University of Liège
Faculty of Engineering

Master Thesis

A workflow for large-scale computer-aided cytology and its applications.

Author : Romain Mormont

Supervisor : Prof. Pierre Geurts

Master thesis submitted for the degree of

MSc in Computer Science and Engineering

Academic year 2015-2016

Contents

Summary

Acknowledgement

Chapter 1

Introduction

Chapter 2

Object detection in large images

2.1 General problem

2.1.1 Formulation

Generic formulation of the object detection problem

2.1.2 Implementation issues

What issues an implementor could face when trying to implement object detection in large images

2.1.3 Related works

What solutions are usually presented in the litterature to solve those problems (shallow overview as this is a wide topic)

2.2 Cytology

2.2.1 An object detection problem ?

Why it is an instance of the "object detection in large images" problem

2.2.2 The thyroid case

Explanation of the Thyroid case

Chapter 3

A generic workflow : Segment Locate Dispatch Classify

In this chapter, a generic workflow for solving problems of objects detection and classification in images is presented. The history of this workflow is presented in Section 3.1. Section 3.2 introduces the workflow itself. Finally, the implementation carried out in the context of this thesis is presented and discussed in Section 3.3.

3.1 History and first developments

The Segment-Locate-Dispatch-Classify (SLDC) workflow was first imagined by ?? Jean-Michel Begon ?? as a generalization of the work on thyroid nodule malignancy detection made in [Deb13]. In the context of his master thesis, the author had implemented a processing workflow for detecting cells with inclusion and proliferative architectural patterns (see ?? (thyroid)) in digitized thyroid punctions slides. The cells and architectural patterns were detected by segmenting the images and then classified using machine learning techniques. As explained in the Section ?? (thyroid), some patterns could themselves contain cells with inclusion. Therefore, the author implemented a second processing workflow to detect those cells. This workflow was similar to the first because it relied on a segmentation algorithm to isolate cells in patterns and then used machine learning to assess their malignity.

From those workflows, a common pattern emerged: performing detection using a segmentation algorithm and then classifying the detected objects using machine learning techniques. In 2015, ?? Jean-Michel Begon ?? developed a first version of a generic workflow based on this pattern and gave it the name *Segment-Locate-Dispatch-Classify*. Then, he applied its workflow to the thyroid case. Unfortunately, this implementation suffered from some drawbacks which made it hard to reuse in other contexts. The workflow was therefore re-worked in the context of this master thesis.

3.2 Principle

3.2.1 Algorithm

The workflow is a meta-algorithm[1] that detects and classifies objects contained in images. Particularly, given a two-dimensional[2] image $\mathcal{I}$ as input, it is expected to output the information about the objects of interest in this image. Those information include the shape of the object, its location in the image as well as a classification label. Formally, the workflow can be seen as an operator $\mathcal{W}$:

Definition 1. *Let $\mathcal{W}$ be an operator such that*

$$\mathcal{W}(\cdot) : \mathcal{I} \rightarrow \{(o_1, C_1), ..., (o_N, C_N)\} \tag{3.1}$$

where N is the number of objects of interest in $\mathcal{I}$ and (o_i, C_i) is a tuple. The first element of this tuple, o_i, is a representation of the information (shape and location) about the i^{th} object of interest found in $\mathcal{I}$ and the second, C_i, its classification label.

It is worth noting that genericity is of the essence. That is, the meta-algorithm should be able to solve the widest possible range of object detection and classification problems. Moreover, as explained in Section 3.1, it should produce those outputs using image segmentation and machine learning. As far as the segmentation is concerned, genericity is usually hard to obtain because of the high variability of images across different problems. In order to ensure genericity, the workflow doesn't impose a particular segmentation procedure but expects the implementer to provide one that suits the problem. The same goes for the classification models used for predicting the labels of the objects.

In the subsequent sections, some additional operators are defined and used to build the $\mathcal{W}$ operator. First, a basic version of the algorithm is presented and then refined in order to achieve an acceptable level of genericity.

3.2.2 Additional operators

Segmentation is the first operation applied to the image. This step of the algorithm is where the detection is actually carried out:

Definition 2. *Let $\mathcal{S}$ be the **segment** operator. It is applied to an image $\mathcal{I}$ and produces a binary mask $\mathcal{B}$. The pixel b_{ij} of $\mathcal{B}$ is 1 if the pixel p_{ij} of $\mathcal{I}$ is located in an object of interest, otherwise it is 0. Formally:*

$$\mathcal{S}(\cdot) : \mathcal{I} \rightarrow \mathcal{B} \tag{3.2}$$

While the segmented image theoretically contains the necessary information about the detected objects (i.e. shape and position in the image), the format of this information is inconvenient to query mostly because it is embedded into the binary mask and a single object cannot be trivially extracted. An intermediate step that would convert this information into a more convenient format is therefore needed. This format should encode both the shape of the object and its position in the image. It appears that polygons match this specification.

[1] In this context, a meta-algorithm is an algorithm that coordinates the execution of other algorithms.

[2] A third dimension can be dedicated to the images channel (i.e. 3 channels for RGB images, 4 channels for RGBA images).

Definition 3. *Let $\mathcal{L}$ be the **location** operator. It is applied to a binary mask and produces a set of polygons encoding the shapes and positions of every object in the image. Formally:*

$$\mathcal{L}(\cdot) : \mathcal{B} \to \{P_1, ..., P_N\} \tag{3.3}$$

where $\mathcal{B}$ is a binary mask as defined in Definition 2, N is the number of objects of interest in $\mathcal{B}$ and P_i is the polygon representing the geometric contour of the i^{th} object in $\mathcal{B}$.

The final step of the workflow is the object classification and is performed by a classifier which is passed a representation of the object (e.g. image, geometric information,...) and produces a classification label. In this theory, there is no restriction about the nature or representation of the objects processed by the classifiers.

Definition 4. *Let $\mathcal{T}$ be the **classifier** operator. It is applied to an object of interest and produces a classification label. Formally:*

$$\mathcal{T}(\cdot) : o \to C \tag{3.4}$$

where o is the object and C, the classification label.

Definition 5. *Let $\mathcal{T}^*$ be an extension of $\mathcal{T}$ which is given a set of objects and produces labels for all of them. Formally:*

$$\mathcal{T}^*(\cdot) : \{o_1, ..., o_N\} \to \{C_1, ..., C_N\} \tag{3.5}$$

3.2.3 Single segmentation, single classifier

The most simple construction of $\mathcal{W}$ would be the composition of the operators defined in Section 3.2.2. Particularly, the compositions $\mathcal{S} \circ \mathcal{L}$ and $\mathcal{S} \circ \mathcal{L} \circ \mathcal{T}^*$ would respectively produce the polygons representing the objects and and their labels. This construction is summarized in Algorithm 1:

Algorithm 1. *Construction of $\mathcal{W}$ using one segmentation and one classifier:*

1. *Return* $(\mathcal{S} \circ \mathcal{L})(\mathcal{I}) \times (\mathcal{S} \circ \mathcal{L} \circ \mathcal{T}^*)(\mathcal{I})$

As explained in Section 3.2.1, the definition of $\mathcal{S}$ and $\mathcal{T}^*$ would be left at the implementer's hands. As far as the $\mathcal{L}$ operator is concerned, it could be imposed by the workflow without loss of genericity. Such an construction of $\mathcal{W}$ could already solve any object detection and classification problem on image in which the labels can be predicted by a single classifier. However, in some cases, one classifier is not enough. This happen, for instance, when the image contains objects of very different nature and using several classifiers would yield better results than using a single one. An extension is therefore needed.

3.2.4 Single segmentation, several classifiers

In this attempt to construct a generic $\mathcal{W}$ operator, the image is assumed to contain M distinct types of objects and the workflow uses M classifiers (the i^{th} classifier being noted as $\mathcal{T}_i$ with $i \in \{1, ..., M\}$) to classify those objects. As an object should only be processed by one classifier, the workflow has to be added a new step which consists in dispatching each polygon to its most appropriate classifier.

Definition 6. *Let $\mathcal{D}$ be the dispatch operator. It is applied to a polygon and produces an integer which identifies the most appropriate classifier for processing this polygon:*

$$\mathcal{D}(\cdot) : P \rightarrow i, i \in \{1, ..., M\} \tag{3.6}$$

This step being problem dependent, it is the responsibility of the implementer to define the rules used for dispatching the polygons. However, the format of these rules can be defined.

Definition 7. *Let $\mathcal{P}$ be a set of M predicates $p_1, ..., p_M$ which associate truth values to polygons:*

$$p_i(\cdot) : P \rightarrow t \in \{true, false\}, i \in \{1, ..., M\} \tag{3.7}$$

where p_i is the predicate associated with the i^{th} classifier. The polygon P is dispatched to a classifier $\mathcal{T}_i$ if p_i associates true to this polygon. To avoid dispatching an object to several classifier, the predicates should verify the following property:

$$p_i = true \Leftrightarrow p_j = false, \forall j \neq i \tag{3.8}$$

Given this format, the $\mathcal{D}$ operator can be trivially constructed as it returns i if p_i is *true*. The algorithm resulting from this construction of $\mathcal{W}$ starts the same way as in Section 3.2.3: the image is applied the segment and locate operators. Then, the resulting polygons are dispatched and classified to produce the classification label. The resulting algorithm is summarized in Algorithm 2.

While the range of problems that can be solved using this algorithm has been increased compared to the version with a single classifier, there are still some problems that cannot be. In particular, if some objects are included into other bigger objects, they won't be considered as independent objects.

Before extending the algorithm for handling this case, it is worth noting that Algorithm 2 is completely compatible with Algorithm 1. Indeed, if there is only one classifier (i.e. $M = 1$) and the predicate p_1 always returns *true*, then both algorithms are exactly the same.

Algorithm 2. *Construction of the $\mathcal{W}$ operator with a single segmentation and several classifiers.*

1. *Apply the $\mathcal{S} \circ \mathcal{L}$ composition to the input image $\mathcal{I}$ to extract the objects of interest as the set of polygons $S_p \leftarrow \{P_1, ..., P_N\}$*
2. *Initialize the labels set $L \leftarrow \emptyset$*
3. *For each polygon $P \in S_p$:*
   (a) *Compute the classification label $C \leftarrow \mathcal{T}_{\mathcal{D}(P)}(P)$*
   (b) *Place the label in the labels set $L \leftarrow L \cup \{C\}$*
4. *Build and return objects and labels set $S_p \times L$.*

3.2.5 Chaining workflows

To handle the case when objects of interest can be included into objects of bigger size in the image, a solution consists in executing several instances of Algorithm 2 sequentially one after another.

Definition 8. *Let $\mathcal{W}_1, ..., \mathcal{W}_K$ be a set of K instances of Algorithm 2. Each algorithm $\mathcal{W}_i$ has its own segmentation procedure $\mathcal{S}_i$ and proper sets of dispatching predicates $\mathcal{P}_i$ and classifiers $S_{\mathcal{T},i}$.*

While $\mathcal{W}_1$ would be applied to the full image $\mathcal{I}$ to extract all the objects of interest, $\mathcal{W}_2, ..., \mathcal{W}_K$ are only passed image windows containing the previously detected objects.

Definition 9. *Let $\mathcal{I}_P$ be an image window extracted from image $\mathcal{I}$ and containing the object represented by the polygon P. The window is the minimum bounding box containing this polygon.*

A further refinement would be to provide a way for the implementer to filter the polygons of which the windows are passed to a given workflow instance. Indeed, a given instance $\mathcal{W}_i$ might be designed to process only a certain category of objects and therefore should not be passed windows of objects that doesn't fall in this category.

Definition 10. *Let $\mathcal{F}$ be the **filter** operator. It is given a set of polygons S_P and returns a subset S'_P of polygons:*

$$\mathcal{F}(\cdot) : S_P \rightarrow S'_P, S'_P \subseteq S_P \tag{3.9}$$

Each instance of the workflow $\mathcal{W}_i$ except $\mathcal{W}_1$ is therefore associated a filter operator $\mathcal{F}_i$. The resulting algorithm is given in Algorithm 3 and has now reached an acceptable level of genericity.

Algorithm 3. *Construction of the $\mathcal{W}$ operator with K instances of Algorithm 2:*

1. *Execute the first workflow and save the results in the results set R: $R \leftarrow \mathcal{W}_1(\mathcal{I})$*
2. *Create the polygons set and initializes it with the polygons found from the execution of $\mathcal{W}_1$: $S_P \leftarrow \{P_1, ..., P_N\}$*
3. *For each $i \in \{2, ..., K\}$:*
   (a) *Extract polygons to be processed by $\mathcal{W}_i$: $S'_P \leftarrow \mathcal{F}_i(S_P)$*
   (b) *For polygon $P \in S'_P$:*
      i. *Execute workflow $\mathcal{W}_i$ on the image window and saves the results: $R \leftarrow R \cup \mathcal{W}_i(\mathcal{I}_P)$*
      ii. *Add the extracted polygons to the polygons set: $S_P \leftarrow S_P \cup \{P_1, ..., P_M\}$*
4. *Return the results set R*

3.3 Implementation

The workflow presented in Section 3.2 is now defined. This section details the design choices and architecture of the implementation. In the subsequent sections, the term workflow will refer to the algorithm while the term framework will refer to the implementation.

3.3.1 Requirements

The main requirements for the framework are listed hereafter.

Genericity As for the algorithm, the framework should be able to solve the widest possible range of objects detection and classification problems in any context. This property has more implication in the case of the framework design than for the algorithm design, especially when it comes to fixing the representation of the various involved data types (i.e. image, polygon,...). Moreover, the implementer should be able to do whatever he wants to do with the results of the computations.

Efficiency While the framework has no control over the efficiency of the algorithms defined by the implementer (i.e. segmentation or classification procedures), the coordination of those algorithms should not induce a significant overhead in the overall execution.

Large images While large images handling was irrelevant at the algorithm design stage, it becomes critical at this point. To remain generic, the framework should not make any assumption about the size of the images to be processed. Especially, a whole image should not be assumed to fit into memory.

Robustness The framework should be robust to errors. That is, a single error should not interrupt the whole execution. For instance, if the framework executes a set of independent computations and one of them fails, it should only be stopped if this failure is unrecoverable and affects all the other computations. Otherwise, the failure should be reported and those others computations should execute until completion.

Transparency The framework should provide a built-in way to communicate its progress, the duration of each steps as well as the errors it encounters with the user. The level of verbosity of this communication tool should be adjustable.

Parallelism Whenever possible the framework should take advantage of parallelism to reduce its execution time. However, the implementer should be given a way to disable parallelism and switch to sequential execution. Moreover, the implementer should be able to change the level of parallelism (i.e. the number of jobs on whi).

Ease of use The work of the implementer should be kept as minimal as possible. He should only have to define the logic of problem dependent elements of the workflow: segmentation, dispatching rules, classifiers,...

3.3.2 Language

The first design choice occurring in the implementation of an existing algorithm is obviously the language in which this implementation will be made. As far as the workflow is concerned, the chosen language was Python. Indeed, this language provides a very complete environment for developing objects detection and segmentation algorithms which would obviously contribute to the overall ease of use the framework.

First of all, the language has many features which allows developers to quickly come up with solutions to problems. Especially, it is strongly and dynamically typed, multi-paradigm (imperative, functional, object oriented,...), interactive (it can be used in an interactive console), interpreted and garbage-collected. It also support usual data structures such as lists, arrays, dictionaries and sets natively and provide operations for manipulating them in a concise way.

In addition to its built-in features, Python has become a great language for scientific computing as it has been augmented with excellent open source libraries over the years. First, the SciPy ecosystem which includes the SciPy [Oli07] and NumPy [VCV11] libraries. The first is a collection of numerical algorithms and domain-specific toolboxes (signal processing, optimization, statistics,...). The second is a fundamental package for numerical computations which provides an efficient representation of multi-dimensional arrays and operations on them. Built on top of the SciPy ecosystem comes Scikit-Learn [Ped+11], a library that provides simple, efficient and reusable tools for data mining and machine learning. Image processing is not outdone with a Python binding for the huge OpenCV library [Bra00]. Two alternatives are scikit-image [Wal+14] which is built on top of the SciPy ecosystem or the Pillow library [Cla16]. All of them provide a collection of well-known image processing algorithms. Another useful library is Shapely [Gil13] which provides a representation for geometric objects such as polygons as well as operations to apply on them.

Then, Python was also chosen because the workflow was implemented to be integrated with Cytomine (see Section ??). Particularly, the final goal was the detection and classification of objects in images stored on Cytomine servers. As those images and their metadata are exposed through an API interfaced by a Python client, it was essential that the workflow could use this client to communicate with the back-end.

3.3.3 Software architecture

3.3.3.1 Image representation

The image representation manipulated by the framework is a critical point of the architecture. Indeed, on the one hand, it should be abstract enough so that implementers can apply the workflow to problems where they have their own image format. On the other hand, it should provide access to a concrete representation that the steps of the workflow fixed by the framework (i.e. location) could use to extract necessary information. This is achieved by using two representations: an abstract one and a concrete one. As much as possible the framework works with the abstract representation and only switch to the other one when required.

The representation should also provide a way of extracting sub-windows from an image. The application of this feature would be twofold. First, it is needed by the workflow (see Definition 9). Then, it could be used to address the large images handling requirement and to overcome the fact that a whole image is not assumed to fit into memory. Especially, the image would be splitted into smaller chunks called tiles and those tiles would be processed one after another.

The final design is shown in Figure 3.1.

Especially, the concrete representation is passed to the segmentation procedure (see operator $\mathcal{D}$ in Definiton 2) and is also use to represent the binary mask produced by the segmentation and processed by the locate procedure (see $\mathcal{L}$ in 3).

3.3.4 How to use the framework

A toy example: finding disks in an image with grey background and guessing whether they're black or white

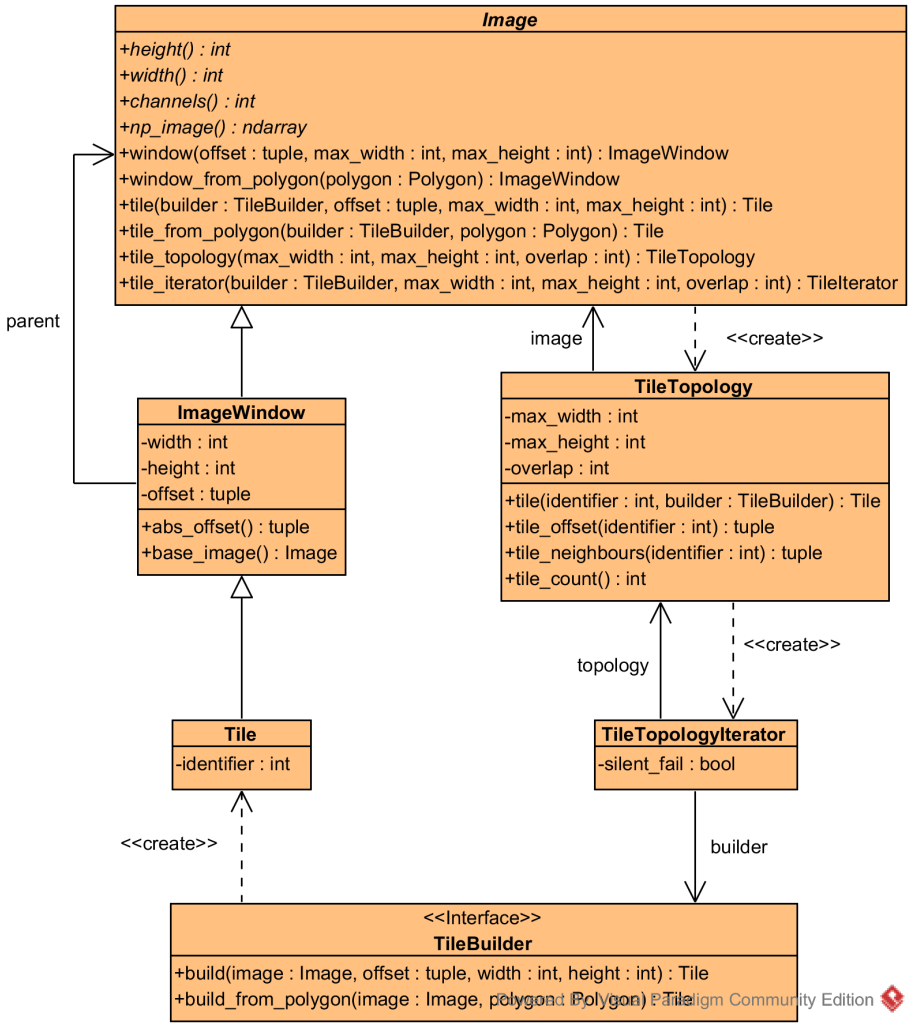

Figure 3.1: Image representation classes

Chapter 4

SLDC at work : the thyroid case

4.1 Cytomine

Presentation of cytomine

4.2 Implementation issues

Presentation of implementation issues related to the thyroid case (image size, over HTTP, image quality, human annotation vs computer annotation, presence of inclusions in patterns, dispatching ...)

4.3 Implementation

Actual implementation of the processing using the workflow

4.4 Performance analysis

4.4.1 Detection

4.4.2 Execution time

Chapter 5

Conclusion

Notations

Image :

$\mathcal{I}$	An image
w	The width of an image
h	The height of an image
c	The number of channels of an image
$\mathcal{I}_{hw}$	An two dimensional image of width w and height h
p_{ij}	A pixel at row i and column j of a two dimensional image
$\mathcal{B}$	A binary image
$b_{ij} \in \{0, 1\}$	A pixel of a binary image
P	A polygon

Machine learning :

$T(\cdot)$	A classifier
C	A classification label

List of Tables

List of Figures

Bibliography

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

[Cla16] Alex Clark. *Pillow (version 3.2.x)*. 2016. URL: `https://github.com/python-pillow/Pillow/tree/3.2.x`.

[Deb13] Antoine Deblire. "Segmentation et classification automatiques de cytoponctions de la thyroïde." fr. MA thesis. Université de Liège, Liège, Belgique, 2013, p. 74.

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

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

[Ped+11] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *The Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: `http://scikit-learn.org`.

[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] Stéfan van der Walt et al. "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`.