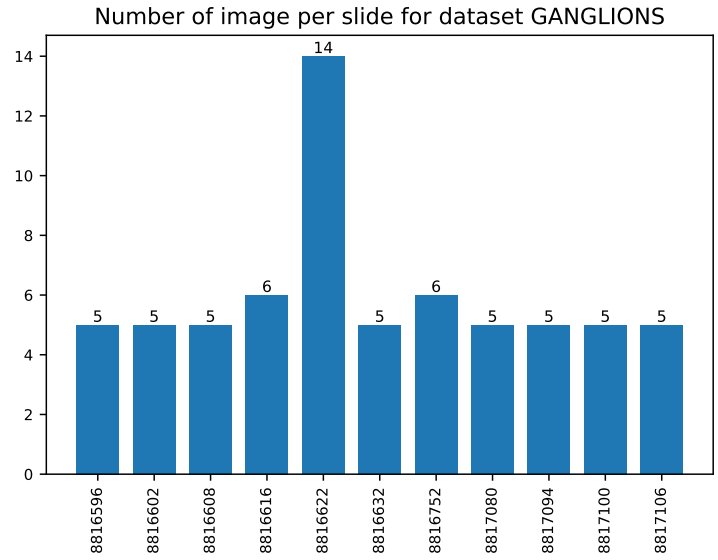


Figure 2.10: Number of ROI per slide in dataset GANGLIONS

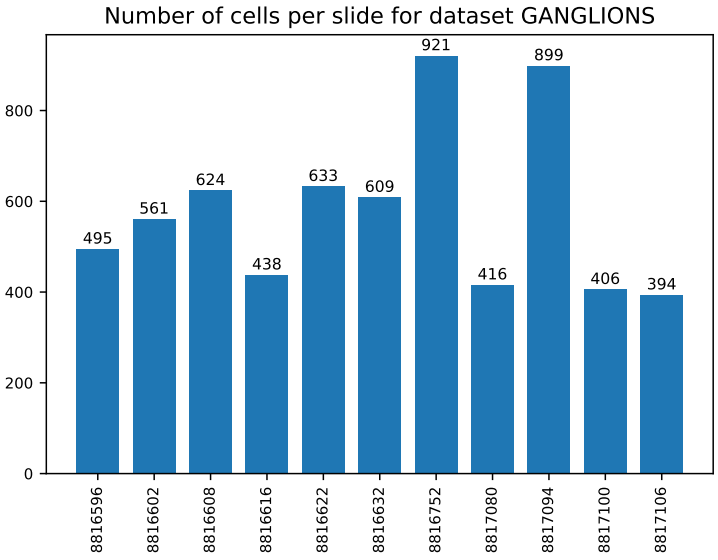
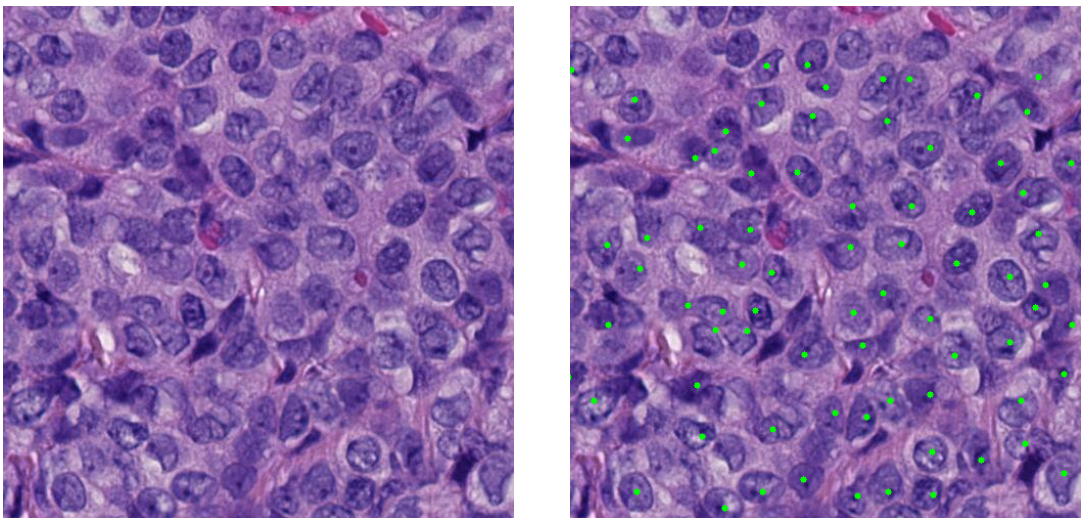


Figure 2.11: Number of cells per slide in dataset GANGLIONS

2.4.4 ANAPATH dataset

This dataset contains a single whole slide of cancer tissue originating from breast. This database differs from others in the sense that it consists of 6 rectangular regions of interests (rougly 10 megapixels each) coming from a single 24-gigapixels whole slide scanned at a pixel resolution of $0.227 \mu m$/pixel (magnitude 40x). Even if it can potentially biases the results since methods can tend to learn characteristics particular to this image, it corresponds to a possible real use case where the user

annotates only some regions of the image and applies a cell counting or detection method on the whole slide. In this case, two experts from the GIGA-Research of the University of Liège [Lon+16] have annotated 16414 cell nuclei in these 6 ROIs on the Cytomine platform with 2736 ± 223 cells per region of interest on average (Figure 2.13). A crop of a region of interest is shown in Figure 2.12 as example.



(a) without dot annotations (b) with dot annotations

Figure 2.12: Example of ANAPATH dataset region of interest.

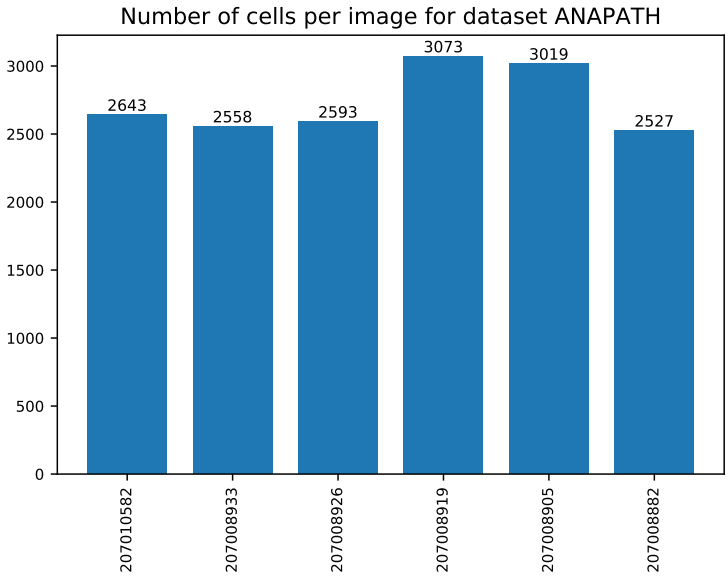


Figure 2.13: Number of cells per image in dataset ANAPATH

2.4.5 Summary

The Table 2.1 summarizes the main characteristics of each dataset.

Dataset	**Slides**	**ROIs (images)**	**Object annotations**
BM GRAZ	11		4205
CRC	100		29756
GANGLIONS	11	66	6396
ANAPATH	1	6	16414

Table 2.1: List of datasets

Chapter 3

Survey of existing approaches

This chapter introduces various approaches to detect and/or count cells into multi-gigapixels images. First, Section 3.1 gives a brief remainder of the supervised learning problem and of two related algorithms, randomized trees and convolutional neural networks. The problem of cell counting is then formulated in two distinct ways. It can indeed be seen as a detection problem or a density estimation problem (Section 3.2). Also, as randomized trees are not directly applicable to images, a special form of samples based on subwindows extracted from images is presented in Section 3.3. The different approaches are described in Section 3.4 for tree-based ensemble ones and in Section 3.5 for neural networks ones.

3.1 Supervised learning

The machine learning approaches studied in this work fall into the category of *supervised learning*. In this kind of problem, a *learning set* $LS = \{(\mathbf{x}_i, \mathbf{y}_i) | i = 1, ..., N\}$ must be available. Each of the N objects of this set is described by an *input vector* $\mathbf{x} \in \mathcal{X}$ and associated to an *output vector* $\mathbf{y} \in \mathcal{Y}$. When the output of the supervised problem is symbolic, one talks about *classification*, while the term *regression* is used when the output is numeric. The learning samples have to be gathered in order to be representative and expose the diversity of real use-case samples.

Given this set of examples, the goal is to *learn* a function (a *model*) $f : \mathcal{X} \to \mathcal{Y}$ only using the inputs and that maps at best the inputs with the outputs. This model can then be applied on new, unseen objects (for which the outputs are unknown) and *predict* these outputs.

The learning of f is the job of a *learning algorithm* A. It is an algorithm that takes as input the learning set LS and returns as output a model. In this work, randomized trees (see Section 3.1.1) and convolutional neural networks (see Section 3.1.2) have been used. These learning algorithms require some *hyperparameters* that will influence the *complexity* of the produced model. Ideally, the model should be able to predict at best the outputs for unseen objects.

All the art of supervised learning is to find the hyperparameters that lead to a

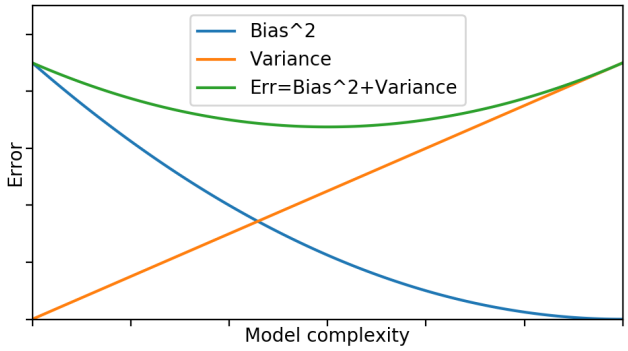


Figure 3.1: Bias and variance in function of model complexity

model with the best generalization performance, that is, with the best prediction capability on independent and unseen test data (*test set*). For a given *loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ measuring the discordance between its arguments, the *generalization error* that f has to minimize for a particular learning set LS and an independent test set is given by

$$Err_{LS} = E_{\mathbf{x},\mathbf{y}}\{\ell(\mathbf{y}, f_{LS}(\mathbf{x}))\}$$

The *expected* generalization error $Err = E_{LS}\{Err_{LS}\}$ is another quantity useful to characterize a learning algorithm. It can be shown that Err can be decomposed in a *noise* term (the residual error), a *bias* term and a *variance* term. As shown in Figure 3.1, there is a trade-off between bias and variance. Also, the bias is usually a decreasing function of the model complexity while the variance increases with model complexity. A model with low variance but high bias leads to *underfitting* because the learning algorithm does not fit enough the data. On the other hand, a model with high variance but low bias leads to *overfitting* since the learning algorithm starts to fit noise [HTF09].

3.1.1 Randomized trees

Random forests [Bre01] are an ensemble learning method using averaging techniques, especially tailored for *decision trees*. The idea behind this is that a single model of decision tree has inevitably some limitations and will make errors. By growing several models independently and averaging their predictions, the variance can be decreased which improves the ensemble model performance.

Let T be the number of decision tree estimators in the forest. Each tree is built from a bootstrap sample, or put differently a sample with replacement drawn from the learning set.

Decision tree growth The construction of a decision tree [Bre+84] initially starts with all learning samples available for that tree at its root. These samples are then partitioned according to the feature A and the associated discretization threshold τ such that (A, τ) is the best split according to a splitting criterion. A typical

splitting criterion in classification tasks is the Gini index impurity measure, the best split being the one providing the highest reduction of impurity of the data, or equivalently, the one for which the resulting subsets are as pure as possible. A set is said to be pure if all its samples belong to the same class. The Gini impurity measure for a learning set LS is given by $\sum_j p_j(1-p_j)$ where p_j is the proportion of objects of class j in LS. For regression, the best split is the one leading to the highest decrease of output variance. A new node is added for the feature A and the procedure is repeated recursively on the new subsets to grow the corresponding subtrees. The nodes are expanded until all leaves contain less than n_{min} samples, where n_{min} is thus the minimum number of samples required to split an internal node of the decision tree. The tree is fully developed when $n_{min} = 2$. Regarding the tree structure, each interior node tests a feature, each branch corresponds to a feature value (or range of values) and the leaves are labelled by the most frequent class among the objects belonging to that leaf in the case of a classification. For a regression, the output average among the objects belonging to the leaf is used.

When decision trees are used in random forests, the best split at a node is selected among a subset F_k of k features randomly chosen instead of looking for the best split among all features. There is bias-variance trade-off with this maximum number of features k, a smaller k decreases variance but increases bias.

The output is simply the average of predictions of each tree of the forest in regression tasks, while a majority class vote is used for classification (Figure 3.2).

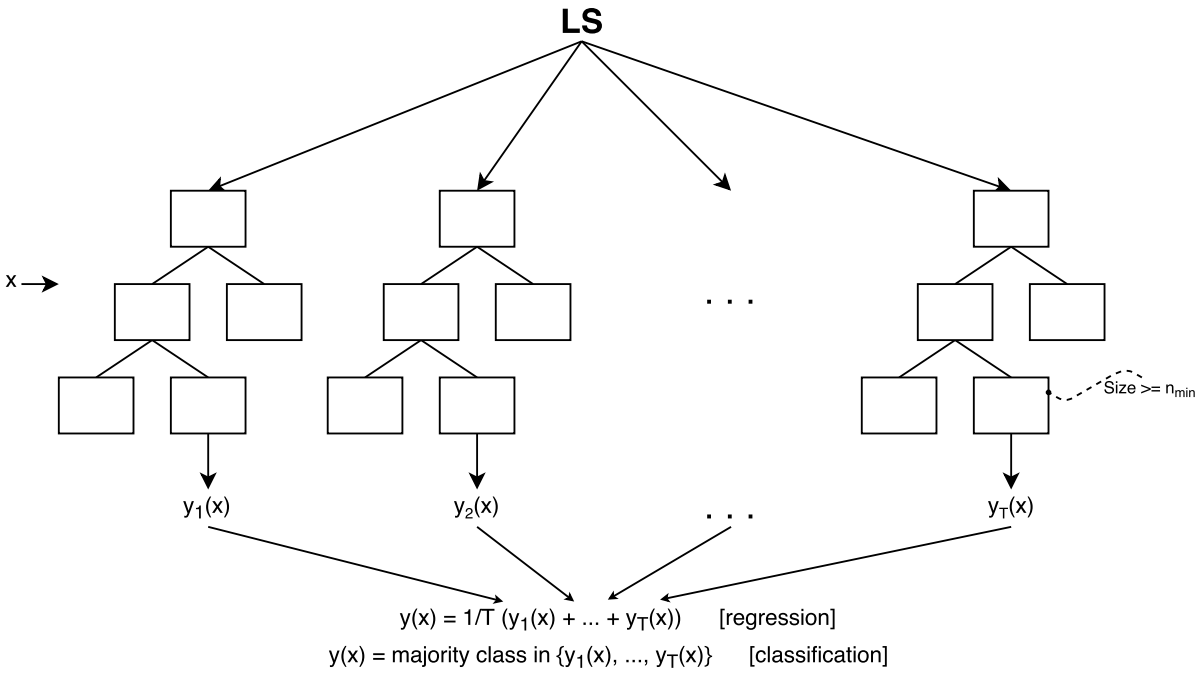


Figure 3.2: Randomized trees algorithms structure

Extra-Trees [GEW06] is another tree-based ensemble method for supervised learning where randomization during decision tree nodes splitting goes one step

further. It reduces variance and decreases training times. The main distinctions with random forests are:

1. The discretization thresholds τ_i associated to the features $A_i \in F_k$ where F_k is the subset of features when looking for the best split at a node, are no more optimized but uniformly drawn between the extreme values of the features A_i in the learning samples reaching this node.

2. By default, each decision tree in the forest is initialized with all the learning set without any sampling of training data.

3.1.2 Convolutional neural networks

Convolutional neural networks (ConvNets or CNNs) are a particular form of neural networks especially tailored for images that fall into the category of deep learning. The content of this Section is based on [LJY17], [Kar16] and [GW16]. A simple and intuitive way to define a neural network f is to see it as a composition of several functions f_i. In the deep learning world, these successive functions f_i are called *layers*. A model with L layers takes the form

$$\begin{aligned} \mathbf{y} &= f(\mathbf{x}; \mathbf{w}_1, ..., \mathbf{w}_L) \\ &= f_L(f_{L-1}(...(f_1(\mathbf{x}, \mathbf{w}_1); ...); \mathbf{w}_{L-1}); \mathbf{w}_L) \\ &= (f_L(\cdot; \mathbf{w}_L) \circ f_{L-1}(\cdot; \mathbf{w}_{L-1}) \circ ... \circ f_1(\cdot; \mathbf{w}_1))(\mathbf{x}) \end{aligned}$$

where the operator $\circ$ denotes the composition of function, *i.e.* $(g \circ f)(x) = g(f(x))$ and $\mathbf{w}_l$ is the *weight and bias vector* of the layer l.

The weight and bias vectors $\mathbf{w}_1, ..., \mathbf{w}_L$ have to be numerically optimized in the learning phase using the learning set LS of N objects, by solving the optimization problem

$$\arg \min_{\mathbf{w}_1,...,\mathbf{w}_L} \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{y}_i, f_{LS}(\mathbf{x}_i; \mathbf{w}_1, ..., \mathbf{w}_L)) + \frac{1}{2}\lambda \sum_{l=1}^{L-1} \mathbf{w}_l^2$$

where the last term of the minimization is a regularization term to penalize too large weights. λ is the *weight decay* and is an hyperparameter that has to be tuned. It allows to avoid overfitting by controlling complexity since more non linearity is introduced with large weights.

Without entering too much into details, the optimization problem is solved through stochastic gradient descent and backpropagation of derivatives. It calculates the gradients of the error with respect to all weights in the network and uses gradient descent to adjust all weights (depending on their contribution to the total error) to minimize the total error:

$$\mathbf{w}_l \leftarrow \mathbf{w}_l - \eta \left(\frac{\partial \ell(\mathbf{y}_i, f(\mathbf{x}_i; \mathbf{w}_1, ..., \mathbf{w}_L))}{\partial \mathbf{w}_l} + \lambda \mathbf{w}_l \right)$$

for a sample $\langle \mathbf{x}_i, \mathbf{y}_i \rangle \in LS$ chosen randomly and in a cyclic way.

The hyperparameter η is the *learning rate*. This parameter is often adjusted over time during training phase. In the step decay approach, η is reduced by some factor every few *epochs*. An epoch means that every samples of the learning set has been seen once. The number of epochs to perform e is an other hyperparameter. There is also the batch size b, which is the number of training samples that are used for one gradient update.

In the case of convolutional neural networks, the layer f_l typically takes the form either a *convolution* (Section 3.1.2.1), a spatial *pooling* (Section 3.1.2.2), a *non-linear activation* (Section 3.1.2.3), or a *fully connected* layer (Section 3.1.2.4). In general, a convolution layer is always followed by a spatial pooling and a non-linear activation.

3.1.2.1 Convolution

The goal of convolution layers (CONV) is to extract features from the input image. The convolution is a linear operation that keeps the spatial relationship between pixels by learning image features using small squares of input data. A *kernel* or *filter* is a $K \times K$ matrix smaller than input images.

The *activation map* or *feature map* is the matrix obtained by sliding the kernel over all the image and calculating the dot product between the input and the kernel. The filters thus behaves like feature detectors from the input image. For an input image of size $W \times H$, the size of the feature map is $W' \times H' \times D$ with

$$W' = \frac{W - K + 2P}{S} + 1 \qquad ; \qquad H' = \frac{H - K + 2P}{S} + 1$$

and where

- D (depth) is the number of filters used for convolution, producing thus D 2D feature maps.
- S (stride) is the number of pixels by which the filter matrix is slided over the input matrix.
- P (padding) is the number of pixels by which the input is extended on each border in order to apply the filters to side elements of the input image matrix.

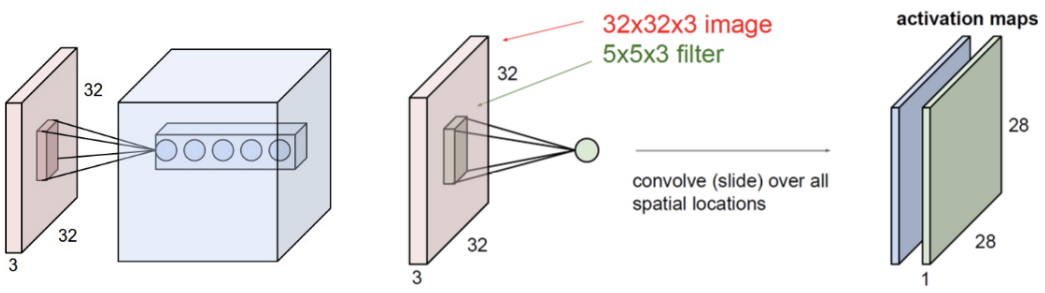


Figure 3.3: A convolutional layer [GW16]

Figure 3.3 illustrates a convolutional layer with a 5×5 filter on a $32 \times 32 \times 3$ image. The layer has a depth $D = 5$, a stride $S = 1$ and no padding. The output activation map thus has a size $28 \times 28 \times 5$ since $W' = H' = (32 - 5)/1 + 1 = 28$.

3.1.2.2 Spatial pooling

The pooling operation (POOL) reduces the quantity of information to only keep the most important one. As a benefit, it reduces the amount of parameters and thus the computation time but also limits overfitting. Small square blocks are merged together according to an aggregation function such *max* or *sum*. For an input map $W \times H \times D$, the pooling operation leads to an output map $W' \times H' \times D$ with

$$W' = \frac{W - F}{S} + 1 \qquad ; \qquad H' = \frac{H - F}{S} + 1$$

and where

- F is the spatial extent of pooling square blocks. Typically, $F = 2$.
- S is the stride. Typically, $S = 2$.

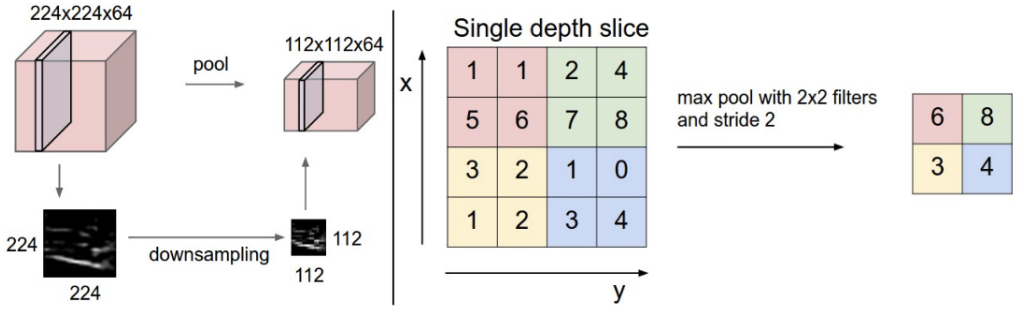


Figure 3.4: A max pooling layer [LJY17]

Figure 3.4 shows a max pooling layer with the typical values, $F = 2$ and $S = 2$. The output map has the same depth as in input but with and height are divided by 2.

3.1.2.3 Non linearity

Most of real-world input-output relationships are non linear. However, the convolution operation is a linear operation. To introduce non linearity in the convolutional neural network, an activation function is employed. In the *ReLU* (Rectifier Linear Unit) one, each element x of the input matrix is substituted by the non linear function $f(x) = \max(0, x)$. A smoothed alternative is the *SoftPlus* activation function $f(x) = \ln(1 + \exp(x))$ or the so-called sigmoïd function $\frac{1}{1+e^{-x}}$.

3.1.2.4 Fully connected layers

The fully connected layer (FC) is a traditional *Multi Layer Perceptron* (MLP) where every neuron in the previous layer is connected to every neuron on the next layer. Their activations can then be computed with a matrix multiplication.

It is worth to notice that a fully connected layer can be re-interpreted as a convolution layer. Indeed, for an input of size $W \times H \times D$, a fully connected layer of s neurons can be seen as a convolution layer with a kernel of size $W \times H$, $S = 1$ and $P = 0$ and an output feature map of size $1 \times 1 \times s$ where s is the depth. In fact, the only difference between fully connected and convolutional layers is that the neurons in the convolutional layer are connected only to a local region in the input, and that many of the neurons in a convolutional layer share parameters. However, the neurons in both layers still compute dot products, so their functional form is identical [LSD15].

3.2 Problem formulation

Before entering deeply into the problem formulation, a bunch of annotations is put in place.

It is assumed that a set of N training color images I_i, $i \in 1, ..., N$ is given. Each image is a 3-dimensional matrix $I_i \in \mathbb{R}^{W \times H \times D}$ where W is its width, H is its height and $D = \{3, 4\}$ is the number of channels in the image, where $D = 3$ corresponds to a RGB image and $D = 4$ corresponds to a RGBA image, whose the three first components can be decoupled from the last channel (alpha channel) which can be interpreted as a mask M_i such that for every pixel x of image I,

$$M_i(x) = \begin{cases} 1 & \text{if } x \text{ belongs to the region of interest in the image} \\ 0 & \text{otherwise} \end{cases}$$

For each image i, it is assumed that experts have annotated cells with a dot in their center. Let $\mathcal{C}_i$ the set of all cell annotations for image i. The binary map of annotations A_i is such that, for every pixel x of image I,

$$A_i(x) = \begin{cases} 1 & \text{if } x \in \mathcal{C}_i \\ 0 & \text{otherwise} \end{cases}$$

An example of mask M and annotation map A is given in Figure 3.5 with an image crop from the BRCA dataset. The number C_i of cells in the training image i is given either by $|\mathcal{C}_i|$ either by $\sum_x A_i(x)$.

According to these notations, the learning set is the set

$$LS = \{\langle I_i, A_i \rangle | i = 1, ..., N\}$$

Researches in scientific literature show that the supervised problem of cell counting and more generally the counting of objects in images can be approached using two distinct perspectives: counting by detection and counting by density estimation.

(a) Image I (b) Mask M (c) Annotations A (d) $A + I$

Figure 3.5: A BRCA image crop (a) with the corresponding mask (b), the binary map of annotations (c) and the annotations superposed on the image crop (d).

3.2.1 Counting by detection

This approach is taken from the different but related problem of *object detection*. A problem of object detection can be trivially castes to a problem of object counting since the only supplementary required step is to count the number of detections. As a bonus, the location of cells is provided. However, this approach inherits of all issues coming from object detection. Object detectors especially have difficulties with clumping or overlapping objects which is unfortunately a rather frequent case when objects are cells.

Object detectors operate in two steps. In the first stage, the goal is to predict a *score map* (or confidence map) $S \in [0, 1]^{W \times H}$ where each element $S(x)$ is a score denoting the probability of presence of an object at location x. It is expected that $S(x) = 1$ if $x \in \mathcal{C}_i$ and $S(x)$ moves towards 0 for any x far away from any $x' \in \mathcal{C}_i$.

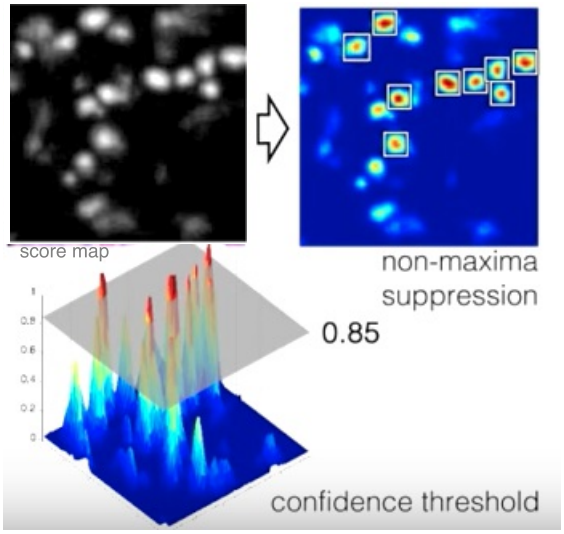


Figure 3.6: Score map and non-maximum suppression [Kai+15]

The second stage consists in applying a *thresholding* operation and a *non maximum suppression* operation on the score map S which results in the prediction

binary map of cell locations $\hat{A}$ as shown in Figure 3.6.

We will use this approach in the euclidean distance transform randomized trees based classifier (Section 3.4.1) and regressor (Section 3.4.2) but also in the spatially constrained fully convolutional neural network method (Section 3.5.2).

3.2.2 Counting by density estimation

In this approach, an estimate of the object count $\hat{C}$ is obtained directly without any prior object detection or segmentation, simply by integrating an estimated density function $D \in [0,1]^{W \times H}$ over the image domain [Fia+12]:

$$\hat{C} = \int_I D(x) \, \mathrm{d}x$$

where each element $D(x)$ is the density of objects per pixel at location x.

In this way, the issues of object detection approaches are avoided. However locations of counted objects are always desirable in real case application, at least to have an idea of what has been counted. The second stage of the counting by detection approach can be wisely employed in that purpose. The density map can be seen as a particular form of score map. The location of cells can thus be also obtained with this approach, which lead to an other number of cells $\hat{C}$ by counting localized cells. It is worth noticing that two distinct cell counts are gathered in this approach. In order to distinguish them, the count by density estimation will be denoted by $\hat{C}_{raw}$.

We will investigate this approach in the density randomized trees regressor (Section 3.4.3) and in the fully convolutional regression network (Section 3.5.1).

3.3 Subwindow-based samples

As randomized trees are not directly applicable to images, a special form of samples based on subwindows extracted from images is adopted. As [MWG13] explains, the input images I_i cannot be directly used as learning set instances. The first reason is the corresponding input vectors have to be of fixed size, while input images I_i are of various size. Secondly, supervised learning methods are not suited to take into account the 2D spatial arrangement of pixels in the image.

The approach to solve these issues is to extract a large number of subwindows N_{sw} of fixed size from each of the N images, optionally transform them and finally describe them using a fixed number of features.

At the training stage, the first step is to extract a large set of $N' = N \times N_{sw}$ subwindows from a learning set $LS = \{\langle I_i, A_i \rangle | i = 1, ..., N\}$. These subwindows consists of images of smaller size than the original ones and that are possibly overlapping. A subwindow $s = s(x)$ is entirely defined by its center coordinates x and its size $w \times h$. Typically, the subwindow size is chosen such that squared subwindows

are extracted. The extraction strategy is method dependant and three strategies have been used: random way, exhaustive way or even constrained by the map of annotations.

An output is associated to each input subwindow, which is obtained from the output map (A_i or a modified version of it such that the score map S_i or the density map D_i) of its parent image I_i. The output is once again a subwindow of size $w' \times h'$, where $w' \leq w$ and $h' \leq h$ is expected, extracted this time from the output map. The particular case when $w' = h' = 1$ leads to a single output problem while it corresponds to a multi-output ($w'h'$ outputs, each output being one of the pixel of the output map) in the general case.

Thus, a new learning set

$$LS' = \{\langle s_j^{in}, s_j^{out} \rangle | j = 1, ..., N'\}$$

is obtained, where the input and output subwindow's center x coincide, *i.e.* $s^{in}(x) = s^{out}(x)\ \forall j$. LS' can be directly used by CNN-based methods while input vectors $\mathbf{x}$ of fixed size must be derived from the input subwindows in order to be used by standard supervised machine learning methods, such as RT-based methods. It results in a learning set

$$LS'' = \{\langle \mathbf{x}(s_j^{in}), s_j^{out} \rangle | j = 1, ..., N'\}$$

At the prediction stage, in order the produced subwindows-based model to be applicable, subwindows are also extracted from the unseen image I_{test} and must have the same size, in input and output, as during the training stage. If required, the same transformations and feature extraction have to be applied.

However, the subwindow extraction strategy may differ. Indeed during prediction, for each pixel $x \in I_{test}$, a subwindow such that x is its pixel center is extracted. The number of subwindows is thus $N_{sw,test} = W_{test}H_{test}$. Each input vector of the set $\{\mathbf{x}(s_j^{in}) | j = 1, ..., N_{sw,test}\}$ is applied to the model and produces a predicted output subwindow. The predictions of these output subwindows are aggregated with a spatial average which results in a predicted map of size $W_{test} \times H_{test}$. The nature of this map depends on the one that has been used as output during training stage (annotation map $\hat{A}_{test}$, score map $\hat{S}_{test}$ or density map $\hat{D}_{test}$). In the particular case of 1×1 output subwindows (single output problem), each prediction directly corresponds to a pixel of the predicted output map.

3.3.1 Subwindow extraction strategy

3.3.1.1 Random strategy

Algorithm 1 extracts a total of $N \times N_{sw}$ subwindows at random.

3.3.1.2 Exhaustive strategy

Algorithm 2 extracts all possible subwindows from the given images.

Algorithm 1. RandomSubwindowExtraction

Input: Number of subwindows per image N_{sw}, input subwindow size $w \times h$, output subwindow size $w' \times h'$
Data: A set of pairs image/map $\mathcal{I}$
Output: A set of pairs input/output subwindows SW

$SW = \emptyset$
foreach $\langle I, M \rangle \in \mathcal{I}$ **do**
    Let C the set of N_{sw} positions in I, chosen randomly
    Extend I and M by $w/2$ pixels on the left and on the right
    Extend I and M by $h/2$ pixels on top and bottom
    foreach $x \in C$ **do**
        $s^{in} \leftarrow$ subwindow from I of size $w \times h$ such that x is its center
        $s^{out} \leftarrow$ subwindow from M of size $w' \times h'$ such that x is its center
        Append to SW the pair $\langle s^{in}, s^{out} \rangle$
    end
end
return SW

Algorithm 2. ExhaustiveSubwindowExtraction

Input: Input subwindow size $w \times h$, output subwindow size $w' \times h'$
Data: A set of pairs image/map $\mathcal{I}$
Output: A set of pairs input/output subwindows SW

$SW = \emptyset$
foreach $\langle I, M \rangle \in \mathcal{I}$ **do**
    Let C the set of all positions in I such that $|C| = W_I H_I$
    Extend I and M by $w/2$ pixels on the left and on the right
    Extend I and M by $h/2$ pixels on top and bottom
    foreach $x \in C$ **do**
        $s^{in} \leftarrow$ subwindow from I of size $w \times h$ such that x is its center
        $s^{out} \leftarrow$ subwindow from M of size $w' \times h'$ such that x is its center
        Append to SW the pair $\langle s^{in}, s^{out} \rangle$
    end
end
return SW

3.3.1.3 Scoremap constrained way

Algorithm 3 extracts all subwindows from the given images whose center pixel is linked to a score higher than a fixed score threshold. The number of extracted subwindows is thus scoremap dependent. The set of subwindows is completed by a certain amount (by default, an equal amount) of subwindows randomly chosen among remaining positions: a half for which the score is less than the threshold but non zero and a half for which the score is equal to zero.

Algorithm 3. ConstrainedSubwindowExtraction

Input: Input subwindow size $w \times h$, output subwindow size $w' \times h'$, threshold t, ratio q (by default, $q = 1$)
Data: A set of pairs image/scoremap $\mathcal{I}$
Output: A set of pairs input/output subwindows SW

$SW = \emptyset$
foreach $\langle I, S \rangle \in \mathcal{I}$ **do**
  $C \leftarrow$ set of all positions $x \in I$ such that $S(x) > t$
  $C' \leftarrow$ set of $q|C|/2$ randomly chosen positions $x \in I$ such that $0 < S(x) \leq t$
  $C'' \leftarrow$ set of $q|C|/2$ randomly chosen positions $x \in I$ such that $S(x) = 0$
  Extend I and S by $w/2$ pixels on the left and on the right
  Extend I and S by $h/2$ pixels on top and bottom
  foreach $x \in C \cup C' \cup C''$ **do**
    $s^{in} \leftarrow$ subwindow from I of size $w \times h$ such that x is its center
    $s^{out} \leftarrow$ subwindow from S of size $w' \times h'$ such that x is its center
    Append to SW the pair $\langle s^{in}, s^{out} \rangle$
  end
end
return SW

3.3.2 Feature extraction

As far as a RT-based method is involved, each subwindow can be described by a set of image filters that are used to build the corresponding input feature vector. The choice of filters is application dependant. This work focused on analysis of images of H& E stained tissues, just like [Kai+15]. Inspired by it, the following filters are applied

1. Each pixel is decomposed in the Red, Green and Blue components of the RGB colorspace (3 features per pixel).

2. Each pixel is decomposed in the L^*, u^* and v^* components of the CIELUV colorspace, a colorspace system exhibiting perceptual uniformity, *i.e.* a change step in the data corresponds to an approximately equally perceptible change step across the colorspace [Poy97] (3 features per pixel).

3. Each pixel is decomposed in the Hue, Saturation and Value components of the HSV colorspace which has the benefit to increase robustness to illumination changes (3 features per pixel).

4. Each pixel is extracted from the grayscale colorspace after an histogram equalization in order to improve constrast in the image (1 feature per pixel).

5. The Sobel operator at first order is used as edge detector and is applied on each pixel in the grayscale space. It is a discrete differentiation operator which computes an approximation of the derivatives of an image intensity function

obtained by using two 3×3 kernels, one for horizontal changes and one for vertical changes, that are convolved with the input image I. The horizontal changes are obtained by

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

Analogously, the vertical changes are obtained by

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

where $*$ denotes the convolution operator. Moreover, the resulting gradient approximations are combined to give the gradient magnitude

$$G = \sqrt{G_x^2 + G_y^2}$$

Three supplementary features per pixel are thus extracted.

This bank of filters is inspired from [Kai+15] with three notable differences:

1. the second order derivative edge detection with the Sobel operator is not used as it does not seem to be widely used in literature nor provide useful information in the studied images (a possible explanation of error source is that it is amplified with derivatives due to the discretized nature of images)

2. in addition to single pixel values, the authors of [Kai+15] compute as features, for each filter, pixel value differences between two random pixels of the same subwindow, Haar-like features, and a constrained pixel value difference, where the second pixel location is chosen within the 10 pixels clamped at the subwindow borders.

3. the HSV colorspace is introduced as it has already proven its interest in other histology image analysis problems [MWG13], [MGW16].

Figure 3.7 shows a visual representation of input features for an image crop extracted from BM GRAZ dataset, where a perceptually uniform colormap has been used.

In the end, the size d of the input vector $\mathbf{x}$ describing the visual content of a subwindow is equal to $(3 + 3 + 3 + 1 + 3) \times w \times h = 13 \times w \times h$ numerical input features.

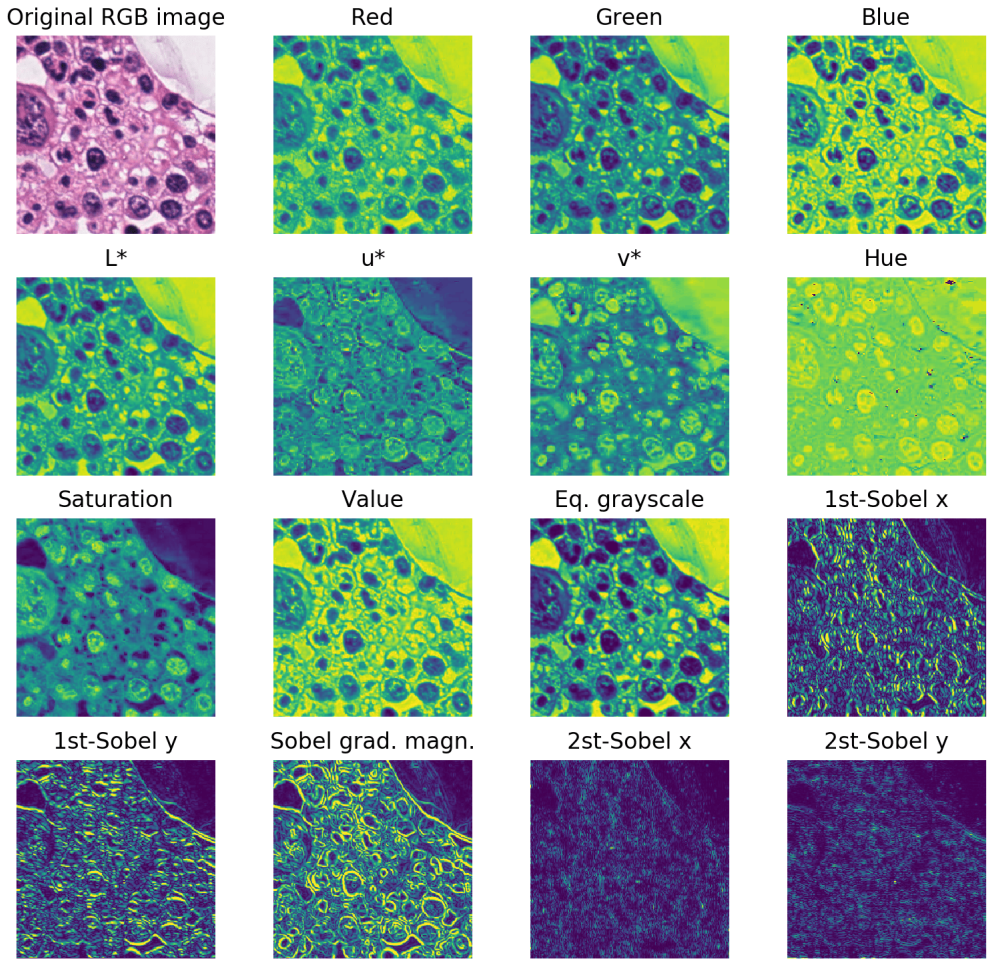


Figure 3.7: Visual representation of input features

3.4 Randomized trees-based methods

The main building blocks having being introduced, we will now explore methods in details. Each of them takes the form of a localization or counting workflow with two phases. First, during the training stage, the model is built from a set of images for which cells have been marked. In the second stage, the prediction is performed on a new, unseen image using the model produced at the previous step. For reference sake in the remainder of this text, we will attribute an acronym to each method.

3.4.1 Standard Classification (SRTC)

This method is described in [Kai+15]. A random forest classifier is trained to predict whether the center pixel of a subwindow extracted from a new, unseen image is of interest (center of a cell) or not. We refer this approach as Standard Randomized Trees Classification (SRTC).

3.4.1.1 Training stage

The learning set is built to be balanced between positive (foreground, cell center) and negative (background) samples.

All subwindows s whose centers correspond to a cell center are extracted from the images I_i. In order to increase the number of positive samples in the learning set, the same operation is repeated on flipped versions of images I_i (vertically, horizontally and both). Negative samples (subwindows whose centers is background) randomly chosen are added to balance the learning set.

The input subwindows s^{in} are transformed to input vectors using the input subwindow feature extractor $\mathbf{v}$ presented in Section 3.3.2. On the other hand, the output s^{out} associated to an extracted subwindow s^{in} is the output vector of size $w'h'$ which is such that pixel (i, j) of the output subwindow is at location $iw' + j$ in the output vector, and whose value is the class, foreground (1) or background (0), corresponding to the pixel (i, j). The learning procedure is summarized in Algorithm 4.

Algorithm 4. ClassificationTraining

Input: randomized trees classifier algorithm $algo = \{\text{RF, ET}\}$, number of tree estimators T, maximum number of features when looking for best split k, minimum number of samples to split a node n_{min}, input subwindow size $w \times h$, output subwindow size $w' \times h'$, input subwindow feature extractor $\mathbf{v}$

Data: a learning set of N images with annotation maps

$$LS = \{\langle I_i, A_i\rangle | i = 1, ..., N\}$$

Output: a tree-based ensemble model f ($w'h'$ outputs) built from subwindows

$LS' =$ ConstrainedSubwindowExtraction(LS, $w \times h$, $w' \times h', 0.5, 2$)
$LS'' = \emptyset$
foreach $\langle s_j^{in}, s_j^{out}\rangle \in LS'$ **do**
    $LS'' = LS'' \cup \{\langle \mathbf{v}(s_j^{in}), s_j^{out}\rangle\}$
    $LS'' = LS'' \cup$ Augment($\langle \mathbf{v}(s_j^{in}), s_j^{out}\rangle$)
end
$f = algo(T, k, n_{min};\ LS'')$
return f

3.4.1.2 Prediction stage

The goal is to predict the binary annotation map $\hat{A}$ associated to a new, unseen image I_{test}. The trained model f is used to predict each pixel of this map, in a sliding window fashion, that is, each pixel of I_{test} is the center of an input subwindow. In the end, $\hat{A}$ should provide the predicted positions of cell centers and the number of cells would correspond to the number of dots in the predicted binary map $\hat{A}$ where $\hat{A}(x) \in \{0, 1\}\ \forall x \in \hat{A}$. We note that authors did not considered probabilities of belonging to a class instead of predicted classes. It could be an interesting alternative but we have not tried it in our tests.

This method can suffer from several drawbacks as stated by [Kai+15]. Firstly, the cell centers can be not well localized due an inconclusive plateau-like classifier response on the cell nuclei. Secondly, it may lead to multiple peak responses for single objects, and finally individual adjacent cell objects can be merged if they are too close or if their appearance is too similar. A solution to overcome these issues is to smooth $\hat{A}$ using a Gaussian kernel in order to group multiple peaks into a single one. However, it implies also the risk to merge peaks actually belonging to different cell objects.

After such a smoothing, the map takes now no longer values in $\{0, 1\}$ but rather values in $[0, 1]$. A *non maximum suppression algorithm* has to be performed in order to find a binary map again.

Non maximum suppression This process resumed in Algorithm 5 is composed of the following steps

1. The input real-value map is normalized in the $[0, 1]$ space, such that the highest value in the map is equal to 1 after this step

2. The real-value map is optionally smoothed with a Gaussian filter in order to merge neighboring maxima. Indeed, in image processing, this filter typically blurs the original image, reducing detail and image noise. The result is obtained by a convolution of the original image with a 2D Gaussian function

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

   where σ is the standard deviation of the Gaussian distribution.

3. A maximum filter is used for finding local maxima in the real-value map. This operation, also known as dilation filter, dilates the original image and merges neighboring local maxima closer than the size of the dilation. Its size is given by $2\xi + 1$ where ξ is the minimum distance between two expected maxima. ξ is determined empirically for a dataset.

4. Locations where the original image is equal to the dilated image and whose value is above a fixed discarding threshold κ are returned in a binary map.

Algorithm 5. PostProcessing

Input: standard deviation of the merging Gaussian kernel σ, minimum distance between 2 objects ξ, discarding threshold for non maximum suppression κ

Data: a real-value map $M_{\mathbb{R}}$ such that $M_{\mathbb{R}}(x) \in [0, 1]\ \forall x \in M_{\mathbb{R}}$

Output: a binary map $M_{\mathbb{Z}}$ such that $M_{\mathbb{Z}}(x) \in \{0, 1\}\ \forall x \in M_{\mathbb{Z}}$

$M_{\mathbb{R}} = M_{\mathbb{R}} / \max(M_{\mathbb{R}})$ *//Normalize in* $[0, 1]$ *space*
if $\sigma > 0$ **then**
    $M_{\mathbb{R}} =$ GaussianFilter($M_{\mathbb{R}}$, σ) *//Optional smoothing*
end
$dilation\_size = 2\xi + 1$
$M_{\mathbb{Z}} = (M_{\mathbb{R}} ==$ MaxFilter($M_{\mathbb{R}}$, *dilation_size*))
foreach $x \in M_{\mathbb{Z}}$ **do**
    if $M_{\mathbb{R}}(x) < \kappa$ **then**
        $M_{\mathbb{Z}}(x) = 0$
    end
end
return $M_{\mathbb{Z}}$

The complete algorithm for prediction phase is given in Algorithm 6. In [Kai+15], the authors only considered the case of single output classifier, using thus 1×1 output subwindows.

Algorithm 6. ClassificationPrediction

Input: a tree-based model f, input subwindow feature extractor $\mathbf{v}$, input subwindows size $w \times h$, standard deviation of the merging Gaussian kernel σ, minimum distance between 2 objects ξ, discarding threshold for non maximum suppression κ

Data: a new, unseen test image I_{test}

Output: a predicted binary map of the same shape as I_{test} giving predicted cell positions

$TS =$ ExhaustiveSubwindowExtraction(TS, $w \times h$, 1×1)
$TS' = \emptyset$
foreach $s_j^{in} \in TS$ **do** $TS' = TS' \cup \{\mathbf{v}(s_j^{in})\}$
$\hat{A} = f(TS')$
$\hat{A} =$ PostProcessing($\hat{A}, \sigma, \xi, \kappa$)
return $\hat{A}$

3.4.2 Regression by proximity score estimation (PRTR)

The key idea behind this method [Kai+15] is to train a regressor to predict a proximity score for each pixel of a new, unseen image and extract local maxima from

the score map that would correspond to cell centers. It should overcome the issues experienced with standard classification. This method will be refered as Proximity Randomized Trees Regression (PRTR).

3.4.2.1 Training stage

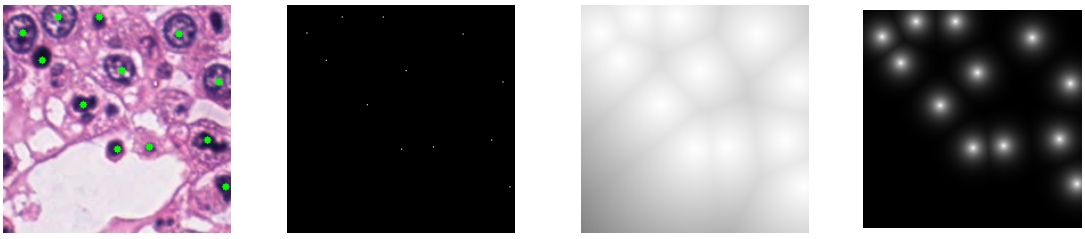
A pre-processing step is required where a score map S_i is computed for each image I_i. Each pixel of this score map encodes a score which varies as a function of its distance to the nearest cell center. We expect it produces high scores only in foreground areas with remarkable peaks at cell center and a low score in background areas. The scores are then computed by

$$S_i(x) = \begin{cases} e^{\alpha\left(1-\frac{\mathcal{D}_{A_i}(x)}{r}\right)} - 1 & \text{if } \mathcal{D}_{A_i}(x) < r \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where

- α is a positive integer controlling the shape of the exponential
- r is the mean object radius
- $\mathcal{D}_{A_i}(x)$ is the Euclidean distance transform of the cell centers on the annotation map A_i

The distance transform operator $\mathcal{D}$ maps a binary image into a grayscale image where the pixel values represent the distance to the nearest obstacle pixel in the binary image, according to a specified metric. Figure 3.8 shows an image crop containing cells, with the corresponding annotation A, the computed distance transform $\mathcal{D}_A$ and the score map S. The usage of a non-linear function S instead of the direct application of the distance transform has two advantages (see Figure 3.9). First, the object positions are precisely localized and the exponential shape clearly distinguishes the object centers from the rest of the object. Also, regions without any markers (such as bottom left in Figure 3.8) are encoded with a single and same score on the contrary to the direct application of $\mathcal{D}$.



(a) (b) (c) (d)

Figure 3.8: A BM GRAZ image crop I (a) with the corresponding annotation map A (b), the raw Euclidean distance transform $\mathcal{D}_A$ (c) and the score map S with $\alpha = 3$ and $r = 19$ (d).

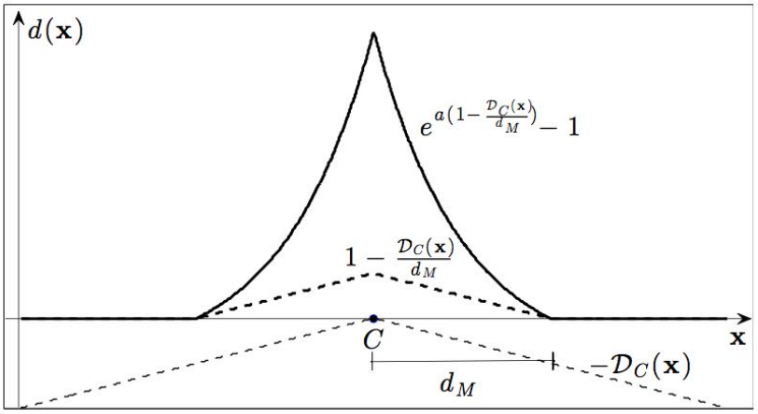


Figure 3.9: The score map S with a cell center in C [Sir+14]. With our notations, $d(x)$ should be seen as $S(x)$ and d_M as r.

Once the score maps are computed, the learning set is built with all subwindows s^{in} whose center pixel x is such that $S_i(x) > t$ where t is a score threshold to decide if the score is foreground or not. Let P be the the number of such subwindows. The learning set is completed with qP subwindows chosen at random locations, where q is the ratio between foreground and background in the learning set. Authors restrict $q \in [0, 1]$ while $a > 1$ would deserve to be tried as there are more background pixels on an image.

The output s^{out} associated to an extracted input subwindow s^{in} is the output vector of size $w'h'$ which is such that pixel (j, k) of the output subwindow is at location $jw' + h'$ in the output vector, and whose value corresponds to the pixel (j, k) in S_i. The input feature extractor $\mathbf{v}$ is the same as the one used with SRTC.

A random forest regressor is trained using the given subwindow-based learning set. The full procedure is given in Algorithm 7.

3.4.2.2 Prediction stage

The goal is to predict a score map $\hat{S}$ associated to a new, unseen image. Each pixel of $\hat{S}$ is predicted using the model, with a sliding subwindow. The procedure is somehow the same as prediction in SRTC with the exception that the learning algorithm can produce multiple outputs if the output subwindows have a size larger than 1. The predicted score map is obtained by averaging all the predicted overlapping subwindows :

$$\hat{S}(x) = \frac{1}{\mathcal{S}(x)} \sum_{\hat{s}^{out} \in \mathcal{S}(x) \in \mathcal{P}} \hat{s}^{out}(x) \tag{3.2}$$

where $\mathcal{S}(x)$ is the set of predicted subwindows s that have pixel x in their scope. $\mathcal{S}(x)$ is a subset of the set of predictions $\mathcal{P}$.

Also, the smoothing in the post-processing step should be less essential than in classification. Indeed, the predicted map is already a real-value map in this case.

Algorithm 7. ProximityRegressionTraining

Input: random forest regressor algorithm $algo = \{\text{RF, ET}\}$, number of tree estimators T, maximum number of features when looking for best split k, minimum number of samples to split a node n_{min}, input subwindows size $w \times h$, output subwindows size $w' \times h'$, input subwindow feature extractor $\mathbf{v}$, mean cell radius r, exponential shape of score peaks α, score threshold t, ratio q

Data: a learning set of N images with annotation maps

$$LS = \{\langle I_i, A_i \rangle | i = 1, ..., N\}$$

Output: a tree-based ensemble model f ($w'h'$ outputs) built from subwindows

$LS' = \emptyset$
foreach $\langle I_i, A_i \rangle \in LS$ **do**
    $LS' = LS' \cup \{\langle I_i, S_i \rangle\}$
    where S_i is obtained using Equation 3.1 with parameters α and r
end
$LS'' =$ ConstrainedSubwindowExtraction(LS, $w \times h$, $w' \times h'$, t, q)
$LS''' = \emptyset$
foreach $\langle s_j^{in}, s_j^{out} \rangle \in LS''$ **do** $LS''' = LS''' \cup \{\langle \mathbf{v}(s_j^{in}), \mathbf{w}(s_j^{out}) \rangle\}$
$f = algo(T, k, n_{min};\ LS''')$
return f

Algorithm 8 recapitulates the procedure.

Algorithm 8. RegressionPrediction

Input: a tree-based model f, input subwindow feature extractor $\mathbf{v}$, input subwindows size $w \times h$, output subwindows size $w' \times h'$, standard deviation of the merging Gaussian kernel σ, minimum distance between 2 objects ξ, discarding threshold for non maximum suppression κ

Data: a new, unseen test image I_{test}

Output: a predicted binary map of the same shape as I_{test} giving predicted cell positions

$TS =$ ExhaustiveSubwindowExtraction(TS, $w \times h$, $w' \times h'$)
$TS' = \emptyset$
Let $\mathcal{P} = \emptyset$ be the set of predictions.
foreach $s_j^{in} \in TS$ **do**
    Append $\hat{s}_j^{out} = f(\mathbf{v}(s_j^{in}))$ to $\mathcal{P}$.
end
Build $\hat{S}_{\mathbb{R}}$ using formula 3.2 with $\mathcal{P}$.
$\hat{S}_{\mathbb{Z}} =$ PostProcessing($\hat{S}_{\mathbb{R}}, \sigma, \xi, \kappa$)
return $\hat{S}_{\mathbb{Z}}$

3.4.3 Regression by density estimation (DRTR)

Proposed in [Fia+12], the method aims to count objects by integrating a density map that is predicted from an input image. We will call this approach the Density Randomized Trees Regression (DRTR).

3.4.3.1 Training stage

A pre-processing step is required where a density map D_i is computed for each image I_i. In the same way as population density measures the number of inhabitants per area unit in demography, we build an object density map where the area unit is the squared pixel. Each pixel of this density map encodes then a density of objects per pixel. The density map is given by

$$D_i(x) = \sum_{\mu \in \mathcal{C}_i} \mathcal{N}(x; \mu, \sigma) \tag{3.3}$$

where μ is the mean and σ is the standard-deviation of a normal or Gaussian distribution $\mathcal{N}$, and $\mathcal{C}_i$ is the set of all cell annotations for image I_i.

The key point of the method is to notice that this equation describes in fact the application of a Gaussian filter on the annotation mask A_i. Indeed, by definition, $\forall x$ such that $x \notin \mathcal{C}_i : A_i(x) = 0$. We thus have

$$D_i = \textsc{GaussianFilter}(A_i, \sigma)$$

A set of randomly chosen (possibly overlapping) subwindows of size $w \times h$ makes the learning set. N_{sw} subwindows per image I_i are selected. The output s^{out} associated to an input subwindow s^{in} is a subwindow $w' \times h'$ extracted from D_i such that the input and output subwindow centers are concurrent. The method thus must use a multi-output regression if $w'h' > 1$. Also, in [Fia+12] input and output subwindows have the same size, i.e. $w = w'$ and $h = h'$. Algorithm 9 summarizes the procedure.

3.4.3.2 Prediction stage

The procedure is similar to PRTR approach, with a sliding window over the image. The number of objects $\hat{C}$ in image I_{test} is given by the integration of $\hat{D}_{\mathbb{R}}$ over the image domain

$$\hat{C}_{raw} = \int_{I_{test}} \hat{D}_{\mathbb{R}}(x) \, \mathrm{d}x$$

Inspired by PRTR, we propose an extension to the method. By performing non maximum suppression on $\hat{D}_{\mathbb{R}}$ with Algorithm 5, we obtain a binary mask $\hat{D}_{\mathbb{Z}}$ with the positions of local density maxima, which may correspond to some extent to the cell centers.

The Algorithm 8 can hence be re-employed here with density map instead of proximity score map. Instead of only returning the binary map $\hat{D}_{\mathbb{Z}}$ obtained with non maximum suppression, the predicted density map with real values $\hat{D}_{\mathbb{R}}$ is also returned. Two counts are thus obtained, $\hat{C}$ from detection and $hatC_{raw}$

Algorithm 9. DensityRegressionTraining

Input: random forest regressor algorithm $algo = \{\text{RF, ET}\}$, number of tree estimators T, maximum number of features when looking for best split k, minimum number of samples to split a node n_{min}, subwindows size $w \times h$, input subwindow feature extractor $\mathbf{v}$, pre-processing standard deviation σ, number of random subwindows par image N_{sw}

Data: a learning set of N images with annotation maps

$$LS = \{\langle I_i, A_i \rangle | i = 1, ..., N\}$$

Output: a tree-based ensemble model f (wh outputs) built from subwindows

$LS' = \emptyset$
foreach $\langle I_i, A_i \rangle \in LS$ **do**
    $D_i = \text{GaussianFilter}(A_i, \sigma)$
    $LS' = LS' \cup \{\langle I_i, D_i \rangle\}$
end
$LS'' = \text{RandomSubwindowExtraction}(LS,\ w \times h,\ w \times h,\ N_{sw})$
$LS''' = \emptyset$
foreach $\langle s_j^{in}, s_j^{out} \rangle \in LS''$ **do** $LS''' = LS''' \cup \{\langle \mathbf{v}(s_j^{in}), \mathbf{w}(s_j^{out}) \rangle\}$
$f = algo(T, k, n_{min};\ LS''')$
return f

3.5 Convolutional neural networks-based methods

3.5.1 Regression by density estimation (FCRN)

This approach is proposed by [XNZ15] and is based on the work of [LSD15] who developed a Fully Convolutional Regression Network (FCRN) for semantic labelling. By reinterpreting the fully connected layers of a classification net as convolutional ones and using upsampling filters, it can take an input of arbitrary size and produce a correspondingly-sized output both for end-to-end training and for prediction.

[XNZ15] proposes two architectures: FCRN-A (Table 3.1) and FCRN-B (Table 3.2). Unfortunately, it does not provide any justification for these architectures. Figure 3.10 gives a graphical representation of these two networks. The first several layers of the network contain regular convolution-ReLU-pooling, then the spatial reduction is removed by performing upsampling-ReLU-convolution. It maps the feature maps back to the original dimension. Upsampling is performed by bilinear interpolation.

A key parameter in each convolution layer is the padding P. It is automatically chosen such that the width and height of the output volume are the same as in the input volume. Each convolutional layer only changes the depth of a volume.

Layer	Type	Parameters	Filter size	Output size
0	INPUT			$w \times h \times d$
1	CONV+ReLU	$K = 3; D = 32; S = 1$	$3 \times 3 \times 32$	$w \times h \times 32$
2	POOL	$F = 2; S = 2$	2×2	$w/2 \times h/2 \times 32$
3	CONV+ReLU	$K = 3; D = 64; S = 1$	$3 \times 3 \times 64$	$w/2 \times h/2 \times 64$
4	POOL	$F = 2; S = 2$	2×2	$w/4 \times h/4 \times 64$
5	CONV+ReLU	$K = 3; D = 128; S = 1$	$3 \times 3 \times 128$	$w/4 \times h/4 \times 128$
6	POOL	$F = 2; S = 2$	2×2	$w/8 \times h/8 \times 128$
7	FC+ReLU	$D = 512$	$3 \times 3 \times 512$	$w/8 \times h/8 \times 512$
8	UPSAMPLING	$F = 2; S = 2$	2×2	$w/8 \times h/8 \times 512$
9	CONV+ReLU	$K = 3; D = 128; S = 1$	$3 \times 3 \times 128$	$w/4 \times h/4 \times 128$
10	UPSAMPLING	$F = 2; S = 2$	2×2	$w/2 \times h/2 \times 128$
11	CONV+ReLU	$K = 3; D = 64; S = 1$	$3 \times 3 \times 64$	$w/2 \times h/2 \times 64$
12	UPSAMPLING	$F = 2; S = 2$	2×2	$w \times h \times 64$
13	CONV+ReLU	$K = 3; D = 32; S = 1$	$3 \times 3 \times 32$	$w \times h \times 32$
14	CONV	$K = 1; D = 1; S = 1$	$1 \times 1 \times 1$	$w \times h \times 1$

Table 3.1: FCRN-A architecture

Layer	Type	Parameters	Filter size	Output size
0	INPUT			$w \times h \times d$
1	CONV+ReLU	$K = 3; D = 32; S = 1$	$3 \times 3 \times 32$	$w \times h \times 32$
2	CONV+ReLU	$K = 3; D = 64; S = 1$	$3 \times 3 \times 64$	$w \times h \times 64$
3	POOL	$F = 2; S = 2$	2×2	$w/2 \times h/2 \times 64$
4	CONV+ReLU	$K = 3; D = 128; S = 1$	$3 \times 3 \times 128$	$w/2 \times h/2 \times 128$
5	CONV+ReLU	$K = 5; D = 256; S = 1$	$5 \times 5 \times 256$	$w/2 \times h/2 \times 256$
6	POOL	$F = 2; S = 2$	2×2	$w/4 \times h/4 \times 256$
7	FC+ReLU	$D = 256$	$3 \times 3 \times 256$	$w/4 \times h/4 \times 256$
8	UPSAMPLING	$F = 2; S = 2$	2×2	$w/2 \times h/2 \times 256$
9	CONV+ReLU	$K = 5; D = 256; S = 1$	$5 \times 5 \times 256$	$w/2 \times h/2 \times 256$
10	UPSAMPLING	$F = 2; S = 2$	2×2	$w \times h \times 256$
11	CONV	$K = 1; D = 1; S = 1$	$1 \times 1 \times 1$	$w \times h \times 1$

Table 3.2: FCRN-B architecture

3.5.1.1 Training stage

The learning set has to be augmented since the more examples are used to train the model, the better the model is performing. In that purpose, possibly overlapping input subwindows $w \times h$ are extracted from input images in a random way. It clearly increases the amount of available data for training. Also, rotated $(0°, 90°, 180°, 270°)$ and flipped (along horizontal or vertical axis) versions of these subwindows are added to the learning set. Each input subwindow is normalized by subtracting its own mean value and then dividing by the standard deviation.

As in other approaches, an output subwindow is associated to each input. It is

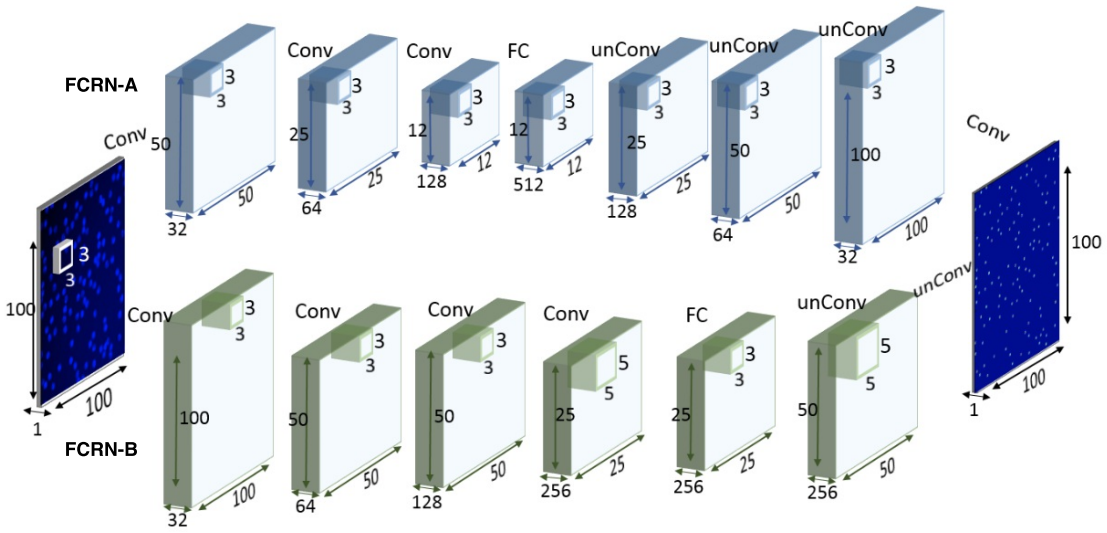


Figure 3.10: The two FCRN architectures for a $100 \times 100 \times 3$ input RGB image [XNZ15]

extracted from the ground truth density map D_i linked to each image I_i of the learning set. A key point of the method is that input and output subwindows have the same size. [XNZ15] argues that the content of output subwindows must be scaled, for example multiplying the ground truth annotation by 100. Without this operation, the difference between peak values and background is too small. Therefore, the networks tend to be more focusing on fitting the background zero rather than Gaussian shapes.

It is worth to remark that subwindows are here used only to augment the dataset. The method would also work if input images are directly given, provided that the size of learning set is large enough.

Algorithm 10 summarizes the training step.

3.5.1.2 Prediction stage

To predict the density map of a new, unseen image I_{test} with the FCRN model f, it is enough just to apply I_{test} to the model. Mathematically, we have

$$\hat{D} = f(I_{test})$$

To obtain a binary mask from the predicted real-value density map, it is enough to apply the non maximum suppression algorithm (Algorithm 5) as previously, requiring a discarding threshold κ and optionally, the standard deviation σ of a Gaussian kernel to merge neighboring multiple responses into a single one. As in DRTR, it is an extension of the original method that we propose here.

Algorithm 10. FCRNTraining

Input: architecture $arch = \{A, B\}$, number of epochs e, batch size b, learning rate $\eta(e)$, weight decay λ, pre-processing standard deviation σ, subwindow size $w \times h$, number of extracted subwindows per image N_{sw}

Data: a learning set of N images with annotation maps

$$LS = \{\langle I_i, A_i \rangle | i = 1, ..., N\}$$

Output: a CNN model f

$LS' =$ RandomSubwindowExtraction(LS, $w \times h$, $w \times h$, N_{sw})
Augment LS' with rotated and flipped versions of its elements
$LS'' = \emptyset$
foreach $\langle I_i, A_i \rangle \in LS'$ **do**
    $I_{i,norm} = [I_i - \text{mean}(I_i)] / \text{std}(I_i)$
    $D_i =$ GaussianFilter(A_i, σ)
    $LS'' = LS'' \cup \{\langle I_{i,norm}, D_i \rangle\}$
end
$f = \text{CNN}(arch, e, b, \eta(e), \lambda;\ LS'')$
return f

3.5.2 Regression by proximity score estimation (SC-CNN)

[Sir+16] presents this method as a spatially constrained convolutional neural network (SC-CNN) which includes two customized layers: a parameter estimation layer and a spatially constrained layer for spatial regression. As a result, it produces a probability score map based on the distance from the center of the nearest object. In other words, it is very similar to PRTR (see Section 3.4.2). Indeed, a score map is computed in a similar way and linked to the input images of the learning set. The goal is to predict the score map for unseen images.

As far as neural networks are involved, an architecture of successive layers must be defined. Table 3.3 gives the sequence of layers as in [Sir+16] whereas no justification is given. It takes in input, as every CNN, an image of size $w \times h \times d$ where d is the depth of the image. Then, two blocks of "convolution + ReLU + max pooling" are added. The fifth and sixth layers are fully connected layers re-interpreted as convolutional ones. As a remainder, this practice is explained in Section 3.1.2. It can be noticed that the filter of fifth layer has a size which depends on previous output size and thus on the size of the input image. Finally, the two last layers (S1 and S2) are the new spatially constrained layers introduced by SC-CNN, as shown in Figure 3.11.

The penultimate layer $S1$ is the *parameter estimation* layer. Knowing that output subwindow size is fixed to $w' \times h'$, let M be the mean number of objects that should be contained into a single output subwindow. While M is empirically set in [Sir+16], a simple improvement we can propose to have a fully automated procedure

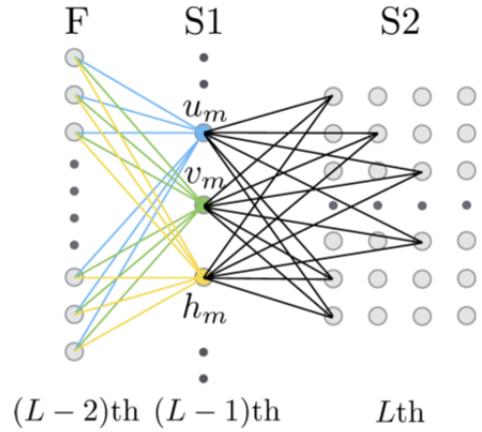


Figure 3.11: Last three layers of SC-CNN, where L is the number of layers

Layer	**Type**	**Parameters**	**Filter size**	**Output size**
0	INPUT			$w \times h \times d$
1	CONV+ReLU	$K = 4; D = 36; S = 1; P = 0$	$4 \times 4 \times 36$	$W_1 \times H_1 \times 36$
2	POOL	$F = 2; S = 2$	2×2	$W_2 \times H_2 \times 36$
3	CONV+ReLU	$K = 3; D = 48; S = 1; P = 0$	$3 \times 3 \times 48$	$W_3 \times H_3 \times 48$
4	POOL	$F = 2; S = 2$	2×2	$W_4 \times H_4 \times 48$
5	FC+ReLU	$D = 512$	$W_4 \times H_4 \times 512$	1×512
6	FC+ReLU	$D = 512$	$1 \times 1 \times 512$	1×512
7	S1		$1 \times 1 \times 3M$	$1 \times 3M$
8	S2			$w' \times h'$

Table 3.3: SC-CNN architecture

is to compute the mean of M over randomly extracted output subwindows from the learning set, before to start training the network. The number of extracted subwindows has still to be set but this hyperparameter is by far less dataset dependent.

The authors of [Sir+16] expect that the number of predicted objects per subwindow would be comprised between 0 and M. For each expected object o_m, with $m = 1, ..., M$, three quantities are computed: u_m, v_m and h_m where $(u_m, v_m) \in s^{out}$ is the location of o_m object and h_m is its probability of existence. According to [Sir+16], they are computed by

$$\begin{aligned} u_m &= w' \ \text{sigm}(\mathbf{w}_{S1,u_m} \cdot \mathbf{x}_{FC}) \\ v_m &= h' \ \text{sigm}(\mathbf{w}_{S1,v_m} \cdot \mathbf{x}_{FC}) \\ h_m &= \text{sigm}(\mathbf{w}_{S1,h_m} \cdot \mathbf{x}_{FC}) \end{aligned}$$

where $\text{sigm}(x)$ is the sigmoid activation function

$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

The number of neurons in the $S1$ layers is thus $3M$.

The last layer $S2$ is the *spatially constrained* layer. It aims to build the predicted output map $\hat{s}^{out}$ of size $w' \times h'$. The score map of the subwindow is calculated using Equation 3.1 where the Euclidean distance transform is obtained from the M objects whose position is given by the (u_m, v_m) of the previous layer. Each pixel x of the resulting subwindow is then scaled by the probability of existence of o_m such that o_m is the nearest object with respect to pixel x. Mathematically,

$$\hat{s}^{out}(x) = S(x)h_m$$

where m is such that $\forall m \neq m', \text{dist}((u_m, v_m), x) < \text{dist}((u_{m'}, v_{m'}), x)$.

Contrary to previous approaches, we had not been able to implement this method. Indeed, it seems difficult to justify theoretically the choices of [Sir+16]. In particular, the usage of layer $S1$ is rather strange. We do not quite understand its usefulness as it would seem more natural to directly predict a score map as output of the convolutional neural network, and without any prior knowledge on the expected number of objects in an output subwindow.

3.5.2.1 Training stage

The SC-CNN model is built using Algorithm 11. The learning set used by the neural networks is composed of all possible subwindows of size $w \times h$ (and their linked output subwindow of size $w' \times h'$) extracted from the N input images. As the number of image examples is crucial in convolutional neural networks, the set is augmented with rotated $(0°, 90°, 180°, 270°)$ and flipped (along horizontal and vertical axis) versions of subwindows.

3.5.2.2 Prediction stage

As the only main difference between SC-CNN and PRTR is the learning algorithm, the prediction stage is similar to Algorithm 8, with the exception of the input subwindow feature extractor $\mathbf{v}$ which can be removed since the features are directly learned by convolutional neural networks.

Algorithm 11. SC-CNNTraining

Input: number of epochs e, batch size b, learning rate $\eta(e)$, weight decay λ, input subwindows size $w \times h$, output subwindows size $w' \times h'$, mean cell radius r, exponential shape of score peaks α, number of subwindows per image N_{sw} (to determine M)

Data: a learning set of N images with annotation maps

$$LS = \{\langle I_i, A_i \rangle | i = 1, ..., N\}$$

Output: a CNN model f built from subwindows

$LS' = \emptyset$
foreach $\langle I_i, A_i \rangle \in LS$ **do**
    $LS' = LS' \cup \{\langle I_i, S_i \rangle\}$
    where S_i is obtained using Equation 3.1 with parameters α and r
end
LS_M =RandomSubwindowExtraction(LS', $w \times h$, $w' \times h'$, N_{sw})
Let M be the mean number of objects per output subwindow from all $s^{out} \in LS_M$
$LS'' =$ ExhaustiveSubwindowExtraction(LS', $w \times h$, $w' \times h'$)
Augment LS'' with rotated and flipped versions of its elements
$f =$ SC-CNN$(e, b, \eta(e), \lambda, M;\ LS'')$
return f

Chapter 4

Experiments and comparison

This chapter applies methods introduced in Chapter 3 to various datasets of microscopy images of human tissues and aims at assessing and validating these methods. The evaluation methodology and protocol regarding model selection and assessment are first described in Section 4.1.

Then, Section 4.2 introduces the design of experiments, explaining in details how the tests will be conducted while the associated obtained results are presented per dataset and per method in Section 4.3. A comparison analysis between the different approaches is given in Section 4.4. Finally, some implementation details are given in Section 4.5.

4.1 Evaluation methodology

4.1.1 Model selection and assessment

Each method has a set of hyperparameters that can be tuned to influence model performances. When it comes to estimate the performance of several models learned from a dataset, the procedure takes place in two stages : *model selection* and *model assessment*. Model selection is the task of selecting the model with the best performance among a set of candidate models learned from a same dataset but with different hyperparameters. Model assessment is the task of assessing the finally chosen model by estimating the *generalization error*, i.e. a measure of the accuracy of model predictions on new, unseen data.

4.1.1.1 Model assessment and test set

A typical approach to assess a supervised machine learning model is to split samples from a dataset into a *learning set* on which the model is trained and a *test set* on which the generalization error is evaluated. This procedure is also known as the *holdout set*. The split must be performed in order to reach two objectives. Firstly, the learning set should contain enough samples to be able to build relevant models. Secondly, the test set should contain enough samples so that the evaluation is significant enough. A common strategy when the number of input samples is high

enough is to keep approximately 70% of available data in the learning set and uses the remaining 30% as test set.

Several scores can be computed for each model of each approach and for each dataset. It allows to compare methods between them based on the scores obtained by the best models.

4.1.1.2 Model selection and cross validation

Values of hyperparameters of the different cell detection or cell counting algorithms have an influence on the model performance. To improve it, the best combination of hyperparameters must be found among a set of hyperparameter combinations provided by the experimenter. The procedure consists in the building of a set of models, namely one for each parameter combination, and assessing them in order to discover the best model and thus the best combination.

The assessment of each model is done through cross-validation. When it was possible, we opted for a *leave one slide out* cross-validation strategy. Concretely, given a learning set of N whole-slide images, N models are learned in turn where the learning set consists in the samples of the $N-1$ slides, the test set being the samples of the slide taken out. The score associated to the model is the average of the N scores generated by the process which is illustrated in Figure 4.1 with $N = 4$. The usage of such a strategy produces almost unbiased estimates since removing only one slide from the learning set does not change too much its size. However, the procedure is slow as it requires to train N distinct models. In the case where N was too large for a leave one slide out CV, *per-slide k-fold* cross-validation had been employed. It is a variation of traditional k-fold strategy which ensures that samples belonging to the same slide are not represented in both testing and training sets. The leave one slide out cross-validation is just a particular case of per-slide k-fold cross-validation where $k = N$.

The choice of *per-slide* cross-validation aims at limiting an overestimation of accuracy. Indeed, having objects coming from a same whole-slide image I both in the learning and the test set could lead to an artificially increased score of a too complex model learning some specific characteristics proper to I.

4.1.1.3 Full procedure

Eventually, the full assessment procedure (see Figure 4.1) consists in the following steps:

1. split the dataset into a learning and a test set of whole-slide images so that they respectively contain 70% and 30% of data.
2. determine the best hyperparameters through an hyperparameter optimization strategy using per-slide k-fold cross-validation on the learning set.
3. learn a model on the entire learning set, using the combination of parameters found in previous step.

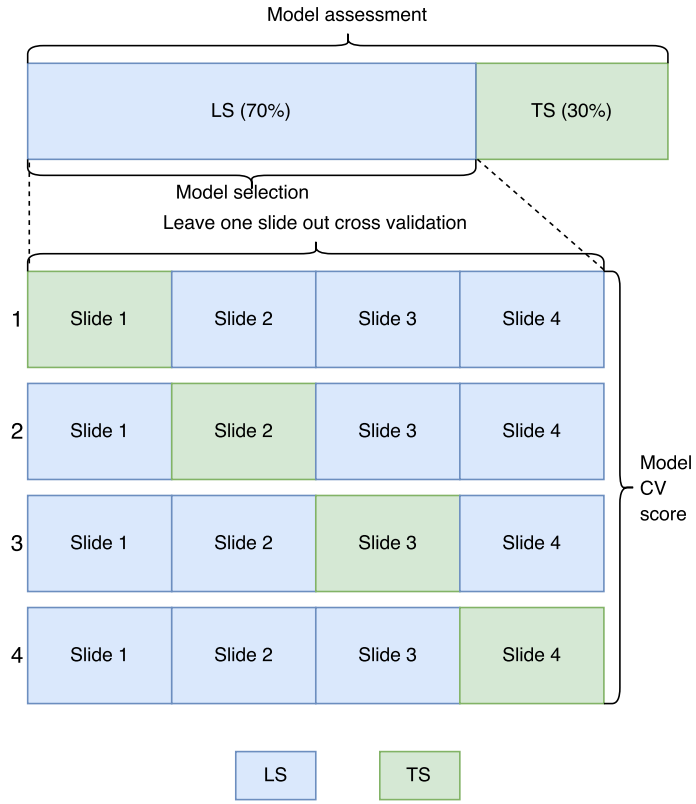


Figure 4.1: Model assessment using test set and model selection using leave one slide out cross-validation.

4. assess the model on the test set.

The selection of *best* parameters and assessment is performed using some defined metrics. It is the object of the next section.

4.1.2 Metrics

Methods yielding to cell detections ([Kai+15], [Sir+16]) are typically evaluated in terms of a binary classification. Results can be summed up in a *confusion matrix*, a $N \times N$ matrix (N being the number of classes) where its element m_{ij} corresponds to the number of objects that are actually associated to the i^{th} class and that are predicted to be the j^{th} one by a model. In the case of a binary classification ($N = 2$), one typically denotes by m_{00} the number of *true positive* (TP), m_{10} the number of *false positive* (FP), m_{01} the number of *false negative* (FN) and m_{11} the number of *true negative* (TN).

Four common metrics can be computed from this binary confusion matrix:

1. **Accuracy :** proportion of correct predictions among all predictions. It is a simple criterion since it does not provide information about how errors are distributed. Accuracy is given by

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

2. **Precision :** proportion of correct predictions among positive predictions. It is thus a measure of results relevancy and is given by

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

3. **Recall :** proportion of positives that are correctly detected. Intuitively, it is the ability of the classifier to find all the positive samples. It is given by

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

4. **F1-Score :** harmonic mean of precision and recall. It can be interpreted as a trade-off between precision and recall and is given by

$$\text{F1-score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

In accordance with [GMR10] who inspired the subsequent publications introducing methods we are studying, the confusion matrix is build using the following guidelines:

1. Consider a distance threshold ε (in pixels) to decide if a detection actually corresponds to a groundtruth dot annotation or not.
2. If the distance between a detection and a groundtruth annotation is less or equal than ε, the detection is a true positive (TP).
3. If several detections have a distance to a same groundtruth annotation which is less or equal than ε, the most confident one (i.e. the one with the highest predicted score) is a true positive (TP) and others are false positives (FP).
4. Detections farther away than ε from any groundtruth dot annotation are false positives (FP).
5. Groundtruth annotations without any close detections are false negatives (FN).

This new distance threshold ε is responsible of localization accuracy. A smaller value of ε will increase the detection confidence since it is harder for a detection to be considered as a true positive. The value of ε is application and dataset dependent. As far as the position of cells is required, a rather strict value is intended.

For those cases where cell center position is significant, the mean Euclidean distance between a true positive and its correctly assigned groundtruth is computed, or equivalently, the mean absolute error (MAE) of the distance between a TP and its correctly assigned groundtruth. Mathematically, if $\mathcal{TP}$ is the set of locations of the true positives detections,

$$\mathbf{MAE}_{dist} = \frac{1}{|\mathcal{TP}|} \sum_{(x_i, y_i) \in \mathcal{TP}} \sqrt{(x_i - x_{gt_i})^2 + (y_i - y_{gt_i})^2}$$

where (x_{gt_i}, y_{gt_i}) is the location of the groundtruth annotation assigned to prediction i. We might notice that we always have $\mathbf{MAE}_{dist} < \varepsilon$.

Methods are also evaluated on the total count of objects identified, computing the absolute difference between the true number of objects and the number of objects found by the method [GMR10], independently of the distance threshold ε. In mathematical terms,

$$\mathbf{MAE}_{count} = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{C}_i - C_i\right|$$

where N is the number of images, $\hat{C}_i = \sum_x \hat{A}_i(x)$ and $C_i = \sum_x A_i(x)$ are respectively the number of detected annotations and the number of ground truth annotations for the image i.

In [Kai+15] this quantity is calculated as the absolute difference between the number of true objects and the number of true positives. It is in fact the mean of the number of false negatives per image which is computed and the number of false positives is not taken into account. It is incorrect to refers to this quantity as $\mathbf{MAE}_{count}$.

It can also be helpful to express the counting mean absolute error in terms of percentage, that can be easily compared, independently of the total number of annotations. We have

$$\mathbf{MAE}_{count,\%} = 100\frac{\sum_{i=1}^{N}\left|\hat{C}_i - C_i\right|}{\sum_{i=1}^{N} C_i}$$

Methods based on density estimation only return a number which is the estimated count of object in the images ([Fia+12], [XNZ15]) and use as metric the mean absolute error on this number. Put differently, it is the mean absolute difference between the estimated count of objects and the number of groundtruth annotations. Mathematically,

$$\mathbf{MAE}_{raw\ count} = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{C}_{raw_i} - C_i\right|$$

where N is the number of images. Also a percentage version $\mathbf{MAE}_{raw\ count,\%}$ can be computed.

As stated in Chapter 3, object locations can be inferred by extraction of local maxima from the density map. Confusion matrix can then be computed and it allows the MAE on the estimated count of object ($\mathbf{MAE}_{raw\ count}$) to be compared with the MAE on the number of detected objects ($\mathbf{MAE}_{count}$).

4.1.2.1 Optimization criteria

At the end of model selection, the *best* model is chosen among a set of candidate models learned from a same dataset. This choice is made by selecting the model

that has the best score according to one of the metrics presented before and is called the *optimization criterion*. Typically, methods coming from the detection world use the F1-score as optimization criterion while methods directly related to counting use the $\mathbf{MAE}_{count}$ (or its equivalent in percentage). In our case, we are interest both in detection and counting. However, it is not at all guaranteed that the model with the best F1-score leads to the best $\mathbf{MAE}_{count}$ and inversely.

An example is shown in Figures 4.2 and 4.3 where F1-score and $\mathbf{MAE}_{count,\%}$ are drawn in function of the value taken by the hyperparameter κ, the post-processing threshold. If we are in the case of Figure 4.2, we could expect good results in terms of counting by choosing the κ that leads to the best F1-score. On the other hand, choosing the κ with the best F1-score in the case of Figure 4.3 conducts to a very poor counting performance.

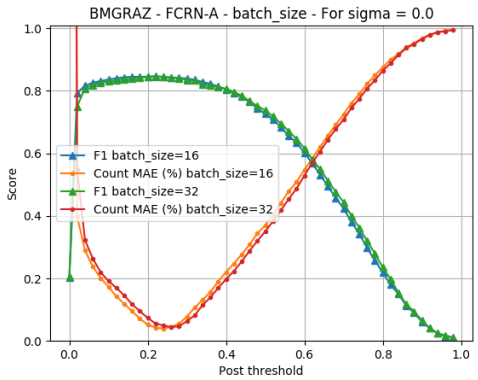


Figure 4.2: The best F1-score leads to a good counting performance

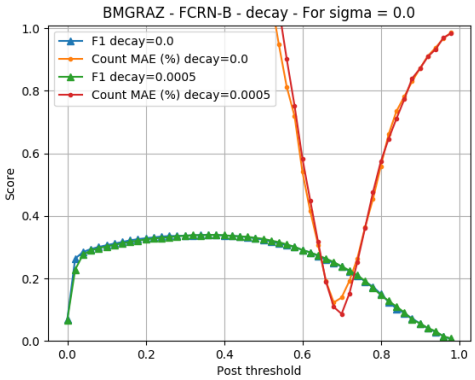


Figure 4.3: The best F1-score leads to a poor counting performance

In order to overcome this issue, we propose a new optimization criterion (OC) based on a weighted average between the normalized F1-score and counting mean absolute error. In this way both criteria are taken into account.

$$OC = \beta\ \overline{\text{F1-score}} + (1-\beta)\ (1 - \overline{\mathbf{MAE}_{count,\%}})$$

where $\overline{x} = \frac{x}{\max x}$ is the normalized score function whose definition domain is the set of all trained models. β is a weight that allows to give more importance to the detection performance ($\beta > 0.5$) or to counting performance ($\beta < 0.5$). We opted for $\beta = 0.5$, which put on a same footing both criteria. It is worth to notice that OC has no physical significance, it only provides a ranking between models.

4.2 Design of experiments

In order to make comparisons between methods, the procedure described in Section 4.1.1.3 is applied for each method and each dataset. During model selection, the hyperparameter optimization is typically performed using a *grid search* strategy which consists in generating a set model candidates in order to make an exhaustive

search across a subset of the hyperparameter space. However, this strategy leads to a highly combinatorial number of experiments to conduct if the number of hyperparameters and their possible values is relatively large.

A possible alternative to overcome this problem is to define a *default model* learned with hyperparameter values fixed either by suggestions from publications proposing some methods, either by the knowledge and the intuition that the experimenter has on the behaviour of these hyperparameters in order to have a great trade-off between computational complexity and model performance. Then, the set of candidate models is completed by taking each hyperparameter and varying its value while other remain fixed to their default value. This strategy has the drawback not to take into account hyperparameter dependencies. On the other hand, it allows to detect trends in terms of model performance with respect to hyperparameter values. This can be wisely used to define a smaller subset of the hyperparameter space on which an exhaustive grid search can be performed.

We opted for a default model search for the hyperparameters, excepting the post-processing hyperparameters for which a grid search strategy is adopted because these hyperparameters do not imply any new model training and require thus low new computation efforts. Other hyperparameters that have to be tuned are described hereafter, each of them with a default value and a set of hyperparameter values to test during model selection.

4.2.1 Pre-processing

When the construction of score map is based on the euclidean distance between cells, a parameter α controlling the exponential shape of score peaks is set by default to 3 as in [Kai+15]. After some preliminary tests, we choose not to tune this parameter as it has low influence on the performances. The mean distance of object is dataset dependent and empirically determined.

When the score map is based on the density of cells per pixel, the standard deviation σ of the Gaussian filter producing density scores is set by default to 1 as suggested in [XNZ15]. Also [Fia+12] and [XNZ15] argue that the parameter does not significantly influence the performances.

4.2.2 Randomized trees

Only squared subwindows have been used in input and output. The size $w \times h$ of input subwindows has been tuned using the following set of values : $\{8 \times 8, 16 \times 16, 24 \times 24, 32 \times 32\}$ and setting the default case to 16×16 input subwindows. The intuition suggests that large subwindows can yield to better results as they capture cell shapes and surrounding environment.

Multi-output regressors or classifiers can be used to predict subwindows larger than 1×1 that can be spatially averaged to produce stronger predictions. The size

$w' \times h'$ of output subwindows has been tuned among $\{1\times1, 2\times2, 4\times4, 8\times8, 16\times16\}$. 1×1 output subwindows have been used as default.

When subwindows extraction is constrained by the score map, t is the score threshold so that all subwindows whose center has a score higher than t are extracted (*positive subwindows*). As suggested in [Kai+15], a default value of $t = 0.4$ has been fixed. The parameter is tuned with the set of values : $\{0.3, 0.4, 0.5\}$ as it is difficult to have an intuition regarding this parameter. Moreover, q is the proportion of negative subwindows (i.e. whose center has a score lower than t) that are added. By default $q = 1.0$, which correspond to a set of samples which an equal amount of positive and negative samples) is used. In classification, a ratio $q = 1.0$ is ideal to balance the set of samples. In the case of regression, a ratio lower than 1 could be used to reduce the number of required samples to train the algorithm. The provided values to be tested are $\{0.25, 0.5, 0.75, 1.0\}$.

When subwindows extraction is performed at random, N_{sw} is the number of randomly chosen input subwindows. By default, it has been set to $N_{sw} = 10000$. Values that have been tested comes from $\{10000, 50000, 100000\}$.

When randomized trees are involved in methods, the number of tree estimators T in the ensemble has to be tuned. The intuition suggests that the higher T is, the better performances are, as the predictions of T trees are averaged which reduces variance. The number of tree estimators T had been set to 10, a moderate value, and trends were estimated with the following set of values : $\{1, 10, 32, 64, 100\}$ where $T = 1$ comes down to a simple decision tree and $T = 32$ and $T = 64$ are determined by cross-validation respectively by [Fia+12] and [Kai+15] for their respective methods and datasets.

The maximum number of features k to evaluate when the decision tree algorithm is looking for the best split has been tuned using the set $\left\{1, \sqrt{d}, d/2\right\}$ with $d = 13 \times w \times h$ where w and h are the size of input subwindows and 13 corresponds to the number of image planes (3 for RGB, 3 for Luv, 3 for HSV, 1 for grayscale, 2 for Sobel filter in x and y direction at first order and 1 for gradient magnitude) as stated in Section 3.3. Moreover in the default case, the subwindow size has been fixed to $w = h = 16$ so that the set of values for k is $\{1, 58, 1664\}$ for a total number of 3328 features. The default value has been set to $\sqrt{d} = 58$.

The minimum number of samples n_{min} required to split a node during tree building has also been tuned. It prevents the algorithm to learn features that are too specific and aims at limiting overfitting. The set $\{2, 10, 100, 200, 600\}$ has been used to tune n_{min}, with a default value of 2 which is the smallest possible value for this parameter and leads to fully developed trees.

It remains to choose the randomized trees algorithm A among standard random forest and Extra-Trees. We chose Extra-Trees as default, in the same way as [Fia+12].

Table 4.1 gives a summary of randomized trees parameters to tune.

Parameter	Description	Default value	Values to test
T	number of tree estimators	10	$\{1, 10, 32, 64, 100\}$
k	max features	$\sqrt{d}$	$\left\{1, \sqrt{d}, d/2\right\}$
n_{min}	min samples split	2	$\{2, 10, 100, 200, 600\}$

Table 4.1: Randomized trees parameters to tune

4.2.3 Convolutional neural networks

The number of epochs e is the number of entire passes over all the dataset during training phase. We use by default $e = 72$ to speed up the computations compared to [Sir+16] which uses 120 epochs and [XNZ15] which uses 192 epochs. These values are nevertheless tested. The batch size b, which is the number of training samples that are used for one gradient update has been tuned with the set $\{16, 32\}$ which are two values suggested in the Keras documentation. By default each batch has a size of 32 samples.

Regarding the learning rate η, its initial value is set by default to 0.01 as suggested by [XNZ15] and this value is tuned using the set $\{0.01, 0.001, 0.0001\}$. These values correspond to the *initial* learning rate. Indeed, as explained in Section 3.1.2, a common practice with CNN is to adjust the learning rate over epochs during training. In all tests, the learning rate is diminished by half its current value every 20 epochs.

The weight decay λ is a regularization term that allows to limit potential overfitting. By default, no regularization is applied ($\lambda = 0$). The value $\lambda = 0.0005$ is also tested. In all cases, the parameters in the network are initialized with an orthogonal basis and no dropout is used as in [XNZ15].

For CNNs, a high number of training images is usually required. In that purpose, the dataset is augmented with randomly flipped and rotated versions of training images. Also, a set of large subwindows is extracted at random from the initial images and are used as training images. In that way, the amount of training data is drastically increased. By default, we use training subwindows of size $w \times h = 256 \times 256$ and we test the values among $\{64 \times 64, 128 \times 128, 256 \times 256, 512 \times 512\}$. The number of training subwindows per image is set by default to 500 and has been tuned using the set $\{50, 500, 1000\}$.

Table 4.2 sums up the convolutional neural network hyperparameters to tune.

Parameter	Description	Default value	Values to test
e	epochs	72	$\{24, 72, 120, 192\}$
b	batch size	32	$\{16, 32\}$
η	initial learning rate	0.01	$\{0.01, 0.001, 0.0001\}$
λ	weight decay	0.0	$\{0.0, 0.0005\}$
$w \times h$	subwindow size (training)	256	$\{64, 128, 256, 512\}$
N_{sw}	subwindow per image (training)	500	$\{50, 500, 1000\}$

Table 4.2: FCRN parameters to tune

4.2.4 Post-processing

The parameters involved during prediction stage influence the performance of a model without influencing the model itself. As previously stated, a grid search is systematically performed to find the best post-processing parameters combination. These parameters are

1. σ the standard deviation of an optional Gaussian kernel used before non maximum suppression in order to merge multiple peaks into a single one, even if such a smoothing can merge together multiple peaks coming from different cells though.

2. κ, the discarding threshold, which is a crucial parameter that influence performance as it discards all local maxima lower than κ.

All discarding thresholds between 0 and 1 are tested by step of 0.02. Regarding the Gaussian kernel the standard deviation is tested among 1.0 and 4.0 as well as without this smoothing procedure. Table 4.3 recapitulates the prediction parameters values tested through a grid search.

Parameter	Description	Values to test
σ	std-dev of optional Gaussian filter	$\{\text{None}, 1.0, 4.0\}$
κ	discarding threshold	$\{0.0, 0.02, ..., 0.98, 1.0\}$

Table 4.3: Parameters for post-processing non maximum suppression

4.3 Tests and results

The content of this section aims at presenting our results and discussing them. For each dataset, we will adopt the following protocol:

1. The dataset is split into a learning set and a test set. Approximately, the former will contain 70% of available data while the test set will receive the remaining 30%. Except for the case where a distribution is already provided, the construction of the two sets has been done empirically in order to achieve as much as possible a 70-30 division in terms of whole slides, images and number of ground truth annotations.

2. Each method is then evaluated according to our design of experiments:

   (a) A first model selection with cross-validation is performed with the default model approach on the learning set, to detect the intuition and trends on the behaviour of each hyperparameter. These are tested while the others remain fixed to their default value. However, as the post processing parameters are required to compute metrics, the results of a test are the ones with the best post processing parameters, chosen from a grid search.

   (b) From the observations of the previous step, a few set of hyperparameters are tested with a grid search approach with cross-validation. It is the model selection as such, returning the set of best hyperparameters.

   (c) The model assessment is then performed on the test set, using the set of best hyperparameters previously determined. It gives the performance estimate for the dataset and the approach.

3. Finally, the different approaches can be compared thanks to the performance estimates of each method. Moreover, they are compared to a simple baseline which is expected to produce lower performance. The principle is simple. First, the mean density of objects per squared pixel D is computed from the learning set. A prediction of the number of objects for an image $w \times h$ is simply Dwh.

The following Table 4.4 gives a summary of all the tests that have been conducted per dataset and per method. The whole procedure has been respected for the datasets BMGRAZ, ANAPATH and GANGLIONS. For the last dataset, CRC, a simple assessment with two fold cross-validation on the whole dataset has been done. The hyperparameters have been inferred from the results of previous experiments and do not have been tuned. It allows to compare the scores with [Sir+16] which has used the same protocol. The same two folds as in [Sir+16] have been used to reduce result variability due to image heterogeneity.

	BMGRAZ	**ANAPATH**	**GANGLIONS**	**CRC**
SRTC	LOO-CV MS + [Kai+15]	LOO-CV MS	LOO-CV MS	2-CV
PRTR	LOO-CV MS + [Kai+15]	LOO-CV MS	LOO-CV MS	2-CV
DRTR	LOO-CV MS	LOO-CV MS	LOO-CV MS	2-CV
FCRN-A	5-CV MS	2-CV MS	3-CV MS	2-CV
FCRN-B	5-CV MS	2-CV MS	3-CV MS	2-CV
SC-CNN	-	-	-	2-CV [Sir+16]

Table 4.4: Our tests per method and per dataset. MS stands for Model Selection.

All experiments have been conducted on supercomputers. Especially, convolutional neural networks have been tested on a Nvidia DGX-1 supercomputer, speeding-up the deep-learning workflows with the usage of GPU-based computations. With 8

GPUs of 16Gb of RAM each, it can gives a computation power equivalent to dozens of CPU-based servers. For randomized trees approaches, the GIGA cluster has been employed.

4.3.1 BMGRAZ dataset

Following what has been done in [Kai+15], the first eight images are used as learning set which represent an amount of 3153 cell centers. The three remaining images make the test set, i.e. 1052 cell centers (Table 4.5).

	images		**cells**	
	11	100%	4205	100%
LS	8	72.72%	3153	74.98%
TS	3	27.28%	1052	25.02%

Table 4.5: Learning set / Test set distribution for BMGRAZ

The mean radius of cells is set to 16 according to [Kai+15]. Also, the minimum distance between two cells, *i.e.* the distance threshold in pixels for metrics computation is $\varepsilon = 8$.

4.3.1.1 Experiments with SRTC

For model selection, the strategy is a leave-one-out cross-validation, that is a 8-fold cross validation as the learning set has eight images. The trends of the studied hyperparameters are given in Table 4.6. In second column, values in italic are those used by the default model. For each line, the results of cross-validation model selection that are given are those with the best post-processing hyperparameters (κ and σ) according to our optimization criterion, required to compute metrics. From this table, several observations can be made:

- As expected, the performance is not satisfying when the number of trees N is small. It seems that $N = 32$ or $N = 64$ is a reasonable choice. $N = 64$ is used in [Kai+15] with the same conditions. It is surprising to notice that the usage of 100 estimators lead to a lower performance than $N = 64$.
- Better results are found when the trees in the forest are fully developed, i.e. the minimum number of samples to split a node is $n_{min} = 2$.
- The maximum number of features when looking for the best split gives better performance when it is set to 1.
- Regarding the size of input subwindows, it is more difficult to decide. We kept sizes of 16 and 32 as possible choices.
- It can be noticed that, as expected, best results are obtained with a post-processing treatment to merge multiple responses for a same object, as we always have $\sigma = 4$.

Param.	Value	κ	σ	F1	Count	%	Dist.
N	1	0.57	4	0.8226	52.38	13.71	3.17
N	*10*	0.60	4	0.8377	41.88	10.96	2.83
N	32	0.59	4	**0.8412**	38.62	10.11	2.80
N	64	0.58	4	0.8402	**36.12**	**9.46**	**2.78**
N	100	0.56	4	0.8396	41.00	10.73	**2.78**
n_{min}	*2*	0.60	4	**0.8377**	**41.88**	**10.96**	2.83
n_{min}	10	0.52	4	0.8342	43.50	11.39	2.81
n_{min}	100	0.51	4	0.8323	59.12	15.48	**2.79**
k	1	0.62	4	**0.8421**	**39.12**	**10.24**	**2.78**
k	*58*	0.60	4	0.8377	41.88	10.96	2.83
k	1664	0.58	4	0.8356	44.75	11.71	2.80
$w \times h$	8	0.63	4	0.8390	43.12	11.29	2.80
$w \times h$	*16*	0.6	4	0.8377	**41.88**	**10.96**	2.83
$w \times h$	24	0.63	4	0.8415	45.75	11.98	2.78
$w \times h$	32	0.63	4	0.8406	48.75	12.76	**2.73**
$w \times h$	50	0.60	4	0.8417	45.12	11.81	2.76
$w \times h$	64	0.62	4	**0.8436**	49.50	12.96	2.77

Table 4.6: STRC hyperparameters trends for BMGRAZ

From this analysis, the search space is reduced and a grid search can now be performed in order to take into account possible dependencies between hyperparameters. Four models are tested with the hyperparameters: $N = \{32, 64\}$, $n_{min} = 2$, $k = 1$, $w = h = \{16, 32\}$. As for other tests, the Extra-Trees algorithm is used with a single output classification. It follows that best model according to our optimization criterion is found with $N = 64$, $w = h = 16$ and the post-processing parameters $\kappa = 0.58$ and $\sigma = 5$. We obtain a F1-score of 0.843, a counting mean absolute error of 36.12 (9.46%) and a mean distance error of 2.778. The model is retrained on the whole learning set with this combination of hyperparameters and assessed on the test set (see Section 4.3.1.6).

4.3.1.2 Experiments with PRTR

For PRTR, the hyperparameter trends are given in Table 4.7, acquired with a leave one image out cross-validation. Again, we try to identify which parameters have an influence on the performance in order to reduce the search space.

Especially, we can deduce that

- better performance is observed with a moderated number of trees ($N = 32$ or $N = 64$), as stated by [Kai+15]. The remark relative to the number of trees made for SRTC is also valid here.
- the minimum number of samples required to split a node n_{min} has a significant influence here, and controls overfitting. Best results are obtained with $n_{min} = 200$.

Param.	Value	κ	σ	F1	Count	%	Dist.
N	1	0.60	1	0.6887	144.25	37.76	**2.77**
N	*10*	0.79	0	0.7910	68.62	17.96	3.32
N	32	0.80	0	**0.8032**	45.38	11.88	3.05
N	64	0.80	0	0.7904	**44.00**	**11.52**	2.91
N	100	0.79	0	0.7749	46.00	12.04	2.86
n_{min}	*2*	0.79	0	0.7910	68.62	17.96	3.32
n_{min}	10	0.80	0	0.8017	74.75	19.57	3.27
n_{min}	100	0.65	1	0.8518	32.00	8.38	2.63
n_{min}	200	0.78	1	**0.8529**	**22.25**	**5.84**	**2.50**
n_{min}	600	0.78	1	0.8406	23.12	6.10	2.53
k	1	0.78	0	0.7863	110.12	28.83	3.57
k	*58*	0.79	0	**0.7910**	**68.62**	**17.96**	**3.32**
k	1664	0.78	0	0.7798	103.42	27.02	3.42
$w \times h$	8	0.77	0	0.7352	199.38	52.19	4.13
$w \times h$	*16*	0.79	0	0.7910	68.62	17.96	3.32
$w \times h$	24	0.80	0	0.8151	40.62	10.63	2.95
$w \times h$	32	0.80	0	**0.8227**	39.12	10.24	**2.87**
$w \times h$	50	0.79	0	0.8163	**37.75**	**9.88**	2.98
$w' \times h'$	*1*	0.79	0	**0.7910**	68.62	17.96	3.32
$w' \times h'$	2	0.80	0	0.7982	40.25	10.54	3.02
$w' \times h'$	4	0.82	0	0.7708	40.88	10.70	2.75
$w' \times h'$	8	0.82	0	0.7684	44.75	11.71	2.63
$w' \times h'$	16	0.75	0	0.7767	**21.75**	**5.69**	**2.56**
t	0.2	0.53	1	0.8201	44.50	11.65	**2.58**
t	0.3	0.56	1	**0.8372**	**34.25**	**8.97**	2.61
t	*0.4*	0.79	0	0.7910	68.62	17.96	3.32
t	0.5	0.80	0	0.7822	142.62	37.34	3.37
q	0.25	0.79	0	0.7853	107.50	28.14	3.35
q	0.5	0.79	0	0.7939	98.25	25.72	3.35
q	0.75	0.79	0	**0.7965**	99.50	26.05	3.36
q	*1*	0.79	0	0.7910	**68.62**	**17.96**	**3.32**

Table 4.7: PRTR hyperparameters trends for BMGRAZ

- $k = \sqrt{d} = \sqrt{16 \cdot 16 \cdot 13}$ i.e. the square root of the number of features is the best choice.
- larger input subwindows gives better results. We choose to keep the subwindows' sizes of 24 and 32 as subwindows of 50×50 gives small improvement but also require much more time during training.
- surprisingly, slightly better results are obtained in terms of F1-score with a single output regression. However, our improvement to the original method with spatial averaging with multi-output regression clearly reduce the counting and distance error. We notice a huge decrease of counting error between output subwindows of size 8 and 16.

- as expected, increasing the size of learning set improves the performance. Indeed, by decreasing the threshold t, more subwindows are extracted and $t = 0.3$ gives best results. Also, balancing the dataset lead to better performance ($q = 1$).

Thanks to these observations, we make a grid search over the following parameters: $N = \{32, 64\}$, $n_{min} = 200$, $k = \sqrt{13wh}$, $w = h = \{24, 32\}$, $w' = h' = \{1, 16\}$, $t = 0.3$, $q = 1$ and Extra-Trees as learning algorithm. Among the four models considered, it results that the best one takes $N = 32$, 24×24 input subwindows, 16×16 output subwindows, $\kappa = 0.63$ and $\sigma = 1$. In terms of performances, we have a F1-score of 0.857, a counting mean absolute error of 29.97 cells per image (7.64%) and a distance error of 2.534. The assessment of this model on the test set is detailed in Section 4.3.1.6.

4.3.1.3 Experiments with DRTR

Param.	Value	κ	σ	F1	Count	%	Raw	%	Dist.
N	1	0.10	4	0.8520	46.62	12.21	**45.01**	**11.78**	2.66
N	*10*	0.10	4	0.8546	**45.75**	**11.98**	45.21	11.84	2.61
N	32	0.11	4	0.8559	50.00	13.09	45.08	11.80	**2.59**
N	64	0.10	4	0.8555	54.62	14.30	45.97	12.03	2.61
N	100	0.10	4	**0.8563**	54.38	14.23	46.09	12.07	2.60
n_{min}	*2*	0.10	4	0.8546	**45.75**	**11.98**	45.21	11.84	**2.61**
n_{min}	10	0.10	4	**0.8549**	54.25	14.20	45.46	11.90	**2.61**
n_{min}	100	0.13	4	0.8526	46.62	12.21	**38.25**	**10.01**	2.62
k	1	0.14	4	0.8553	46.00	12.04	45.67	11.96	2.64
k	*58*	0.10	4	0.8546	45.75	11.98	**45.21**	**11.84**	2.61
k	1664	0.11	4	**0.8584**	**41.00**	**10.73**	47.94	12.55	**2.59**
N_{sw}	*10000*	0.10	4	0.8546	45.75	11.98	**45.21**	**11.84**	2.61
N_{sw}	50000	0.11	4	**0.8586**	41.25	10.80	46.16	12.08	2.56
N_{sw}	100000	0.11	4	0.8575	**38.62**	**10.11**	47.11	12.33	**2.52**
$w \times h$	8	0.12	4	0.8480	**35.75**	**9.36**	58.80	15.39	2.83
$w \times h$	*16*	0.10	4	0.8546	45.75	11.98	45.21	11.84	2.61
$w \times h$	24	0.11	4	**0.8568**	50.12	13.12	41.52	10.87	2.54
$w \times h$	32	0.11	4	0.8538	47.12	12.34	**40.68**	**10.65**	**2.52**

Table 4.8: DRTR hyperparameters trends for BMGRAZ

A set of experiments is conducted to reduce the hyperparameter search space as previously. With DRTR, it is difficult to detect what are the best hyperparameters. Indeed, all metrics do not significantly vary. Moreover, choosing hyperparameters improving raw count very often reduces the performance from a detection point of view. However, we are particularly interested in the counting with visualization of counted objects. Therefore, we try two models, either with subwindow size (input and output) of 16×16 either with 32×32. Other parameters are fixed: $n_{min} = 2$, $k = \sqrt{13wh}$ and $N_{sw} = 50000$. The best model is finally the one with 32×32

subwindows, according to our optimization criterion. The F1-score is 0.861 and the counting error corresponds to 38.12 cells per image (10.05%). Contrary to what might have been thought, the raw counting error is also decreased, and even better than the counting error: 33.802 cells per image (8.23%). We also have a distance error of 2.508.

4.3.1.4 Experiments with FCRN-A

Due to limited time slots on the GDX-1 supercomputer, we adopted a 5-fold cross validation strategy for model selection. Although the method is initially designed to count by density (assessed with the mean raw count absolute error), a first ascertainment that can be made from Table 4.9 is the fact that the method tends to produce better results in terms of cell detection (assessed with the F1-score and the mean counting absolute error). As for PRTR, we made the optimization of hyperparameters with detection performance in mind.

Param.	**Value**	κ	σ	**F1**	**Count**	**%**	**Raw**	**%**	**Dist.**
e	24	0.24	0	0.8380	19.38	4.96	93.24	29.65	2.47
e	*72*	0.18	0	**0.8423**	17.16	4.41	**34.53**	**8.81**	2.41
e	120	0.26	0	0.8415	**12.00**	**3.04**	56.98	19.32	**2.37**
e	192	0.24	0	0.8379	16.12	4.01	42.24	9.19	2.38
b	16	0.18	1	**0.8461**	**15.75**	**4.03**	45.57	10.99	**2.36**
b	*32*	0.18	0	0.8423	17.16	4.41	**34.53**	**8.81**	2.41
η	0.0001	0.20	1	0.8216	18.75	4.70	**32.20**	**7.65**	2.51
η	0.001	0.26	0	0.8404	**14.62**	**3.70**	34.43	8.79	**2.35**
η	*0.01*	0.18	0	**0.8423**	17.16	4.41	34.53	8.81	2.41
λ	*0.0*	0.18	0	**0.8423**	**17.16**	**4.41**	**34.53**	**8.81**	**2.41**
λ	0.0005	0.26	0	0.8388	17.25	4.42	43.15	9.38	2.44
N_{sw}	50	0.18	0	0.8220	23.00	5.86	135.59	34.33	2.57
N_{sw}	*500*	0.18	0	0.8423	17.16	4.41	**34.53**	**8.81**	**2.41**
N_{sw}	1000	0.18	1	**0.8434**	**16.25**	**4.13**	40.01	10.21	2.38
$w \times h$	64	0.24	0	**0.8475**	20.38	5.04	85.20	28.99	2.44
$w \times h$	128	0.24	0	0.8314	24.62	6.23	123.30	33.13	2.47
$w \times h$	*256*	0.18	0	0.8423	**17.16**	**4.41**	34.53	8.81	**2.41**
$w \times h$	512	0.24	0	0.8329	22.88	5.89	**34.30**	**8.45**	2.46

Table 4.9: FCRN-A hyperparameters trends for BMGRAZ

From these observations, we perform a grid search with four models built from the four possible combinations of hyperparameters' values that we have retained. We set the number of epochs e to 120, the batch size b to 16 and no decay. The initial learning rate η is tested among the set of values $\{0.01, 0.001\}$. For training 1000 subwindows are extracted per image, with a size either of 64 or 256 pixels. The model with the best performance has a learning rate of 0.01 and uses subwindows of size 64×64. The F1-score is 0.85, the mean count error is 17.5 cells per image (thus 4.37% of error per image). Regarding the raw count, the error is of 14.56 cells per images (3.67% of error per image). It is worth to notice that even if we optimized

on a criterion only based on detection performance, the raw count error is very low. The assessment of the model on the test set is given in Section 4.3.1.6.

4.3.1.5 Experiments with FCRN-B

The tests have been conducted in the same way as with FCRN-A. The same observations can be made, and the same four combination of hyperparameters is tested on this architecture, with the exception of the number of epochs e which is set to 192 here. The major difference between FCRN-A and FCRN-B results is the post-processing σ, which is essential to obtain acceptable performances. Indeed, we noticed in our tests that with $\sigma = 0$, whatever the test, the F1-score drops to around 0.20.

Param.	Value	κ	σ	F1	Count	%	Raw	%	Dist.
e	24	0.20	1	0.8132	28.37	7.25	84.61	28.35	2.45
e	*72*	0.21	1	0.8301	17.84	4.52	67.40	16.93	2.42
e	120	0.22	1	0.8227	20.00	5.09	37.39	9.51	2.43
e	192	0.22	1	**0.8366**	**17.25**	**4.39**	**27.68**	**7.04**	**2.39**
b	16	0.22	1	**0.8411**	**17.75**	**4.52**	**39.10**	**9.68**	**2.40**
b	*32*	0.21	1	0.8301	17.84	4.59	67.40	16.93	2.42
η	0.0001	0.20	1	0.8151	23.88	6.07	**63.4**	**16.06**	2.59
η	0.001	0.20	1	**0.8359**	**17.62**	**4.48**	66.97	16.96	**2.37**
η	*0.01*	0.21	1	0.8301	17.84	4.52	67.40	16.93	2.42
λ	*0.0*	0.21	1	**0.8301**	**17.84**	**4.52**	67.4	16.93	**2.42**
λ	0.0005	0.22	1	0.8244	29.25	7.65	**42.60**	**10.35**	2.48
N_{sw}	50	0.22	1	0.7916	27.38	6.95	118.61	30.03	2.85
N_{sw}	*500*	0.21	1	0.8301	17.84	4.52	67.40	16.93	2.42
N_{sw}	1000	0.06	4	**0.8441**	**17.12**	**4.36**	**20.14**	**5.14**	**2.37**
$w \times h$	64	0.22	1	0.8170	22.5	5.72	**41.40**	**10.53**	2.50
$w \times h$	128	0.06	4	0.8265	26.00	6.63	140.33	35.83	2.47
$w \times h$	*256*	0.21	1	0.8301	**17.84**	**4.52**	67.40	16.93	**2.42**
$w \times h$	512	0.08	4	**0.8309**	23.00	5.85	74.96	19.02	2.46

Table 4.10: FCRN-B hyperparameters trends for BMGRAZ

The best model uses 256×256 input subwindows and an initial learning rate of 0.001. A F1-score of 0.86 is achieved during model selection on the learning set, with a counting error of 15.5 (3.89% of error per image) and slightly larger raw counting error of 63.21 (16.83% of error per image).

4.3.1.6 Comparison

The comparison is done on the test set which is composed of new, unseen images. In Table 4.12, each line presents the model selection performance on the learning set and the model assessment performance on the test set, for a particular method. The set of hyperparameters used to build models is given in Table 4.11. These parameters (including post-processing ones) are determined by model selection and

used to train the final model assessed on the test set. The area under curve (AUC) on the test set for each method is shown in Figure 4.4. PR curves are obtained by varying the post-processing threshold κ.

Method	T	n_{min}	k	e	b	η	λ	t	q	N_{sw}	$w = h$	$w' = h'$	κ	σ
SRTC	64	2	1	-	-	-	-	-	-	-	16	1	0.58	5
PRTR	32	200	87	-	-	-	-	0.3	1	-	24	16	0.63	1
DRTR	32	2	115	-	-	-	-	-	-	50000	32	32	0.10	4
FCRN-A	-	-	-	120	16	0.01	0.0	-	-	1000	64	-	0.22	0
FCRN-B	-	-	-	192	16	0.001	0.0	-	-	1000	256	-	0.06	4

Table 4.11: Best models for BMGRAZ

	Model selection on LS						**Model assessment on TS**					
Method	**F1**	**Count**	**%**	**Raw**	**%**	**Dist.**	**F1**	**Count**	**%**	**Raw**	**%**	**Dist.**
Baseline	-	35.06	8.90	35.06	8.90	-	-	43.46	12.39	43.46	12.39	-
SRTC	0.8430	36.12	9.46	-		2.78	0.7926	68.00	19.39	-		2.64
PRTR	0.8570	29.97	7.64	-		2.53	0.8343	38.33	10.93	-		2.32
DRTR	**0.8612**	38.12	10.05	33.80	8.23	2.51	**0.8495**	37.33	10.65	88.56	25.27	2.34
FCRN-A	0.8471	17.50	4.37	**14.56**	**3.67**	**2.36**	0.8440	**21.33**	**6.08**	**34.66**	**9.89**	2.35
FCRN-B	0.8415	**15.50**	**3.89**	63.21	16.83	2.37	0.8441	36.33	10.36	178.28	47.87	**2.30**

Table 4.12: Comparison of best models performance for BMGRAZ

With the exception of SRTC, all methods have better performance than our simple baseline, and yield to nearly similar results. However, we can especially highlight the performance of FCRN-A which is able to reach a counting error of 6.08% and a raw counting error of 9.89%, always better than the counting error of other methods. Also, it is the only model to not require smoothing post-processing ($\sigma = 0$).

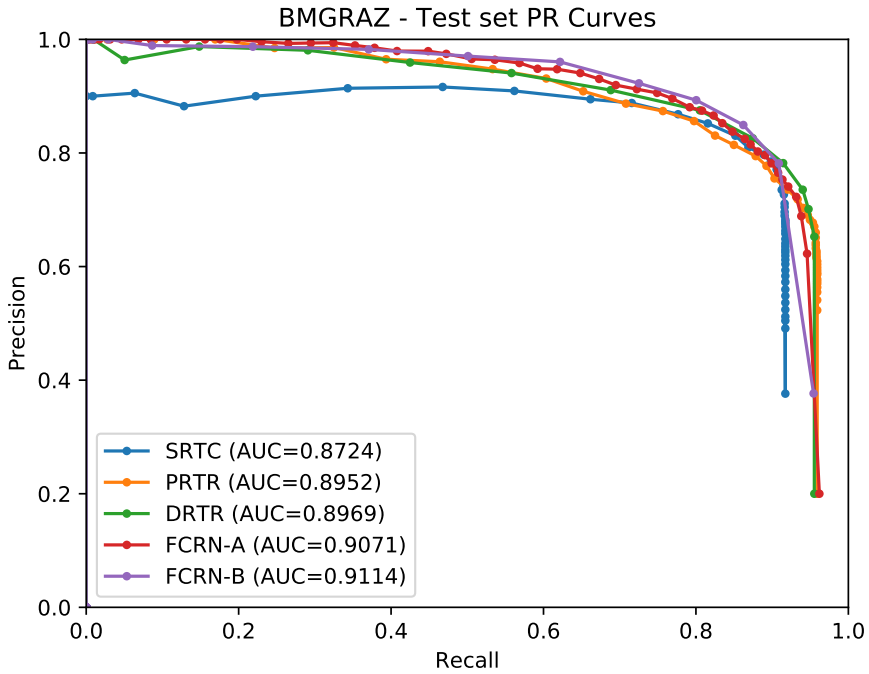


Figure 4.4: PR curves for methods applied on BMGRAZ

4.3.2 GANGLIONS dataset

From the 11 whole slides, the first eight ones are used as learning set which represents 4716 dot annotations for 51 region of interest images. The three remaining slides compose the test set, meaning 1680 dot annotations and 15 images, as shown in Table 4.13.

	slides		**images**		**cells**	
	11	100%	66	100%	6396	100%
LS	8	72.73%	51	77.27%	4716	73.73%
TS	3	27.27%	15	22.73%	1680	26.27%

Table 4.13: Learning set / Test set distribution for GANGLIONS

The mean radius of cells has been determined empirically and set to 2 pixels. Also, the minimum distance between two cells, *i.e.* the distance threshold in pixels for metrics computation is $\varepsilon = 6$.

For the sake of clarity, the results of model selection are not given here. The interested reader can find them in Appendix A. Table 4.14 presents the hyperparameters that lead to the best models during the model selection. These are used to train the final model which is assessed on the test set. Results are given in Table 4.15 and precision-recall curves are drawn in Figure 4.5.

SRTC is again the method with the lowest scores relative to cell localization (AUC and F1-score). Contrary to BMGRAZ, FCRN-A is not the best except on

Method	T	n_{min}	k	e	b	η	λ	t	q	N_{sw}	$w = h$	$w' = h'$	κ	σ
SRTC	32	10	58	-	-	-	-	-	-	-	16	1	0.68	1
PRTR	32	200	58	-	-	-	-	0.3	1.0	-	16	16	0.42	0
DRTR	32	20	115	-	-	-	-	-	-	50000	32	32	0.32	0
FCRN-A	-	-	-	120	16	0.0001	0.0005	-	-	500	256	-	0.26	0
FCRN-B	-	-	-	72	16	0.0001	0.0005	-	-	500	128	-	0.22	1

Table 4.14: Best models for GANGLIONS

	Model selection on LS						Model assessment on TS					
Method	**F1**	**Count**	**%**	**Raw**	**%**	**Dist.**	**F1**	**Count**	**%**	**Raw**	**%**	**Dist.**
Baseline	-	32.67	35.33	32.67	35.33	-	-	40.44	36.11	40.44	36.11	-
SRTC	0.7681	18.04	19.01	-		4.00	0.7634	17.47	15.60	-		4.10
PRTR	0.7914	15.68	16.28	-		3.87	**0.8285**	**14.67**	**13.10**	-		3.62
DRTR	0.7729	15.82	16.31	16.32	17.32	3.92	0.8260	15.80	14.11	22.70	20.26	**3.42**
FCRN-A	**0.8120**	**13.25**	**15.22**	15.19	16.18	4.33	0.7930	25.67	22.92	**10.57**	**9.44**	4.19
FCRN-B	0.8009	13.58	15.95	**14.11**	**16.01**	**3.90**	0.8207	15.33	13.69	13.41	11.97	3.70

Table 4.15: Comparison of best models performance for GANGLIONS

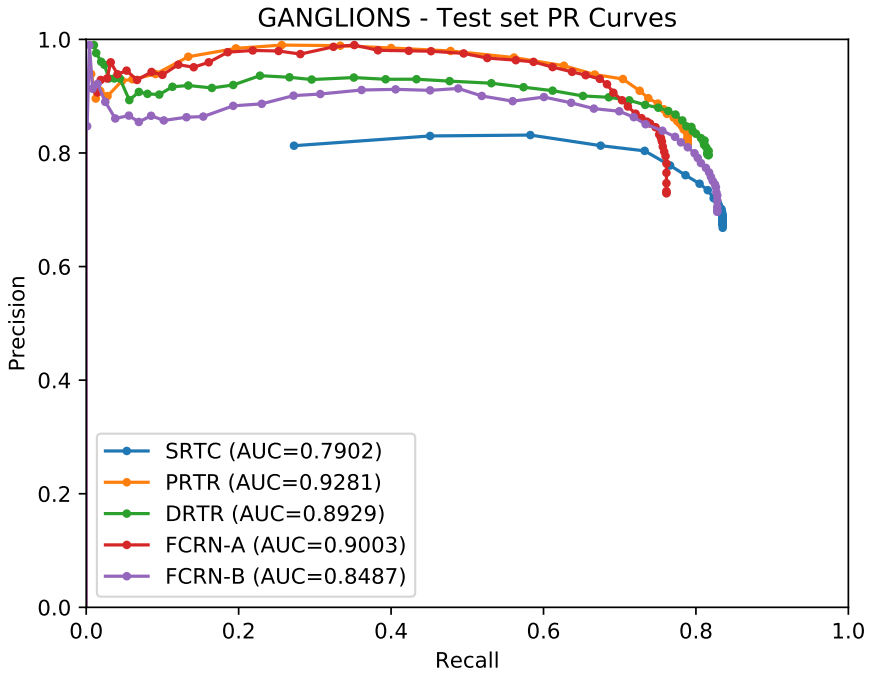


Figure 4.5: PR curves for methods applied on GANGLIONS

the raw counting error which drops to 9.44%. PRTR wins on the counting error from detection and the detection F1-score (13.10% and 0.8285 repectively).

4.3.3 ANAPATH dataset

From the six regions of interest extracted from the single whole slide, the four ones are chosen as learning set and correspond to the ones that achieve at best the goal of 70% of groundtruth annotations. 11328 annotations are in the learning set while the 5086 remaining annotations make the test set (Table 4.16).

	images		**cells**	
	6	100%	16414	100%
LS	4	66.67%	10943	66.67%
TS	2	33.33%	5471	33.33%

Table 4.16: Learning set / Test set distribution for ANAPATH

The mean radius of cells has been determined empirically and set to 2 pixels. Also, the minimum distance between two cells, *i.e.* the distance threshold in pixels for metrics computation is $\varepsilon = 8$.

Again, detailed results regarding model selection on ANAPATH dataset can be found in Appendix B. The parameters determined by this procedure are given in Table 4.11 and the comparison of model performances is in Table 4.12.

Method	T	n_{min}	k	e	b	η	λ	t	q	N_{sw}	$w = h$	$w' = h'$	κ	σ
SRTC	32	200	87	-	-	-	-	-	-	-	24	1	0.70	4
PRTR	32	400	58	-	-	-	-	0.4	1.0	-	16	8	0.52	4
DRTR	32	100	115	-	-	-	-	-	-	50000	32	32	0.52	4
FCRN-A	-	-	-	120	16	0.01	0.0005	-	-	500	256	-	0.20	1
FCRN-B	-	-	-	120	16	0.01	0.0005	-	-	1000	64	-	0.24	5

Table 4.17: Best models for ANAPATH

	Model selection on LS						**Model assessment on TS**					
Method	**F1**	**Count**	**%**	**Raw**	**%**	**Dist.**	**F1**	**Count**	**%**	**Raw**	**%**	**Dist.**
Baseline	-	227.35	8.03	227.35	8.03	-	-	394.88	15.53	394.88	15.53	-
SRTC	0.5833	234.75	8.58	-		5.78	0.4962	493.41	19.41	-		6.01
PRTR	**0.6129**	207.42	7.59	-		5.74	**0.5312**	301.16	11.85	-		5.81
DRTR	0.5985	203.13	7.44	347.47	12.71	5.76	0.5107	310.24	12.20	427.45	16.81	5.80
FCRN-A	0.4649	126.75	4.46	302.19	**10.68**	**4.76**	0.3973	226.50	**8.91**	274.94	**10.81**	**4.81**
FCRN-B	0.4388	95.75	**3.38**	475.02	16.84	4.79	0.3633	31.008	12.51	727.49	28.61	4.82

Table 4.18: Comparison of best models performance for ANAPATH

We can notice that all methods have lower counting error than the simple baseline except for the standard classification with randomized trees (SRTC). Like for BMGRAZ dataset, FCRN-A is the only one with a counting error lower than 10%. In this case, a slight post-processing smoothing is performed ($\sigma = 1$). Other methods need a larger smoothing for this dataset.

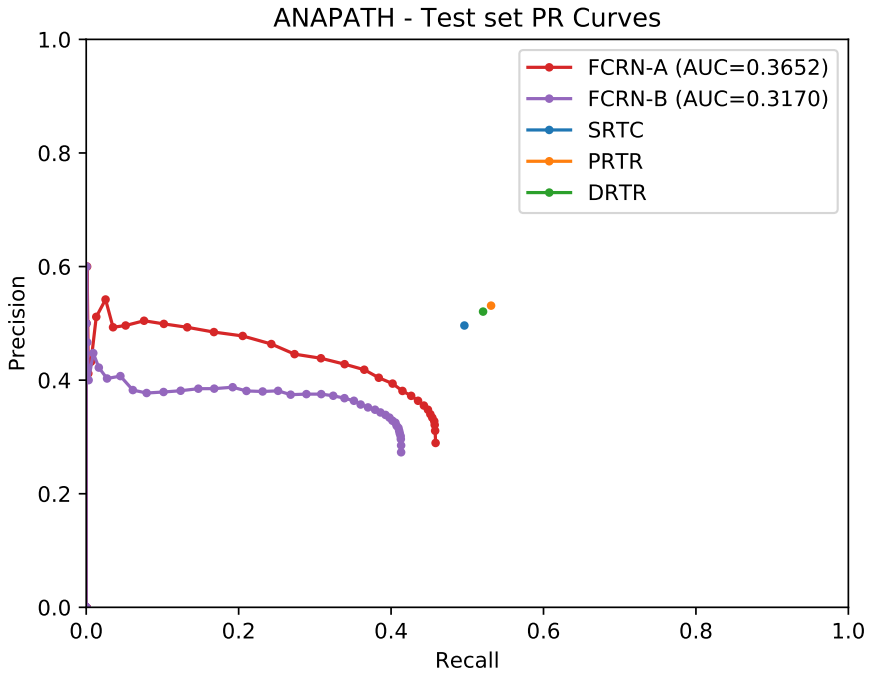


Figure 4.6: PR curves for methods applied on ANAPATH

However, localization results are disappointing, especially for CNN-based approaches. As for GANGLIONS dataset, the best F1-score is given by PRTR. In Figure 4.6, we show the precision-recall curves for the CNN-based methods. For others, we show the results with the best post-processing threshold κ, given from Table 4.17.

4.3.4 CRC dataset

Here, a simple assessment with two fold cross-validation on the whole dataset has been done. The hyperparameters have been inferred from the results of previous experiments (especially GANGLIONS whose objects to detect have similar size) and do not have been tuned, with the exception of post-processing parameters. It allows to compare the scores with [Sir+16] which has used the same 2-fold CV protocol. The same two folds as in [Sir+16] have been used to reduce result variability due to image heterogeneity. The set of hyperparameters is shown in Table 4.19 and the associated results in Table 4.20. The PR curves are in Figure 4.7.

We had been never able to reach the F1-score (0.8020) stated by [Sir+16] with our methods. At best, PRTR shows a F1-score of 0.7365 (-6.55%). In terms of counting, PRTR is the only one to report a error per image lower than 10%.

The results of convolutional neural networks are particularily low on this dataset. We expect that our FCRN models are overfitting. Compared with GANGLIONS, from which the hyperparameter's values are inspired, CRC contains more samples

Method	T	n_{min}	k	e	b	η	λ	t	q	N_{sw}	$w = h$	$w' = h'$	κ	σ
SRTC	32	100	58	-	-	-	-	-	-	-	16	1	0.54	4
PRTR	32	400	58	-	-	-	-	0.35	1.0	-	16	16	0.46	4
DRTR	32	2	115	-	-	-	-	-	-	10000	32	32	0.20	1
FCRN-A	-	-	-	120	16	0.001	0.0005	-	-	50	256	-	0.10	1
FCRN-B	-	-	-	120	16	0.001	0.0005	-	-	50	256	-	0.10	1

Table 4.19: Best models for CRC

	2-fold CV assessment					
Method	**F1**	**Count**	**%**	**Raw**	**%**	**Dist.**
Baseline	-	96.51	32.27	96.51	32.27	-
SRTC	0.6329	77.37	25.87	-		2.89
PRTR	0.7365	**28.74**	**9.61**	-		2.26
DRTR	0.6853	35.12	11.75	**49.62**	**16.61**	2.17
FCRN-A	0.6120	64.39	21.28	59.94	20.00	1.69
FCRN-B	0.6553	57.62	19.42	58.85	19.73	**1.56**
SCCNN (M=1) [Sir+16]	0.7910	-		-		2.24
SCCNN (M=2) [Sir+16]	**0.8020**	-		-		2.24

Table 4.20: Comparison of best models performance for CRC

in its learning set. In order to verfiy this hypothesis, a model selection should be performed, especially on the decay parameters λ. Also, dropout could be used.

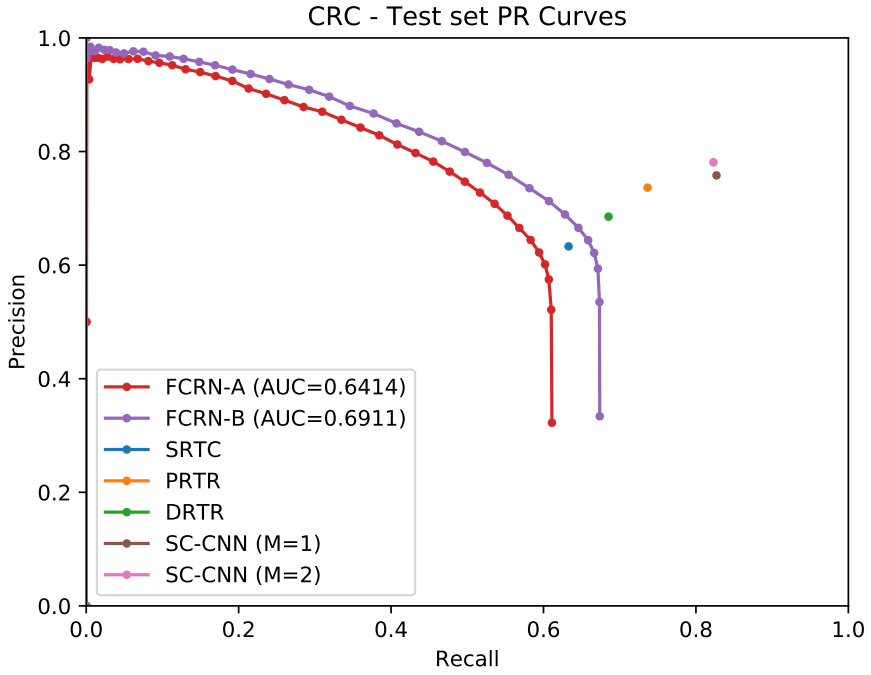


Figure 4.7: PR curves for methods applied on CRC

4.4 Overall comparison

From a per-dataset point of view, best results are reported for BMGRAZ and GANGLIONS to a lower extent, while ANAPATH and CRC globally have mixed results. For CRC, an explanation could be the weakness of the model selection protocol. For ANAPATH, a possible source of error could come from approximative annotations from experts. Indeed we noticed that two experts actually encoded annotations, but these have not been reviewed. It should be investigated in more details but could demonstrates the importance of correctly labelled annotations.

From a per-method perspective, it seems clear that SRTC can be discarded. It never wins on any criteria and in two datasets out of four, a simple baseline is able to do better.

CNN-based methods (FCRN-A and FCRN-B) report impressive results in terms of counting. In two datasets, FCRN-A is the best for the two counting metrics. In a third one, it is the best on the raw counting error. Generally, FCRN-A exhibits slightly better performance than FCRN-B (with the notable exception of CRC, where networks seems to overfit).

In terms of localization, the best F1-score is given by PRTR three times out four and by DRTR once, but differences are not significant. In BMGRAZ and GANGLIONS, convolutional networks also show similar F1-score.

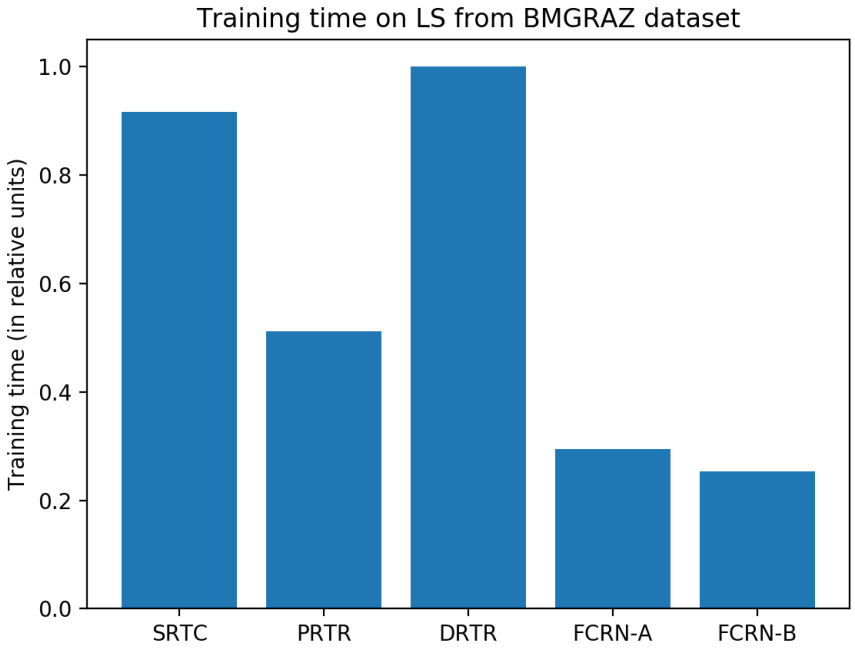


Figure 4.8: Training time for LS from BMGRAZ dataset

Finally, Figure 4.8 is a comparison of training times on the BMGRAZ dataset. For SRTC, PRTR and DRTR, training has been done on 4 CPUs. For FCRN, one CPU and one GPU have been used. It is clear that the training time of forest depends on the depth of each estimator and explains the peaks for DRTR and SRTC which fully develop the trees on this dataset. While the observation we are doing here is empirical and cannot be used as a general conclusion, it shows the power of neural networks training on GPUs. They are known to take several days of training on CPUs. With only one GPU, we are able to observe training times lower than randomized trees (trained on four CPUs).

From these observations, the best compromise seems to be FCRN-A, which shows impressive counting results and reasonable detection performance and low training time, provided that it is trained on GPU.

4.5 Implementation

4.5.1 Language and libraries

One of the purposes of this work is to provide new *softwares* in Cytomine (see Section 2.3) answering the problem of cell counting and/or detection. The approaches previously presented were very often provided with some pieces of code published by their authors. For instance, [Kai+15] provides an hybrid C++/MATLAB implementation while [Art+12], [XNZ15] and [Sir+16] supplies a full MATLAB implementation. MATLAB is a commercial software providing a computing environment and a programming language but suffers from some limitations. First, the built-in algorithms are proprietary which makes impossible to know how these algorithms are implemented. This proprietary nature puts also restrictions on the code portability of MATLAB programs. Finally, the usage licence is quite expensive and its commercial nature do not meet the requirements of an open-source project such as Cytomine.

Beyond that, there exists no binding between Cytomine and MATLAB in order to communicate between the Cytomine back-end server and the actual software implementation. However, such a binding is essential. Indeed, as the final goal is to deal with multi-gigapixels images stored on Cytomine servers and to visualize cells localization on the platform, it is required that the newly developed algorithms exchange images and metadata such as annotations through the Cytomine API. The current Cytomine release provides API bindings for Python and Java. The decision has thus been taken to (re-)implement the algorithms in one of these languages. The choice ultimately focused on Python for a couple of reasons that will be expressed below.

Python is a free, open-source, portable and powerful programming language. It allows several programming paradigms such as imperative, object-oriented and functional ones. Also, the built-in features gives manipulations for high-level data structures such as lists, tuples and dictionaries. Moreover, it benefits of a large

and reactive community of developers and comes with a lot of efficient third-party libraries. First, the SciPy ecosystem which includes

- SciPy [Oli07], a fundamental library for scientific computing such as statistics
- NumPy [VCV11], a fundamental package for efficient representation and manipulation of multi-dimensional arrays, but also tools for random numbers
- Matplotlib [Hun07], a powerful library for 2D plotting
- pandas [McK10], a library to simplify data representation and data analysis

Scikit-learn [Ped+11] comes on top of the SciPy ecosystem and gives easy-of-use and efficient tools for standard machine learning. These libraries have gain a large popularity among the scientific community and Python is now one of the languages of choice for data sciences. The particular case of deep learning has seen the emergence of specific libraries, such as Keras [Cho+15], a high-level API for neural network built on top on TensorFlow [Mar+15], a CPU/GPU multidimensional array processing tailored for deep learning.

Regarding image processing, the well-known OpenCV library [Bra00] is provided with a Python binding. Scikit-image [Wal+14] is an alternative based on the SciPy ecosystem which fills the gaps of the former. To complete the list, Joblib [Var10] has been used to perform parallel computations and Shapely [Gil13] has been used to represent geometrical objects.

4.5.2 Other details

The implementation itself was not often the main issue, the trick was to elegantly combine the best of all libraries. However, we rapidly realized that methods with randomized trees tend to consume a very high and significant amount of memory. Each subwindow is stored as an array of floating numbers. In these methods, some memory-consuming situations can occurs. It is the case for example if the subwindow size is large, the number of subwindows to extract is high and the feature extractor uses a lot of image filters. It was especially the case in the current release of Scikit-learn (0.18) for the randomized trees implementation of prediction phase. The development version (0.19) fixes the issue. The problem was that predictions were done on each estimator of the forest and combined only at the end. The new implementation substantially decreases the memory usage by directly summing each output prediction[1].

Also, it has been envisaged to use SLDC [Mor16], a framework which aims at accelerating the development and execution of multi-gigapixel images analysis workflows. It would allow to reduce prediction time of our algorithms. However, the integration of this framework has been set aside because it does not fully meet our needs. Indeed, SLDC is designed to directly segment an image and to extract objects (polygon) from it while our algorithms produce a map of real values and not always

[1] See https://github.com/scikit-learn/scikit-learn/pull/8672

objects. For example, the approach taken by SLDC would not make it possible to compute the counting by integration on the image domain.

The implementation of model selection and assessment is inspired from the code of the Scikit-Learn module `sklearn.model_selection`. Some adaptations were necessary. First, the way our metrics are computed is particular. For example, the generic implementation of metrics in Scikit-Learn restricts the usage of confusion matrix to pure classification problems, while we estimate the performance of regression estimators as a binary classification (see Section 4.1.2).

Also, Scikit-Learn does not have the notion of post-processing parameters while they are of great interest in our case. Our implementation allows to make a grid search with post-processing parameters, which are seen as any hyperparameter in the procedure, except that they do not involve to re-train a new model since post-processing parameters only influence predictions.

Chapter 5

Conclusion and perspectives

Through this work, we have been interested in the problem of cell counting and cell localization within digitized pathological tissues. Especially, we investigated five different supervised machine learning workflows on four datasets of tissue images.

First, we analyzed the problem in a end-user point of view. We opted for ground truths annotations solely based on simple dot annotations to indicate the presence of a cell, as it is the natural way of counting but also makes histopathological experts less prone to errors due to its simplicity.

We performed then a survey of existing approaches with dot-marked annotations, that we have (re-)implemented. Basically, two ways of counting are studied. Firstly, counting can be done by cell detection, the counting trivially being the number of detections. Secondly, counting can be estimated by density estimation, knowing the density for each image's pixel. We also proposed some methodological improvements. In particular, we noticed that these two approaches are ultimately relatively similar. By this way, we established that it is possible to retrieve locations of cells from a pure counting by density approach.

A major part of this thesis involved the assessment of these algorithms on real-case histopathological datasets. For this purpose, we defined some metrics: the localization reliability is evaluated trough a F1-score, the counting error is obtained with a mean absolute error as well as the average Euclidean distance error between a detection and its correctly assigned ground truth annotation. For each method, a rich model selection has been made in order to find their best model. We proposed a two-objective optimization criterion based on the detection F1-score and the mean absolute counting error, since we were interested to have performing models in detection *and* counting, which was not have been done before.

Results are not conclusive but encouraging. In general, at least one algorithm is able to reach a counting error less than 10% for each dataset. It would be interesting to have an histopathological expert's view regarding the expected allowed percentage of error, even if we assume it is application dependent.

This thesis mainly focused on the validation of these workflows such that there is room for improvements. In the short term, our algorithms will be able to be launched from the Cytomine, as all the implementation have been thought in that sense. Moreover, the integration of the template module of Cytomine would allow the user to directly run the prediction workflow from a region that he can define on the image.

Also, the usage of SLDC framework would be welcome to drastically reduce prediction times on large multi-gigapixel images and thus propose a more efficient solution on real usages. Finally, our solution of detection and counting could be integrated in larger workflows with the help of the SLDC framework. It would then be possible to make classification of the detected cells, as done in [Sir+16].

After all, this study is a necessary starting point for future works related to the development of generic counting methods. Indeed such a multi-method and multi-dataset study had not yet been published in literature.

Appendix A

Detailed results for GANGLIONS dataset

This chapter gives the detailed results for GANGLIONS dataset regarding the preliminary model selection with cross-validation and the default model approach on the learning set, to detect trends on the behaviour of each hyperparameter. These are tested while the others remain fixed to their default value. However, as the post processing parameters are required to compute metrics, the results of a test are the ones with the best post processing parameters, chosen from a grid search.

A.1 Experiments with SRTC

Param.	**Value**	κ	σ	**F1**	**Count**	**%**	**Dist.**
N	1	0.70	1	0.7352	18.55	19.12	4.42
N	*10*	0.67	1	0.7616	17.51	18.05	4.03
N	32	0.67	1	0.7792	16.37	16.88	4.02
N	64	0.66	1	0.7774	16.82	17.34	3.98
N	100	0.66	1	0.7800	16.67	17.18	4.00
n_{min}	*2*	0.67	1	0.7616	17.51	18.05	4.03
n_{min}	10	0.65	1	0.7683	17.08	17.61	4.00
n_{min}	100	0.49	1	0.7650	17.16	17.69	4.03
k	1	0.59	1	0.7594	18.88	19.47	4.21
k	*58*	0.67	1	0.7616	17.51	18.05	4.03
k	1664	0.70	1	0.7716	17.47	18.01	4.05
$w \times h$	8	0.65	1	0.7524	19.20	19.79	4.14
$w \times h$	*16*	0.67	1	0.7616	17.51	18.05	4.03
$w \times h$	24	0.66	1	0.7546	17.98	18.54	4.03
$w \times h$	32	0.63	1	0.7592	20.49	21.12	4.13
$w \times h$	50	0.64	1	0.7550	19.41	20.01	4.23
$w \times h$	64	0.63	1	0.7428	21.29	21.95	4.27

Table A.1: STRC hyperparameters trends for GANGLIONS

A.2 Experiments with PRTR

Param.	Value	κ	σ	F1	Count	%	Dist.
N	1	0.53	1	0.6595	40.39	41.64	4.15
N	*10*	0.54	1	0.7749	19.71	20.32	4.09
N	32	0.57	1	0.7751	18.51	19.08	3.98
N	64	0.58	1	0.7694	18.88	19.47	3.97
N	100	0.58	1	0.7723	19.16	19.75	3.94
n_{min}	*2*	0.54	1	0.7749	19.71	20.32	3.96
n_{min}	10	0.56	1	0.7760	21.51	22.18	3.96
n_{min}	100	0.62	1	0.7646	22.49	23.19	3.94
n_{min}	200	0.56	1	0.7900	15.67	17.92	4.06
n_{min}	600	0.62	1	0.7800	14.55	16.91	4.02
k	1	0.52	1	0.7499	20.04	20.66	4.16
k	*58*	0.54	1	0.7749	19.71	20.32	3.96
k	1664	0.54	1	0.7777	19.73	20.34	3.98
t	0.2	0.44	1	0.7611	24.27	25.03	4.00
t	0.3	0.49	1	0.7606	23.16	23.87	3.98
t	*0.4*	0.54	1	0.7749	19.71	20.32	3.96
t	0.5	0.79	0	0.7641	16.43	16.94	4.48
t	0.6	0.82	0	0.7574	17.67	18.21	4.50
t	0.7	0.86	0	0.7533	18.92	19.51	4.62
q	0.25	0.79	0	0.7529	21.14	21.79	4.25
q	0.5	0.57	1	0.7686	18.41	18.98	4.11
q	0.75	0.56	1	0.7767	17.20	17.73	4.07
q	*1*	0.54	1	0.7749	19.71	20.32	3.96
$w \times h$	8	0.75	0	0.7578	19.65	20.25	4.19
$w \times h$	*16*	0.54	1	0.7749	19.71	20.32	3.96
$w \times h$	24	0.54	1	0.7748	21.51	22.18	3.98
$w \times h$	32	0.54	1	0.7823	17.22	17.74	4.07
$w \times h$	50	0.55	1	0.7672	24.06	24.80	4.12
$w' \times h'$	*1*	0.54	1	0.7749	19.71	20.32	3.96
$w' \times h'$	2	0.57	1	0.7729	18.80	19.39	4.10
$w' \times h'$	4	0.76	0	0.7624	18.51	19.08	4.01
$w' \times h'$	8	0.73	0	0.7355	18.75	19.32	3.94
$w' \times h'$	16	0.69	0	0.7484	18.43	19.00	3.89

Table A.2: PRTR hyperparameters trends for GANGLIONS

A.3 Experiments with DRTR

Param.	Value	κ	σ	F1	Count	%	Raw	%	Dist.
N	1	0.35	0	0.7485	17.16	17.59	19.05	19.64	4.00
N	*10*	0.38	0	0.7700	14.82	15.28	19.14	19.74	3.91
N	32	0.39	0	0.7748	14.41	14.86	19.13	19.72	3.90
N	64	0.39	0	0.7725	15.31	15.79	19.10	19.69	3.91
N	100	0.38	0	0.7700	14.82	15.28	17.14	17.68	3.91
n_{min}	*2*	0.38	0	0.7700	14.82	15.28	19.14	19.74	3.91
n_{min}	10	0.25	1	0.7648	15.82	16.31	18.47	19.05	3.91
n_{min}	100	0.25	1	0.7678	17.25	17.79	16.14	16.64	3.93
k	1	0.39	0	0.7472	16.90	17.42	18.04	18.60	3.91
k	*58*	0.38	0	0.7700	14.82	15.28	19.14	19.74	3.91
k	1664	0.38	0	0.7786	14.16	14.59	19.05	19.64	3.90
N_{sw}	*10000*	0.38	0	0.7700	14.82	15.28	19.14	19.74	3.91
N_{sw}	50000	0.37	0	0.7722	16.55	17.06	16.27	16.78	3.91
N_{sw}	100000	0.37	0	0.7732	16.06	16.87	16.90	17.42	3.90
$w \times h$	8	0.38	0	0.7494	14.67	15.12	23.69	24.42	4.10
$w \times h$	*16*	0.38	0	0.7700	14.82	15.28	19.14	19.74	3.91
$w \times h$	24	0.37	0	0.7618	16.98	17.51	17.95	18.51	3.91
$w \times h$	32	0.38	0	0.7700	14.82	15.28	16.14	16.64	3.91

Table A.3: DRTR hyperparameters trends for GANGLIONS

A.4 Experiments with FCRN-A

Param.	Value	κ	σ	F1	Count	%	Raw	%	Dist.
e	24	0.06	0	0.76	14.22	16.76	22.55	22.91	4.18
e	*72*	0.04	0	0.77	13.09	15.79	26.17	25.08	4.46
e	120	0.04	0	0.77	12.16	13.98	26.24	25.12	4.45
e	192	0.04	0	0.77	11.36	13.49	26.17	25.08	4.47
b	16	0.04	0	0.77	11.30	13.46	31.43	30.81	4.55
b	*32*	0.04	0	0.77	13.09	15.79	26.17	25.08	4.46
η	0.0001	0.18	1	0.82	12.18	13.90	13.85	14.63	3.83
η	0.001	0.06	0	0.79	13.06	15.44	22.41	21.97	4.37
η	*0.01*	0.04	0	0.77	13.09	15.79	26.17	25.08	4.46
λ	*0.0*	0.04	0	0.77	13.09	15.79	26.17	25.08	4.46
λ	0.0005	0.14	0	0.80	11.38	13.50	11.90	13.29	3.86
N_{sw}	50	0.22	0	0.81	12.53	14.09	13.54	14.52	3.90
N_{sw}	*500*	0.04	0	0.77	13.09	15.79	26.17	25.08	4.46
N_{sw}	5000	0.02	0	0.76	14.86	16.51	49.40	50.06	4.72
$w \times h$	32	0.04	0	0.76	13.51	15.28	33.83	31.36	4.31
$w \times h$	64	0.06	1	0.79	12.35	14.13	15.27	15.89	4.14
$w \times h$	*128*	0.04	0	0.77	13.09	15.79	26.17	25.08	4.46
$w \times h$	256	0.06	1	0.77	14.69	16.49	22.88	22.00	4.09

Table A.4: FCRN-A hyperparameters trends for GANGLIONS

A.5 Experiments with FCRN-B

Param.	Value	κ	σ	F1	Count	%	Raw	%	Dist.
e	24	0.06	1	0.79	12.65	15.11	49.26	44.93	4.09
e	*72*	0.04	1	0.77	14.61	16.96	28.12	28.07	4.26
e	120	0.04	1	0.77	12.75	15.18	20.79	22.43	4.34
b	16	0.04	1	0.77	15.38	17.08	30.54	28.90	4.49
b	*32*	0.04	1	0.77	14.61	16.96	28.12	28.07	4.26
η	0.0001	0.04	1	0.72	21.84	20.43	34.54	33.37	4.37
η	0.001	0.04	1	0.74	16.97	17.50	25.23	24.93	4.35
η	*0.01*	0.04	1	0.77	14.61	16.96	28.12	28.07	4.26
λ	*0.0*	0.04	1	0.77	14.61	16.96	28.12	28.07	4.26
λ	0.0005	0.12	1	0.80	10.89	13.67	12.96	14.28	3.94
N_{sw}	50	0.22	1	0.72	21.83	20.43	17.82	18.51	3.88
N_{sw}	*500*	0.04	1	0.77	14.61	16.96	28.12	28.07	4.26
N_{sw}	5000	0.02	1	0.66	36.75	35.70	43.25	42.13	4.55
$w \times h$	32	0.06	1	0.79	12.55	14.74	16.63	16.86	4.28
$w \times h$	64	0.04	1	0.78	14.20	16.55	19.86	20.13	4.24
$w \times h$	*128*	0.04	1	0.77	14.61	16.96	28.12	28.07	4.26
$w \times h$	256	0.04	1	0.78	14.59	16.95	24.11	24.18	4.26

Table A.5: FCRN-B hyperparameters trends for GANGLIONS

Appendix B

Detailed results for ANAPATH dataset

This chapter gives the detailed results for ANAPATH dataset regarding the preliminary model selection with cross-validation and the default model approach on the learning set, to detect trends on the behaviour of each hyperparameter. These are tested while the others remain fixed to their default value. However, as the post processing parameters are required to compute metrics, the results of a test are the ones with the best post processing parameters, chosen from a grid search.

B.1 Experiments with SRTC

Param.	**Value**	κ	σ	**F1**	**Count**	**%**	**Dist.**
N	1	0.58	4	0.5238	735.50	26.88	5.91
N	*10*	0.70	4	0.5771	317.75	11.61	5.82
N	32	0.72	4	0.5833	234.75	8.58	5.78
N	64	0.72	4	0.5855	278.00	10.16	5.79
N	100	0.70	4	0.5862	275.75	10.08	5.78
n_{min}	*2*	0.7	4	0.5771	317.75	11.61	5.82
n_{min}	10	0.66	4	0.5754	267.50	9.78	5.82
n_{min}	100	0.57	4	0.5805	227.50	8.32	5.81
k	1	0.68	4	0.5502	284.25	10.39	5.83
k	*58*	0.70	4	0.5771	317.75	11.61	5.82
k	1664	0.66	4	0.5753	252.50	9.23	5.84
$w \times h$	8	0.72	4	0.5574	530.00	19.37	5.86
$w \times h$	*16*	0.70	4	0.5771	317.75	11.61	5.82
$w \times h$	24	0.70	4	0.5783	259.00	9.47	5.79
$w \times h$	32	0.72	4	0.5927	340.50	12.45	5.79

Table B.1: STRC hyperparameters trends for ANAPATH

B.2 Experiments with PRTR

Param.	Value	κ	σ	F1	Count	%	Dist.
N	1	0.55	1	0.2408	4856.25	177.49	5.92
N	*10*	0.53	1	0.4743	2856.50	104.40	6.04
N	32	0.70	0	0.4377	4307.25	157.43	6.09
N	64	0.66	0	0.4538	3771.75	137.86	6.03
N	100	0.68	0	0.4438	2435.34	102.23	5.99
n_{min}	*2*	0.53	1	0.4743	2856.50	104.40	6.04
n_{min}	10	0.74	0	0.4141	4531.25	165.62	6.11
n_{min}	100	0.51	1	0.6085	370.00	13.52	5.74
k	1	0.74	0	0.3749	6215.25	227.17	6.23
k	*58*	0.53	1	0.4743	2856.50	104.40	6.04
k	1664	0.56	1	0.4832	3212.23	113.12	6.02
t	*0.4*	0.53	1	0.4743	2856.50	104.40	6.04
t	0.5	0.78	0	0.4021	5214.75	190.60	6.10
t	0.6	0.82	0	0.4080	4824.00	176.32	6.13
t	0.7	0.86	0	0.4071	4573.00	167.14	6.15
q	0.25	0.77	0	0.3538	7502.50	274.21	6.19
q	0.5	0.76	0	0.3784	6420.00	234.65	6.20
q	0.75	0.75	0	0.3995	5416.00	197.95	6.22
q	*1*	0.53	1	0.4743	2856.50	104.4	6.04
$w \times h$	8	0.74	0	0.3436	7790.25	284.73	6.28
$w \times h$	*16*	0.53	1	0.4743	2856.50	104.40	6.04
$w \times h$	24	0.56	1	0.4845	2835.43	103.24	5.99
$w \times h$	32	0.56	1	0.4889	2790.30	101.89	5.99
$w' \times h'$	*1*	0.53	1	0.4743	2856.50	104.40	6.04
$w' \times h'$	2	0.68	0	0.4293	4466.25	163.24	6.09
$w' \times h'$	4	0.68	0	0.4573	3470.75	126.85	6.03
$w' \times h'$	8	0.66	0	0.4680	2795.25	102.17	6.12

Table B.2: PRTR hyperparameters trends for ANAPATH

We clearly have overfitting here. As shown in Table B.2, limiting overfitting by early stopping the three growths (increase n_{min}) improves the performances.

B.3 Experiments with FCRN-A

Param.	Value	κ	σ	F1	Count	%	Raw	%	Dist.
e	24	0.16	4	0.40	265.75	9.45	556.59	19.44	4.91
e	*72*	0.16	4	0.42	242.88	8.57	624.04	21.77	4.86
e	120	0.14	4	0.42	203.25	7.27	628.65	21.87	4.85
e	192	0.16	4	0.42	201.33	7.21	625.43	21.87	4.85
b	16	0.16	4	0.43	236.00	8.14	652.97	22.49	4.87
b	*32*	0.16	4	0.42	242.88	8.57	624.04	21.77	4.86
η	0.0001	0.16	4	0.37	330.00	11.59	636.40	22.26	4.89
η	0.001	0.14	4	0.42	244.25	8.59	746.04	25.99	4.83
η	*0.01*	0.16	4	0.42	242.88	8.57	624.04	21.77	4.86
λ	*0.0*	0.16	4	0.42	242.88	8.57	624.04	21.77	4.86
λ	0.0005	0.14	4	0.41	197.00	6.96	626.59	21.97	4.83
N_{sw}	50	0.12	4	0.35	231.00	8.25	1194.05	41.90	4.93
N_{sw}	*500*	0.16	4	0.42	242.88	8.57	624.04	21.77	4.86
N_{sw}	1000	0.12	4	0.43	272.00	9.64	716.10	24.65	4.83
$w \times h$	64	0.14	4	0.39	206.50	7.24	769.82	27.36	4.88
$w \times h$	128	0.14	4	0.40	195.50	6.95	578.81	20.40	4.87
$w \times h$	*256*	0.16	4	0.42	242.88	8.57	624.04	21.77	4.86
$w \times h$	512	0.14	4	0.41	270.00	9.54	780.72	27.81	4.92

Table B.3: FCRN-A hyperparameters trends for ANAPATH

B.4 Experiments with FCRN-B

Param.	Value	κ	σ	F1	Count	%	Raw	%	Dist.
e	24	0.16	4	0.37	321.45	11.35	654.87	22.47	4.90
e	*72*	0.16	4	0.39	258.00	9.14	512.37	18.12	4.88
e	120	0.18	4	0.40	246.12	9.03	681.21	23.29	4.86
e	192	0.16	4	0.39	197.25	6.96	475.01	16.77	4.89
b	16	0.18	4	0.40	314.75	11.10	557.98	19.76	4.88
b	*32*	0.18	4	0.40	246.12	9.03	681.21	23.29	4.86
η	0.0001	0.18	4	0.38	172.50	6.09	542.65	19.20	4.83
η	0.001	0.16	4	0.41	214.75	7.57	578.58	20.49	4.87
η	*0.01*	0.18	4	0.40	246.12	9.03	681.21	23.29	4.86
λ	*0.0*	0.18	4	0.40	246.12	9.03	681.21	23.29	4.86
λ	0.0005	0.18	4	0.38	184.00	6.49	384.25	13.59	4.88
N_{sw}	50	0.18	4	0.40	169.50	5.99	612.18	21.59	4.88
N_{sw}	*500*	0.18	4	0.40	246.12	9.03	681.21	23.29	4.86
N_{sw}	1000	0.12	4	0.44	321.75	11.35	569.97	20.07	4.80
$w \times h$	64	0.18	4	0.38	206.00	7.27	377.67	13.43	4.86
$w \times h$	128	0.12	4	0.39	204.25	7.21	755.22	26.77	4.86
$w \times h$	*256*	0.18	4	0.40	246.12	9.03	681.21	23.29	4.86
$w \times h$	512	0.18	4	0.39	259.00	9.16	760.48	26.86	4.88

Table B.4: FCRN-B hyperparameters trends for ANAPATH

List of Tables

2.1 List of datasets 13

3.1 FCRN-A architecture 36
3.2 FCRN-B architecture 36
3.3 SC-CNN architecture 39

4.1 Randomized trees parameters to tune 50
4.2 FCRN parameters to tune 51
4.3 Parameters for post-processing non maximum suppression 51
4.4 Our tests per method and per dataset. MS stands for Model Selection. 52
4.5 Learning set / Test set distribution for BMGRAZ 53
4.6 STRC hyperparameters trends for BMGRAZ 54
4.7 PRTR hyperparameters trends for BMGRAZ 55
4.8 DRTR hyperparameters trends for BMGRAZ 56
4.9 FCRN-A hyperparameters trends for BMGRAZ 57
4.10 FCRN-B hyperparameters trends for BMGRAZ 58
4.11 Best models for BMGRAZ 59
4.12 Comparison of best models performance for BMGRAZ 59
4.13 Learning set / Test set distribution for GANGLIONS 60
4.14 Best models for GANGLIONS 61
4.15 Comparison of best models performance for GANGLIONS 61
4.16 Learning set / Test set distribution for ANAPATH 62
4.17 Best models for ANAPATH 62
4.18 Comparison of best models performance for ANAPATH 62
4.19 Best models for CRC 64
4.20 Comparison of best models performance for CRC 64

A.1 STRC hyperparameters trends for GANGLIONS 71
A.2 PRTR hyperparameters trends for GANGLIONS 72
A.3 DRTR hyperparameters trends for GANGLIONS 73
A.4 FCRN-A hyperparameters trends for GANGLIONS 73
A.5 FCRN-B hyperparameters trends for GANGLIONS 74

B.1 STRC hyperparameters trends for ANAPATH 75
B.2 PRTR hyperparameters trends for ANAPATH 76
B.3 FCRN-A hyperparameters trends for ANAPATH 77
B.4 FCRN-B hyperparameters trends for ANAPATH 77

List of Figures

2.1 Variety of cells with visually similar structures that are not cells (a), with various shapes (b) and cell clumping (c) & (d). Green dots are ground truth cells whose size has been exaggerated. 4
2.2 A slide from ANAPATH dataset (24 gigapixels) and one region of interest (7.8 megapixels) 5
2.3 Cytomine image analysis module 6
2.4 Cytomine software module 7
2.5 A sample of a BM GRAZ dataset region of interest. 8
2.6 Number of cells per slide in dataset BM GRAZ. 8
2.7 A CRC dataset region of interest. 9
2.8 Number of cells per slide in dataset CRC. 9
2.9 Examples of GANGLIONS dataset region of interest (at different scales). 10
2.10 Number of ROI per slide in dataset GANGLIONS 11
2.11 Number of cells per slide in dataset GANGLIONS 11
2.12 Example of ANAPATH dataset region of interest. 12
2.13 Number of cells per image in dataset ANAPATH 12

3.1 Bias and variance in function of model complexity 15
3.2 Randomized trees algorithms structure 16
3.3 A convolutional layer [GW16] 18
3.4 A max pooling layer [LJY17] 19
3.5 A BRCA image crop (a) with the corresponding mask (b), the binary map of annotations (c) and the annotations superposed on the image crop (d). 21
3.6 Score map and non-maximum suppression [Kai+15] 21
3.7 Visual representation of input features 27
3.8 A BM GRAZ image crop I (a) with the corresponding annotation map A (b), the raw Euclidean distance transform $\mathcal{D}_A$ (c) and the score map S with $\alpha = 3$ and $r = 19$ (d). 31
3.9 The score map S with a cell center in C [Sir+14]. With our notations, $d(x)$ should be seen as $S(x)$ and d_M as r. 32
3.10 The two FCRN architectures for a $100 \times 100 \times 3$ input RGB image [XNZ15] 37
3.11 Last three layers of SC-CNN, where L is the number of layers 39

4.1 Model assessment using test set and model selection using leave one slide out cross-validation. 44
4.2 The best F1-score leads to a good counting performance 47
4.3 The best F1-score leads to a poor counting performance 47
4.4 PR curves for methods applied on BMGRAZ 60
4.5 PR curves for methods applied on GANGLIONS 61
4.6 PR curves for methods applied on ANAPATH 63
4.7 PR curves for methods applied on CRC 64
4.8 Training time for LS from BMGRAZ dataset 65

List of Algorithms

1 RandomSubwindowExtraction 24
2 ExhaustiveSubwindowExtraction 24
3 ConstrainedSubwindowExtraction 25
4 ClassificationTraining 28
5 PostProcessing 30
6 ClassificationPrediction 30
7 ProximityRegressionTraining 33
8 RegressionPrediction 33
9 DensityRegressionTraining 35
10 FCRNTraining 38
11 SC-CNNTraining 41

Bibliography

[Art+12] C. Arteta, V. Lempitsky, J. A. Noble, and A. Zisserman. "Learning to Detect Cells Using Non-overlapping Extremal Regions". In: *International Conference on Medical Image Computing and Computer Assisted Intervention*. Ed. by N. Ayache. Lecture Notes in Computer Science. MICCAI. Springer, 2012, pp. 348–356.

[Bra00] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[Bre+84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[Bre01] Leo Breiman. "Random Forests". In: *Mach. Learn.* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: http://dx.doi.org/10.1023/A:1010933404324.

[Cho+15] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[Fia+12] L. Fiaschi, U. Koethe, R. Nair, and F. A. Hamprecht. "Learning to count with regression forest and structured labels". In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. Nov. 2012, pp. 2685–2688.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees". In: *Machine Learning* 63.1 (2006), pp. 3–42. ISSN: 1573-0565. DOI: 10.1007/s10994-006-6226-1. URL: http://dx.doi.org/10.1007/s10994-006-6226-1.

[Gil13] Sean Gillies. *Shapely User Manual (version 1.2 and 1.3)*. Dec. 2013. URL: https://pypi.python.org/pypi/Shapely.

[GMR10] Metin N. Gurcan, Anant Madabhushi, and Nasir Rajpoot. "Pattern Recognition in Histopathological Images: An ICPR 2010 Contest". In: *Recognizing Patterns in Signals, Speech, Images and Videos: ICPR 2010 Contests, Istanbul, Turkey, August 23-26, 2010, Contest Reports*. Ed. by Devrim Ünay, Zehra Çataltepe, and Selim Aksoy. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 226–234. ISBN: 978-3-642-17711-8. DOI: 10.1007/978-3-642-17711-8_23. URL: http://dx.doi.org/10.1007/978-3-642-17711-8_23.

[GW16] Pierre Geurts and Louis Wehenkel. *Introduction to Machine Learning - Lecture 5 : (Deep) neural networks*. Oct. 2016. URL: http://www.montefiore.ulg.ac.be/~lwh/AIA/deep-neural-nets-20-10-2016.pdf (visited on 06/06/2017).

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009. URL: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[Kai+15] P. Kainz, M. Urschler, S. Schulter, P. Wohlhart, and V. Lepetit. "You Should Use Regression to Detect Cells". In: *International Conference on Medical Image Computing and Computer Assisted Intervention*. 2015.

[Kar16] Ujjwal Karn. *An Intuitive Explanation of Convolutional Neural Networks*. Aug. 2016. URL: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ (visited on 06/06/2017).

[Kos+17] Henning. Kost, André;. Homeyer, Jesper. Molin, Claes. Lundstrom, and Horst. Hahn. "Training nuclei detection algorithms with simple annotations". In: vol. 8. 1. 2017, p. 21. DOI: 10.4103/jpi.jpi_3_17.

[LJY17] Fei-Fei Li, Justin Johnson, and Serena Yeung. *CS231n: Convolutional Neural Networks for Visual Recognition*. Jan. 2017. URL: http://vision.stanford.edu/teaching/cs231n/ (visited on 06/06/2017).

[Lon+16] Rémi Longuespée et al. "A laser microdissection-based workflow for FFPE tissue microproteomics: Important considerations for small sample processing". In: *Methods* 104 (2016). Laser-based biological mass spectrometry, pp. 154–162. ISSN: 1046-2023. DOI: http://dx.doi.org/10.1016/j.ymeth.2015.12.008.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.

[LZ10] Victor Lempitsky and Andrew Zisserman. "Learning To Count Objects in Images". In: *Advances in Neural Information Processing Systems 23*. Ed. by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta. Curran Associates, Inc., 2010, pp. 1324–1332.

[Mar+15] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[Mar+16] Raphaël Marée, Loic Rollus, Benjamin Stévens, Renaud Hoyoux, Gilles Louppe, Rémy Vandaele, Jean-Michel Begon, Philipp Kainz, Pierre Geurts, and Louis Wehenkel. "Collaborative analysis of multi-gigapixel imaging data using Cytomine". In: *Bioinformatics* (2016), pp. 1395–1401.

[McK10] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51–56.

[MGW16] Raphaël Marée, Pierre Geurts, and Louis Wehenkel. “Towards generic image classification using tree-based learning: An extensive empirical study”. In: *Pattern Recognition Letters* 74 (2016), pp. 17–23. ISSN: 0167-8655. DOI: http://dx.doi.org/10.1016/j.patrec.2016.01.006.

[Mor16] Romain Mormont. “A workflow for large scale computer-aided cytology and its applications”. MA thesis. University of Liège, 2016.

[MWG13] R. Marée, L. Wehenkel, and P. Geurts. “Extremely Randomized Trees and Random Subwindows for Image Classification, Annotation, and Retrieval”. In: *Decision Forests for Computer Vision and Medical Image Analysis*. Ed. by A. Criminisi and J. Shotton. London: Springer London, 2013, pp. 125–141. ISBN: 978-1-4471-4929-3. DOI: 10.1007/978-1-4471-4929-3_10. URL: http://dx.doi.org/10.1007/978-1-4471-4929-3_10.

[Oli07] Travis E Oliphant. “Python for scientific computing”. In: *CiSE* 9.3 (2007), pp. 10–20.

[Ped+11] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Poy97] Charles Poynton. *Poynton’s color FAQ*. 1997. URL: http://www.poynton.com/ColorFAQ.html.

[Sir+14] A. Sironi, E. Türetken, V. Lepetit, and P. Fua. “Multiscale Centerline Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.7 (July 2014), pp. 1327–1341. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2015.2462363.

[Sir+16] K. Sirinukunwattana, S. E. A. Raza, Y. W. Tsang, D. R. J. Snead, I. A. Cree, and N. M. Rajpoot. “Locality Sensitive Deep Learning for Detection and Classification of Nuclei in Routine Colon Cancer Histology Images”. In: *IEEE Transactions on Medical Imaging* 35.5 (May 2016), pp. 1196–1206. ISSN: 0278-0062. DOI: 10.1109/TMI.2016.2525803.

[Var10] Gael Varoquaux. *Joblib User Manual*. 2010. URL: https://pypi.python.org/pypi/joblib.

[VCV11] S. Van der Walt, S. C. Colbert, and G. Varoquaux. “The NumPy array: a structure for efficient numerical computation”. In: *CiSE* 13.2 (2011), pp. 22–30.

[Wal+14] Stefan van der Walt, Johannes L. Schonberger, Juan Nunez-Iglesias, Francois Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. “scikit-image: image processing in Python”. In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: http://dx.doi.org/10.7717/peerj.453.

[XNZ15] W. Xie, J. A. Noble, and A. Zisserman. “Microscopy Cell Counting with Fully Convolutional Regression Networks”. In: *MICCAI 1st Workshop on Deep Learning in Medical Image Analysis*. 2015.