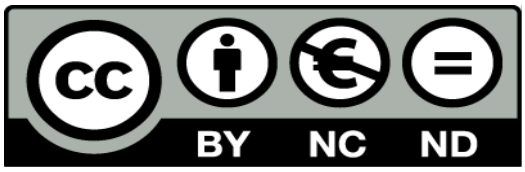
UNIVERSIDAD DE LA RIOJA

TESIS DOCTORAL

Título
Processing biomedial images for the study of treatments related to neurodegenerative diseases
Autor/es
Gadea Mata Martínez
Director/es
Julio Rubio García y Miguel Morales Fuciños
Facultad
Facultad de Ciencia y Tecnología
Titulación
Departamento
Matemáticas y Computación
Curso Académico



publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

DOCTORAL THESIS

Processing Biomedical Images for the Study of Treatments Related to Neurodegenerative Diseases

Author:
Gadea Mata Martínez

Supervisors:
Julio Rubio García, PhD
Miguel Morales Fuciños, PhD

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Grupo de Informática
Departamento de Matemáticas y Computación

2017

This work has been partially supported by an FPI grant from the *Universidad de La Rioja* (FPI-UR-13) and projects MTM2014-54151-P from the Spanish *Ministerio de Economía y Competitividad*, programme of grants ADER2011 from *Agencia de Desarrollo Económico de La Rioja*, grant COLABORA 2010/07 under the title "Formalisation of Mathematics (For Math) FP7 STREP Project, number 243847" and grants ATUR2014, ATUR2015, ATUR2016 from the *Universidad de La Rioja*.

Abstract

The study of neuronal cell morphology and function in neurodegenerative disease processes is essential in order to develop suitable treatments.

In fact, studies such as the quantification of either synapses or the neuronal density are instrumental in measuring the evolution and the behaviour of neurons under the effects of certain physiological conditions.

In order to analyse this data, fully automatic methods are required. To this end, we have studied and developed methods inspired by Computational Algebraic Topology and Machine Learning techniques. Notions such as the definition of connected components, or others related to the persistent homology and zigzag persistence theory have been used to compute the synaptic density or to recognise the neuronal structure. In addition, machine learning methods have been used to determine where neurons are located in large images and to ascertain which are the best features to describe this kind of cells.

Resumen

El estudio de la morfología y de la funcionalidad de céludas neuronales en el proceso de enfermedades neurodegenerativas es de alta importancia para desarrollar fármacos y terapias adecuadas.

De hecho, estudios como la cuantificación de sinapsis o la densidad neuronal son fundamentales para medir la evolución y el comportamiento de neuronas bajo el efecto de ciertas condiciones fisiológicas.

Para el análisis de estos datos se necesitan métodos completamente automatizados. Con esta finalidad, hemos estudiado y desarrollado métodos inspirados en la Topología Algebraíca Computacional y técnicas de aprendizaje automatizado. Nociones como la definición de componente conexa u otras relacionadas con la homología persistente y la teoría de la persistencia zigzag han sido usadas para calcular la densidad sináptica o reconocer la estructura neuronal. Además, se han utilizado métodos de aprendizaje automatizado, para conocer dónde se encuentran las neuronas en imágenes de gran tamaño y para determinar cuáles son las características que mejor describen a este tipo de células.

Contents

Abstract iii

Resumen v

Contents vii

List of Figures xi

List of Tables xiii

1 Introduction 1

2 Synaptic Density and Verification 5
2.1 Synaptic Density . . . 5
2.1.1 Introduction . . . 5
2.1.2 Methodology . . . 6
2.1.3 Experimental Results . . . 10
2.1.4 Scientific Validations of the Computations . . . 11
2.1.5 Discussion . . . 14
2.1.6 Conclusions . . . 15
2.1.7 Availability and Software Requirements . . . 16
2.2 Verification . . . 16
2.2.1 Introduction . . . 16
2.2.2 Context . . . 17
2.2.3 Methodology . . . 18
2.2.4 Experimental Results . . . 25
2.2.5 Conclusions . . . 28

3 Neural Density 29
3.1 Introduction . . . 29
3.2 Methodology . . . 29
3.3 Experimental Results . . . 37
3.4 Conclusions . . . 38
3.5 Availability and Software Requirements . . . 38

4 Neuron detection in stack images 39
4.1 Introduction . . . 39
4.2 A Persistent Homology Interpretation . . . 39
4.2.1 Introduction . . . 40
4.2.2 Methodology . . . 40
4.2.3 Experimental Results . . . 45
4.2.4 Discussion . . . 47
4.2.5 Availability and Software Requirements . . . 48
4.3 Zigzag Persistence Theory . . . 48

4.3.1 Introduction . . . 48
4.3.2 Methodology . . . 49
4.3.3 Experimental Results . . . 56
4.3.4 Conclusions . . . 57
4.3.5 Availability and Software Requirements . . . 57

5 Location of neurons **59**
5.1 Introduction . . . 59
5.2 An Approach Using Intensity Features . . . 59
5.2.1 Introduction . . . 59
5.2.2 Previous work . . . 60
5.2.3 Methodology . . . 62
5.2.4 Annotation of the Image Data . . . 71
5.3 Methodology Used Over Binary Images . . . 71
5.4 Methodology Used Over Textures Features . . . 74
5.4.1 Introduction . . . 74
5.4.2 Context . . . 76
5.4.3 Methodology . . . 78
5.4.4 Experimental Results . . . 78
5.5 Discussion . . . 80
5.6 Deeper Study about Features of Images and Machine Learning Algorithms . . . 80
5.6.1 Introduction . . . 80
5.6.2 Methodology . . . 80
5.6.3 Experimental Results . . . 84
5.6.4 Discussion and Current Work . . . 87

6 Conclusions **89**

7 Future Work **91**

A Definitions **93**

B Biology and Acquisition **101**
B.1 Experimental Phase . . . 101
B.1.1 Experimental Methods . . . 101
B.1.2 Primary Neuronal Cultures . . . 102
B.1.3 Immunocytochemistry of neurons in cultures . . . 102
B.2 Acquisition of Images . . . 103
B.2.1 Images for the Analysis of Synaptic Density . . . 103
B.2.2 Images for the Analysis of the Immunocytochemistry of Neurons in Cultures . . . 104
B.2.3 Images for the Analysis of the Structure of GFP-Transfected Neurons . . . 104
B.2.4 Images for the Analysis of the GFP-Transfected Neurons . . . 106

C Technology **107**

D Validation process of the plug-in NeuronZigzagJ **111**
D.1 First experiment . . . 111
D.2 Second experiment . . . 112

E Results of Machine Learning Experiments **119**

Bibliography **129**

List of Figures

1.1 Draw by Ramón y Cajal . . . 2

2.1 Workflow of SynapCountJ. . . . 7
2.2 Ex. of neuron with two antibody markers . . . 8
2.3 SynapCountJ window to configure the analysis. . . . 8
2.4 Window of threshold and result . . . 8
2.5 Binary image of connected components . . . 9
2.6 Ex. of a high-content microscope image . . . 10
2.7 Syanptic density study . . . 11
2.8 Workflow to compute homology groups . . . 14
2.9 Architecture of the I2EA framework integrating Coq . . . 25
2.10 Proof General with Coq2ACL2 extension . . . 28

3.1 Ex. of neurotoxicity cultures . . . 30
3.2 Ex. of a piece of a mosaic . . . 31
3.3 Binary image with the nuclei . . . 32
3.4 Ex. the regions marked . . . 34
3.5 Study of the local intensity . . . 35
3.6 Ex. of a result image . . . 35
3.7 Workflow of NucleusJ . . . 36
3.8 Interface NucleusJ . . . 37
3.9 Ex. of how the plug-in works . . . 38

4.1 Ex. tests with a transfected neuron . . . 40
4.2 Ex. of a Z-stack pre-processed . . . 41
4.3 Ex. of a process of filtration . . . 42
4.4 Ex. of a simplicial complex . . . 43
4.5 Ex. of a filtration . . . 43
4.6 Ex. of barcode . . . 44
4.7 Summary of the connected components in a projection image . . . 44
4.8 Ex. of NeuronPersistentJ results . . . 46
4.9 Interfaz of NeuronzigzagJ . . . 54
4.10 Ex. of neuron stained with DiI . . . 55
4.11 Part of structure of a DiI-stained neuron . . . 55
4.12 Graph of the results from the plug-in . . . 55

5.1 Patch of a mosaic with neurons . . . 60
5.2 Graph of intensities . . . 61
5.3 Ex. of using of the method . . . 62
5.4 Ex. of continuity in dendrites . . . 63
5.5 Ex. of the neurons found . . . 64
5.6 Tests with different values . . . 65
5.7 Resutls vs solution by an expert . . . 66

5.8 Histograms of the regions with and without neuron 67
5.9 Manually-traced neurons 68
5.10 Ex. of combining different descriptors 69
5.11 Ex. of binary patches 72
5.12 Graph of ROC space 73
5.13 Ex. of a high-content microscope image 75
5.14 Ex. of a result using one of the machine learning techniques 79
5.15 Ex. of method for extracting positive patches 81
5.16 Ex. of a mosaic sampling 82
5.17 Ex. the SIFT-features in neruons and background patches 83
5.18 Error plot (AUROC) 86

A.1 Structure of a pyramidal hippocampal neuron 97

B.1 Ex. of image used to the study of Chapter 2 104
B.2 Ex. of image used to the study of Chapter 3 105
B.3 Ex. of images used for study, of Chapter 4 105
B.4 Ex. of image used to the study of Chapter 5 106

List of Tables

2.1 Features of the analyzed software . . . 14
2.2 Features to quantify of the analyzed software . . . 15

4.1 Percentages of accuracy of NeuronPersistentJ . . . 47

5.1 Optimal values for the method . . . 64
5.2 Optimal values in the 2nd phase . . . 65
5.3 Theoretical confusion matrix . . . 70
5.4 Ex. of confusion matrices . . . 71
5.5 Recall and precision for different combinations . . . 74
5.6 Results of the first experiment . . . 79
5.7 Results of the second experiment . . . 79
5.8 Features in each data set . . . 84
5.9 Number of features . . . 85
5.10 Percentage of the features selected . . . 87
5.11 Percentage of the features used to the selection . . . 87

D.1 Results 1st experiment . . . 113
D.2 Summary of DiI-experiment . . . 115
D.3 Summary of GFP-experiment . . . 116
D.4 Results 1st observer . . . 117
D.5 Results 2nd observer . . . 117
D.6 Results 3rd observer . . . 118

E.1 Table of GLMNET results (1) . . . 120
E.2 Table of GLMNET results (2) . . . 121
E.3 Table of KNN results (1) . . . 122
E.4 Table of KNN results (2) . . . 123
E.5 Table of RF results (1) . . . 124
E.6 Table of RF results (2) . . . 125
E.7 Table of SVM results (1) . . . 126
E.8 Table of SVM results (2) . . . 127

List of Abbreviations

1D	**1** **D**imensional
2D	**2** **D**imensional
3D	**3** **D**imensional
DAPI	(see definition in Appendix A)
JML	**J**ava **M**odelling **L**anguage
FN	**F**alse **N**egative
FP	**F**alse **P**ositive
GLMNET	(see definition in Appendix A)
GFP	**G**reen **F**luorescent **P**rotein
HCA	**H**igh-**C**ontent **A**nalysis (see definition in Appendix A)
KNN	k-**N**earest **N**eighbours
MAP2B	(see definition in Appendix A)
N	(Condition) **N**egative
N′	(Predicted condition) **N**egative
NB	**N**aïve **B**ayes
NMDA	(see definition in Appendix A)
p	**p**recision (see definition in Appendix A)
P	(Condition) **P**ositive
P′	(Predicted condition) **P**ositive
PI3K	(see definition in Appendix A)
PO	**P**roof **O**bligation
PPV	**P**ositive **P**redicted **V**alue
PTD4-PI3K	(see definition in Appendix A)
r	**r**ecall (see definition in Appendix A)
RF	**R**andom **F**orests
RGB	**R**ed **G**reen **B**lue (color model)
ROC	**R**eceiver **O**perating **C**haracteristic (space)
ROI	**R**egion **O**f **I**nterest
SIFT	**S**cale **I**nvarient **F**eature **T**ransform
SVM	**S**upport **V**ector **M**achine
SSP	**S**tructural **S**ynaptic **P**lasticity (group)
TN	**T**rue **N**egative
TP	**T**rue **P**ositive
TPR	**T**rue **P**ositive **R**atio
XLL	**X**mall **L**ogical **L**anguage

List of Concepts

The definitions of these concepts are in Appendix A.

Adjacency
Artefact
Aspect Ratio
Centroid
Confusion Matrix
Connected component
DAPI
Dendrite
Feret' s diameter
GLMNET
Gold Standard
High-Content Analysis
Homology group
Immunostaining
k-Means
k-Nearest Neighbours
Kurtosis
Machine Learning
Making an image binary with ImageJ
MAP2
Maximum
Mean
Median
Median Filter
Morphological operations
Neuron
NMDA
Noise
PI3K
Precision
PTD4-PI3K
Random Forests
Recall
ROC space
ROI
Skewness
Soma
Solidity
Standard Deviation
Supervised Learning
Support Vector Machine
Unsupervised Learning
Z-stack

List of Symbols

$\mathbb{N}$	natural numbers
$\mathbb{Z}$	integer numbers
$\emptyset$	empty set
H_0	homology group of dimension 0
H_1	homology group of dimension 1
XY	XY-plane
Z	Z-plane
μm	microns
mm^2	square millimeters
$\_^c$	set complement
$\cup$	union
$\cap$	intersection
$\subseteq$	subset of
$\oplus$	exclusive or - xor
$\ominus$	symmetric difference

Chapter 1

Introduction

> {Las neuronas son} células de formas delicadas y elegantes, las misteriosas mariposas del alma, cuyo batir de alas quién sabe si esclarecerá algún día el secreto de la vida mental.
>
> *Recuerdos de mi vida*
> Chapter VII of 2nd part
> Santiago Ramón y Cajal.

Santiago Ramón y Cajal said, referring to neurons (Cajal, 1917): "they are cells of delicate and elegant forms, the mysterious butterflies of the soul, whose wings flap who knows if it will someday clarify the secret of the mental life".

Ramón y Cajal spoke these words after spending countless hours examining brain matter under a microscope, while all the time noting down what he saw in pen and ink. Of course, since Ramón y Cajal drew these pictures two-hundred years ago (see Figure 1.1) this technique has certainly advanced.

Nowadays, advanced microscopes acquire images, and their processing is faster and more objective than in Ramón y Cajal's time. These advances led to the possibility of analysing samples in a more efficient and objective way, not to mention time-saving for the researcher. This particular technological advance has been key in the field of digital image processing. In fact, this branch of computer science was originally developed for artificial intelligence and robotics.

A neurodegenerative disease occurs when there is a progressive loss of structure of the neuronal functions, which includes the death of neurons. Diseases such as amyotrophic lateral sclerosis, Alzheimer's, Parkinson's and Huntington's disease are the result of a neurodegenerative process. Until now, these diseases have been incurable, although scientists are constantly searching for the means to understand what occurs throughout these diseases and how the nervous system works in order to discover a cure.

There are approximately 100 billion neurons in a brain. These cells are connected to thousands of others forming a dense signal-processing network. This means that the number of synapses (connections among neurons) is superior to 1,000 billions, surpassing the amount of stars in the milky way. The pioneering works of Ramón y Cajal suggested that *neuronal morphology* and *physiology* were intrinsically correlated. The specificity of connections and information flux, as Ramón y Cajal proposed, were closely dependent of *neuronal structure* (Cajal, 1889).

Different factors can determine the synapses distribution such as the environment where the animals were raised or if they have been trained to perform a specific task (see Gogolla, Galimberti, Deguchi, and Caroni, 2009, Xu, Yu, Perlik, Tobin,

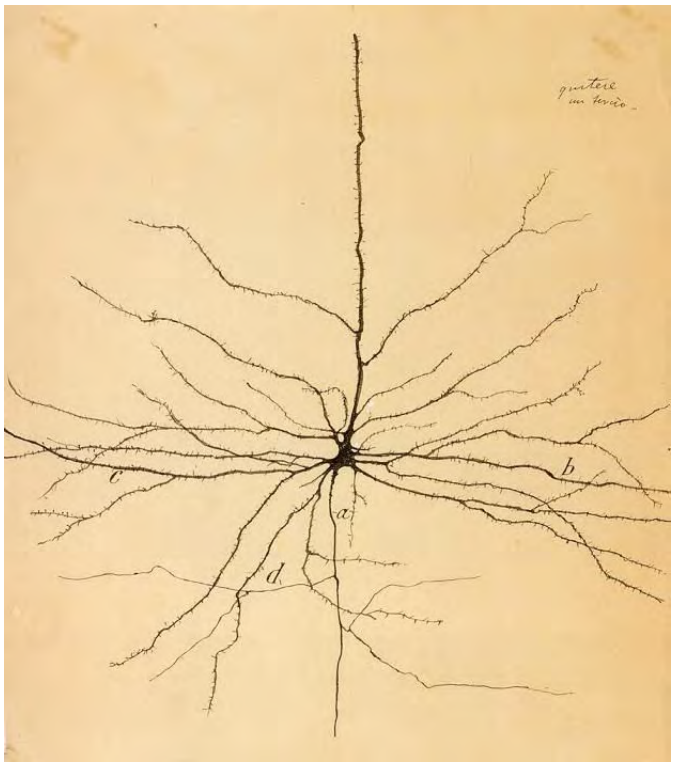


FIGURE 1.1: Neuron drawn by Ramón y Cajal.

Zweig, Tennant, Jones, and Zuo, 2009). In addition, ageing is another factor which influences in the synapses distribution tending to decrease them.

Studies about a type of bird, the zebra finches, have revealed there is and evidence linking synapses formation and learning. Isolated birds present a lower synaptic turnover while others which have been exposed to song tutoring lead to high turnover (see Tschida and Mooney, 2012, Roberts, Tschida, E., and R., 2010). Therefore the main process of these birds is to be able to modify the synaptogenic rate.

As mentioned, ageing and diseases are other factors which affect to the synaptic distribution. The study of synapses formation and their distribution was the focus of the Structural Synaptic Plasticity group(SSP) ((see SpineUp, 2014), a group of biologists lead by Miguel Morales, PhD. The SSP group has previously demonstrated that activation of the PI3K pathway increases synaptogenesis and improves learning in mammals and invertebrates (Acebes and Morales; 2012), thus we have proposed that a regulated activation of this pathway can be a putative target to develop drugs to treat Alzheimer disease.

Currently, counting synapses relies on manual procedures. Although humans are competent at recognizing objects and patterns in an image, these manual approaches can be slow, tedious, error-prone and subjective tasks, especially for samples with a large amount of spines (places where synapses take place) or even neuronal densities. The capacity to rapidly evaluate these processes would help pharmacological studies accelerate the process of compound development.

But before automating the study of synapses, many other tasks are needed. Some of this tasks were carried out manually by the SSP group. The goal of this work is to get an automatic or semi-automatic alternative approach to these previously non-automatic processes. Biologists are the experts who assess the images and the new process.

Roughly speaking, the programmed tasks are related to knowing the numbers of cells in a sample, or delimiting them, etc. Each chapter is devoted to such a task, as we are going to explain in the following paragraphs.

As we mentioned before, the quantification of synapses is instrumental to measuring the evolution of synaptic densities of neurons under the effect of some physiological conditions, such as neurological diseases or drug treatments. An approach to determining the number and density of synapses is presented in **Chapter 2**.

Knowing the connections among neurons is as important as knowing the neuronal density of a sample. One could suppose that if there are a small number of neurons, the number of synapses are low as well. A method for finding the nuclei of the neurons and, thus, knowing the number of neurons in a sample is presented in **Chapter 3**.

Neuronal reconstruction and recognition have been, since Ramón y Cajal's work, hindered by a similar problem: the discerning of a single neuron among hundreds of millions. Several staining techniques are employed to identify a single neuron (Ramón-Moliner, 1970); for instance, Golgi staining, iontophoretic intracellular injection (Elston and DeFelipe, 2002; Ballesteros-Yáñez, Benavides-Piccione, Elston, Yuste, and DeFelipe, 2006) and Diolostic gun (Heck, 2012). The use of optical and confocal microscopy and digital reconstruction of neuronal morphology has become a powerful technique for investigating the nervous system structure, providing us with a large scale collection of images. An approach to discern the structure of a neuron from a stack of images (several planes which depict the volume of the cell in 3D dimensions) is presented in **Chapter 4**.

Next, in **Chapter 5**, we present a study of two methods to find the neurons in large images, also known as images for high-content analysis (HCA). One of the methods is based on hysteresis thresholding, which only studies intensity features; and the second applies machine learning techniques to locate neurons in these kinds of images and find which are the best features to describe a patch of images containing a neuronal structure.

Finally, there are appendices which are divided in the following sections. The first appendix, Appendix A, collects definitions of the most relevant concepts which are used in this work. The biological methodology and the experiments are described in Appendix B next to the description of the each kind of image. The technologies used are described in Appendix C. The last appendices, Appendix D and E, contain the experimental results obtained in Chapters 4 and 5.

Looking for the means to make these tasks automatic, we have developed several programmes which can be used as plug-ins of the platforms for bioimage processing ImageJ and Fiji (Schneider, Rasband, and Eliceiri, 2012; Schindelin, Arganda-Carreras, and Frise, 2012).

As mentioned before, this work is fruit of the collaboration established between the Structural Synaptic Density group (SpineUp) and the Computer Science group of the University of La Rioja. In addition, part of this work has taken place over two three-month stays at the Biomedical Imaging Group Rotterdam of the Erasmus University Medical Center of Rotterdam in the Netherlands, under the supervision of Erik Meijering, PhD. Learning about biomedical analysis took place during a third two-month stay at the Unit of Advanced Optical Microscopy (UMOA) of the Scientific and Technological Centers (CCiT) of the University of Barcelona (UB) in Barcelona, Spain under the supervision of María Calvo, PhD.

Chapter 2

Synaptic Density and Verification

2.1 Synaptic Density

The work explained in this section has been presented and published in several conferences within the field of biology such as XIV Congreso Nacional Sociedad Española de Neurociencia (SENC), held in Salamanca, Spain, 2011. This is also the case within the field of bioimage processing such as the first Congress of the Spanish Network of Advanced Optical Microscopy (REMOA), held in Barcelona, Spain, 2012; the Bioimage Analysis Workshop - Euro-bioimaging, held in Barcelona, Spain, 2012; and the third International Conference on Bioimaging (BIOIMAGING, part of BIOSTEC), held in Rome, Italy, 2016, titled "SynapCountJ, a Tool for Analyzing Synaptic Densities in Neurons". This contribution was later published in the journal of the conference under the title: "SynapCountJ: A Validated Tool for Analyzing Synaptic Densities in Neurons", see Mata, Cuesto, Heras, Morales, Romero, and Rubio, 2017.

2.1.1 Introduction

Synapses are the points of connection between neurons, and they are dynamic structures subject to a continuous process of formation and elimination. Pathological conditions, such as the Alzheimer disease, have been related to synapse loss associated with memory impairments. Hence, the possibility of changing the number of synapses may be an important asset to treat neurological diseases (Selkoe, 2002). To this aim, it is necessary to determine the evolution of synaptic densities of neurons under the effect of some physiological conditions, neuronal diseases or even drug treatments.

The procedure to quantify synaptic density of a neuron is usually based on the colocalization between signals generated by two antibodies (Cuesto and Enriquez-Barreto, 2011). Namely, neuron cultures are permeabilized and treated with two different primary markers (for instance, bassoon and synapsin). These antibodies recognize specifically two presynaptic structures. Then, it is necessary a secondary antibody couple attached to different fluorochromes (for instance red and green; note, that several other combinations of colour are possible) making these two synaptic proteins visible under the fluorescence microscope. The two markers are photographed in two gray-scale images; that, in turn, are overlapped using respectively the red and green channels. In the resultant image, the yellow points (colocalization of the code channels) are the candidates to be the synapses.

The final step in the above procedure is the selection of the yellow points that are localized either on the dendrites of the neuron or adjacent to them. Tools like MetaMorph (Molecular Devices, 2015) or ImageJ (Schneider, Rasband, and Eliceiri, 2012) can be used to manually count the number of synapses; however, such a manual quantification is a tedious, time-consuming, error-prone, and subjective task;

hence, reliable tools that might automate this process are desirable. In this section, we present *SynapCountJ*, an ImageJ plug-in, that semi-automatically quantifies synapses and synaptic densities in neuron cultures. The programme is based on Algebraic Topology techniques and has been validated by comparing some intermediate results with those of Kenzo (Dousson, Rubio, Sergeraert, and Siret, 1999), a (partially) formally verified programme.

2.1.2 Methodology

SynapCountJ supports two execution modes: individual treatment of a neuron and batch processing — the workflow of both modes is provided in Figure 2.1.

Individual treatment of a neuron

The input of SynapCountJ in this execution mode are two images of a neuron marked with two antibodies (an image per antibody), see Figure 2.2. SynapCountJ is able to read tiff (a standard format for biological images) and lif files (obtained from Leica confocal microscopes) — the latter requires the Bio-Formats plug-in (Linkert, Rueden, and Allan, 2010). The following steps are applied to quantify the number of synapses in the given images.

In the first step, from one of the two images, the region of interest (i.e. the dendrites where the quantification of synapses will be performed) is manually specified using NeuronJ (Meijering, Jacob, and Sarria, 2004) — an ImageJ plug-in for tracing elongated image structures. In this way, the background of the image is removed. The result is a file containing the traces of each dendrite of the image.

Subsequently, the user can decide whether to perform a global analysis of the whole neuron, or a local analysis focused on each dendrite of the neuron. In both cases, SynapCountJ requires additional information such as the scale and the mean thickness (that is determined by the size of the subjacent dendrite) of the region to analyze (see Figure 2.3) — these parameters determine the area of the dendrite avoiding the background (i.e. all the non-synaptic marking).

Taking into account the settings provided by the user, SynapCountJ overlaps the two original images of the neuron and the structure of the neuron previously defined. From the resultant image, SynapCountJ identifies the almost white points (the result of green, red, and blue combination) as synaptic candidates, and it allows the user to modify the values of the red and green channels in order to modify the detection threshold (see Figure 2.4).

Once the detection threshold has been fixed, the counting process is started. Such a process is inspired by techniques coming from Computational Algebraic Topology. In spite of being an abstract mathematical subject, Algebraic Topology has been successfully applied in digital image analysis (González-Díaz and Real, 2005; Ségonne, Grimson, and Fischl, 2003; Mata, Morales, Romero, and Rubio, 2015).

In our particular case, the white areas are segmented from the overlapped image, and the colours of the resultant image are inverted — obtaining as a result a black-and-white image (from this point forward, binary image), see Figure 2.5 where the synapses are the black areas. From such an image, the problem of quantifying the number of synapses is reduced to compute the homology group in dimension 0 of the image; this corresponds to the computation of the number of connected components of the image. Our algorithm to count synapses can be summarized as presented in Algorithm 1.

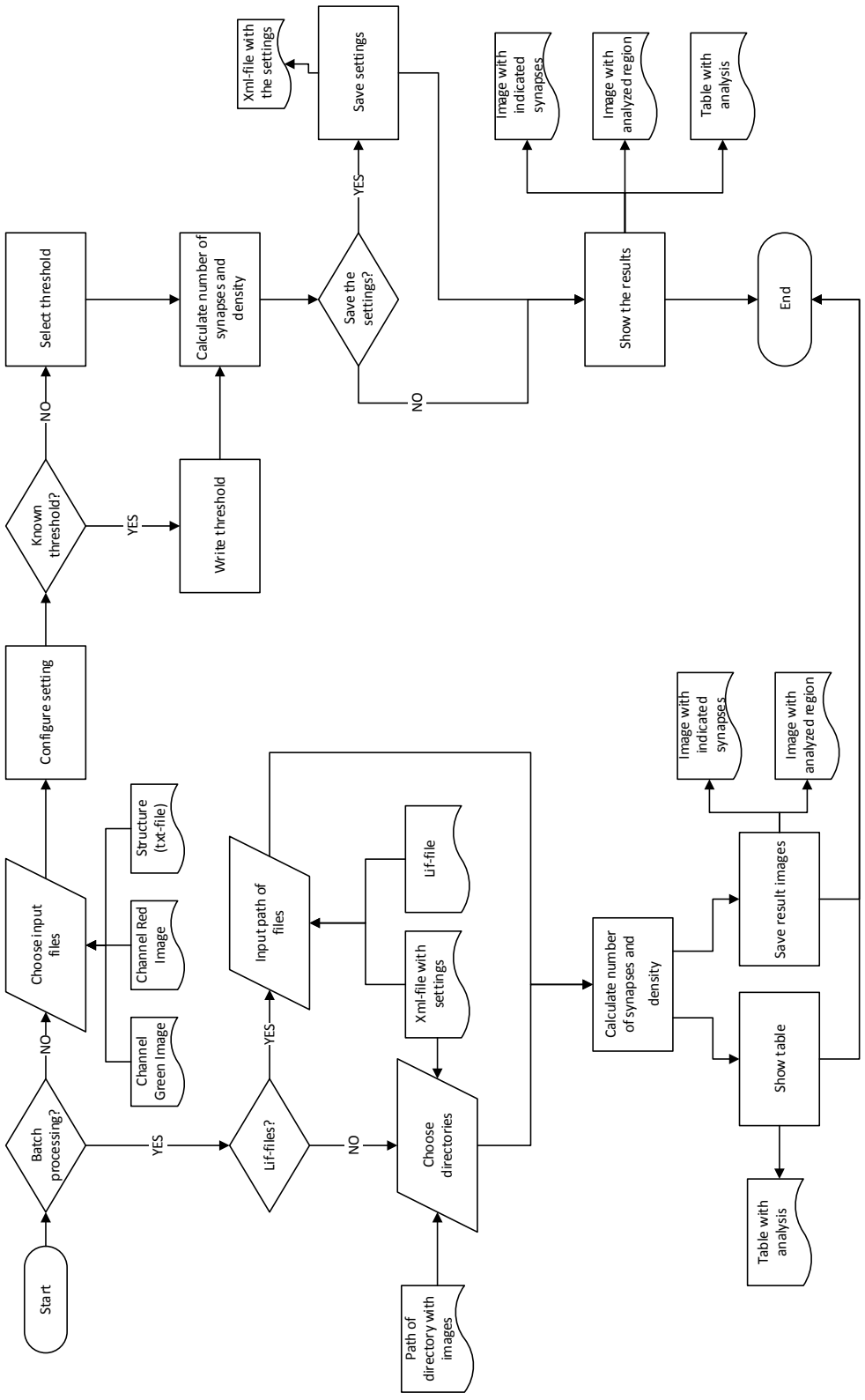
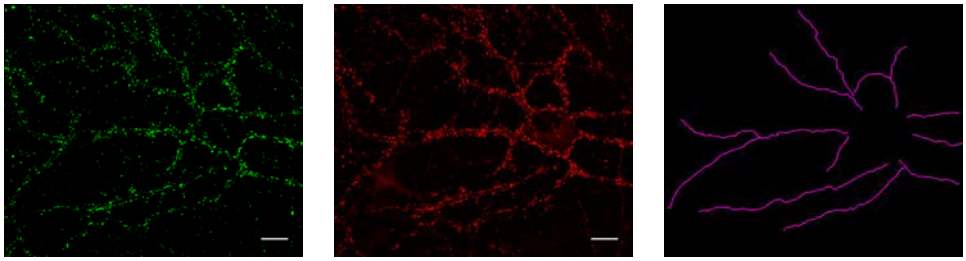


FIGURE 2.1: Workflow of SynapCountJ.



(a) (b) (c)

FIGURE 2.2: Example of neuron with two antibody markers and its structure. (a) Neuron marked with the bassoon antibody marker. (b) Neuron marked with the synapsin antibody marker. (c) Structure of the neuron. Scale bar: 50μm.

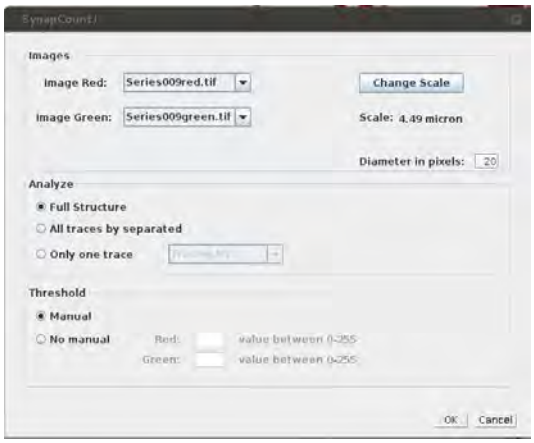
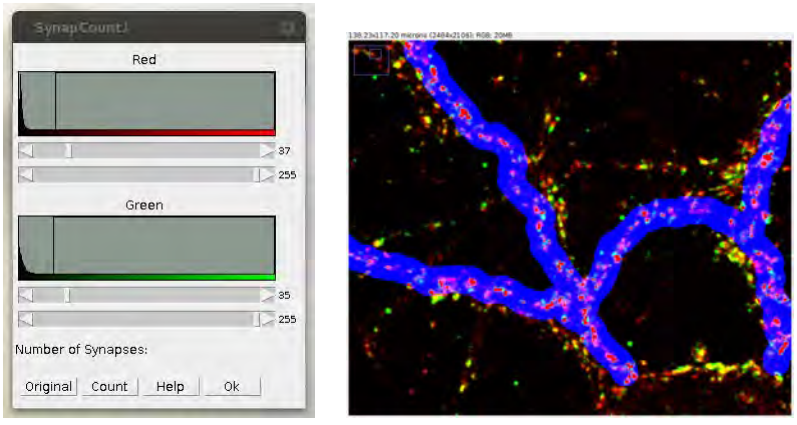


FIGURE 2.3: SynapCountJ window to configure the analysis.



(a) (b)

FIGURE 2.4: SynapCountJ window to modify the threshold of the red and green channels. (a) Window to fix the threshold of the image. (b) Fragment of the neuron image with the synapses indicated as the red areas on the structure of the neuron marked in blue. Moving the scrollbars of left window, the marked areas of the image are changed.

FIGURE 2.5: Binary image where the synapses are the black connected components.

Algorithm 1.
`Input`: Two images of a neuron marked with two antibodies
`Output`: Number of synapses of the neuron

1. Create an image with the structure of the neuron using NeuronJ
2. Overlap the two original images of the neuron and the structure of the neuron
3. Fix the detection threshold
4. Segment the white areas of the overlapped image using the fixed threshold
5. Invert the colours of the segmented image
6. Count the number of connected components of the image.

Finally, SynapCountJ returns a table with the obtained data (length of dendrites both in pixels and micras, number of synapses, and density of synapses per 100 micron) and two images showing, respectively, the analyzed region and the marked synapses (see Figure 2.6).

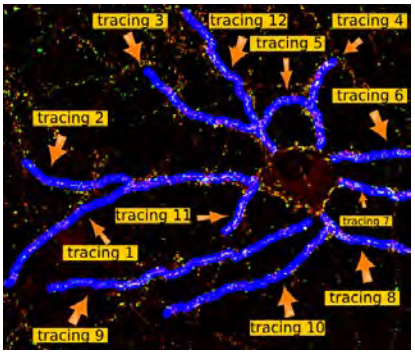
Batch processing

Images obtained from the same biological experiment usually have similar settings; hence, their processing in SynapCountJ will use the same configuration parameters. In order to deal with this situation, SynapCountJ can be applied for batch processing of several images using a configuration file. It is necessary to study at least one image from experiment to get the optimal settings. The parameters are saved in a XML-file (eXtensible Markup Language) and used to process the set of images from the same experiment.

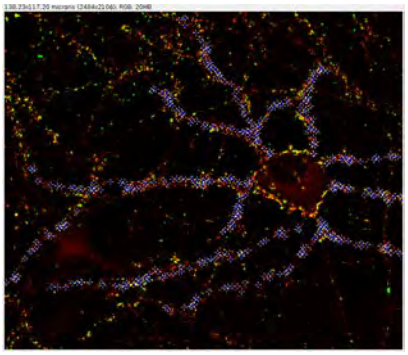
For batch processing, SynapCountJ reads tiff files organized in folders or a lif file (the kind of files produced by Leica confocal microscopes), and using the configuration file processes the different images. As a result, a table with the information related to each neuron from the batch is obtained. The table includes an analysis

	Label	Length in pixels	Length in micras	Synapses	Density	Red	Green
1	Tracing N1:	1833.1058	91.6553	71	77.4642	116	164
2	Tracing N2:	867.7840	43.3892	35	80.6652	116	164
3	Tracing N3:	983.5322	49.1766	53	107.7748	116	164
4	Tracing N4:	599.8320	29.9916	41	136.7049	116	164
5	Tracing N5:	437.7388	21.8869	25	114.2234	116	164
6	Tracing N6:	468.8438	23.4422	26	110.9111	116	164
7	Tracing N7:	447.6296	22.3815	31	138.5074	116	164
8	Tracing N8:	574.3691	28.7185	38	132.3191	116	164
9	Tracing N9:	1776.2572	88.8129	69	77.6915	116	164
10	Tracing N10:	1224.7374	61.2369	45	73.4851	116	164
11	Tracing N11:	355.7054	17.7853	26	146.1884	116	164
12	Tracing N12:	905.3750	45.2688	45	99.4063	116	164
13	Total Neuron	10474.9103	523.7455	479	91.4566	116	164

(a)



(b)



(c)

FIGURE 2.6: Results provided by SynapCountJ. (a) Table with the results obtained by SynapCountJ. (b) Image with the analyzed region of the neuron. (c) Image with the counted synapses indicated by means of blue crosses.

for both the whole neuron and from each of its dendrites. In addition, in the same directory where the lif-file or tiff-files are stored, the plug-in saves all the resultant images for each image from experiment (one of them shows the marked synapses and the other one, the region which has been studied).

2.1.3 Experimental Results

A total of 13 individual images from three independent cultures were analyzed — cultures treated according to the explanation given in Appendix B. In Figure 2.7 we can observe that using a manual method to identify and count synapses, we obtain a mean of 24.12 synapses in the control cultures and 16.74 in the treated cultures. The results obtained with SynapCountJ are similar: there is a mean of 26.03 synapses in the control cultures and 16.50 in the ones which have been treated.

Notwithstanding the differences in the quantification, in both procedures we obtain almost the same inhibition percentage, a 30.51% manually and 36.61% automatically. This shows the suitability of SynapCountJ to count synapses, meaning a considerably reduction of the time employed in the manual process, namely, the manual analysis of an image takes approximately 5 minutes; of a batch, 1 hour; and of a complete study, 4 hours. Using SynapCountJ, the time to analyze an image is 30 seconds; a batch, 2 minutes; and a complete study, 6 minutes.

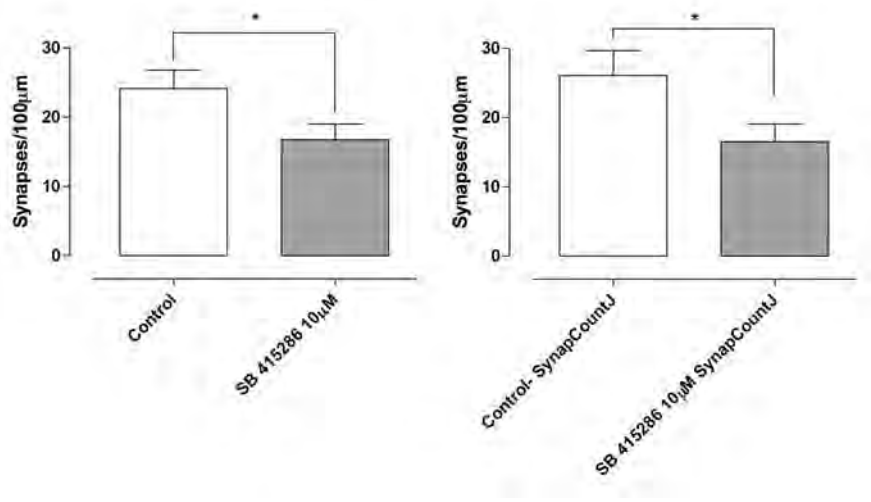


FIGURE 2.7: Hippocampal cultures were treated with the synapses formation inhibitor SB 415286 for 48 hours. *Left*. Manual quantification of synapses. *Right*. Quantification of synapses using *SynapCountJ*.

2.1.4 Scientific Validations of the Computations

Accuracy and reliability are two desirable properties of every software tool, especially in the case of biomedical software. A method to increase the trust in scientific software is the use of mechanised theorem proving technology to verify the correctness of the programmes (Amorim, Collins, DeHon, Hritcu, Pichardie, Pierce, Pollack, and Tolmach, 2014; Hales, 2005). However, such a formal verification is a challenging task (Benton, 2006). As it implies a direct verification of the programmes which is especially complicated in this case, for the following reasons:

- Our programmes are developed in Java (a general-purpose programming language) and the common language to formalize them is a functional programming language (for instance, Haskell).
- Accurate specification in detail is necessary to obtain all possible cases.
- The branch of the mathematics related to the problem has to be formalized, although there are already a lot of basic notions formalized.

We were interested in increasing the reliability of our software; however, due to the difficulty of directly verifying the correctness of our programmes — see a brief explanation in Benton, 2006 —, we have followed an indirect method.

A key component of our algorithm to count synapses is the computation of connected components of a binary image (see Algorithm 1). Such a computation can be performed using two different methods:

- a direct approach, where the pixels of the image are directly processed; and,
- an indirect approach, where the notion of simplicial complex associated with an image, and techniques from Algebraic Topology (namely, homology groups) are employed to compute the connected components of the image.

The former is efficient and can be easily employed in ImageJ — in fact, it is the one implemented in SynapCountJ — however, its formal verification is a challenging

problem. The latter is slower than the former, is difficult to incorporate it into ImageJ; but, it can rely on a previously developed software, the Kenzo system (Dousson, Rubio, Sergeraert, and Siret, 1999), and therefore, it does not require any further development. The formal verification of the Kenzo system is even harder than the verification of the direct approach, but, fortunately, such a task was, at least partially, tackled in the ForMath project (ForMath, 2010–2013) — a European project devoted to the development of libraries of formalised mathematics concerning algebra, linear algebra, real number computation, and Algebraic Topology.

In this context, where we have a fast but unverified algorithm, and a slow but verified algorithm, the following strategy can be employed to increase the reliability of the fast version thanks to the verified version. The strategy consists in performing an intensive automated testing checking whether the results obtained with both versions are the same; if that is the case, the reliability of the fast algorithm is increased. In our particular case, we have employed such a strategy to increase the reliability of the computation of connected components of binary images using the fast version implemented in SynapCountJ (the direct approach) thanks to the verified Kenzo system (the indirect approach).

In the rest of this section, we thoroughly explain the two different approaches to compute connected components of a binary image.

The direct approach

The direct approach to compute connected components of a binary image processes directly the pixels of the image by means of an algorithm included in ImageJ which is called *FindMaxima* (see Díaz de Greñu de Pedro, J., 2014). This algorithm can be applied to black-and-white (or binary), grayscale or colour images and determines the local maxima of the image, provided with segmented regions containing all the pixels of the image whose value differs from the corresponding local maxima in less than a chosen threshold. In the case of binary images, the result corresponds to the different connected components.

The algorithm is divided into two steps:

Algorithm 2.
`Input`: Binary image
`Output`: Number of local maxima points

1. First of all, the local maxima of the image are determined, and they are ordered in a decreasing way.

2. Secondly, a filling algorithm is applied for each local maximum to determine its connected region. If a maximum produces a region which was already filled by a previous maximum, the actual local maximum is discarded.

The first step is done by means of a method called *getSortedMaxPoints*. Here, all the pixels in the image are studied comparing them with their adjacent pixels. A pixel is chosen as local maxima if its value is higher than all their adjacent pixels. A threshold is also considered to discard those pixels with value lower than it. The result is an array with the local maxima (with their coordinates) ordered in a decreasing way.

Once the ordered list of local maxima has been obtained, the second step of the algorithm *FindMaxima* is done by means of a method called *analylizeAndMarkMaxima*. In this method, a filling algorithm is applied to each local maximum going over

the list in a decreasing way. To determine the region associated to a local maximum, an iterative process is applied considering the 8 adjacent pixels to the maximum (see Rosenfeld, 1974), selecting those whose difference with the local maximum is lower than a chosen parameter and studying then the adjacent pixels to those selected in the previous step. If a selected pixel is higher than the local maxima then it is stored as the maximum of the region and the previous one is discarded. If it is equal, the new pixel is also stored in order to be able to compute the mean of all the local maxima in the region as we will explain later. The process finishes when all possible adjacent pixels to the previously selected ones have been studied.

Let us observe that when applying the filling algorithm to a local maximum, we could find other maxima (included in the same connected component as the considered one). In that case, the process stops and the second maximum is discarded. Moreover, in case of having several maxima with the same value in a region, the final maximum is computed as the pixel with the same intensity as the local maximum which is closest to the baricenter of all of them.

For a more complete study of the *FindMaxima* algorithm in ImageJ see Díaz de Greñu de Pedro, J., 2014.

The indirect approach

The indirect approach to compute connected components of a binary image employs the Kenzo system. Kenzo (Dousson, Rubio, Sergeraert, and Siret, 1999) is a Common Lisp system devoted to Algebraic Topology which has obtained some results not confirmed nor refuted by theoretical or computational means (Sergeraert, 1992), and also has been used to refute some computations obtained by theoretical means (Romero, Heras, Rubio, and Sergeraert, 2014; Romero and Rubio, 2013). Then, the question of Kenzo reliability arose in a natural way, and several works have been focussed on studying the correctness of Kenzo key fragments and algorithms (Aransay, Ballarin, and Rubio, 2008; Domínguez and Rubio, 2011; Lambán, Martín-Mateos, Rubio, and Ruiz-Reina, 2013).

The final aim of Kenzo was not the analysis of digital images, but it was extended with a module that tackles such a problem (Heras, Pascual, and Rubio, 2012). In particular, such a module computes homological properties, that measure connected components and holes of binary images. This Kenzo module for digital images has been employed to validate the results obtained in SynapCountJ using the direct approach.

The Kenzo module for digital images works as follows (see Figure 2.8). Given a binary image, a triangulation procedure is employed to obtain a simplicial complex (a generalisation of the notion of graph to higher dimensions) — there are several methods to construct a simplicial complex from a digital image, see Ayala, Domínguez, Francés, and Quintero, 2003. From the simplicial complex, its *boundary (or incidence) matrices* are constructed. Since the size of the boundary matrices coming from biomedical images is too big to be handled directly by Kenzo, a reduction strategy is employed to work with smaller matrices, but preserving their homological properties (Romero and Sergeraert, 2010). From the reduced boundary matrices, homology groups in dimensions 0 and 1 are computed using a diagonalisation process (Munkres, 1984). The homology groups are either null or a direct sum of $\mathbb{Z}$ components, and they should be interpreted as follows: the number of $\mathbb{Z}$ components of the homology groups of dimension 0 and 1 measures respectively the number of connected components and the number of holes of the image. Hence,

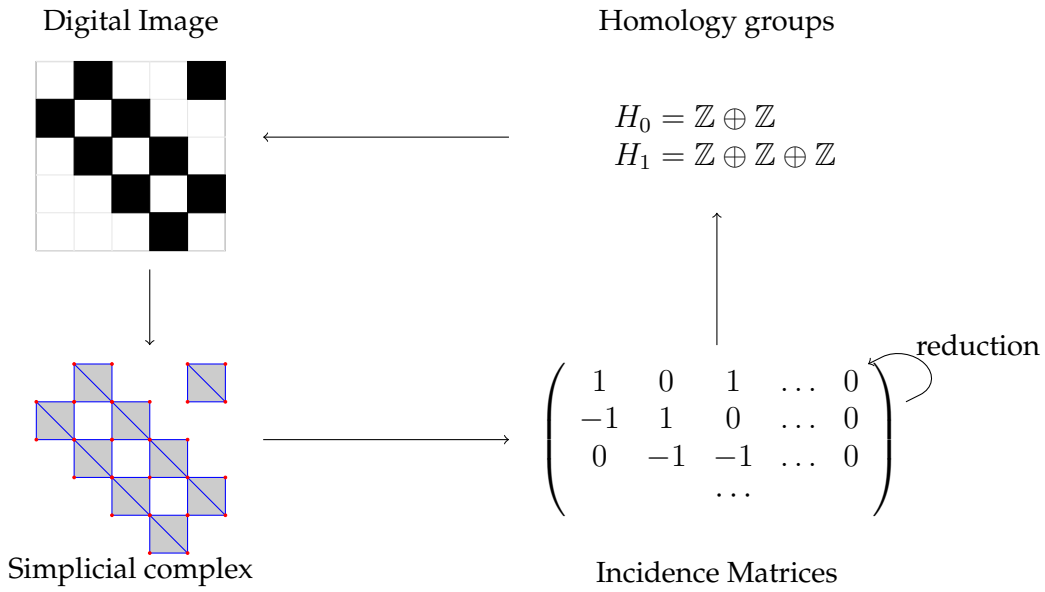


FIGURE 2.8: Workflow to compute homology groups from digital images. The homology groups indicate that the image has two connected components and three holes.

computing the homology groups associated with a digital image, we can obtain the number of connected components of the image.

The aforementioned workflow to compute homology groups from digital images was fully verified, see Heras, Dénès, Mata, Mörtberg, Poza, and Siles, 2012; Poza, Domínguez, Heras, and Rubio, 2014.

2.1.5 Discussion

Up to the best of our knowledge, four tools have been developed to quantify synapses and measure synaptic density: Green and Red Puncta (Shiwarski, Dagda, and T., 2014), Puncta Analyzer (Wark, 2013), SynD(Schmitz, Johannes Hjorth, and Joemail, 2011) and SynPAnal (Danielson and Lee, 2014) — a summary of the general features of these tools can be seen in Table 2.1. The rest of this section is devoted to compare SynapCountJ with these tools — such a comparison is summarized in Table 2.2.

Software	Language	Underlying Technology	Types of Images	Technique for detection
Green and Red Puncta	Java	ImageJ	tiff	Colocalization
Puncta Analyzer	Java	ImageJ2	tiff	Colocalization
SynapCountJ	Java	ImageJ	tiff and lif	Colocalization
SynD	Matlab	Matlab	tiff and lsm	Brightness
SynPAnal	Java		tiff	Brightness

TABLE 2.1: General features of the analyzed software.

There are two approaches to locate synapses in an RGB image (Red-Green-Blue image) either based on colocalization or brightness. In the former, synapses are identified as the colocalization of bright points in the red and green channels — this is the approach followed by Green and Red Puncta, Puncta Analyzer and SynapCountJ — in the latter, synapses are the bright points of a region of an image — the approach employed in SynD and SynPAnal. In both approaches, it is necessary

Software	Detection of dendrites	Threshold	Batch Processing	Dendrites length	Density	Export	Save
Green and Red Puncta	Not used	✓					
Puncta Analyzer	Manual ROI	✓				✓	
SynapCountJ	Manual	✓	✓	✓	✓	✓	✓
SynD	Automatic	✓	✓	✓		✓	✓
SynPAnal	Manual	✓		✓	✓	✓	✓

TABLE 2.2: Features to quantify synapses and synaptic density of the analyzed software

.

a threshold that can be manually adjusted to increase (or decrease) the number of detected synapses; such a functionality is supported by all the tools.

In the quantification of synapses from RGB images, it is instrumental to determine the region of interest (i.e. the dendrites of the neurons where the synapses are located); otherwise, the analysis will not be precise due to noise coming from irrelevant regions or the background of the image — this happens in the Green and Red Puncta tool since it considers the whole image for the analysis. Puncta Analyzer allows the user to fix a rectangle containing the dendrites of the neuron, but this is not completely precise since some regions of the rectangle might contain points considered as synapses that do not belong to the structure of the neuron. SynD is the only software that automatically detects the dendrites of a neuron; however, it can only be applied to neurons with a cell-fill marker, and does not support the analysis from specific regions, such as soma or distal dendrites. SynapCountJ and SynPAnal provide the functionality to manually draw the dendrites of the image; allowing the user to designate the specific areas where quantification is restricted.

The main output produced by all the available tools is the number of synapses of a given image; additionally, SynapCountJ, SynD and SynPAnal provides the length of the dendrites; and, SynapCountJ and SynPAnal are the only tools that output the synaptic density per micron. All the tools but Green and Red Punctua can export the results to an external file for storage and further processing.

Finally, as we have explained in Section 2.1.2, images obtained from the same biological experiment usually have similar settings; hence, batch processing might be useful. This functionality is featured by SynapCountJ and SynD, and requires a previous step of saving the configuration of an individual analysis. SynPAnal does not support batch processing, but the configuration of an individual analysis can be saved to be later applied in other individual analysis.

As a summary, SynapCountJ is more complete than the rest of available programmes. It can use different types of synaptic markers and can process batch images. Furthermore, a differential feature of SynapCountJ is that it is based on a topological algorithm (namely, computing the number of connected components in a combinatorial structure), allowing us to validate the correctness of our approach by means of formal methods in software engineering.

2.1.6 Conclusions

SynapCountJ is an ImageJ plug-in that provides a semi-automatic procedure to quantify synapses and measure synaptic density from immunofluorescence images

obtained from neuron cultures. This plug-in has been tested not only with neurons in development, but also with the neuromuscular union of Drosophila (genus of small flies); therefore, it can be applied to the study of images that contain two synaptic markers and a determined structure. The results obtained with SynapCountJ are consistent with the results obtained manually; and SynapCountJ dramatically reduces the time required for the quantification of synapses. Moreover, the realiability of SynapCountJ has been increased by validating some of its computations using the formally verified module for digital images of Kenzo.

2.1.7 Availability and Software Requirements

SynapCountJ is an ImageJ plug-in that can be downloaded, together with its documentation, from the external reference *SynapCountJ*. SynapCountJ is open source and available for use under the GNU General Public License. This plug-in runs within both ImageJ and Fiji (Schindelin, Arganda-Carreras, and Frise, 2012) and has been tested on Windows, Macintosh and Linux machines.

2.2 Verification

The work explained in this section has been presented and published in the Conference on Intelligent Computer Mathematics (CICM) held in Bath, United Kingdom, 2013, under the title "Verifying a platform for digital imaging: a multi-tool strategy", see Heras, Mata, Romero, Rubio, and Sáenz, 2013.

2.2.1 Introduction

As a consequence of attempting to validate the computation of connected components — explained in Section 2.1.4 —, we considered going one step further and verifying the code used in the processing of images.

As previously mentioned (Section 2.1), we use Fiji in some pre-processing steps before undertaking a homological digital processing of images.

Due to the fact that the reliability of results is instrumental in biomedical research, we are working towards the certification of the programmes that we use to analyse biomedical images — here, certification means verification assisted by computers. In a previous work, see Heras, Poza, and Rubio, 2012; Heras, Coquand, Mörtberg, and Siles, 2013, two homological techniques to process biomedical images were formalised. However, in both cases, the verification of Fiji's pre-processing step was not undertaken.

Being a software built by means of plug-ins developed by several authors, Fiji is messy, very flexible (programme pieces are used in some occasions with a completely different objective from the one they were designed), contains many redundancies and dead code, and so on. In summary, it is a big software system which has not been devised to be formally verified. So, this endeavour is challenging.

There are several approaches to verify Java code; for instance, proving the correctness of the associated Java bytecode, see Liu and S., 2004. In this Section, we use Krakatoa (Filliâtre and Marché, 2007) to specify and prove the correctness of Fiji/Java programmes. This experience allows us to evaluate both the verification of *production* Fiji/Java code, and the Krakatoa tool itself in an unprepared scenario.

Krakatoa uses some automated theorem provers (as Alt-Ergo (Bobot, Conchon, Contejean, Iguernelala, Lescuyer, and Mebsout, 2008) or CVC3 (Barrett and Tinelli,

2007)) to discharge the proof obligations generated by means of the Why tool (Filliâtre and Marché, 2007). When a proof obligation cannot be solved by means of the automated provers, the corresponding statement is generated in Coq (Coq development team, 2012). Then, the user can prove the missing property by interacting with this proof assistant.

In this description, we add the ACL2 theorem prover (Kaufmann and Moore, 2012). ACL2 is an automated theorem prover but more powerful than others. In many aspects, working with ACL2 is more similar to interactive provers than to automated ones (see Kaufmann and Moore, 2012). Instead of integrating ACL2 in the architecture of Why/Krakatoa, we have followed another path leaving untouched the Why/Krakatoa code. Our approach reuses a proposal presented in (Aransay, Divasón, Heras, Lambán, Pascual, Rubio, and Rubio, 2012) to translate first-order Isabelle/HOL theories to ACL2 through an XML specification language called XLL (Aransay, Divasón, Heras, Lambán, Pascual, Rubio, and Rubio, 2012). We have enhanced our previous tools to translate Coq theories to the XLL language, and then apply the tools developed in (Aransay, Divasón, Heras, Lambán, Pascual, Rubio, and Rubio, 2012) to obtain ACL2 files. In this way, we can use, unmodified, the Why/Krakatoa framework; the Coq statements are then translated (if needed) to ACL2, where an automated proof is tried; if it succeeds, Coq is only an intermediary specification step; otherwise, both ACL2 or Coq can be interactively used to complete the proof.

2.2.2 Context

The platforms for processing images, known as Fiji and ImageJ, offer plug-ins and macros with different functionalities which allow us, among other things, to binarize an image via different threshold algorithms, homogenise images through filters such as the "median filter" or obtain the maximum projection of a stack of images.

In the frame of the ForMath European project (ForMath, 2010–2013), one of the tasks is devoted to the topological aspects of digital image processing. The objective consists of formalising enough mathematics to verify programmes in the area of biomedical imaging. In collaboration with the SSP-group (directed by Miguel Morales, PhD), several plug-ins for Fiji have been developed — for instance, the plug-in explained in the previous Section (Section 2.1).

These programmes are devoted to analysing the effects of some drugs on the neuronal structure. At the end of such analyses, some homological processing is needed (standard homology groups in SynapCountJ and persistent homology in NeuronPersistentJ, plug-in explained in Chapter 4). As explained in the introduction, we have verified these last steps (Heras, Poza, and Rubio, 2012; Heras, Coquand, Mörtberg, and Siles, 2013). But all the pre-processing steps, based on already-built Fiji plug-ins and tools, kept unverified. This is the gap we try to fill now, by using the facilities presented in the sequel.

Next, the tools used are explained.

Why/Krakatoa: Specifying and verifying Java code.

The Why/Krakatoa tools (Filliâtre and Marché, 2007) are an environment for proving the correctness of Java programmes annotated with JML (Java Modelling Language, Burdy, 2005) specifications which have been successfully applied in different contexts, see (Barthe, Pointcheval, and Zanella-Béguelin, 2012). The environment involves three distinct components:

- Krakatoa tool, which reads the annotated Java files and produces a representation of the semantics of the Java programme into Why's input language
- Why tool, which computes proof obligations (POs) for a core imperative language annotated with pre- and post-conditions,
- several automated theorem provers which are included in the environment and are used to prove the POs.

When some PO cannot be solved by means of the automated provers, corresponding statements are automatically generated in Coq (COQ development team, 2012), so that the user can prove the missing properties in this interactive theorem prover. The POs generation is based on a *Weakest Precondition calculus* and the validity of all generated POs implies the soundness of the code with respect to the given specification. The Why/Krakatoa tools are available as open source software at the external reference in *Krakatoa*.

Coq and ACL2: Interactive theorem proving.

Coq (COQ development team, 2012) is an interactive proof assistant for constructive higher-order logic based on the Calculus of Inductive Construction. This system provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs. Coq has been successfully used in the formalisation of relevant mathematical results; for instance, the Feit-Thompson Theorem (Gonthier, 2013).

ACL2 (Kaufmann and Moore, 2012) is a programming language, a first order logic and an automated theorem prover. Thus, the system constitutes an environment in which algorithms can be defined and executed, and their properties can be formally specified and proved with the assistance of a mechanical theorem prover. ACL2 has elements of both interactive and automated provers. ACL2 is automatic in the sense that once started on a problem, it proceeds without human assistance. However, non-trivial results are not usually proved in the first attempt, and the user has to lead the prover to a successful proof providing a set of lemmas, inspired by the failed proof generated by ACL2. This system has been used for a variety of important formal methods projects of industrial and commercial interest (Hardin, 2010) and for implementing large proofs in mathematics.

See more information about these tools in Appendix C.

2.2.3 Methodology

The method which we have applied to verify the Fiji code can be split into the following steps.

Algorithm 3.
`Input`: Java code
`Output`: Verified code

1. Transforming Fiji code into compilable Krakatoa code.
2. Specifying Java programmes.
3. Applying the Why tool.

4. If all the proof obligations are discharged automatically by the provers integrated in Krakatoa, stop; the verification has ended.

5. Otherwise, study the failed attempts, and consider if they are under-specified; if it is the case, go again to step (2).

6. Otherwise, consider the Coq expressions of the still-non-proven statements and transform them to ACL2.

7. If all the statements are automatically proved in ACL2, stop; the verification has ended.

8. Otherwise, by inspecting the failed ACL2 proofs, decide if other specifications are needed (go to item (2)); if it is not the case, decide if the missing proofs should be carried out in Coq or ACL2.

The first step is the most sensitive one, because it is the only point where informal (or, rather, semi-formal) methods are needed. Thus, some unsafe, and manual, code transformation can be required. To minimize this drawback, we apply two strategies:

- First, only well-known transformations are applied; for instance, we eliminate inheritance by "flattening" out the code, but without touching the real behaviour of methods.

- Second, the equivalence between the original code and the transformed one is systematically tested.

Employing both points together increases the reliability of our approach; a more detailed description of the transformations needed in step one are explained in subsection titled "Transforming Fiji-Java to Krakatoa-Java". Explanations about step two are provided in subsection under title "Specifying programmes for digital imaging". Steps three to six are mechanized in Krakatoa. The role of ACL2 (steps six to eight) is explained in subsection titled "The role of ACL2" and, by means of an example, in section of experimental resutls (Section 2.2.4).

Transforming Fiji-Java to Krakatoa-Java

In its current state, the Why/Krakatoa system does not support the complete Java programming language and has some limitations. In order to make a Fiji Java programme compilable by Krakatoa we have to take several steps.

Algorithm 4.
`Input`: Java code from Fiji platform
`Output`: Compilable code by Krakatoa

1. Delete annotations. Krakatoa JML annotations will be placed between `\*@` and `@*\`. Therefore, we need to remove other Java Annotations preceded by `@`.

2. Move the classes that are referenced in the file that we want to compile into the directory *whyInstallationDir/java_api/*. For example, the class *RankFilters* uses the class *java.awt.Rectangle*; therefore, we need to create the folder *awt* inside the *java* directory that already exists, and put the file *Rectangle.java* into it. Moreover, we remove the body of the methods because only the headers and

the fields of the classes will be taken into consideration. We must iterate this process over the classes which we add. The files that we add into the *java_api* directory can contain `import`, `extends` and `implements` clauses although the file that we want to compile cannot do it — Krakatoa does not support these mechanisms. This is a tough process: for instance, to make use of the class *Rectangle*, we need to add fifteen classes.

3. Reproduce the behaviour of the class which we want to compile. Considering that we are not able to use `extends` and `implements` clauses, we need to move the code from the upper classes into one whichcompiles and exhibits the same behaviour. For instance, the class *BinaryProcessor* extends from *ByteProcessor* and within its constructor it calls the constructor of *ByteProcessor*; to solve this problem we need to copy the body of the super constructor at the beginning of the constructor of the class *BinaryProcessor*. If we find the use of interfaces, we can ignore them and remove the `implements` clause because the code will be implemented in the class that makes use of the interface.

4. Remove `import` clauses. We need to delete them from the file that we want to compile and change the places where the corresponding classes appear with the full path codes. If for example we use the class *Rectangle* as we have explained in the second Step, we need to replace it by *java.awt.Rectangle*.

5. Owing to the fact that packages of declarations are forbidden, we need to remove them with the purpose of halting "*unknown identifier packageName*" errors.

6. Rebuild native methods. The Java programming language allows the use of *native* methods, which are written in C or C++ (programming languages) and might be specific to a hardware and operating system platform. For example, many of the methods in the class *Math* (which perform basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions) simply call the equivalent method included in a different class named *StrictMath* for their implementation, and then the code in *StrictMath* of these methods is just a *native* call. Since native methods are not written in Java, they cannot be specified and verified in Krakatoa. Therefore, if our Fiji programme uses some native methods, it will be necessary to rewrite them with our own code. See in Section 2.2.4 our implementation (and specification) of the native method `sqrt` computing the square root of a number of type double, based on Newton's algorithm.

7. Add a clause in *if-else* structures in order to remove "*Uncaught exception: Invalid_argument("equal: abstract value")*". We can find an example in the method `filterEdge` of the class *MedianFilter* where we have to replace the last *else...* clause by *else if(true)....*

8. Remove debugging useless references. We have mentioned in a previous step that we can only use certain static methods that we have manually added to the Why core code and therefore we can remove some debugging instructions like `System.out.println(...)`. We can find the usage of standard output printing statement in the method `write` of the class *IJ* (class of the Application Programming Interface (API) from Fiji).

9. Modify the declaration of some variables to avoid syntax errors. There can be some compilation errors with the definition of some floats and double values that match the pattern <*number*>*f* or <*number*>*d*. We can see an example in the line 180 of the file *RankFilters.java*; we have to transform the code: `float f = 50f;` into `float f = 50`.

10. Change the way that Maximum and Minimum float numbers are written. Those two special numbers are located in the file *Float.java* and there are widely used to avoid overflow errors, but they generate an error due to the *eP* exponent. To stop having errors with expressions like `0x1.fffffeP+127d` we need to convert it into `3.4028235e+38f`.

Specifying programmes for digital imaging

As already said and it is explained in Appendix C, Fiji and ImageJ are open source projects and many different people from many different teams (some of them not being computer scientists) are involved in the development of the different plug-ins for these platforms. This implies that the code of these programmes is in general not suitable for its formal verification and a deep previous transformation process, following the steps explained in Section 2.2.3, is necessary before introducing the Java programmes into the Why/Krakatoa system. Even after this initial transformation, Fiji programmes usually remain complex and their specification in Krakatoa is not a direct process. In this section we present some examples of Fiji methods that we have specified in JML (Java Modelling Language) trying to show the difficulties we have faced.

Once that a Fiji Java programme has been adapted, following the ideas of Section 2.2.3, and is accepted by the Why/Krakatoa application, the following step in order to certify its correctness consists in specifying its behaviour (that is, its precondition and its postcondition) by writing annotations in the Java Modelling Language (JML, Burdy, 2005) . The precondition of a method must be a proposition introduced by the keyword `requires` which is supposed to hold in the pre-state, that is, when the method is called. The postcondition is introduced by the keyword `ensures`, and must be satisfied in the post-state, that is, when the method returns normally. The notation `\result` denotes the returned value. To differentiate the value of a variable in the pre- and post- states, we can use the keyword `\old` for the pre-state.

Let us begin by showing a simple example. The following Fiji method, included in the class *Rectangle*, translates an object by given horizontal and vertical increments `dx` and `dy`.

```
/*@ ensures x == \old(x) + dx && y == \old(y) + dy;
  @*/
public void translate(final double dx, final double dy) {
    this.x += dx; this.y += dy;
}
```

The postcondition expresses that the field `x` is modified by incrementing it by `dx`, and the field `y` is increased by `dy`. In this case no precondition is given since all values of `dx` and `dy` are valid, and the keyword `\result` does not appear because the returned type is `void`.

Using this JML specification, Why/Krakatoa generates several lemmas (Proof Obligations) which express the correctness of the programme. In this simple case, the proof obligations are elementary and they are easily discharged by the automated theorem provers Alt-Ergo (Bobot, Conchon, Contejean, Iguernelala, Lescuyer,

and Mebsout, 2008) and CVC3 (Barrett and Tinelli, 2007), which are included in the environment. The proofs of these lemmas guarantee the correctness of the Fiji method `translate` with respect to the given specification.

Unfortunately, this is not the general situation because, as already said, Fiji code has not been designed for its formal verification and can be very complicated; so, in most cases, Krakatoa is not able to prove the validity of a programme from the given precondition and postcondition. In order to formally verify a Fiji method, it is usually necessary to include annotations in the intermediate points of the programme. These annotations, introduced by the keyword `assert`, must hold at the corresponding programme point. For loop constructs (while, for, etc), we must give an *inductive invariant*, introduced by the keyword `loop_invariant`, which is a proposition which must hold at the loop entry and be preserved by any iteration of the loop body. One can also indicate a `loop_variant`, which must be an expression of type integer, which remains non-negative and decreases at each loop iteration, assuring in this way the termination of the loop. It is also possible to declare new logical functions, lemmas and predicates, and to define *ghost variables* which allow one to monitor the programme execution.

Let us consider the following Fiji method included in the class *RankFilters*. It implements Hoare's find algorithm (also known as *quickselect*) for computing the n-th lowest number in part of an unsorted array, generalizing in this way the computation of the median element. This method appears in the implementation of the"median filter", a process very common in digital imaging which is used in order to achieve greater homogeneity in an image and provide continuity, obtaining in this way a good binarization of the image.

```
/*@ requires buf!=null && 1<= bufLength <= buf.length && 0<=n <bufLength;
  @ ensures Permut{Old,Here}(buf,0,bufLength-1)
  @    && (\forall integer k; (0<=k<=n-1 ==> buf[k]<=buf[n])
  @              && (n+1<=k<=bufLength-1 ==> buf[k]>=buf[n]))
  @    && \result==buf[n] ;
  @*/
public final static float findNthLowestNumber
               (float[] buf, int bufLength, int n) {
    int i,j;
    int l=0;
    int m=bufLength-1;
    float med=buf[n];
    float dum ;
    while (l<m) {
        i=l ;
        j=m ;
        do {
            while (buf[i]<med) i++ ;
            while (med<buf[j]) j-- ;
            dum=buf[j];
            buf[j]=buf[i];
            buf[i]=dum;
            i++ ; j-- ;
        } while ((j>=n) && (i<=n)) ;
        if (j<n) l=i ;
        if (n<i) m=j ;
        med=buf[n] ;
    }
    return med ;
}
```

Given an array `buf` and two integers `bufLength` and `n`, the Fiji method `findNthLowestNumber` returns the $(n+1)$-th lowest number in the first `bufLength`

components of `buf`. The precondition expresses that `buf` is not null, `bufLength` must be an integer between 1 and the length of `buf`, and `n` is an integer between 0 and `bufLength` $-$ 1. The definition of the postcondition includes the use of the predicate `Permut`, a predefined predicate, which expresses that when the method returns the (modified) `bufLength` first components of the array `buf` must be a permutation of the initial ones. The array has been reordered such that the components $0, \ldots, n-1$ are smaller than or equal to the component n, and the elements at positions $n+1, \ldots,$ `bufLength` $-$ 1 are greater than or equal to that in n. The returned value must be equal to `buf[n]`, which is therefore the $(n+1)$-th lowest number in the first `bufLength` components of `buf`.

In order to prove the correctness of this programme, we have included different JML annotations in the Java code. First of all, loop invariants must be given for all `while` and `do` structures appearing in the code. Difficulties have been found in order to deduce the adequate properties for invariants which must be strong enough to imply the programme (and other loops) postconditions; automated techniques like discovery of loop invariants (Ireland and Stark, 1997) will be used in the future. We show as an example the loop invariant (and variant) for the exterior while, which is given by the following properties:

```
/*@ loop_invariant
  @ 0<=l<=n+1 && n-1<=m<=bufLength-1 && l<=m+2
  @ && (\forall integer k1 k2; (0<=k1<=n && m+1<=k2<=bufLength-1)
  @          ==> buf[k1]<=buf[k2])
  @ && (\forall integer k1 k2; (0<=k1<=l-1 && n<=k2<=bufLength-1)
  @          ==> buf[k1]<=buf[k2])
  @ && Permut{Pre,Here}(buf,0,buf.length-1) && med==buf[n]
  @ && ((l<m)==> ((l<=n)&&(m>=n)));
  @ loop_variant m - l+2;
  @*/
```

To help the automated provers to verify the programme and prove the generated proof obligations it is also necessary to introduce several assertions in some intermediate points of the programme and to use ghost variables which allow the system to deduce that the loop variant decreases.

The final specification of this method includes 78 lines of JML annotations (for only 24 Java code lines). Krakatoa/Why produces 175 proof obligations expressing the validity of the programme. The automated theorem prover Alt-Ergo is able to demonstrate all of them, although in some cases more than a minute (in an ordinary computer) is needed; another prover included in Krakatoa, CVC3, is, on the contrary, only capable of proving 171. The proofs of the lemmas obtained by means of Alt-Ergo certify the correctness of the method with respect to the given specification.

In this particular example, the automated theorem provers integrated in Krakatoa are enough to discharge all the proof obligations. In other cases, some properties are not proven, and then one should try to prove them using *interactive* theorem provers, as Coq. In this architecture, we also introduce the ACL2 theorem prover, as explained in the next subsection.

The role of ACL2

Over the following section, the role played by ACL2 in this infrastructure to verify the correctness of Java programmes is explained. The Why platform relies on automated provers, such as Alt-Ergo or CVC3, and interactive provers, such as Coq or PVS, to discharge proof obligations; however, it does not consider the ACL2 theorem

prover to that aim. We believe that the use of ACL2 can help in the proof verification process. The reason is twofold.

- The scope of automated provers is smaller than the one of ACL2; therefore, ACL2 can prove some of the proof obligations which cannot be discharged by automated provers.
- Moreover, interactive provers lack automation; then, ACL2 can automatically discharge proof obligations which would require user interaction in interactive provers.

A Proof General extension called *Coq2ACL2* was developed, which integrates ACL2 in this infrastructure to verify Java programmes; in particular, we work with ACL2(r) a variant of ACL2 which supports the real numbers (Gamboa and Kaufmann, 2001) — the formalisation of real analysis in theorem provers is an outstanding topic, see Boldo, Lelay, and Melquiond, 2013. Coq2ACL2 features three main functions:

F1. it transforms Coq statements generated by Why to ACL2;

F2. it automatically sends the ACL2 statements to ACL2; and

F3. it displays the proof attempt generated by ACL2.

If all the statements are proved in ACL2; then, the verification process is ended. Otherwise, the statements must be manually proved either in Coq or ACL2.

The major challenge in the development of Coq2ACL2 was the transformation of Coq statements to ACL2. There is a considerable number of proposals documented in the literature related to the area of theorem proving interoperability. It is possible to classify the translations between proof assistants in two groups: *deep* (Gordon, Kaufmann, and Ray, 2011; Jacquel, Berkani, Delahaye, and Dubois, 2011; Codescu, 2012) and *shallow* (Keller and Werner, 2011; Obua and Skalberg, 2006; Denney, 2000).

In this work, we took advantage of a previous shallow development presented in Aransay, Divasón, Heras, Lambán, Pascual, Rubio, and Rubio, 2012, where a framework called *I2EA* to import Isabelle/HOL theories into ACL2 was introduced. That approach can be summarized as follows. Due to the different nature of Isabelle/HOL and ACL2, it is not feasible to replay proofs that have been recorded in Isabelle/HOL within ACL2. Nevertheless, Isabelle/HOL statements dealing with first order expressions can be transformed to ACL2; and then, they can be used as a schema to guide the proof in ACL2.

A key component in the framework presented in (Aransay, Divasón, Heras, Lambán, Pascual, Rubio, and Rubio, 2012) was an XML-based specification language called *XLL* (that stands for Xmall Logical Language). XLL was developed to act as an intermediate language to port Isabelle/HOL theories to both ACL2 and an Ecore model (given by UML class definitions and OCL restrictions) — the translation to Ecore serves as a general purpose formal specification of the theory carried out. The transformations among the different languages are done by means of XSLT (Extensible Stylesheet Language Transformations) and some Java programmes. Coq system has been integrated into the I2EA framework as can be seen in Figure 2.9; in this way, we can reuse both the XLL language and some of the XSLT files developed in (Aransay, Divasón, Heras, Lambán, Pascual, Rubio, and Rubio, 2012) to transform (first-order like) Coq statements to ACL2.

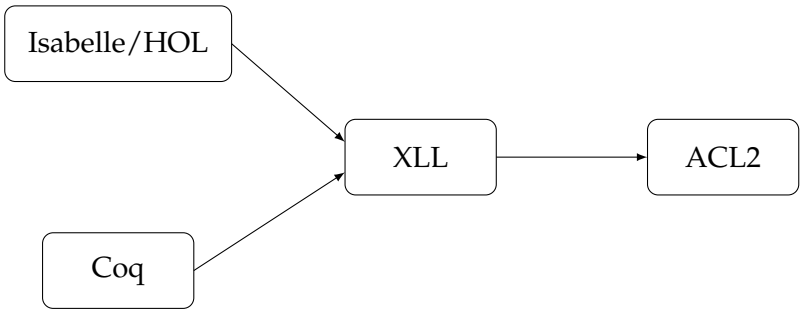


FIGURE 2.9: Architecture of the I2EA framework integrating Coq.

In particular, functionality **F1** of Coq2ACL2 can be split into two steps:

1. given a Coq statement, Coq2ACL2 transforms it to an XLL file using a Common Lisp translator programme; then,
2. the XLL file is transformed to ACL2 using an XSLT file previously developed in (Aransay, Divasón, Heras, Lambán, Pascual, Rubio, and Rubio, 2012).

In this way, ACL2 has been integrated into our environment to verify Java programmes. As we will see in the following section, this has meant an improvement to automatically discharge proof obligations.

2.2.4 Experimental Results

An immediate consequence of the collaboration with the SSP-group, was dealing with images acquired by microscopy techniques from biological samples. These samples have volume and the object of interest is not always in the same plane. For this reason, it is necessary to obtain different planes from the same sample to get more information. This means that several images are acquired in the same XY plane at different levels of Z. To work with this stack of images, it is often necessary to make their *maximum projection*. To this aim, Fiji provides several methods such as maximum intensity or standard deviation to obtain the maximum projection of a set of images.

We consider the Fiji code for computing the maximum projection of a set of images based on the standard deviation, which uses in particular the method `calculateStdDev` located in the class *ImageStatistics*.

```
double calculateStdDev(double n, double sum, double sum2) {
    double stdDev = 0.0;
    if (n>0.0) {
       stdDev = (n*sum2-sum*sum)/n;
       if (stdDev>0.0)
          stdDev = Math.sqrt(stdDev/(n-1.0));
       else
          stdDev = 0.0;
    } else
    stdDev = 0.0;
}
```

The inputs are `n` (the number of data to be considered), `sum` (the sum of all considered values; in our case, these values will obtained from the pixels in an image) and `sum2` (the sum of the squares of the data values). The method `calculateStdDev`

computes the standard deviation from these inputs and assigns it to the field `stdDev`. The specification of this method is given by the following JML annotation.

```
/*@ requires ((n==1.0)==> sum2==sum*sum) && ((n<=0.0) || (n>=1.0)) ;
  @ behaviour negative_n :
  @   assumes  n<=0.0 || (n>0.0 && (n*sum2-sum*sum)/n <=0.0);
  @   ensures stdDev == 0.0;
  @ behaviour normal_behaviour :
  @   assumes n>=1.0 && ((n*sum2-sum*sum)/n  > 0.0);
  @   ensures is_sqrt(stdDev,(double)((n*sum2-sum*sum)/n/(n-1.0)));
@*/
```

The precondition, introduced by the keyword `requires`, expresses that in the case $\texttt{n} = 1$ (that is, there is only one element in the data) the inputs `sum` and `sum2` must satisfy $\texttt{sum2} = \texttt{sum} * \texttt{sum}$. Moreover we must require that `n` is less than or equal to 0 or greater than or equal to 1 to avoid the possible values in the interval $(0, 1)$; for `n` in this interval one has $\texttt{n} - 1 < 0$ and then it is not possible to apply the square root function to the given argument $\texttt{stdDev}/(\texttt{n} - 1.0)$. This fact has not been taken into account by the author of the Fiji programme because in all real applications the method will be called with `n` being a natural number; however, to formalise the method we must specify this particular situation in the precondition. For the postcondition we distinguish two different behaviours: if `n` is non-positive or `sum` and `sum2` are such that $\texttt{n} * \texttt{sum2} - \texttt{sum} * \texttt{sum} < 0$, the field `stdDev` is assigned to 0; otherwise, the standard deviation formula is applied and the result is assigned to the field `stdDev`. The predicate `is_sqrt` is previously defined.

For the proof of correctness of the method `calculateStdDev` in Krakatoa, it is necessary to specify (and verify) the method `sqrt`. The problem here, as already explained in previous section, is that the method `sqrt` of the class *Math* simply calls the equivalent method in the class *StrictMath*, and the code in *StrictMath* of the method `sqrt` is just a native call and might be implemented differently on different Java platforms. In order to give a JML specification of the method `sqrt` is necessary then to rewrite it with our own code. The documentation of *StrictMath* states "*To help ensure portability of Java programmes, the definitions of some of the numeric functions in this package require that they produce the same results as certain published algorithms. These algorithms are available from the well-known network library netlib as the package "Freely Distributable Math Library", fdlibm*". In the case of the square root, one of these *recommended* algorithms is Newton's method; based on it, we have implemented and specified in JML the computation of the square root of a given (non-negative) input of type double.

```
/*@ requires c>=0 && epsi > 0 ;
  @ ensures \result >=0 &&  (\result*\result>=c)
  @    && \result*\result - c < epsi ;
  @*/
public double sqrt(double c, double epsi){
    double t;
    if (c>1) t= c;
        else t=1.1;
    /*@ loop_invariant
      @ (t >= 0) && (t*t> c) ;
      @*/
    while (t* t - c  >= epsi) {
        t = (c/t + t) / 2.0;
    }
    return t;
}
```

```
/*@ requires c>=0  ;
  @ ensures (\result >=0) && (\result*\result>=c)
  @ && (\result*\result - c < 1.2E-7);
  @*/
public double sqrt(double c){
    double eps=1.2E-7;
    return sqrt(c,eps);
}
```

The first method computes the square root of a double `x` with a given precision `epsi`; the second one calls the previous method with a precision less than $1.2E-7$. Using JUnit, we have run one million tests between $1E9$ and $1E-9$ to show that the results of our method `sqrt` have similar precision to those obtained by the *original* method `Math.sqrt`. Here, we applied the "first test, then verify" approach — intensive testing can be really useful to find bugs (and can save us time) before starting the verification process.

From the given JML specification for the Fiji method `calculateStdDev` and our `sqrt` method, Why/Krakatoa produces 52 proof obligations, 9 of them corresponding to lemmas that we have introduced and which are used in order to prove the correctness of the programmes. Alt-Ergo is able to prove 50 of these proof obligations, but two of the lemmas that we have defined remain unsolved. CVC3 on the contrary only proves 44 proof obligations.

The two lemmas that Alt-Ergo (and CVC3) are not able to prove are the following ones:

```
/*@ lemma double_div_pos :
  @ \forall double x y; x>0 && y > 0 ==> x / y > 0;
  @*/
/*@ lemma double_div_zero :
  @ \forall double x y; x==0.0 && y > 0 ==> x / y == 0.0;
  @*/
```

In order to discharge these two proof obligations, we can manually prove their associated Coq expressions.

```
Lemma double_div_zero : (forall (x_0_0:R), (forall (y_0:R),
  ((eq x_0_0 (0)%R) /\ (Rgt y_0 (0)%R) -> (eq (Rdiv x_0_0 y_0) (0)%R)))).

Lemma double_div_pos : (forall (x_13:R), (forall (y:R),
  ((Rgt x_13 (0)%R) /\ (Rgt y (0)%R) -> (Rgt (Rdiv x_13 y) (0)%R)))).
```

Both lemmas can be proven in Coq in less than 4 lines, but, of course, it is necessary some experience working with Coq. Therefore, it makes sense to delegate those proofs to ACL2. Coq2ACL2 translates the Coq lemmas to the following ACL2 ones. ACL2 can prove both lemmas without any user interaction (a screenshot of the proof of one of this lemmas in ACL2 is shown in Figure 2.10).

```
(defthm double_div_zero
 (implies (and (realp x_0_0) (realp y_0) (and (equal x_0_0 0) (> y_0 0)))
          (equal (/ x_0_0 y_0) 0)))

(defthm double_div_pos
 (implies (and (realp x_13) (realp y) (and (> x_13 0) (> y 0)))
          (> (/ x_13 y) 0))
```

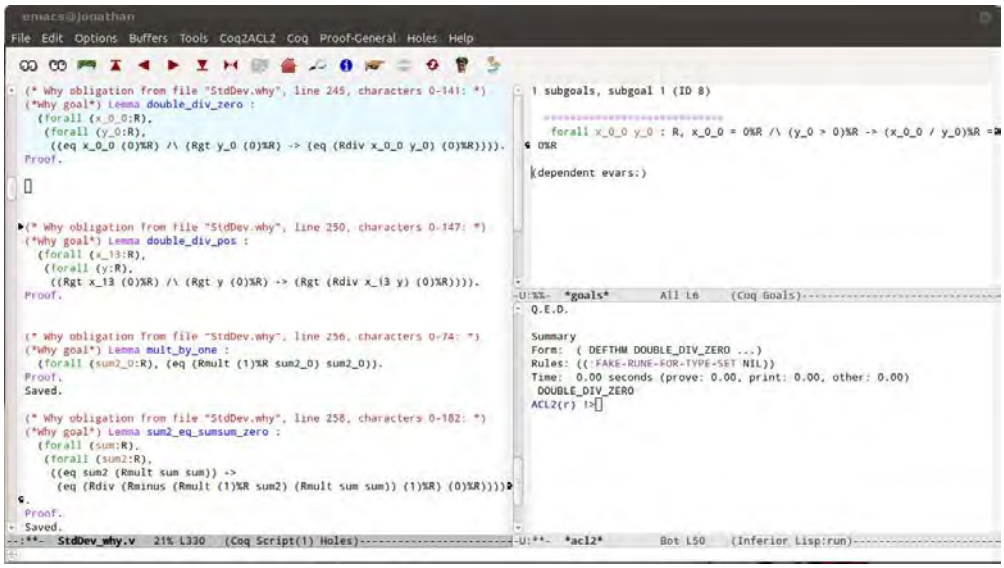


FIGURE 2.10: The Coq2ACL2 extension consists of the Coq2ACL2 menu and the right-most button of the toolbar. Left: the Coq file generated by the Why tool. Top Right: current state of the Coq proof. Bottom Right: ACL2 proof of the lemma.

2.2.5 Conclusions

This work reports an experience to verify actual Java code, as generated by different-skilled programmers, in a multi-programmer tool called Fiji. As one could suspect, the task is challenging and, in some sense, the objectives are impossible to accomplish, at least in their full extent – after our experiments, we have found that the Fiji system is *unsound*, but the errors are minor (e.g. a variable declared as a real number but which should be declared as an integer) and can be easily corrected.

Nevertheless, we defend the interest of this kind of experimental work. It is useful to evaluate the degree of maturity of the verification tools (Krakatoa, in our case). In addition, by a careful examination of the code really needed for a concrete application, it is possible to isolate the relevant parts of the code, and then it is possible to achieve a complete formalisation. As it was mentioned, several examples in our text showed this feature.

In addition to Krakatoa, several theorem provers (Coq and ACL2) have been used to discharge some proof obligations that were not automatically proved by Krakatoa. To this aim, it has been necessary the integration of several tools, and our approach can be considered as semi-formal: we keep transformations as simple as possible, and substantiate the process by systematic testing.

As a further interest of our work, we have reused a previous interoperability-proposal (Aransay, Divasón, Heras, Lambán, Pascual, Rubio, and Rubio, 2012), between Isabelle and ACL2, to get an integration of ACL2 (through a partial mapping from Coq to ACL2), without touching the Krakatoa kernel.

Chapter 3

Neural Density

The work explained in this chapter has been presented as a poster in the conference within the field of biology: XV Congreso Nacional Sociedad Española de Neurociencia (SENC), held in Oviedo, Spain, 2013, under the title: "NucleusJ: Developing of a plug-in for FIJI to analyze neuronal death model".

3.1 Introduction

When an ictus occurs in a brain region, part of the neurons which are in that region die. This damage can be more or less severe depending on the region it happens in, and the damaged cerebral area. Several pharmaceutical treatments have been developed aimed at minimizing this damage.

In this particular case, experts wish to observe whether it is possible to study the action of a neuroprotector when an ictus is taking place, specifically the one which prevents the death of the greatest number of neurons.

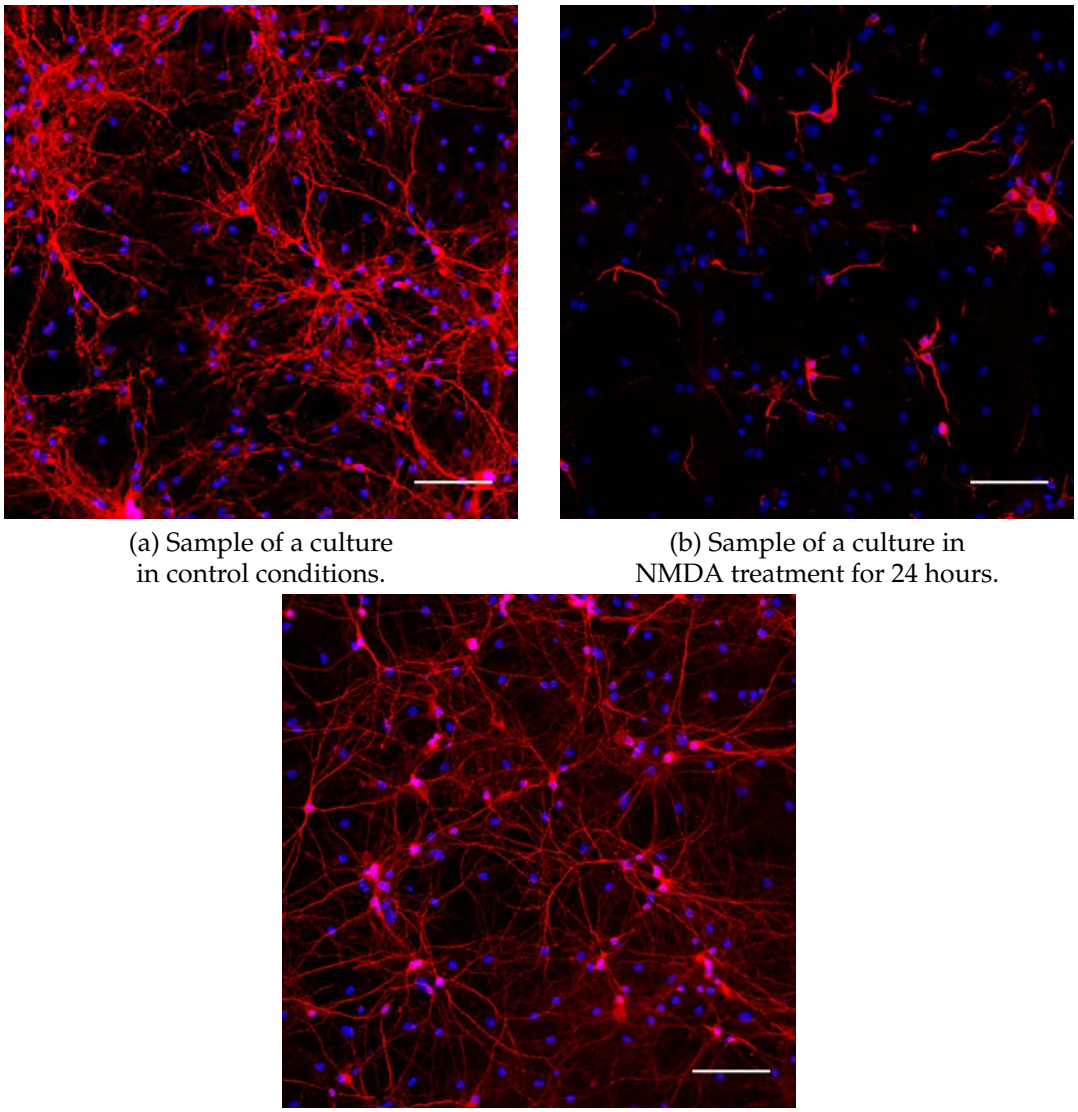
To this aim, a tool counting the number of living neurons in an image would be of great help for the neurobiologist. This is the goal of the programme described in this chapter.

In our case, the biological context for the experimental study is the following. It is known as PI3K pathway activation has previously been implicated in neuronal survival. In fact, this experiment wants to study the potential neuroprotective activity of a new PI3K activator compound (PTD4-PI3K Ac).

To this end, growing rat primary hippocampal cultures were employed. A complete description of the experimental procedure is in Appendix B, however we want to emphasise in some details. Cells were seeded in 12mm glass coverslips and treated with NMDA — glutamatergic agonist — to induce controlled neuronal death. Neuronal cultures were treated with different concentrations of NMDA in combination with a fixed concentration of the PTD4-PI3K — peptide to control of the PI3K-activity. Samples were then fixed and processed for immunostaining, employing DAPI (a specific dye for nuclei in blue) and MAP2B (a specific marker only for neuronal structures in red), see Figure 3.1. The efficacy of the treatment was estimated by calculating the percentage of survival neurons after the injury.

3.2 Methodology

This kind of image has two channels: one of them shows the nucleus of all the cells which are in the sample, and the other shows the neurons, see Figure 3.2. In principle, one could expect that the number of nuclei has to be the same as the number of neurons. However, it is common for the number of nuclei to be higher since there are other cells in the sample, such as, for example, astrocytes.



(a) Sample of a culture in control conditions.

(b) Sample of a culture in NMDA treatment for 24 hours.

(c) Sample of a culture in NMDA and PI3K-Activator treatment for 24 hours.

FIGURE 3.1: Example of hippocampal neurons cultures in control conditions (a), treated with NMDA (b) and the PI3K-Activator (c). Scale bar: 100μm.

In order to count the neurons, it is necessary to work with the split channels of the image. One of them shows the channel where the nuclei are located (from this point forward, first channel) and the other image shows the channel with the neurons (from now on, second channel).

Before advancing with the method, it is important to remove any noise which could appear in the image. To this aim, it is necessary to apply a filter. In this case, the median filter is used as it has been observed, through several computations, to perform better with this kind of image (see other alternatives to pre-process images of nuclei in Miura, 2016). This filter is applied to both channels.

This phase of pre-processing and the rest of phases, are summarized in this outline of the algorithm:

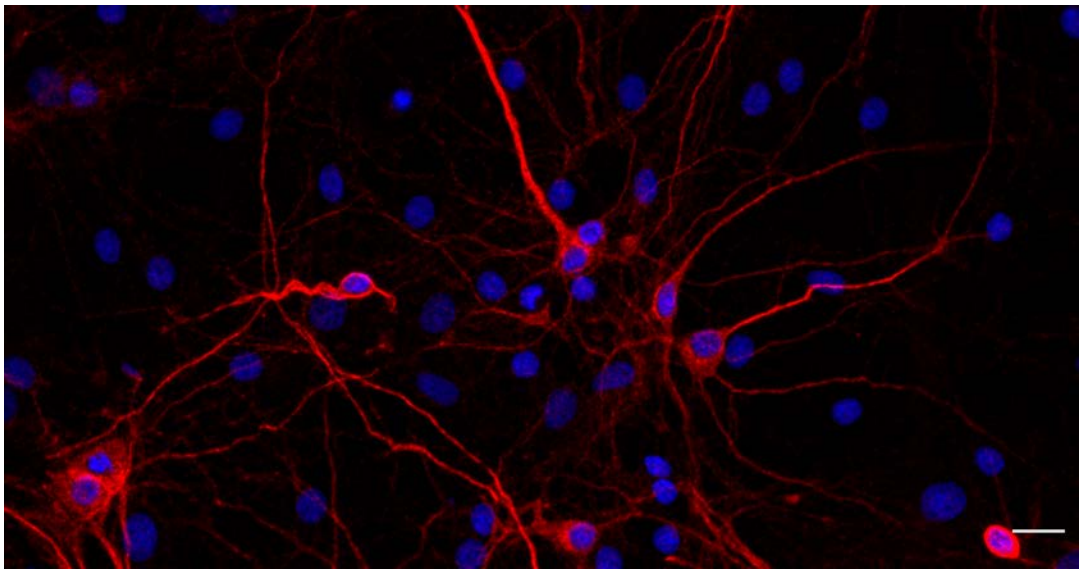


FIGURE 3.2: Example of a patch from a mosaic where the neurons are in the red channel and the nuclei are in the blue channel. Scale bar: 20μm.

Algorithm 5.
`Input`: An image with two channels and some parameters given by the user
`Output`: Three results:

a Total number of cells

b Number of neurons

c List with the regions of the nuclei of neurons marked

1st phase: Pre-processing of the image

1. Split the channels in two images
2. Remove noise applying the median filter

2nd phase: Count the number of cells in the image

1. Work on the first channel (nuclei)
   1.1. Make the image binary
   1.2. Apply morphological operations
   1.3. Select the objects by size
   1.4. Count connected components — Total number of cells (output (a))

3rd phase: Count the number of neurons and localise them

1. Work on the first channel (nuclei)
   1.1. Make the image binary through a manual threshold
   1.2. Apply morphological operations
   1.3. Select the objects by size
2. Work on the second channel (cell bodies of neurons)
   2.1. Make the image binary
3. Work on both pre-processed channels

3.1. Logical operation between the two channels (union)
3.2. Select the objects by their shape and the mode value of their intensity
3.3. Local study of the statistical mode of the objects
3.4. Count the connected components — Number of neurons (output (b))
3.5. Obtain the regions of each nucleus of neuron (output (c))

The second phase starts working with the first channel (or the channel of nuclei). This image has a lot of information about where the neurons (and the rest of the cells) can be found. In fact, there is a cell for each nucleus in the image. Therefore, it is necessary to have two criteria of size to be able to differentiate first between the cells and, the possible artefacts or noise of the image, and second, between the nuclei of the neurons and the remainder of cells.

These two parameters of size are introduced by an expert who is able to capture the differences among different experimental situations. In order to determine an approximation of these two criteria, the nuclei of all the cells and the nuclei of the neurons were measured and it was observed that a good approximation of the first size is 20-300 microns and the area for the nuclei of the neurons is between 40-200 microns.

In order to select the nuclei of the cells, the image is converted to a binary image using a variation of the IsoData algorithm, also known as iterative intermeans (Ridler and Calvard, 1978). Subsequently, morphological operations such as dilating and filling holes, are applied to obtain a better definition of the objects. The Watershed algorithm (Vincent and Soille, 1991, Roerdink and Meijster, 2000) is used to divide the nuclei which are most likely creating a cluster, see Figure 3.3.

The nuclei of the image are counted using the first criterion of size, thus, the number of cells is computed. It corresponds to the first output obtained by this method (output (a) of the outline shown in algorithm 5).

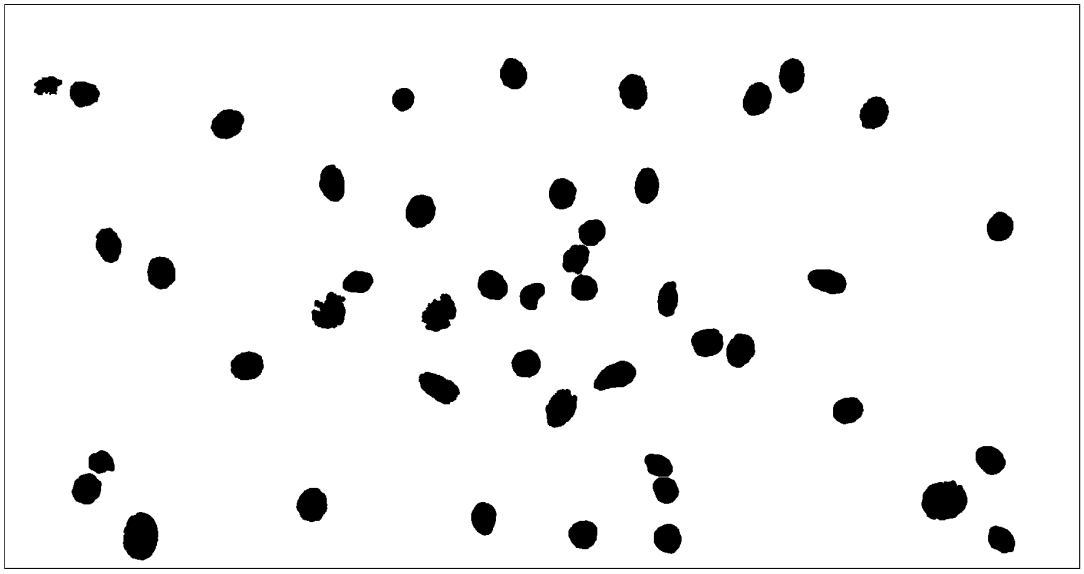


FIGURE 3.3: Binary image of all the nuclei.

After this phase, the following step (the third phase) is to determine the number of neurons. Beginning with the filtered image, the value of a threshold of intensity is chosen manually, instead of automatically. While an automatic method was considered, its use would limit the use of the same method on other types of images. In the first phase, it is possible to do it automatically since the nuclei belong to the cells

(in general); however, in this step, we only want to segment the nucleus of a specific cell, such as a neuron.

Before going any further, it is worth explaining that the next step of the study is based on comparing different computations with some values given by the expert. These values, as the size of the nuclei used in the previous step, are based on the experience of the expert in this kind of images and analysis. These parameters are necessary to determine a threshold in different aspects. Although the use of these thresholds will be explained later, let us announce that there is a threshold to determine the optimal 'aspect ratio', another threshold for the statistical mode and one more to define the number of circles fixed to perform a local study of the mode of each object selected.

The next steps are similar to those in the first phase: apply the morphological operations to the binary image, and run the Watershed algorithm to divide the clusters. Finally, the second size of the nuclei, the relation with the nuclei of neurons, is used to select the connected components which satisfy that size condition. The connected components are identified and studied with the same process explained in Chapter 2. This step returns a binary mask which will be used later.

Up to this point some of the nuclei have been selected to determine wether they belong to the nuclei of a neuron or not.

This phase consists in discriminating whose candidates are really part of a neuron. The main idea is to check in the candidate nucleus are overlapping with a binary version of the second channel. The process begins with the filtered image of the second channel; it is then made a binary image using the same method as in the first phase (a variation of the IsoData method).

A logical operation is then used to choose the objects which overlap with neurons. This operation is the union between a mask of the regions of interest selected as the possible nuclei of the neurons (previously obtained) and the binary image of the second channel. The nuclei which belong to a neuron will produce an object more circular than the rest of nuclei. This operation then gives back a new binary image where we can find the parts of the connected components which appear in both images. These connected components are studied and classified according to their shape. The ratio of the width to height of a connected component is determined, and all with a higher value of this ratio, known as the aspect ratio, are removed. In other words, the long objects are discarded as nucleus of neurons. It is the case, for instance, when the nucleus of a cell is underneath the dendrite of a neuron, see Figure 3.4.

The remaining objects are still not yet considered to be nuclei of neurons; they overlap on the second channel of the original image (without filtering). A selection is created for each object (or ROI, region of interest), and its statistical mode is computed. If this value is less than a fixed value by the user, then those particular regions of interest are removed.

The last process consists of studying locally the variability of intensity in the second channel (the channel of neurons). Our hypothesis is that a nucleus of a neuron has a determined intensity which is clearly lower around it. To measure this variability of intensity, we have chosen to study the statistical mode around the ROIs selected up to this point. This study is a local way to analyze what is happening in the area encircling a nucleus. For this step we needed another parameter given by the user: the percentage of variability which is maximum to consider that the intensity changes.

The outline of the algorithm for this step is the following.

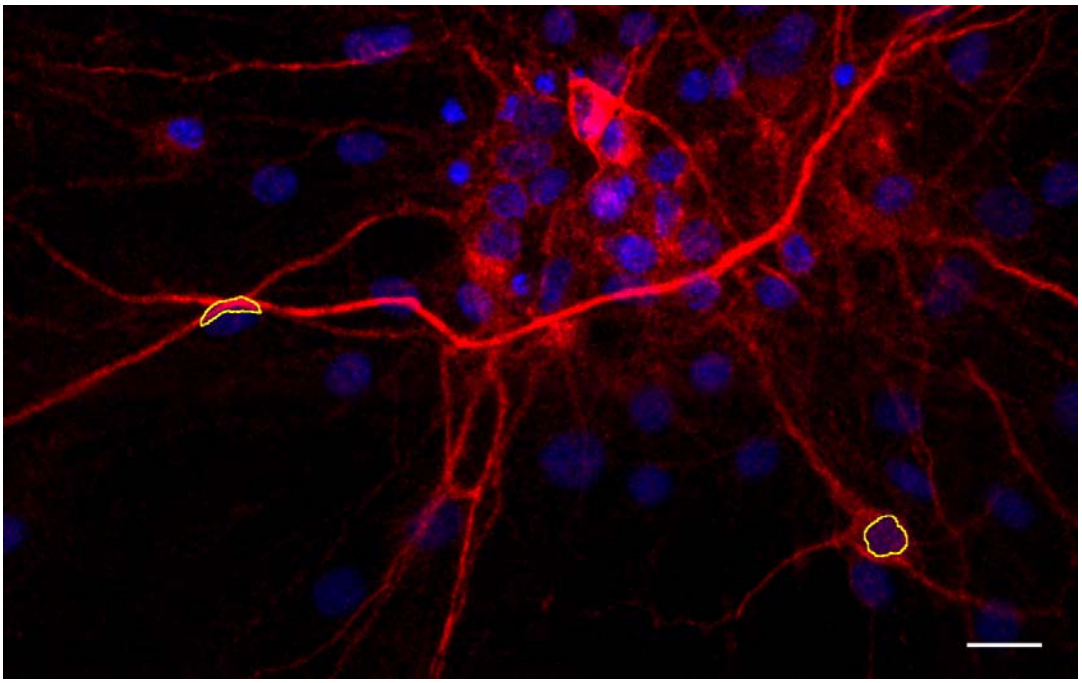


FIGURE 3.4: Example of an object (left region) to be removed due to its high aspect ratio. However, the shape of the object on the right of the image is more circular and its aspect ratio has a value close to 1; thus, it is treated as a possible neuron. Scale bar: 20μm.

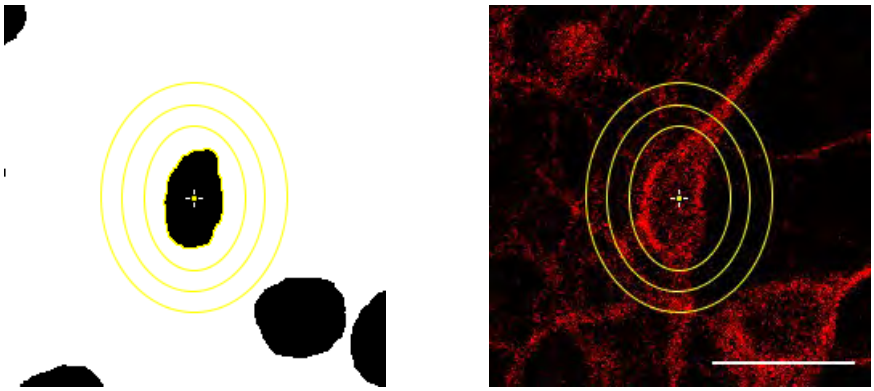
Algorithm 6.
`Input`: An image with two channels and the centroids of the regions of interest
`Output`: Number of neurons found

1. Tracing n circles: $c_1, c_2, \ldots, c_{n-1}, c_n$; Let c_1 be the smallest circle.
2. Computing the mode for each circle in the second channel of the image (channel of neurons). Let $m_1, \ldots, m_n$ be the corresponding modes for each circle.
3. Computing the ratio of the each circle to the smallest: $\frac{m_i}{m_1}$ with $i = \{2, \ldots, n\}$.
4. Comparing if the ratio computed is less than the minimum given by the user.

The centroid is computed for each ROI in the binary mask image. This point is the center point of the component and the center of the n circles used to study the variability of the intensity. In addition to computing the center, extracting the radius is mandatory for tracing the circles. In the case that the smallest rectangle is a square, we use circles for the study, by contrast, ovals will be used and we need two radii for the axis. Anyway, the process to obtain them is the same in both cases (the first case is a particular case of the second one). We are explaining for the case of an irregular region because since is more general.

The radii for the smallest oval are obtained from the bounds of the smallest rectangle which enclose to a region. The radii for the oval are the value of these bounds plus 6 μm. This number is choice like the best option analysed to increase the ovals and to be able to study of the intensity.

Afterwards, this operation is repeated $n - 1$ times to trace the other circles. For each circle the radii increase 3 μm or in other words, 6 μm are adding to the bounds of the last circle traced (see Figure 3.5 (a)). When all circles are defined, the mode for each one is computed over the second channel of the image (see Figure 3.5 (b)).



(a) Circles traced in the binary mask (b) Circles traced in the second channel

FIGURE 3.5: Example of the study of the variability of the intensity in a component or region. Scale bar: 20μm.

The next step consists of comparing the ratio of the circles to the smallest circle (see formula in the point 3 of the algorithm 6) with a value given by the user. If this ratio is lower than the percentage of variability considered as the maximum, it means that the component is the nucleus of a neuron. On the other hand, if the ratio is higher than this fixed value, the intensity around the component does not change and thus, it is difficult to determine whether the component is a neuron or not. This can happen when there is a cluster of neurons, noise or even artefacts in the image.

Finally, this method returns the number of neurons found and the regions where their nuclei are located, see Figure 3.6 — outputs (b) and (c) of the outline shown in algorithm 5. However, this is not an accurate process, and it is possible that neurons may be lost along the way. This is due to the fact that the neurons appear in clusters and sometimes is difficult to discriminate among them. Other usual aspects making it difficult to detect all the neuron are related to the contrast and brightest of the image, and to the role of the fixed parameters, which could distort the process.

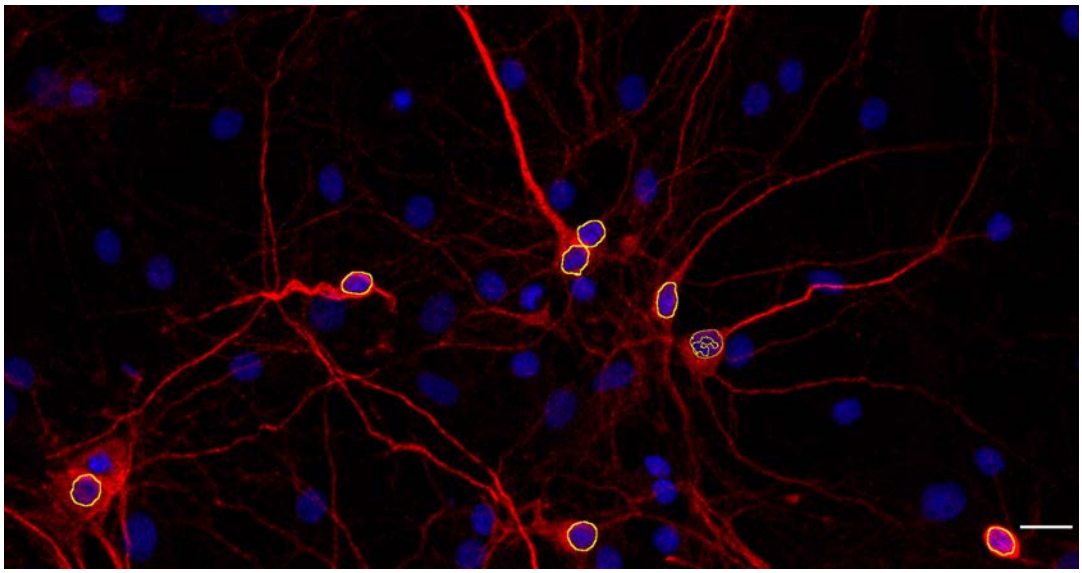


FIGURE 3.6: Example of neurons found with this method. Scale bar: 20μm.

This process has been implemented in Java and it is a plug-in of *ImageJ / Fiji* called *NucleusJ*. This programme allows the biologist to work in a semi-automatic way, and its workflow is provided in Figure 3.7.

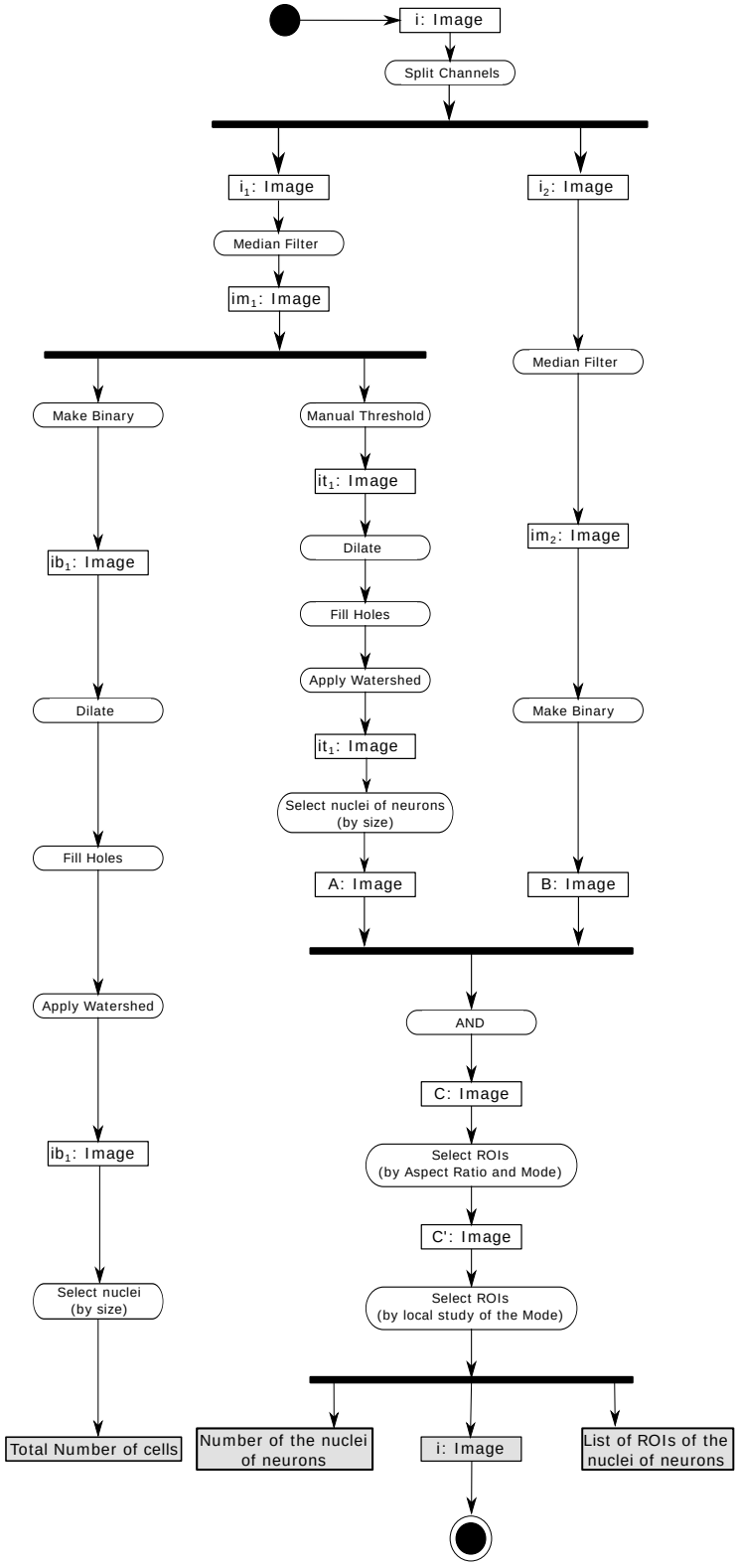


FIGURE 3.7: Workflow of NucleusJ.

Next, the interface of the plug-in and the results obtained are presented.

The first window of the plug-in shows the parameters which the user is required to configure, see Figure 3.8, given that the method employs different criteria.

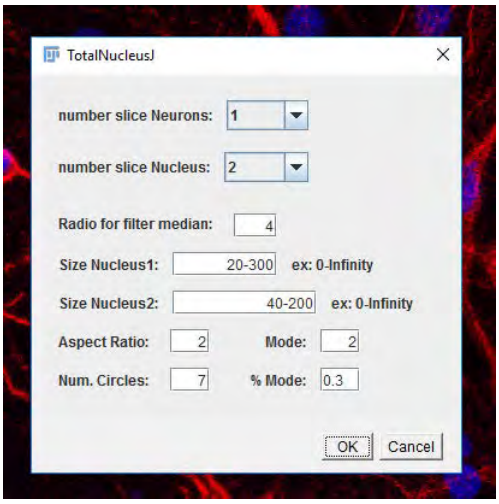


FIGURE 3.8: Interface of NucleusJ plug-in to configure the input parameters.

Firstly, the expert has to say which channel corresponds to the neurons and which one to the nuclei. Depending on the way the images are acquired, the first channel may reference to neurons or vice versa.

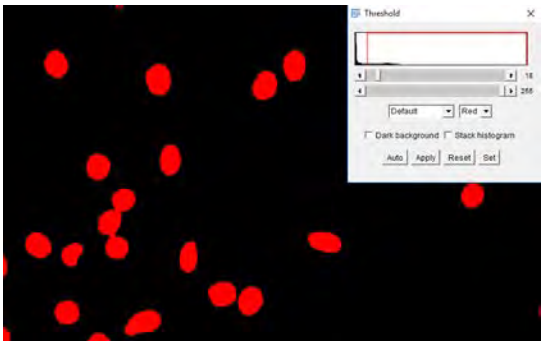
Likewise, the value for the radius of the median filter has to be introduced to be able to begin the preprocessing of the image and to remove noise. Other settings are related to the algorithms for discriminating the neuronal nuclei from other cells, such as the maximum and minimum sizes of all the nuclei, or only those of the neuronal nuclei. In addition, the values of the aspect ratio and the mode refer to the geometrical criteria to discard the nuclei which do not have a circular shape or a good intensity. The values for the number of circles and the percentages of the modes are related to the study of the evolution of the mode in successive circular areas and the removal of the nuclei which, for example, belong to a cluster.

Subsequently, the user has to choose a manual threshold to select the nuclei which are neurons, see Figure 3.9 (a), and the number of total nuclei of all the cells is shown in a table entitled Summary (Figure 3.9 (c)).

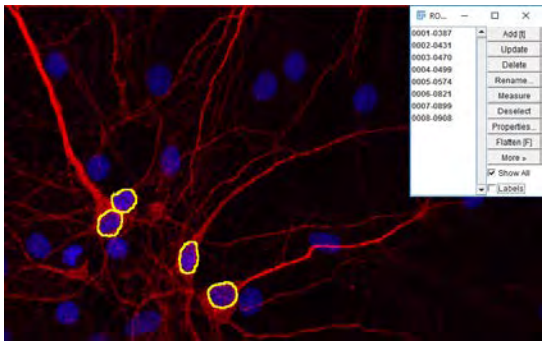
Finally, the plug-in returns the original image with the nuclei of the neurons marked, and they are listed in a window called *ROI Manager*, see Figure 3.9 (b). This window allows the user to select each nucleus individually and to check if the expert agrees or not. Should a nucleus not be found, the expert can add it using the functions of the *ROI Manager* window. By contrast, if the plug-in selects something which is not a nucleus, the expert can remove it from the list using the corresponding function. Furthermore, the plug-in returns the number of nuclei of neurons found in the window entitled Summary, see Figure 3.9 (d).

3.3 Experimental Results

This method, developed in the plug-in, called NucleusJ, is in phase of validation by expert users and there are still not enough results to be able to compare this method to the manual one.



(a) Patch of channel of nuclei with the threshold window (top left). The figures in red are the pixels which are between the limits of the threshold.



(b) Patch of image with the nuclei of neurons marked and listed in a ROI manager window (top left).

Summary

File Edit Font

Slice	Count	Total Area	Average Size	%Area
Total Nucleus	46	4619.215	100.418	5.346

(c) Summary of the number of total cells.

Summary

File Edit Font

Slice	Count	Total Area	Average Size	%Area
Total Nucleus	46	4619.215	100.418	5.346
Neurons Found	8	708.617	88.577	0.820

(d) Final summary of the number of the total cells and the neurons found.

FIGURE 3.9: Example of how the plug-in works and the results obtained.

3.4 Conclusions

The work presented in this chapter offers an approach to locate neurons in large images using intensity and geometrical criteria. The images analysed have been acquired as it is explained in Appendix B. However it is possible to use the plug-in for other kinds of images with similar features.

This work includes the option for the user checks the results. This method can be considered as a semi-automatic approach. However, this is a great advantage since this task requires less time than to count manually neurons. In addition, it is possible to compare the results among different experiments because the process is based on objective criteria.

3.5 Availability and Software Requirements

NucleusJ is an ImageJ plug-in that can be downloaded, together with its documentation, from the external reference *NucleusJ*. NucleusJ is open source and available for use under the GNU General Public License. This plug-in runs within both ImageJ and Fiji and has been tested on Windows, Macintosh and Linux machines.

Chapter 4

Neuron detection in stack images

4.1 Introduction

Dendritic neuronal trees and axonal growing are involved in *neuronal computation* and *brain functions*. Dendritic growing and axonal pathfinding are modified during brain development (Landmesser, 1994; Tessier-Lavigne and Goodman, 1996) neuronal plasticity process (Govindarajan, 2011) and neural disorders such autism (Calderón de Anda, 2012) or degenerative diseases such Alzheimer; for instance, in this neurodegenerative process brains are characterized by the presence of numerous atrophic neurons near the amyloidal plaques (Velez-Pardo, 2004; Goedert and Spillantini, 2006). Therefore, visualization and analysis of neuronal morphology and structure is of a critical importance to elucidate physiological changes.

The majority of reconstruction available software are manual or semiautomatic (Meijering, 2010), in which axonal and dendritic process are drawing by hand and consequently are not suitable for the analysis of large arrays of data sets. Subsequently the traces would transform into a geometrical format suitable for quantitative analysis and computational modeling. Algorithmic automation of neuronal tracing promises to increase the speed, accuracy, and reproducibility of morphological reconstructions. In this way, large scale analysis is feasible and would allow a high throughput strategy for the study of nervous system morphology in pharmacology or degenerative diseases (Donohue and Ascoli, 2011). The properties of optical microscopes images make it difficult to identify and automatically trace dendrites accurately, the presence of noise and biological contaminations, i.e. dendritic segments from neighbours neurons make difficult the digital encoding and reconstruction of a single neuronal structure.

In order to find a solution to this problem, two approaches based on a branch of mathematics called Algebraic Topology are explained in this chapter. The first is based on the *persistent homology theory* (Edelsbrunner, Letscher, and Zomorodian, 2002; Zomorodian, 2001) and the second, an improvement on the first, is based on the *theory of zigzag persistence* (Carlsson and DeSilva, 2010).

The following is an explanation of both approaches and their results.

4.2 A Persistent Homology Interpretation

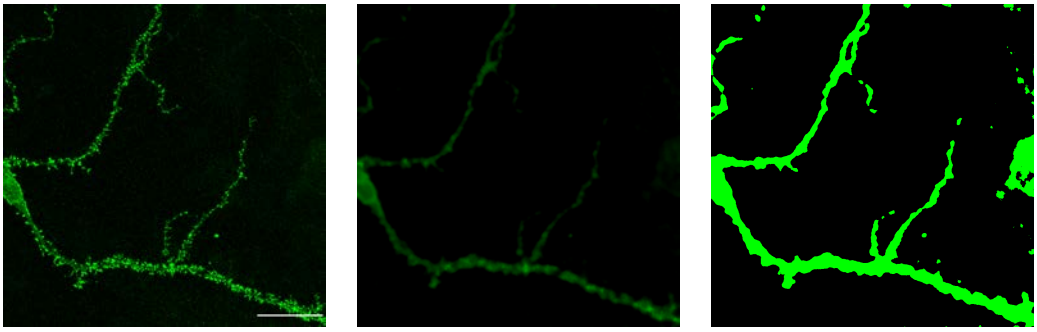
The work explained in this section has been presented as a poster in the conference within the field of bioimage processing: the first Congress of the Spanish Network of Advanced Optical Microscopy (REMOA), held in Barcelona, Spain, 2012, under the title: "Developing new tools to analyze neuronal morphology, spine and synaptic density"

4.2.1 Introduction

In this section we explain how, using geometric persistence models, it is possible to extract the dendrites and neuronal morphology from a series of inmunohistochemical images. The application developed is based on the idea that the neuron that we want to study *persists* in all the levels of the Z-stack. In this case, it is important to regard the neurons as cells with volume, that is, objects which are not on one, single plane. In fact, the image is a stack of images in which each slice depicts the cell a different height of the sample. The cultures were obtained according to the experiment explained in Appendix B, Section B.1.1 and the acquisition of the images following the criteria is detailed in the same appendix, Section B.2.

The method presented is not just theoretical but has also been implemented as a new plug-in, called *NeuronPersistentJ* (*NeuronPersistentJ*), for the systems ImageJ and Fiji.

4.2.2 Methodology



(a) (b) (c)

FIGURE 4.1: A 21 days in culture rat hippocampal neuron in culture, transfected with Actin-GFP. (a) Maximum intensity projection from a Z-stack. (b) Median filter of the same image. (c) Huang's thresholding method applied to the same image. Scale bar: 10μm.

This method to detect the neuronal structure from images, like the one of Figure 4.1 (a), can be split into two steps, which will be called respectively *salt-and-pepper removal* and *persistent*. In the former one, we reduce the salt-and-pepper noise, and in the latter one we dismiss the elements which appear in the image but which are not part of the structure of the main neuron (astrocytes, other neurons and so on).

In order to carry out the task of reducing the salt-and-pepper noise, we apply the following process both to the images of the stack and to the maximum intensity projection image. Firstly, we apply a *low-pass filter* (Castleman, 1996) to the images. In our case, the filter which fits better with our problem is the *median* one, since such a filter reduces speckle noise while retaining sharp edges. The filter radius is set to 10 pixels for the situation described in Appendix B, this value has been pragmatically determined and it is the only parameter of the whole method which must be changed if the acquisition procedure is modified. The result produced for the maximum intensity projection image of Figure 4.1 (a) is shown in Figure 4.1 (b).

Afterwards, we obtain binary images using *Huang's method* (Huang and Wang, 1995). This procedure automatically determines an adequate threshold value for the images. Applying that method to the image of Figure 4.1 (b), we obtain the result depicted in Figure 4.1 (c).

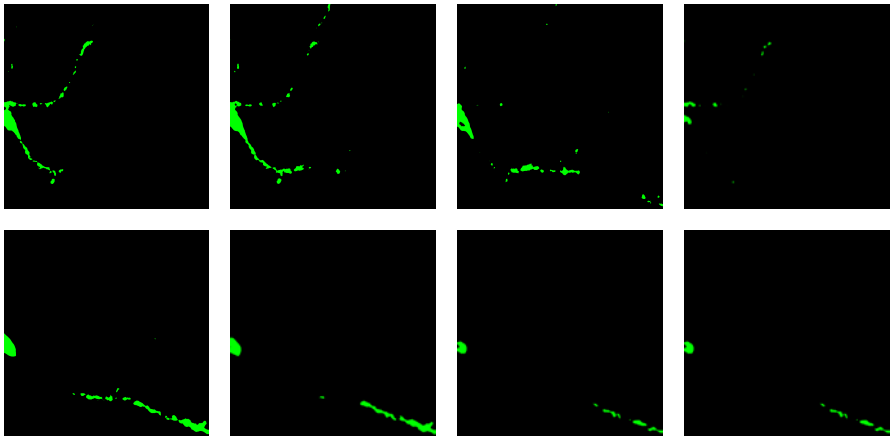


FIGURE 4.2: Processed median and Huang's filter of each Z-stack plane from Figure 4.1 neuron.

However, in the image of Figure 4.1 (c) we can see elements which does not belong to the main neuronal structure. Let us explain how we manage to remove those undesirable elements.

It is worth noting that part of the neuronal structure appears in every slide of a Z-stack. On the contrary, irrelevant elements just appear in some of the slides. This will be the key idea of our method.

More concretely, we proceed as follows. As it was explained previously, we apply the *salt-and-pepper removal* step to all the slides of the Z-stack, the result of that in our case study can be seen in Figure 4.2.

In the *persistent* step, we firstly construct a *filtration* of the binary image associated with the maximum projection image. A monochromatic image, $\mathcal{D}$, can be seen as a set of black pixels (which represent the foreground of the image), and a *filtration* of $\mathcal{D}$ is a nested subsequence of images $D^0 \subseteq D^1 \subseteq \ldots \subseteq D^m = \mathcal{D}$.

In order to construct a filtration of the binary image associated with the maximum projection image we proceed as follows. D^m is the maximum projection image. D^{m-1} consists of the connected components of D^m whose intersection with the first slide of the stack is non empty. D^{m-2} consists of the connected components of D^{m-1} whose intersection with the second slide of the stack is non empty, and so on. In general, D^{m-n} consists of the connected components of D^{m-n+1} whose intersection with the n-th slide of the stack is non empty. In this way, a filtration of the maximum projection image is obtained, see Figure 4.3.

As we know that the neuron appears in all the slides of the stack, the component D^0 of our filtration will be the structure of the neuron. As a final remark, we can notice that the construction of the filtration reaches a point where it is stable; that is, a level of the filtration D^i of the filtration such that D^j is equal to D^i for all $0 \leq j < i$. An example can be seen in the components D^0 to D^4 of Figure 4.3. This observation will be important in the next subsection.

Interpretation in terms of persistent homology

The persistent adjective of the second step of the method presented in the previous subsection comes from the nice interpretation which can be given in terms of the *persistent homology theory* (Edelsbrunner, Letscher, and Zomorodian, 2002), a branch of Algebraic Topology (Maunder, 1996). In a nutshell, persistent homology is a technique which allows one to study the *lifetimes* of *topological attributes*. For a detailed

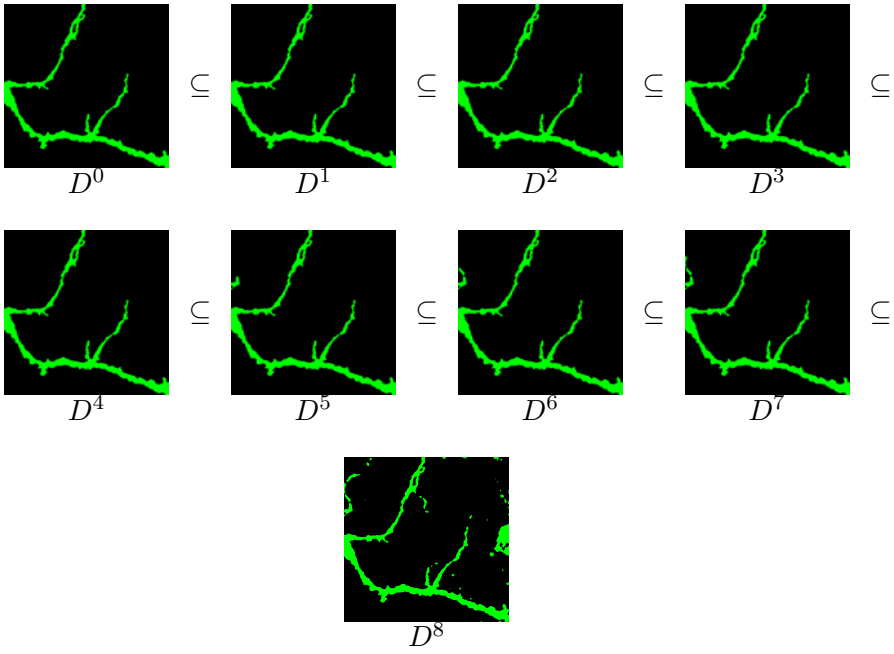


FIGURE 4.3: A series of pictures depicting the process of filtration from the Z-stack of Figure 4.1. From D^0 to D^8: Starting on D^0 and following to D^8 each level of the filtration represent the containing, $\subseteq$ information from the previous level. D^8 contains all the connected components from the image.

description of persistent homology see Edelsbrunner, Letscher, and Zomorodian, 2002; Zomorodian, 2001.

One of the most important notions in Algebraic Topology is the one of *homology groups*. Next, we remember a brief description of homology group, which was also explained in a previous chapter. However, we want to emphasise in this notion since it is key for the method which is explained in this chapter.

The homology group in dimension n of an object X, denoted by $H_n(X)$, is a set which consists of the n-dimensional holes of X, also called *n-dimensional homology classes* of X. To be more concrete, $H_0(X)$ measures the number of connected components of X, and the homology groups $H_n(X)$, with $n > 0$, measure higher dimensional connectedness. In the case of 2 dimensional monochromatic or binary images, the 0 and 1-dimensional homology classes are, respectively, the connected components and the holes of the image; there are not homology classes in higher dimensions.

Definition 4.2.1. An *(ordered abstract) simplicial complex* over V is a set of simplexes $\mathcal{K}$ over V such that it is closed by taking faces (subsets); that is to say:

$$\forall \alpha \in \mathcal{K},\ if\ \beta \subseteq \alpha \Rightarrow \beta \in \mathcal{K}$$

An example of a simplicial complex can be seen in Figure 4.4.

A *subcomplex* of $\mathcal{K}$ is a subset $\mathcal{L} \subseteq \mathcal{K}$ that is also a simplicial complex.

Definition 4.2.2. A *filtration* of a simplicial complex $\mathcal{K}$ is a nested subsequence of simplicial complexes

$$K^0 \subseteq K^1 \subseteq \ldots \subseteq K^m = \mathcal{K}$$

An example of a filtration can be seen in Figure 4.5.

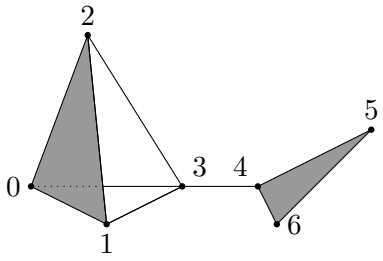


FIGURE 4.4: Butterfly Simplicial Complex.

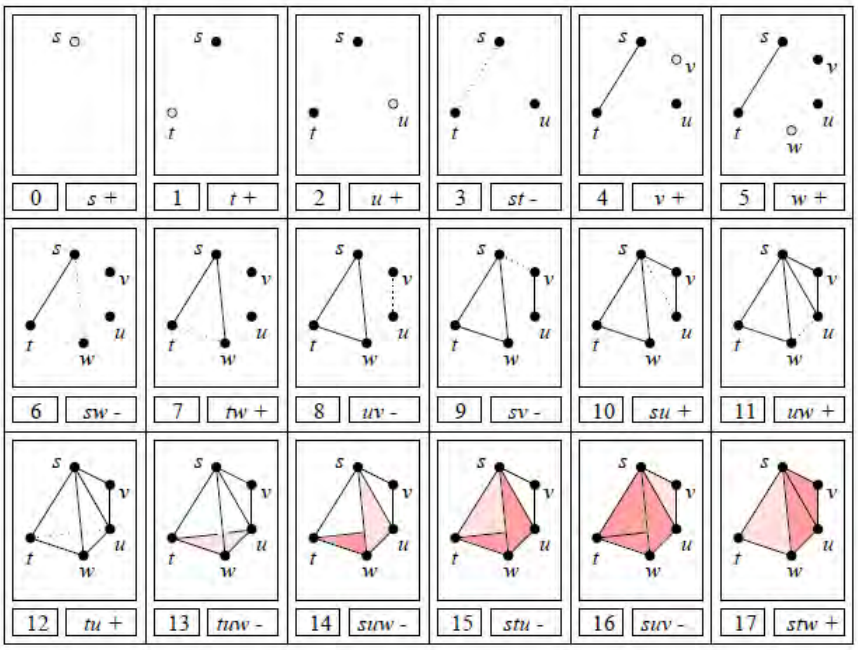


FIGURE 4.5: Example of a filtration.

Definition 4.2.3. Given a filtration $K^0 \subseteq K^1 \subseteq \ldots \subseteq K^m = \mathcal{K}$, a homology class α *is born* at K^i if it is not in the image of the map induced by the inclusion $K^{i-1} \subseteq K^i$. Furthermore, if α is born at K^i it *dies entering* K^j if the image of the map induced by $K^{i-1} \subseteq K^{j-1}$ does not contain the image of α but the image of the map induced by $K^{i-1} \subseteq K^j$ does. The *persistence* of α is $j - i$.

Now, it means, given a 2 dimensional monochromatic digital image $\mathcal{D}$ and a filtration $D^0 \subseteq D^1 \subseteq \ldots \subseteq D^m = \mathcal{D}$ of $\mathcal{D}$, a n-homology class α *is born* at D^i if it belongs to the set $H_n(D^i)$ but not to $H_n(D^{i-1})$. Furthermore, if α is born at D^i it *dies entering* D^j, with $i < j$, if it belongs to the set $H_n(D^{j-1})$ but not to $H_n(D^j)$. The *persistence* of α is $j - i$. We may represent the lifetime of a homology class as an interval, and we define a *barcode* to be the set of resulting intervals of a filtration.

In the case of the filtrations presented in the previous subsection, the outstanding barcode is the one of 0-dimensional homology classes. It is worth noting that the structure of the neuron lives from the beginning to the end of the filtration while external elements are short-lived.

For example, the barcode associated with the filtration of Figure 4.3 is the one depicted in Figure 4.6.

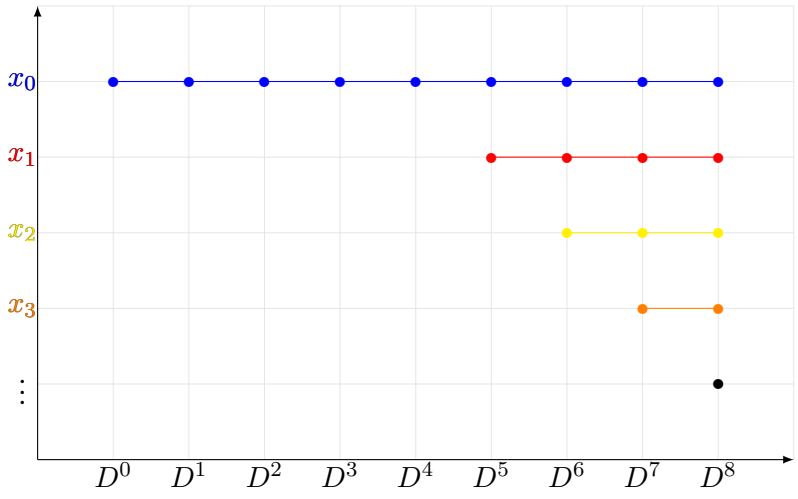


FIGURE 4.6: Barcode of the filtration of the Figure 4.3.

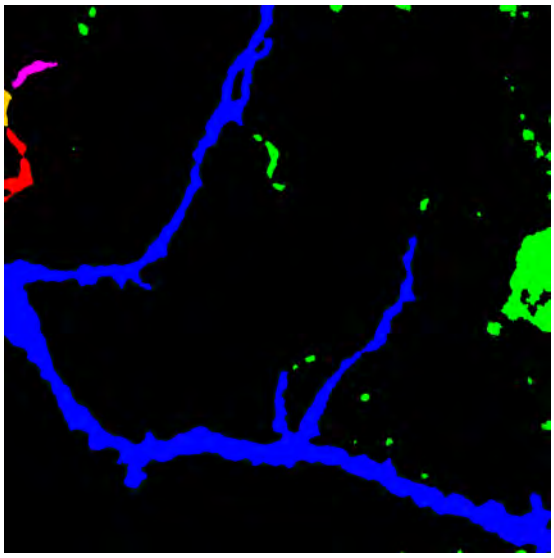


FIGURE 4.7: Summary picture of the connected components from Figure 4.1 projection. Colour code: green, components that last one plane. Orange, 2 planes. Yellow, three planes. Red, four planes and blue, components that are present in the eight planes.

Let us analyze the information which can be extracted from such barcode. There are several connected components whose life is reduced to the maximum projection image, the green connected components of Figure 4.7, and can be considered as noise. Notwithstanding that the components x_1, x_2 and x_3 (which are respectively the red, yellow and orange connected components of Figure 4.7) live a bit longer than green components; they are also short-lived; so, they cannot be part of the main structure of the neuron, it is likely that these components come from other biological elements. Eventually, we have the x_0 component, the blue connected component of Figure 4.7, which lives from the beginning to the end of the filtration; therefore, as it lives from the beginning to the end of the filtration, it represents the structure of the neuron.

We have devised an efficient algorithm to obtain the barcode of 0-dimensional homology classes associated with the images that we have presented in the previous subsection. This method takes advantage of both the way of building the filtration

and the stability of such a filtration. Firstly, we obtain the connected components of the level 0 of the filtration, D^0; this is a well-known process called *connected component labeling* which can be solved using different efficient algorithms (see, Rakhmadi, 2010; Lee, 2007). Such connected components are 0-dimensional homology classes which are born at D^0 and live until the end of the filtration, this fact comes from the filtration construction process. Now, we focus on the level 1 of the filtration, D^1. The filtration has a stability level; therefore, we consider two feasible cases. If D^0 is equal to D^1, we can pass to the next level of the filtration. Otherwise, we obtain the connected components which appear at D^1 but not at D^0, such components are 0-dimensional homology classes which are born at D^1 and live until the end of the filtration. In order to check if D^0 and D^1 are equal, we use the *MD6 Message-Digest Algorithm* (Rivest, 2008). Such algorithm is a cryptographic hash function which given an image returns a *unique* string; therefore, if the result produced for D^0 and D^1 is the same, we can claim that both images are equal. This procedure is faster than comparing pixel by pixel the images.

The above process is iterated for the rest of the levels of the filtration. In general, if we are in the level i of the filtration, there are two cases: if $D^{i-1} = D^i$ (this is tested with MD6 algorithm) then pass to level $i + 1$; otherwise the connected components which appear in D^i but not in D^{i-1} are the 0-dimensional homology classes which are born at D^i and live until the end of the filtration. In this way, we can obtain the barcode of 0-dimensional homology classes without explicitly computing persistent homology.

4.2.3 Experimental Results

The procedure to detect neural structure presented in the previous section has been implemented as a new plug-in for *ImageJ* called *NeuronPersistentJ*. Figure 4.8 illustrates the results which are obtained with NeuronPersistentJ using three different examples considering 10 as the radius (or the length) of the median filter. As can be seen in such examples both the noise and structures of neighbour neurons are removed from the final result.

We have validated our method and plug-in with a set of image stacks of real 3D neuron dendrites acquired using the procedure explained in Appendix B. In order to test the suitability of our software, we have compared a manual selection of the region of interest with the results obtained using NeuronPersistentJ. The manual selection was performed using the polygonal selection tool from ImageJ. In order to compare the two tracings (the manual and the one obtained using NeuronPersistentJ), we have considered both the *accuracy* and the *efficiency*.

The accuracy of the plug-in is measured with the three following relevant features:

(1) the number of branches obtained with the manual tracing compared with the number of branches detected with NeuronPersistentJ,

(2) the area of the region selected manually recognized with NeuronPersistentJ (that is, the intersection, $\cap$, of the region selected manually and the one obtained with NeuronPersistentJ),

(3) the area of the region detected by NeuronPersistentJ which does not appear in the manual tracing with respect to the area which does not contain the manual tracing (i.e. the area of the region recognized by NeuronPersistentJ minus, $\setminus$, the region manually selected with respect to the complement, $\_^C$, of the region manually selected).

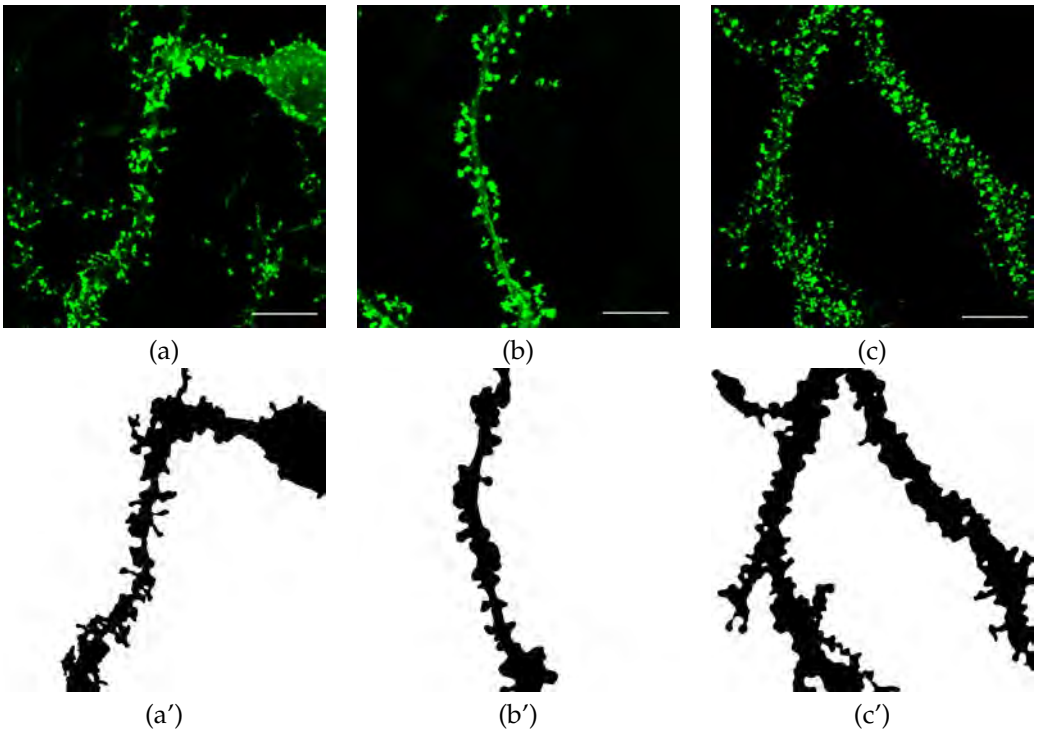


FIGURE 4.8: From a to c: Three examples of dendritic fragments of hippocampal neurons in culture transfected with Actin-GFP. From a′ to c′: Structures obtained with the NeuronPersistentJ application with a median filter 10. Scale bar: 10μm.

To compute the percentages associated with these features, we use the following formulas.

$$(1) = \frac{\texttt{Number of branches of NeuronPersistentJ tracing}}{\texttt{Number of branches of manual tracing}} \times 100$$

$$(2) = \frac{\texttt{Area (NeuronPersistentJ tracing} \cap \texttt{Manual tracing)}}{\texttt{Area (Manual tracing)}} \times 100$$

$$(3) = \frac{\texttt{Area (NeuronPersistentJ tracing} \setminus \texttt{Manual tracing)}}{\texttt{Area((Manual tracing)}^{C}\texttt{)}} \times 100$$

It is worth noting that the higher the values for both (1) and (2) the better since this means that we are close to detect all the branches and the whole region of interest. On the contrary, the value of (3) should be small in order to avoid the inclusion of regions which are not relevant.

The experimental results that we have obtained with our data set, considering different radii for the median filter, using NeuronPersistentJ are shown in Table 4.1. As we are seeking an equilibrium between the values of the features (2) and (3), the best value for the radius of the filter is 10.

Let us consider the efficiency of the plug-in. As we have explained previously the manual method to select the region of interest consists in using the polygonal tool of ImageJ in the maximum projection image. This manual procedure takes approximately three minutes per neuron. On the contrary, the results are obtained in half the time using NeuronPersistentJ. This is quite relevant since in order to test the effect of some experimental treatments over neurons we do not study just one

radius of filter \ percentage	(1)	(2)	(3)
5	96.2%	78.06%	4.43%
10	98.2%	93.3%	4.19%
15	98.7%	94.9%	6.25%

TABLE 4.1: Percentages of accuracy of NeuronPersistentJ. (1) Percentage of components detects with NeuronPersistentJ versus manual tracking. (2) Percentage of area detected with NeuronPersistentJ versus manual tracking. (3) Percentage of area draw by NeuronPersintentJ not present in the manual tracking. Percentage is the mean value from eight images.

neuron but batteries of neurons. Therefore, the use of NeuronPersistentJ means a decreasing of the time invested to detect the neuronal structure.

In view of the results, our method can be considered as an approach, both from the accuracy and efficiency point of view, to automatically trace neuronal morphology from Z-stacks.

4.2.4 Discussion

The geometric persistence method reported here has been used to develop a plug-in to extract the neuronal structure. In particular, the contour of the neuron is segmented and therefore the region of interest is recognized.

The application is based on the fact that the neuronal structure is present, "or persists", in all the levels of Z-stack images. This plug-in, automatically, generates a digital 2D representation of a three-dimensional neuron in the final picture. After the extraction process, structures from neighbour neurons, background noise and unspecific staining are eliminated from the final image.

The plug-in works analyzing every optical plane and comparing the maximum intensity projection with the slides of the Z-stack. After a preprocessing step where the salt-and-paper noise is removed, using the median filter, from the slides of the Z-stack and the maximum intensity projection, the plug-in removes from the maximum intensity projection the elements which does not live enough (i.e. the elements which do not appear in all the slices of the stack) obtaining as result the structure of the neuron.

Transfected neurons were used to test this method. The protocol to obtain these kind of cell and its acquisition is described in Appendix B.

In order to validate this method we have compared a manual surface tracking employing the polygonal selection from ImageJ. The validation used as a control the total area delimiting by a manual tracing and compared it with the area delimited by NeuronPersistentJ. It is worth noting that the result of the comparison depends on the value of the radius of the low pass filter selected; large values will led to a broad structure, on the contrary small values will produce sharp and more defined images. Employing this validation method our results indicate that NeuronPersistentJ is suitable to carry out the recognition of the neuron structure.

The number of manipulations during the reconstruction process is always a drawback for a fully automatic process. NeuronPersitentJ requires a set of binary images; thus, selection of the radius of a low-pass filter value that retains the maximal information from the Z-stack pictures is the only parameter determined by the

experimenter and clearly it depends on the images conditions. NeuronPersitentJ, as it is usual in this kind of technique, works better with highly contrasted images, such the ones obtained by inmunofluorescence.

The topological approach employed here and the use of binary pictures is independent from the nature of the picture. However, as mentioned, highly contrasted pictures and a clear and continuous staining are key elements for a fine reconstruction.

Skeletonization of neuronal structure has been a popular solution to neuronal reconstruction and structure extraction (Meijering, 2010), and this method could be used a basic towards the automatic detection and classification of different features of neuronal structure, such spine density or dendritic arborization.

4.2.5 Availability and Software Requirements

NeuronPersistentJ is an ImageJ plug-in that can be downloaded, together with its documentation, from the external reference *NeuronPersistentJ*. NeuronPersistentJ is open source and available for use under the GNU General Public License. This plug-in runs within both ImageJ and Fiji and has been tested on Windows, Macintosh and Linux machines.

4.3 Zigzag Persistence Theory

The work explained in this section has been presented and published in several conferences within the field of Computational Algebra such as XIV Encuentro de Álgebra Computacional y Aplicaciones (EACA), held in Barcelona, Spain, 2014. This is also the case within the field of bioimage processing such as the first NEUBIAS 2020 Conference - Network of European Bioimage Analysts, held in Lisbon, Portugal, 2017; In addition, this work was published in the Pattern Recognition Letters journal under the title: "Zigzag persistent homology for processing neuronal images", see Mata, Morales, Romero, and Rubio, 2015.

4.3.1 Introduction

The goal of this section is to improve the approach seen in the previous section, to isolate the object of interest in a Z-stack of images. Up till now, we have described the plug-in used to this end in Section 4.2, which was called NeuronPersistentJ. It processes each layer of the stack in a cumulative manner and, doing a *maximal projection* at each step, allows us to find a covering of the neuron in the noisy images, discarding some spurious elements in the picture (such as other cellular bodies or dendrites coming from other neurons). Nevertheless, this algorithm is unable, by its very nature, to distinguish between two dendrites that are crossing in space (since they would necessarily intersect in any Z-projection).

The NeuronPersistentJ program proceeds by examining each projection, and looking for the number of connected components (the 0-homology group of the digital images), until it stabilizes. This method admits an elegant interpretation in terms of persistent homology, although this requires introducing a filtration which is unnatural from the persistence perspective. Indeed, the layered structure of the stack of images does not define a filtration.

Building on this weakness of the previous method, we take advantage now of the *zigzag* persistence ideas, providing a new plug-in which maintains the performance of NeuronPersistentJ and, in addition, has been capable of distinguishing among

dendrites crossing in space, in actual images acquired from biological experiments. This kind of object tracking has also been dealt with by other authors (Mori and Zijl, 2002; Weeden and Wang, 2008; Dellani and Glaser, 2007), in a different context from ours (that of magnetic resonance imaging).

This plug-in, called *NeuronzigzagJ*, is reported in this section.

4.3.2 Methodology

The theory of zigzag persistence (Carlsson and DeSilva, 2010) is defined for diagrams of topological spaces of the form:

$$X_1 \leftrightarrow X_2 \leftrightarrow \cdots \leftrightarrow X_m$$

where each arrow can point either left or right.

Considering the homology groups of each topological space and the induced morphisms, for each $n \in \mathbb{N}$ one obtains a sequence of vector spaces and linear maps:

$$V_1 \equiv H_n(X_1) \leftrightarrow V_2 \equiv H_n(X_2) \leftrightarrow \cdots \leftrightarrow V_m \equiv H_n(X_m)$$

which is called a *zigzag module*.

In Carlsson and DeSilva, 2010, it is shown that zigzag modules can be decomposed as a direct sum of submodules W^i of the form

$$0 \leftrightarrow \cdots \leftrightarrow 0 \leftrightarrow W^i_{a_i} = \mathbb{F} \leftrightarrow \cdots \leftrightarrow W^i_{b_i} = \mathbb{F} \leftrightarrow 0 \leftrightarrow \cdots \leftrightarrow 0$$

for some $1 \leq a_i \leq b_i \leq m$, where $\mathbb{F}$ is the base field and all arrows are the identity map. In this way, zigzag modules can be classified up to isomorphism by a multi-set of intervals $\{[a_i, b_i]\}$ with $1 \leq a_i \leq b_i \leq m$; drawing all of them as a disjoint union of intervals leads to the graphical representation of zigzag modules by means of *barcode diagrams* (see Carlsson and DeSilva, 2010), which can be used to get a visual description of the evolution of the different homology classes along the different spaces.

One of possible applications of zigzag persistence consists in studying the relations of the homology classes of different subspaces $X_1, \ldots, X_m$ of a topological space X. To this aim, the following sequence is considered:

$$X_1 \hookrightarrow X_1 \cup X_2 \hookleftarrow X_2 \hookrightarrow X_2 \cup X_3 \hookleftarrow \cdots \hookrightarrow X_{m-1} \cup X_m \hookleftarrow X_m$$

The associated zigzag module determines the continuity of homology classes between the terms in the above sequence, which allows one to know whether the subspaces X_i contain the same homology classes in X or different ones.

Zigzag persistence provides an extension of persistent homology (explained in the previous approach, Section 4.2), which is defined for diagrams of topological spaces with inclusions:

$$X_1 \hookrightarrow X_2 \hookrightarrow \cdots \hookrightarrow X_m$$

In other words, in persistent homology theory a filtration of a global space $X = X_m$ is needed, whereas zigzag modules can be defined in a more general situation.

Zigzag persistence for 3D digital images

Let us consider a binary 3D digital image in the 3D integer grid; in other words, we consider finitely many voxels having two possible values, usually black and white

(in this case, white pixels correspond with the *figure* in the image and black pixels with the *background*). By cutting at heights $z = j + 1/2$ (with j an integer) we get horizontal slices which define a series of binary 2D images, denoted S_i, for $1 \leq i \leq m$. The maximal projection of such a sequence of 2D-images consists, in general (in the case of greyscale images), of projecting in a unique plane the pixels with maximum intensities on every vertical view; for binary images the maximal projection is simply given by the union of the white pixels of all slices $S_1, \ldots, S_m$.

In order to compute the different connected components of a binary 3D image, it is well-known that graph theory can be used, for instance, in Shapiro and Stockman, 2002 there are some algorithms for 2D-images which can be generalized to arbitrary dimension. If we are interested in how 3D connected components are mapped on the maximal projection, we could simply project each component, since it is given by a set of voxels. Let us observe that when in $3D$ images 26-adjacency is considered, for $2D$ images 8-adjacency is obtained (see, Rosenfeld, 1974).

As an alternative, and based on the ideas of zigzag persistence presented in this subsection, the following algorithm also allows us to mark over the maximal projection the fragments coming from different connected components of the 3D image (they could be indicated graphically using different colours; see Figure 4.12).

Algorithm 7.
`Input`: Binary stack of images
`Output`: List whose elements correspond to the connected components of each slice over the maximal projection

1. Compute separately (by means of some of the previously cited algorithms, for instance, from Shapiro and Stockman, 2002) the connected components of all slices S_i and of the unions $S_i \cup S_{i+1}$ for all i.

2. Consider each connected component C_1^j of the first slice S_1 and store it as the first element of a new list l_1^j; we want to determine the projection over the maximal projection of the 3D connected component corresponding to C_1^j.

3. Obtain the connected component $C_{1\cup 2}^k$ of the union $S_1 \cup S_2$ corresponding to C_1^j (that is, we take $C_{1\cup 2}^k$ such that $C_1^j \subseteq C_{1\cup 2}^k$).

4. Compute the connected components $C_2^{i_1}, \ldots, C_2^{i_r}$ of the slice S_2 which have a non-empty intersection with $C_{1\cup 2}^k$. The result corresponds to the pixels of S_2 which in the $3D$ object are in the same connected component as C_1^j. The components $C_2^{i_1}, \ldots, C_2^{i_r}$ are added to the list l_1^j.

5. Repeat steps 3 and 4 for each component $C_2^{i_s}$, considering now the following slice S_3 and the union $S_2 \cup S_3$. We continue the process for every slice.

6. Once all the slices have been studied, the desired projection of C_1^j over the maximal projection of the image is given by the union of all elements in the list l_1^j. It is a set of pixels that we denote by PC_1^j, and that is stored in the final output list l.

7. Repeat steps 2-6 for all connected components of the first slice S_1.

8. Repeat steps 2-7 for all connected components of the other slices $S_2, \ldots, S_m$ which have not been previously considered (that is, we consider only connected components which are not present in any list l_i^j constructed before).

9. The output of the algorithm is the list l, whose elements are sets of pixels corresponding to the projection over the maximal projection of the different connected components of the $3D$ image.

The algorithm allows us, in particular, to distinguish in the maximal projection information coming from different connected components in the 3D image which are *crossing* in space (that is to say, connected components that do not intersect, but whose projections do).

The ideas of this algorithm can be formalised by means of zigzag persistence as follows. Given a binary 2D-image, we consider the simplicial complex obtained by triangulating each pixel by means of a diagonal edge (see Kozlov, 2008; a 2D-cubical complex could also be used); this produces 8-adjacency, as required. Then, for each slice S_i, let us consider the associated simplicial complex, denoted by X_i. It is a topological space, and the number of connected components of the image S_i can be determined as the rank of the homology groups in degree 0 of X_i. Similarly, we can consider the simplicial complex associated with the union $S_i \cup S_{i+1}$, that we denote by $X_{i\cup i+1}$, that is in fact equal to the union of the simplicial complexes X_i and X_{i+1}. The 0-homology of $X_{i\cup i+1} = X_i \cup X_{i+1}$ is related to the connected components of the image $S_i \cup S_{i+1}$. Then, we have the following diagram:

$$X_1 \hookrightarrow X_1 \cup X_2 \hookleftarrow X_2 \hookrightarrow X_2 \cup X_3 \hookleftarrow \cdots \hookrightarrow X_{m-1} \cup X_m \hookleftarrow X_m$$

and the corresponding zigzag module for degree 0:

$$H_0(X_1) \to H_0(X_1 \cup X_2) \leftarrow H_0(X_2) \to H_0(X_2 \cup X_3) \leftarrow \\ \cdots \to H_0(X_{m-1} \cup X_m) \leftarrow H_0(X_m)$$

Thanks to the ideas explained in Algorithm 7, it is not difficult to observe that the different intervals of this zigzag module correspond to the connected components of the $3D$ image. If homology with generators is computed at different steps, then it is possible to determine in the simplicial complex the $2D$ connected component associated with each generator in S_i or $S_i \cup S_{i+1}$. Subsequently, the looked-for projection of each $3D$ connected component over the maximal projection of the image can be determined as the union of the $2D$ connected components in each step S_i of the corresponding zigzag interval.

In this way, the computation of zigzag homology of the previous module allows us to determine the different connected components in the $3D$ image. More concretely, each connected component appearing in the initial $3D$ body is described in the barcode diagram by means of an interval starting at X_1 or at $X_i \cup X_{i+1}$ for some i, and dying at X_m or at $X_j \cup X_{j+1}$ for some j. We can also observe that intervals starting at some X_i (with $i > 1$) correspond to different connected components in S_i which merge in $S_{i-1} \cup S_i$. Similarly, intervals dying at some X_j (with $j < m$) represent components which merge with another one in $S_j \cup S_{j+1}$.

Zigzag persistence for Z-stacks of images

In a more realistic situation, the microscope provides only a stack of several $2D$ images $I_1, \ldots, I_m$ corresponding to different levels of the Z-axis, and therefore, the complete 3D body is not available. In this case, we can binarize each slice I_i to get a binary image S_i, and then determine the maximal projection of $S_1, \ldots, S_m$ which

is given by the union of white pixels of all of them — the white pixels are the foreground.

In this case, information about the $3D$ volume is not complete and, therefore, it is not possible to compute directly the connected components of the body (and their projection over the maximal projection). However, we can apply Algorithm 7 to the binary $2D$ images $S_1, \ldots, S_m$ as in the "ideal" situation of the previous subsection. More formally, we can also determine the barcode for the zigzag module

$$H_0(X_1) \to H_0(X_1 \cup X_2) \leftarrow H_0(X_2) \to H_0(X_2 \cup X_3) \leftarrow \\ \cdots \to H_0(X_{m-1} \cup X_m) \leftarrow H_0(X_m)$$

where each X_i is the simplicial complex associated to the binary image S_i.

The result of Algorithm 7 is again a list whose elements are sets of pixels, which, in this case, can be considered as an *approximation* of the projection over the maximal projection of the different connected components of the $3D$ object. The associated barcode diagram expresses again the continuity of 0-homology classes between the different slices and represents graphically the results of Algorithm 7. As before, connected components appearing in the initial $3D$ body correspond in the barcode diagram to intervals starting at X_1 or at $X_i \cup X_{i+1}$ for some i and dying at X_m or at $X_j \cup X_{j+1}$ for some j. Intervals starting at some X_i (with $i > 1$) correspond to different connected components in S_i which merge in $S_{i-1} \cup S_i$. Similarly, intervals dying at some X_j (with $j < m$) represent components which merge with another one in $S_j \cup S_{j+1}$.

It is worth emphasizing in this point that Algorithm 7 returns an *approximation* of the desired projection of the different connected components of the $3D$ object, but we cannot guarantee that the results are totally correct. The results depend on several factors related to the quality of the Z-stack of images, factors as resolution, noise or the real distance between slices (it could imply that some information is lost or confused in the space between two consecutive slices). In general, Algorithm 7 returns many connected components which are produced due to noise in the image and do not correspond to the real situation. To avoid this problem, we considered that *real* connected components correspond to zigzag intervals which persist *many* slices, and those components which were present only in a few steps, were discarded. Let us remark that, although in the ideal case of a 3D image, one connected component could completely lie on a Z-plane (being discarded as noise), this could not occur in the real situation, because a dendrite tends to be a quite warped object and the way of acquiring images is precise enough to reflect this fact (see details in the Appendix B).

In addition to this *homological* criterion, we considered another one based on metric information (more concretely, related to the radius of the different connected components), because in some uncommon occasions isolated points can also persist. Moreover, depending on the type of the image to be studied, the union of the binary slices S_i and S_{i+1} is replaced by the binarization of the maximal projection of the initial images I_i and I_{i+1}.

Description of the plug-in

Once images have been acquired, zigzag persistence can be used to detect the different dendrites appearing in an image and to discard non-relevant elements considered as noise. This approach is based on the following three hypotheses which are supported by experimental evidences (these evidences are "a priori" ones, led by

the expert knowledge of biologists; they are different from the conclusions extracted from our *computer* experiments explained later on):

- A neuron is a *continuous* (connected) $3D$ body.
- The neuron which is the object of interest in the image appears in most layers of the stack.
- It is not frequent that two different dendrites intersect over the same slice (in that case, our algorithm would be unable to differentiate them).

With these assumptions, this aim is to apply Algorithm 7 considering that the *long* intervals in the zigzag barcode will correspond to significant dendrite fragments. Since we consider that starting from the upper or lower slices could be a bad strategy (in general, relevant neuronal information will be concentrated on some intermediate slices), the plug-in works in such a way that an initial slice is chosen by the user, and then Algorithm 7 is deployed.

Before defining the outputs of our plug-in, it is necessary to explain how we transform the *actual* (greyscale) images in binary 2D-images where our homological algorithm can be safely applied. In order to *binarize* a greyscale image it is necessary to determine a *threshold* by means of some algorithm, and to transform the pixels with intensity greater than the threshold into white pixels, the rest being converted into black pixels. Notice that on greyscale images the binarization of a maximal projection of two images is, in general, different from the union (maximal projection) of the binarization of the two initial images (the reason being that the threshold is computed on the whole image).

The algorithms employed to binarize and to compute the zigzag information are different according to the two kinds of images which were used to test this method. Both types of images are explained in Appendix B; they are images with GFP-transfected neurons, and images with DiI-stained neurons.

Due to the fact that Actin-GFP staining produces poorly contrasted images, they were preprocess by means of the median filter and then the operation $S_i \cup S_{i+1}$ on slices was replaced by the binarization of the *maximal projection* of the initial greyscale images I_i and I_{i+1}. In order to binarize the images, the *Huang's* threshold algorithm (Huang and Wang, 1995) was used. This strategy allows us to avoid many of the distortions produced during the acquisition process, getting a better representation of the topological information. On the contrary, on DiI images, it was enough to apply a Gauss filter (used to homogenize the noisy background), then the "*Default*" method of auto thresholding available in Fiji/ImageJ, which is a variation of the *IsoData* algorithm (Ridler and Calvard, 1978). Finally, taking as operation among slices the standard union of the binarized images. All these decisions have been validated through experimental evidences (see subsection 4.3.3).

In order to get homological information about a single image, we used an algorithm to determine local maximal intensities in images. To this end, we used the same algorithm which was explained in the previous chapters and it is known as *FindMaxima*. As it was mentioned, this algorithm computes the pixels with local maximal intensity in a greyscale image, calculating an area of *influence* for each maximum. In the case of binary images, this amounts to determine the connected components (with respect to the 8-adjacency) in the image. *FindMaxima* produces three kind of results: the number of connected components (in other words, it determines the H_0 of the simplicial complex associated with the image), a point in each component (homology with generators) and, in addition, a drawing of each connected

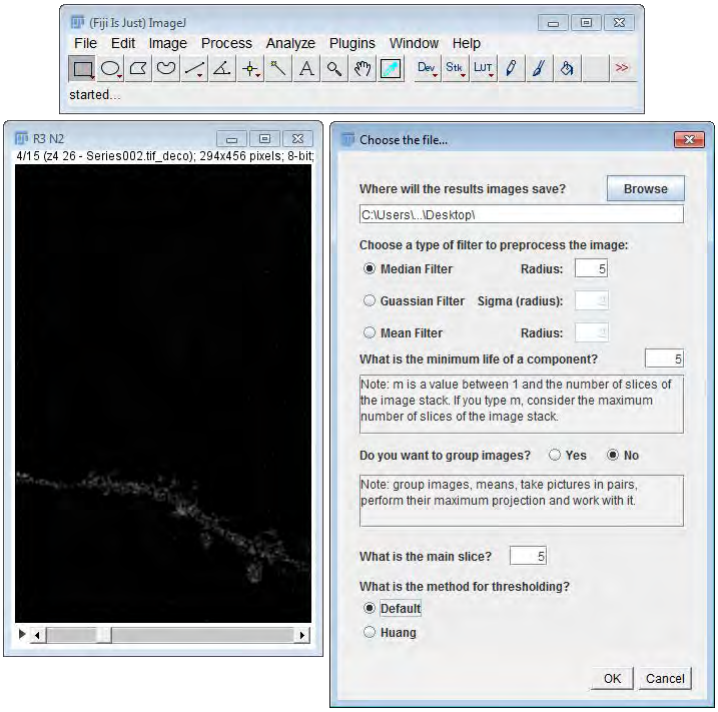


FIGURE 4.9: Interface of NeuronzigzagJ plug-in to configure the input parameters.

component (producing, in our case, a segmentation of dendrite fragments). Thus, by applying *FindMaxima*, all the functionality needed to implement Algorithm 7 obtained. Let us observe that, since *FindMaxima* always determines the same generator for the same component, it provides an efficient management of the component lists involved in Algorithm 7.

After explaining some of the internal processing in our plug-in, let us briefly describe the user interface. The user sets a number of parameters before starting the execution. The slice where the processing will start from, and also the minimum *life* of each component are fixed. The second value referes to the length of the barcode to be considered with biological meaning. Furthermore, the visual interface allows the user to choose freely among different filters and thresholding methods (even if, for the two kind of images considered in our experimental study, some of these parameters are strongly correlated), foreseeing that other kinds of images could need other kinds of preprocessing (see Figure 4.9).

We explain now the outputs of the plug-in by means of an example. The plug-in is applied on a selected part of the image where the object of interest is concentrated. For instance, in Figure 4.11 we have selected the part of Figure 4.10 which is inside the white rectangle, because there is a possible crossing (a situation difficult to analyze by automated methods; in particular it would be out of reaching for our previous plug-in NeuronPersistentJ).

The plug-in studies all connected components contained in the different slices, and studies their evolution throughout the Z-stack by means of the zigzag module. Then, NeuronzigzagJ calculates the main structure and is able to check automatically whether two dendrites that are linked in the maximal projection of the stack are really the same, or are overlapped on the plane but there is no intersection between them in the space.

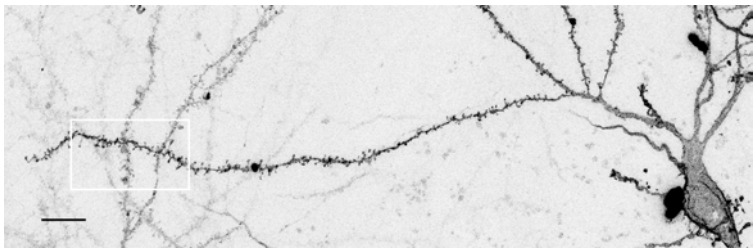


FIGURE 4.10: Basal dendritic segment of a CA1 mouse hippocampal neuron stained with a DiI biolistic protocol. Scale bar: 10μm. Image inverted.

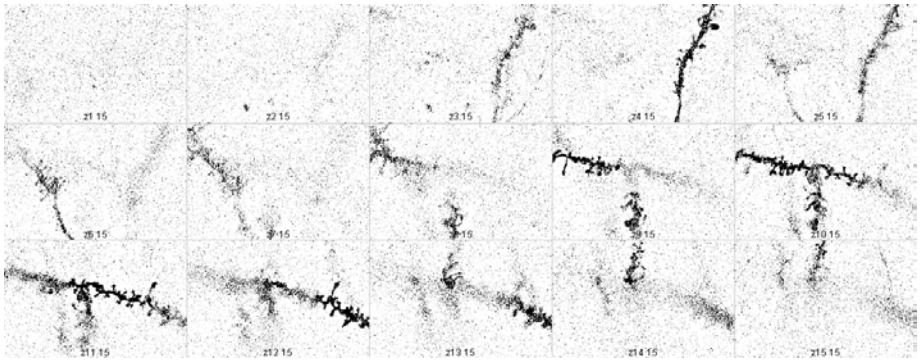
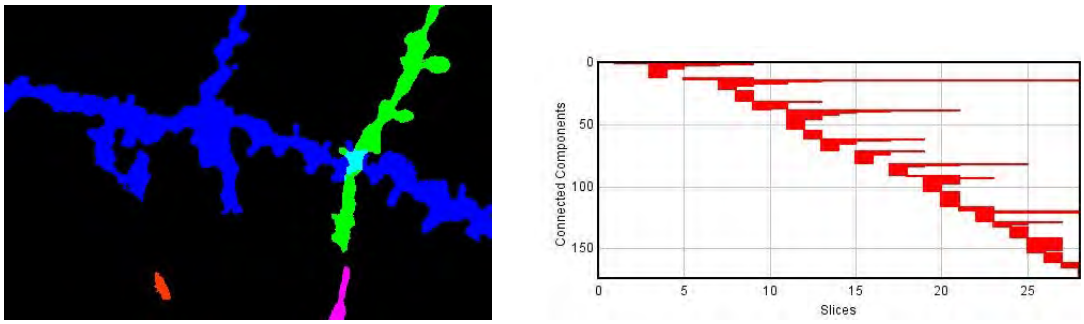


FIGURE 4.11: A piece of Figure 4.10. This is a relevant part for our study because we can see there two crossing dendrites. Image inverted.



(a) Summary picture of the main connected components.

(b) Barcode of all connected components.

FIGURE 4.12: Graphical results from the plug-in.

The final outputs are two images, see Figure 4.12. The first image (Figure 4.12 (a)) depicts the main connected components of the neuron structure, with different colours. The plug-in considers that the main connected components are those that live longer. In this example, there are 4 connected components which live in 5 or more slices (the number 5 corresponds to the "life" parameter fixed by the user). This Z-stack has 15 slices and these components have the longest life. We can see that they represent the main dendritic structure in this image. The second image (Figure 4.12 (b)) is a barcode summarizing the life of all connected components which are in the Z-stack. The barcode shows an abridged version of all the homological information; in particular, the four longest dendrite fragments are clearly visualized. Let us stress that even if the stack has 15 slices, the number of columns in the barcode is 29 (twice 15 minus 1), because the compound images $S_i \cup S_{i+1}$ (got by union or by maximal

projection) are also considered there; concretely, even columns correspond to actual slices, and odd ones to compound images.

4.3.3 Experimental Results

In order to validate our program and assumptions, we have undertaken two experimental studies. In the first experiment, the aim was to investigate the role of the *maximal projection* ($\sqcup$) and the *union*($\cup$) strategies (with the corresponding filter and thresholding methods). In the second experiment, a more systematic study was carried out to estimate the accuracy of our program with respect to the observations of some human experts.

In the first study, we applied this plug-in starting from 1146 slices (coming from 12 Actin-GFP stacks of images, and 11 DiI stacks). The gathered-data were the number of slices and crossings with respect of each kind of strategy, together with a control by a human who inspected the relevant number of connected components (i.e., significant fragments of dendrites) in each stack. From the experimental data, we obtained the confirmation of some of our assumptions about the role of the $\sqcup$ and $\cup$ operations, but also some unexpected insights about the importance of the starting slice for the process, and the relevance of the barcode information. These experiments suggest a strategy to work with this plug-in:

1. First, focus on the barcode diagram, looking for long bars and also considering when two longs bars are not intersecting in the same column (in this second case, it implies that no starting slice could produce all the relevant homological information).

2. Second, produce the corresponding graphical outputs starting from one or from several slices, determined by the barcode examination.

In the second experimental study, 60 images were selected (30 GFP images and 30 Dil images). They were randomly divided into three blocks of 30 images, and each block was sent to a researcher to make a manual analysis. In that way, some images were analyzed by more than one person, trying to measure the influence of subjective behavior in the study. Two of the observers were biologists, and the third one a computer scientist.

For each image, the human observer annotated the number of crossings, the number of connected components, and also the exact location of each crossing and each dendrite.

The result of the crossover study was clear: there was not any relevant discrepancy among the interpretations of the three observers, when looking at the same image.

This increases the reliability of the aggregated results obtained in the final report.

The success rate of the plug-in was remarkable with respect to GFP images (90% of hits) and reasonable with respect to Dil images (77.6% of hits). Here *hit* means that the plug-in found the same results than the human observers, up to some small ambiguity present in the image. If we consider *full hits*, that is to say, exact equality with respect to the four measured features (number of crossings, number of dendrites, and location of both), the figures are 86% for GFP images, and 66.6% for Dil images. The poorer performance for Dil images is explained because each image has approximately 45 slices (theses are pictures from actual slices of brain tissue where neurons growth in three-dimensional space); then, the human eye does not perceive all the intricacies contained in the image; on the contrary, GFP pictures came from

neurons in culture that tend to growth in a more limited space (growing in this case is limited to the surface of the culture chamber).

A complete report on the validation performed is in Appendix D together with some summary tables.

4.3.4 Conclusions

This section has presented the ideas underlying a Fiji/ImageJ plug-in for analyzing neuronal structures in a stack of images. The algorithm is based on *zigzag* persistence, a tool from Computational Topology. This approach is an improvement on the previous one and offers an objective method on finding the structure of cells in Z-stack images.

4.3.5 Availability and Software Requirements

NeuronzigzagJ is an ImageJ plug-in that can be downloaded, together with its documentation, from the external reference *NeuronzigzagJ*. NeuronzigzagJ is open source and available for use under the GNU General Public License. This plug-in runs within both ImageJ and Fiji and has been tested on Windows, Macintosh and Linux machines.

Chapter 5

Location of neurons

5.1 Introduction

The images studied in the previous chapters were acquired by an expert under fixed conditions, according to a planned experiment. The sample was observed using a microscope and the expert attempted to find the cells to be subsequently analysed.

In order to facilitate this task of searching and acquiring cells, the following idea was devised.

By an automated process, an image with a low resolution is obtained, containing a big part of the sample (High Content Screening). Once this is achieved, we should design an algorithm to find cells from this poor image and obtain their location. The last phase would consist of changing automatically the objective in the microscope (by using the coordinates found by the location procedure) and acquiring a new image for any cell located , in particular with a higher resolution. The programs explained in previous chapters could then be applied on this rich pictures.

To summarise, the process includes obtaining a large image of the sample, finding the cells within it, and finally, reconfiguring the microscope to acquire new images of those cells with better conditions.

The goal of this section is to explain a method which locates neurons in large images. As explained in the previous chapters, the features based on the intensity of the image help to segment and measure the objects of an image. However, we are also explaining how other features and techniques can offer a good approach to this aim.

5.2 An Approach Using Intensity Features

5.2.1 Introduction

Considering that transfected neurons ideally have clearly different intensities to the background, in this approach only intensity and connectivity information is used to detect neurons in the images. The main method described in this section is based on hysteresis thresholding (Canny, 1986). It starts with a breadth-first search for image pixels with an intensity above a global user-defined threshold. The chosen threshold is sufficiently broad to include pixels with a very high probability of being part of a neuron (true positives), and a very low probability of being part of the background (false positives), although many turn out to be missed neuronal pixels (false negatives). The latter are largely added in the second round — segmented pixels from the first round are taken as seeds for a depth-first search to find connected pixels with an intensity superior to a second, lower user-defined threshold. This a sensible approach due to the way cells are dyed in the experiments and it allows for “graceful degradation” of intensity within neuronal image structures.

5.2.2 Previous work

The objective is to find an effective method of locating neurons in a large image based on the intensity of the pixels (an example of a fragment of such an image can be found in Figure 5.1).

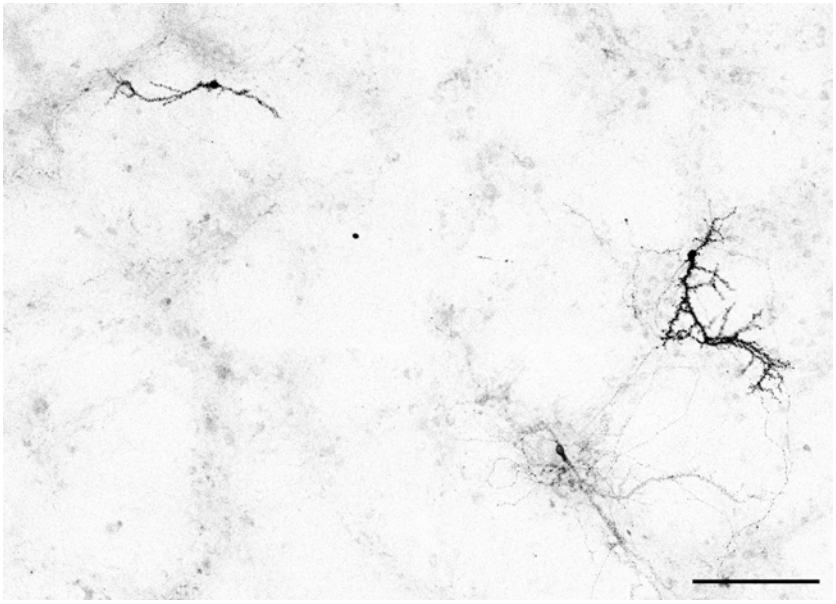


FIGURE 5.1: Patch of mosaic with neurons. Scale bar: 200μm. Image inverted.

The process was divided into two phases, the first of which was to find the soma. If it was located, a large part of the problem would be solved since the study of the image only occurs around these smaller regions. The second phase was to segment the rest of the body cell.

To this aim, different strategies were tested and are explained below.

The first strategy, based on the process described in Zhang, Zhou, Degterev, Lipinski, Adjeroh, Yuan, and Wong, 2007, requires a pre-processing of the image in gray-scale. The first step consists of applying the morphological operation 5.1.

$$J = I + THT(I, E) - BHT(I, E) \tag{5.1}$$

Where I is the original image, J is the resulting image and E is a structuring element (a disc with a determined radius). The operators corresponding to these operations are the following:

- Dilation: D=I $\oplus$ E
- Erosion: Er=I $\ominus$ E
- Opening: O=I $\circ$ E=(I $\ominus$ E) $\oplus$ E
- Closing: C=I $\bullet$ E=(I $\oplus$ E) $\ominus$ E
- Morphological Gradient: D-Er=(I $\oplus$ E)-(I $\ominus$ E)
- Top-Hat Transform (THT): I-O=I-(I $\circ$ E)
- Bottom-Hat Transform (BHT): C-I=(I $\bullet$ E)-I

The second step is to convert the 2D image into a 1D structure. Hence, the pixels are studied and classified into three groups using the algorithm called 'fuzzy c-means' (Zhou, Wang, Dougherty, Russ, and Suh, 2004; Pham and Crane, 2004).

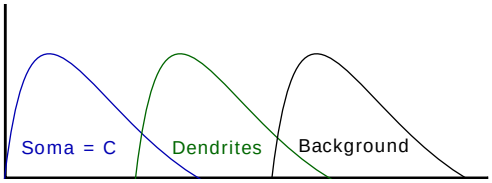


FIGURE 5.2: This graph shows the different groups regarding the values of intensity

The group with the highest values of intensity {C} (see Figure 5.2) is selected to generate the threshold:

$$T_{soma} = min_C\{I\} \tag{5.2}$$

The image is binarized and then only the somas are marked. After testing this process in our group of images, we observed that several somas were segmented, although this number was not high enough. There were too many neurons which were not found.

The second strategy is based on the proposal described in Al-Kofahi, Lasek, Szarowski, Pace, Nagy, Turner, and Roysam, 2002. While their process was designed for 3D images, their idea could be adapted to 2D images. First, the authors suggest using the morphological operation called 'closing'. Therefore, the structural element is adapted to the different structures in the image (soma or dendrites).

In order to segment the soma, an adaptive threshold is used to binarize the image. The threshold is defined by the median intensity of all the images and the maximum of such medians (see more detail in Al-Kofahi, Lasek, Szarowski, Pace, Nagy, Turner, and Roysam, 2002). This thresholding criterion ensures that the signal does not produce an out-of-focus image.

Finally, an analysis of the connected components is completed in the binary image to give a unique label to all the somas.

Both strategies inspired us to refine the following method. We pre-processed the image, applying the median filter with different radii. The aim was to observe whether the noise of the image was removed and the objects were more homogenised. However, the results obtained were the same as the results without the filter, so we concluded that this step was not necessary.

Therefore, in a second attempt to segment the soma, we used the algorithm called 'Intermodes', described in Prewitt and Mendelsohn, 1966. This algorithm uses a bimodal histogram which is iteratively smoothed until there are only two local maxima (j and k). Then, the threshold is computed as $\frac{(j+k)}{2}$. Once we have a binary image, the morphological operation known as 'open' is applied to remove the noise or small artefacts.

In the end, the connected components, depicted the somas, are analysed.

All these strategies have an automatic process in order to obtain the threshold which binarizes the image. While this offers the greatest advantage, it is also more inconvenient, at least for the kinds of images which are object of study in this chapter.

These are sizeable images which depict a substantial region of the sample. The images were acquired by a confocal microscope, which focuses on one plane. This

technique is very good to clearly see the objects which are in the same plane, but given that the cells are 3D structures, they are not only in a single plane. Most of time it is possible to acquire the soma in a plane, however the dendrites can be found on several planes. This is one of the reasons why the intensity of a neuron is not the same in all its structure. Some areas may appear unfocused — with lower intensity values — but the expert may consider them are clear enough to be taken into account. A detailed description about the acquisition of the images used is in Appendix B.

In light of this, we consider a user-given threshold. The specific method is described in the following section.

5.2.3 Methodology

It is common to observe in images through this technique that the soma is brighter than the rest of the neuron, like the dendrites. Hence, two thresholds are necessary, one of them to distinguish the soma (first phase) and the second, to find the dendrites (second phase).

On account of the considerable size of the image, it must be studied in smaller patches. To this aim, a grid is created in the image where each individual tile has a size fixed by the user. The process consists of looking for the tiles which have a number of pixels with an intensity value higher than the threshold given for the soma of the neurons. The user has to fix the minimum percentage of the area which a tile has to have to meet the criterion, and to give the value of the threshold too. This value determines whether a pixel is part of a soma or not.

To this aim, the process uses a linear or sequential search (Knuth, 1997) to check all the tiles. In fact, the tiles which meet the criteria are termed candidate cells (see an example of a candidate tile in Figure 5.3 (a)). As a result, several squares are labeled *candidates*. They depict the somas found in the neurons of the image.

The second phase consists of studying the neighbouring squares of each candidate tile to find the structure of the neuron. To study the "neighbourhood" we use what is termed the 4-adjacency (see Rosenfeld, 1974) — the tiles which are located above, below, to the left and the right (see Figure 5.3 (b)).

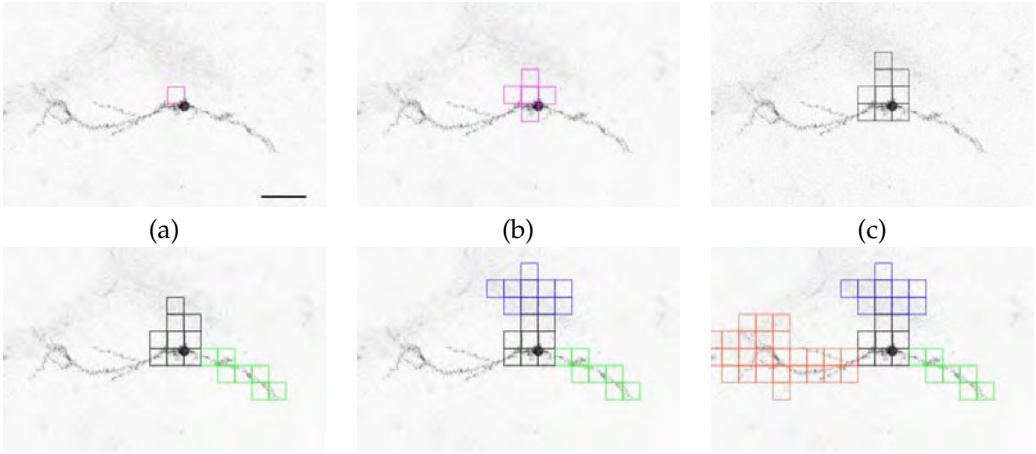


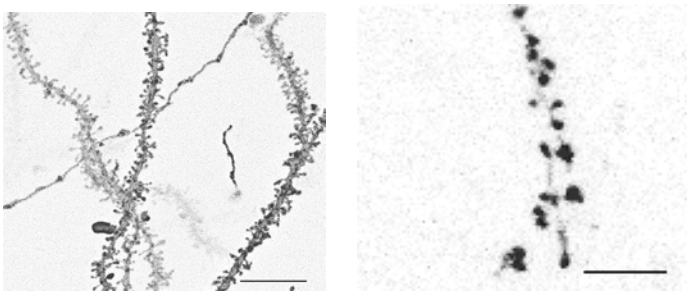
FIGURE 5.3: Neuron from the top left of Figure 5.1. Example of how the method described is applied in a particular case. Scale bar: 50μm. Images inverted.

The process for each candidate tile follows the same order for the search within its neighbourhood: up, right, down and left. The method begins looking for the brightest pixel in the tile to act as the starting point for the path — a set of pixels which have an intensity value higher than a second fixed threshold. This threshold is given by the user and determines wether a pixel belongs to a dendrite or not.

Two algorithms of 'Backtracking' (Gurari, 1999) are used in this second phase. The first one is known as 'Breadth First Search' (Skiena, 2008). Beginning at the brightest pixel, the algorithm begins searching for a path which meet all the criteria. It builds paths which are possible candidates and abandons each partial path when it determines that the path cannot be completed nor meets all the criteria. Given that these circumstances would deem it an invalid path, the algorithm continues searching.

At the moment that a valid path is found, that particular square is considered a candidate tile. The search algorithm applied in this process takes into account the 8-adjacency (see Rosenfeld, 1974) to look for a valid path. Furthermore, a path is considered valid by the algorithm if it at least has a determined length.

One of the problems with the digital images is that, while the objects can seem continuous, sometimes they are not, see Figure 5.4. This idea is related to the notions of distance in a discrete space and of generalised adjacency. With this in mind, the user can allow gaps in the path of a determined number of pixels. To avoid a path constituting more gaps that real pixels, the user can fix the number pixels which have to be followed into the path, to allow a gap.



(a) (b)

FIGURE 5.4: (a) Example of dendrites which appear to contain a continuing trace. Scale bar: 10μm. (b) However, upon focusing on one of the dendrites, the trace, while clearer, turns out to be discontinued. Scale bar: 2μm. Both images inverted.

This process finishes when every candidate tile and its neighbourhood are studied (see Figure 5.3). In this case, the search algorithm used is 'Depth First Search' (Tarjan, 1972). The optimal values which have been found for this process and this kind of image are in Table 5.1.

As a result, the method creates a rectangle around each group of connected tiles. This region is created taking into account the tiles which are in the upper left and lower right corner, see Figure 5.5.

This process was developed as a plug-in of ImageJ, called LocationJ.

Extrapolating the algorithm to detect the neuronal structure

Upon testing the method, results were observed such as those in Figure 5.5. This led us to consider using the same algorithm to locate and define the structure of the neurons.

Parameter	Value
size of square	25x25px
percentage of the area	20%
th_S	70
th_d	20
lenght of a path	15px
size of a gap allowed	3
min. length of a path before a gap	5px

TABLE 5.1: Optimal values found upon running the method for this kind of images. The term th_S is the threshold chosen for finding somas and th_d is the threshold to find dendritical structure.

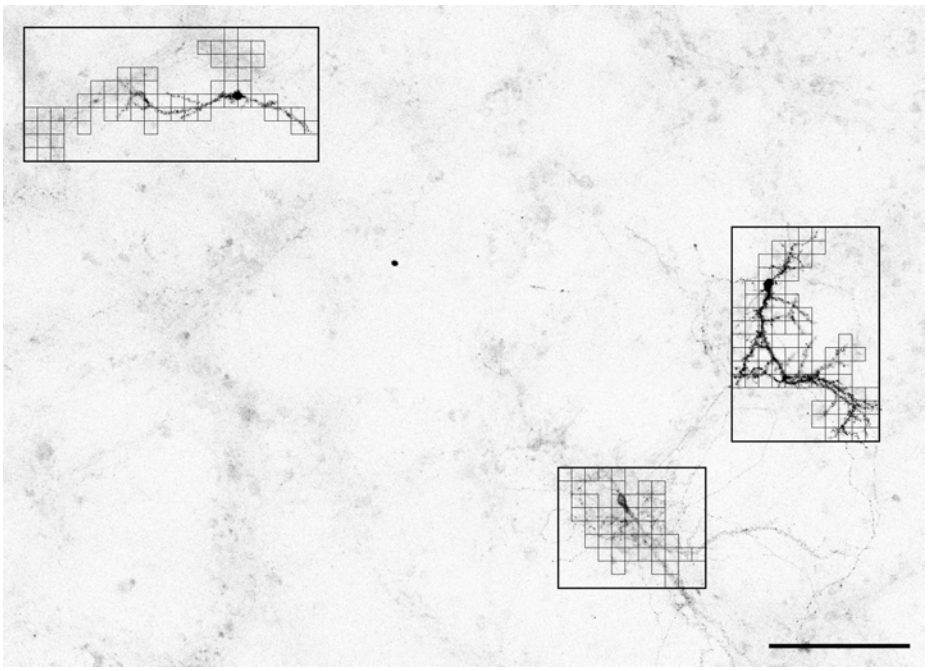


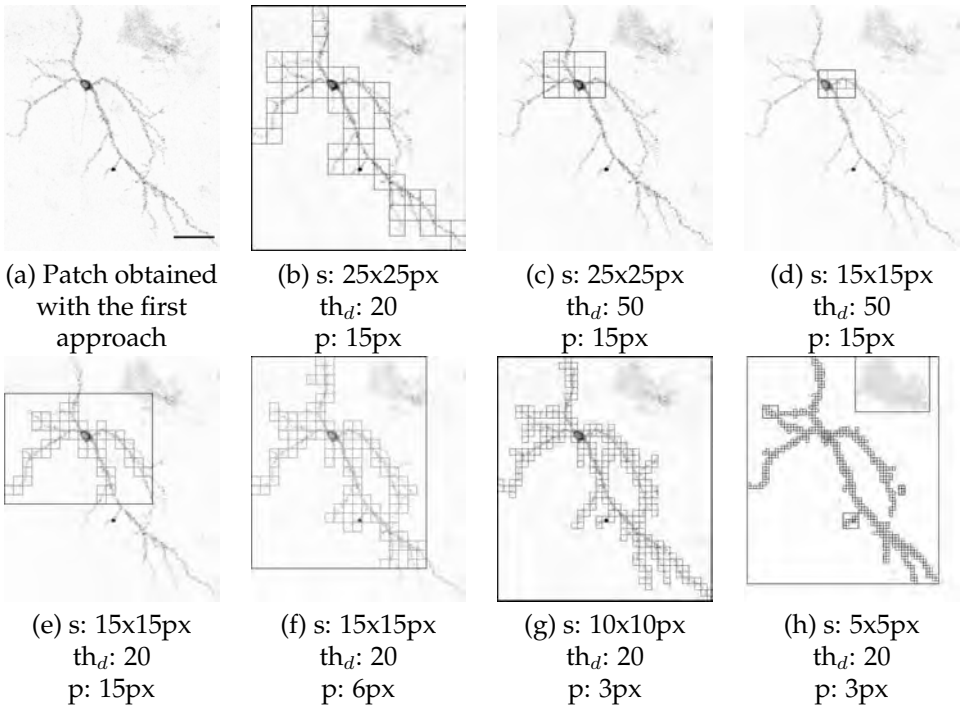
FIGURE 5.5: Example of the result given by the method in a patch of the mosaic. Scale bar: 200μm. Image inverted.

To that end, the algorithm is applied to the patches obtained through the process described in the previous section. The images are smaller than before, since they are a patch of the mosaic, so the values of the parameters are smaller too; for example, the size of the squares or the length of the path, see Figure 5.6.

The optimal conditions found are in the Table 5.2. An example of the result obtained with these parameters is in Figure 5.7 (a), together with another example, Figure 5.7 (b). The green squares are the tiles selected in the first phase of the process. These squares meet the criteria of a soma. On the other hand, the blue squares meet the criteria of part of the cell body. They are selected in the second phase of the process.

Looking at Figure 5.7 (a or c), the result obtained is very accurate with respect to the structure of the neuron. However, the second example (see Figure 5.7 (b or d)) shows that the area selected has all the structure of the neuron but also a region with noise.

Looking at the pictures, it is very clear that the cell body has more intensity than the rest of the image. Nevertheless, the previous process is not clever enough to



(a) Patch obtained with the first approach

(b) s: 25x25px th_d: 20 p: 15px

(c) s: 25x25px th_d: 50 p: 15px

(d) s: 15x15px th_d: 50 p: 15px

(e) s: 15x15px th_d: 20 p: 15px

(f) s: 15x15px th_d: 20 p: 6px

(g) s: 10x10px th_d: 20 p: 3px

(h) s: 5x5px th_d: 20 p: 3px

FIGURE 5.6: Results of applying different values in order to determine the best choice. The threshold for the soma is 70 in all examples, and the percentage of each area with this intensity is equal or higher than 20%. The rest of the parameters are indicated below each example. In the pictures, variable called s means size of the square, th_d is the threshold for the dendrites and p means the minimum length of the path. Scale bar: 50μm. Images inverted.

Parameter	**Value**
size of square	10x10px
percentage of the area	20%
th_S	70
th_d	20
lenght of a path	15px
size of a gap allowed	3
min. length of a path before a gap	5px

TABLE 5.2: Optimal values found upon running the method in this phase. The term th_S is the threshold chosen for finding somas and th_d is the threshold to find dendritical structure.

generalize since some images, such as Figure 5.7 (b or d), have not produced good results.

In order to get a better understanding, the squares in the background that are considered body cells, the histograms of samples containing squares both with and without neuronal structures (the background) were obtained, see Figure 5.8. Upon observing the results it was clear whether a square contains a neuronal structure or not.

If we observe the histograms (see an example in Figure 5.8), instead of the image, it is possible to distinguish them too.

This suggests that there are one or more parameters which define each stage of the squares. Six statistical measurements related to the intensity are studied, since

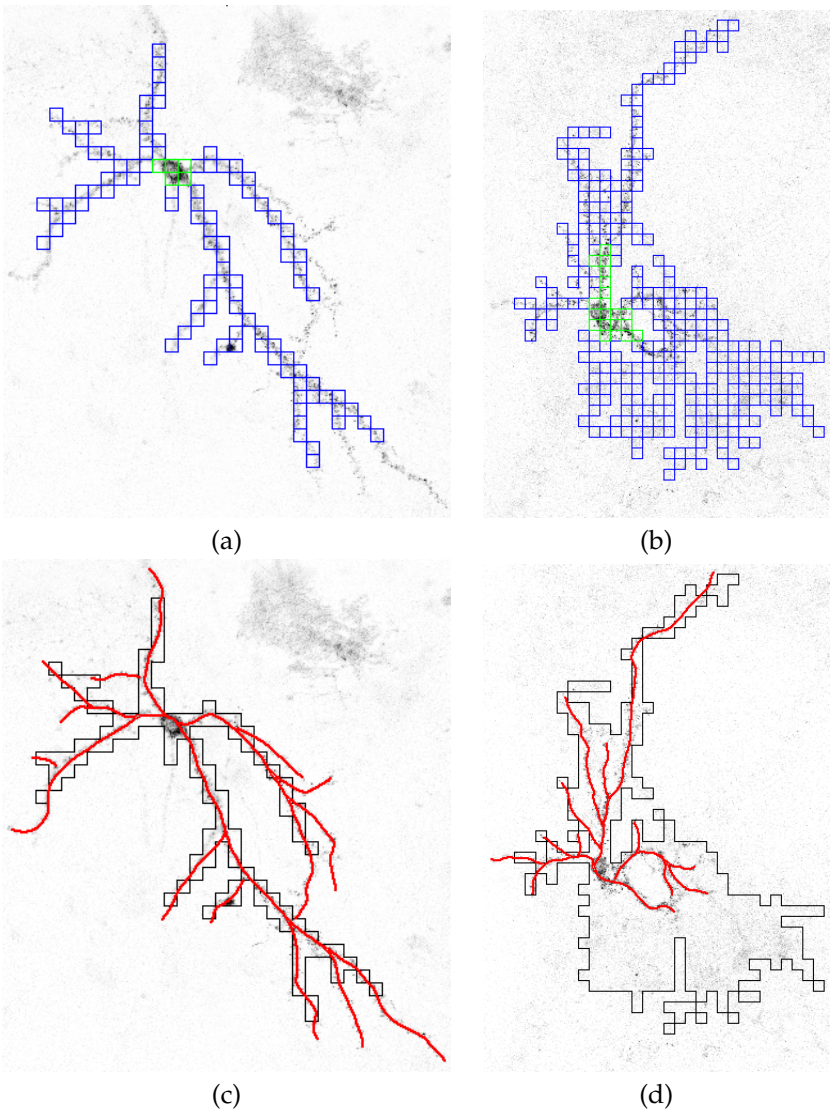
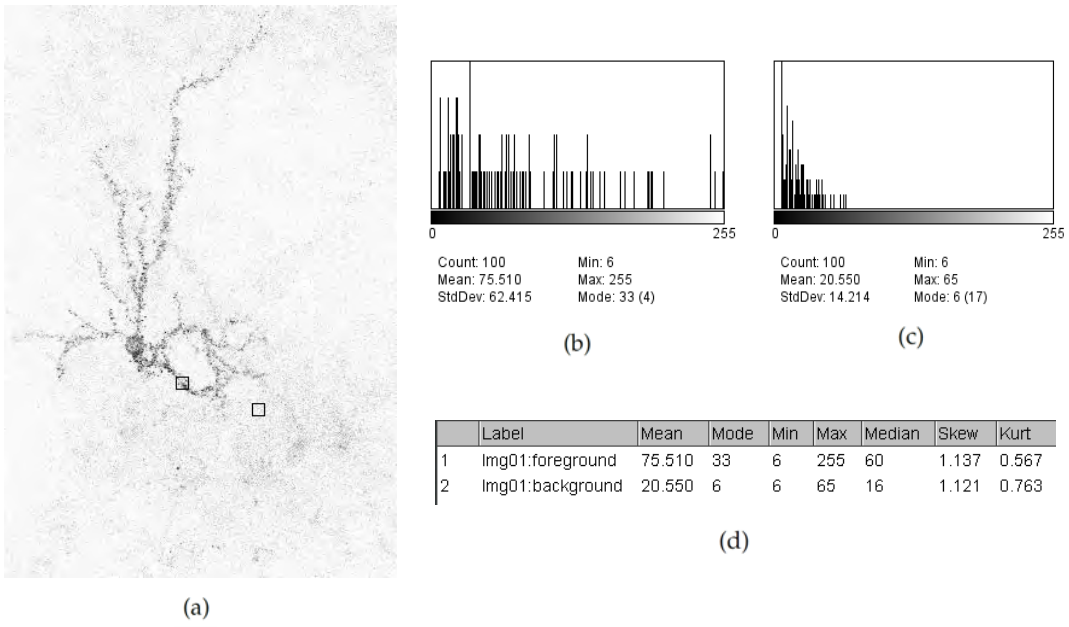


FIGURE 5.7: (a) and (b) are examples of the patches of images after running the process. (c) and (d) show the structure marked by the expert in red and the region marked by the algorithm in black. Images inverted.

the histogram depicts the amount of intensity values which there are in the selection. The features are the following:

- mean
- median
- maximum
- standard deviation (StDev)
- Skewness
- Kurtosis

The measurements known as Kurtosis and Skewness provide information about the distribution of a histogram of a selected area.



	Label	Mean	Mode	Min	Max	Median	Skew	Kurt
1	Img01:foreground	75.510	33	6	255	60	1.137	0.567
2	Img01:background	20.550	6	6	65	16	1.121	0.763

(d)

FIGURE 5.8: (a) Neuron with two selected regions. Scale bar: 50μm. Image inverted. (b) Histogram of the region with part of the neuronal structure (foreground). It is the square on the left. (c) Histogram of the region in the background. It is the square on the right. (d) Table with the statistical measurements of both regions.

With this in mind, these specific features were extracted for a set of images. We observed that they define the stage of a square (whether it contains a neuronal structure or not), however it is difficult to determine the weight of each one.

At this point, we considered the use of Machine Learning techniques to study and determine whether a square is, in fact, part of a neuron or it simply belongs in the background.

Upon considering it as a clustering problem, the k-means algorithm (Lloyd, 1982), an unsupervised technique, was applied.

This algorithm consists of partitioning the data into k clusters. Each value belongs to the cluster with the nearest mean or centroid. The Euclidean distance is used to compute the distance among values. Each value is considered a point in an n-dimensional space, n being the number of features which are used to describe the data.

In our case, the number of clusters is two ($k = 2$): one containing the squares with neuronal structure, and the other composed of the squares belonging to the background. In addition, the six intensity features of each square previously mentioned are computed and the clustering algorithm is applied to this data.

With this in mind, the following algorithm is outlined.

Algorithm 8.
`Input`: A greyscale image
`Output`: Binary image in which each colour corresponds to the labels: neuron or background

1. Make a grid in the image.
2. Compute the mean, mode, StDev, maximum, Skewness and Kurtosis for each tile of the grid.

3. Apply the k-means algorithm.

4. Analyse the result.

In order to study the features, we use the ImageJ programme and the version of the k-means algorithm implemented in Weka (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten, 2009). In fact, we use these programmes to obtain results as shown in Figure 5.10.

What we are mainly concerned with here is to define the structure of a neuron and, to this aim, we use the patches obtained in section 5.2.3, with the plug-in LocationJ.

This algorithm was run on each image, creating a grid with tiles of 5×5 px. Firstly, all features were used as descriptors of the squares and the results were similar to those of the examples in Figure 5.10 (a). Then a descriptor was removed in order to determine whether the results would improve. In the part (b) of Figure 5.10, it is possible to observe some of the examples obtained with all features barring Skewness. Likewise, this study was repeated removing each one of the descriptors. The next step consisted of running the algorithm again, but in this case, with just four descriptors, and so forth (see Figure 5.10).

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

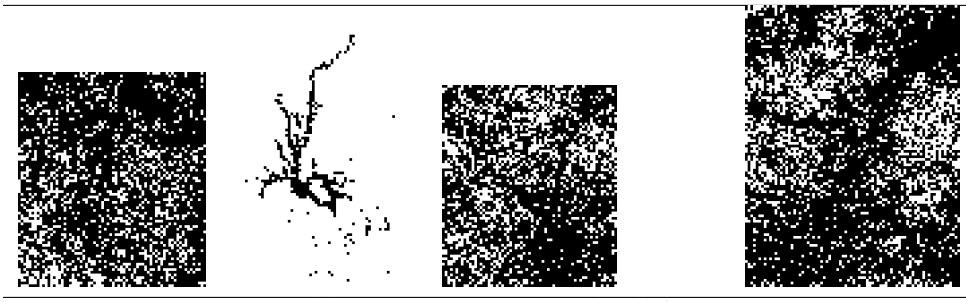
FIGURE 5.9: Example of four manually-traced neurons used in the study (inverted images).

In order to determine which of these combinations offers a more accurate result, we did the following study. We compared the results obtained with algorithm 8 and with the manually traced neurons drawn by an expert.

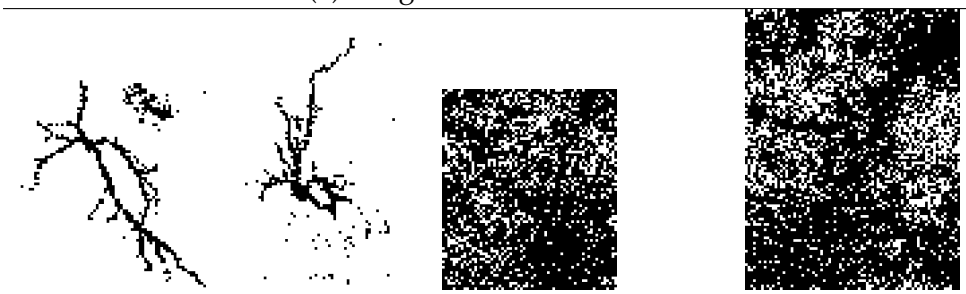
After applying different combinations of the previously mentioned features, we concluded that the best results were obtained using the followings two features: maximum and standard deviation. However, this was further checked comparing the results in this case to the manually traced neurons given by the expert.

Before explaining which process is used to study the results, certain keywords are defined below.

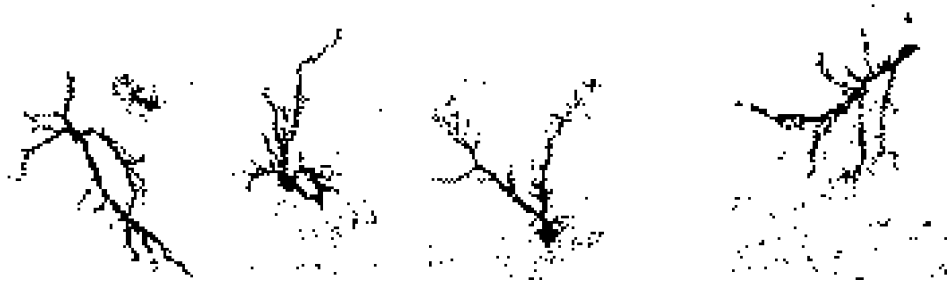
Positive squares, or only positive (P), if the square contains neuronal structure, as opposed to negative squares, or only negative (N), if the square only shows background; results obtained by the algorithm are called positive (P') and negative (N') predictions; a square is true positive (TP) if the prediction done with an algorithm and the description done by the expert is positive in both cases; and finally, a square is false positive (FP) if the prediction done with an algorithm is positive but the description done by the expert is negative. In this case, the prediction of the algorithm is erroneous. Analogous definitions also have the concepts true and false negative (TN and FN respectively).



(a) Images studied with all features.



(b) Images studied with just 5 features (no Skewness descriptor).



(c) Images studied with just 3 features: mean, maximum and standard deviation.

(d) Images studied with just 2 features: maximum and standard deviation.

FIGURE 5.10: Examples of the results obtained from applying Algorithm 8 to different combinations of descriptors. From left to right are the same examples as Figure 5.9, respectively.

	Predicted condition Positive (P')	Predicted condition Negative (N')
Condition Positive (P)	True Positive (TP)	False Positive (FP)
Condition Negative (N)	False Negative (FN)	True Negative (TN)

TABLE 5.3: Theoretical confusion matrix.

All these outcomes can be formulated in a 2×2 matrix, called a confusion matrix or contingency matrix, see Table 5.3. The confusion matrix allows the visualization of the performance of an algorithm. Each column of the matrix depicts the predictions of the algorithm while each row represents the 'real' instances - those defined by the expert - or vice versa.

In order to compare the results obtained by the algorithm (with two features: maximum and standard deviation) and the expert, we compute the confusion matrix for each one of the images studied, see Table 5.4. In addition, terms and formulas can be derived as a result of this matrix. The most commonly used parameters are recall and precision, which are also analysed for each image. The parameter called recall is also known as sensitivity or true positive rate (TPR). It measures the proportion of positives which are correctly identified as such, see Equation 5.3 (i.e. the percentage of squares containing neuronal structure which are correctly identified as meeting a criteriom). On the other hand, precision (also called positive predictive value (PPV)) is the fraction of relevant instances among the ones classified with the same predicted condition (i.e. the percentage of squares which are correctly identify as positive (TP) among all the squares classified with the same condition (P')), see Equation 5.3.

$$recall = TPR = \frac{TP}{TP + FN} \qquad \text{and} \qquad precision = PPV = \frac{TP}{TP + FP} \tag{5.3}$$

The values shown in Table 5.4 are an example of the values obtained in this study. While we can observe that the recall is high, which is a favourable result, the precision parameter is only approximately 50%, which is less than favourable, implying these results could be considered random.

This study made it evident that, while the machine learning technique provided neuronal structures, there was still a need to conduct a more in-depth study of the features defining neurons and their processing.

Using the intensity features implies losing information in most cases. To segment or, for instance, apply the morphological operations requires a threshold which results in losing information. It was evident we needed to choose another way to study the problem. The starting point of this method had been, up to this point, the brightest pixel in a region. We decided to change this process and studying the whole mosaic by making a fixed grid and focusing on the tiles. This is explained in the next section, Section 5.3.

In the light of this strategy, it is necessary to describe which is the process to make a grid in the mosaic or large images.

	P′	N′	
P	312	37	349
N	209	5042	5251
	521	5079	

r:	0.894
p:	0.599

(a)

	P′	N′	
P	231	18	249
N	180	5746	5926
	411	5764	

r:	0.928
p:	0.562

(b)

	P′	N′	
P	227	36	263
N	215	4397	4612
	442	4433	

r:	0.863
p:	0.514

(c)

	P′	N′	
P	324	44	368
N	226	7806	8032
	550	7850	

r:	0.880
p:	0.5889

(d)

TABLE 5.4: Example of confusion matrices and the parameters: recall (r) and precision (p) of the images of Figure 5.9. Data (a) corresponds to image (a) and so on.

5.2.4 Annotation of the Image Data

When using the machine learning techniques, the patches to be classified must first be sampled from the image, since it is computationally very costly to exhaustively check all possible patch locations in an image.

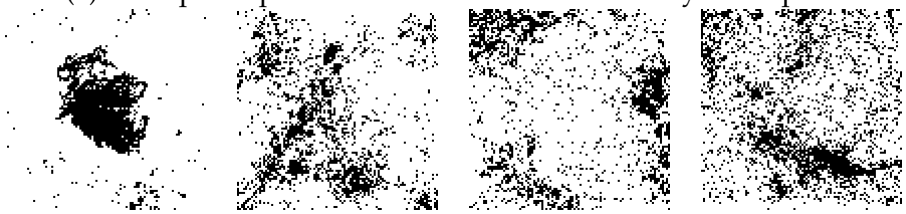
An expert neurobiologist manually marked all regions in these images that contained neurons (in our case, 409 neurons were marked). Through this we learned that neuron regions in our images are typically 500×500 pixels. From the non-neuron regions in the images we randomly sampled approximately 1,000 patches of this size to serve as negative examples (background). This sample of patches is also known as gold standard (gs). It is a term used to refer to information provided by a direct observation obtained by an expert.

5.3 Methodology Used Over Binary Images

We used this set of images to study the features which provide a better description of a neuron. We based on the previous section to work on images with less information, we mean, to transform each image of gray scale into a binary image. The binary images are obtained applying the method described in Algorithm 8, although two features were considered as descriptors. In this case the size of the square for the grid made in the patch is 5×5 — it is the value of the size which gave us best results — and only the standard deviation and maximum values are computed to obtain the binary image. One of advantages of this method is that we avoided that the expert decided a threshold to binarize the image. Figure 5.11 shows some examples of patches binarized using this method.

It is possible to observe in Figure 5.11 that the binary images of neurons are different from the images of background. However there is a problem with the images which are neurons (see Figure 5.11 (a.4)), but contain a lot of noise, or vice versa,

(a) Example of patches classified like a neuron by the expert.



(b) Example of patches classified as background by the expert.

FIGURE 5.11: Patches binarized using a method described in Algorithm 8.

images which are not neurons, but they contain another cell as, for instance, an astrocyte (see Figure 5.11 (b.1)).

Taking this fact into account, the process used was the following. With these new patches (binary patches) other features were studied such as the number of connected components, the area of the background and others related to the biggest connected component such as the solidity or Feret's diameter. The parameter called solidity is computed through of the formula $solidity = \frac{area}{convex\ area}$, convex area being the area enclosed by the convex hull of the outer contour of an object. On the other hand, the value known as Feret's diameter is the longest distance between any two points along the selected boundary, also known as maximum caliper.

These features were computed for each patch and they determined a vector of descriptors which was the input for the unsupervised clustering algorithm. In fact, different combinations of the descriptors were analysed by the clustering algorithm. The results were compared with the real results and the recall and precision were computed for each case. After several combinations of the features, we observed that the best results obtained were with these features: the number of black pixels which were in the patch (*bp*), the number of connected component (*cc*), the solidity and the inverse number of connected component (*inv(cc)*).

To observe which was the best option among them, we study the ROC space (*receiver operating characteristic* space, see Fawcett, 2006) compound of the combinations which were being studied. The best possible method would yield a point in the upper left corner (or coordinate $(0, 1)$) of the space. This corner represents 100% sensitivity (no false negatives) and 100% specificity (no false positives). Therefore, we obtained Figure 5.12.

Taking this into account, the features chosen to continue with the study are:

- (bp) the number of black pixels, which means that if there are a lot of foreground or not
- (cc) the number of connected components, this value implies if the black pixels are dispersed or as opposed to they are closed and they compound a big connected components

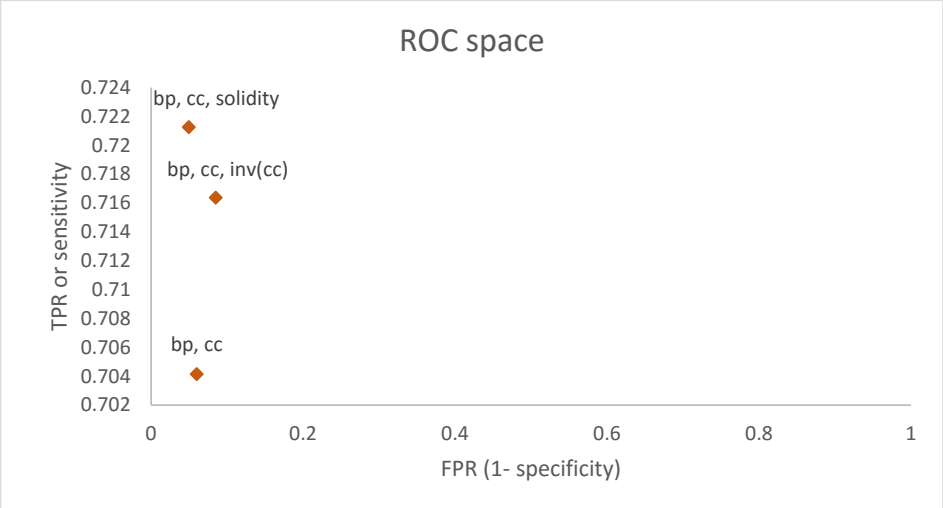


FIGURE 5.12: ROC space of same method with different features.

- (solidity) is the measurement of the overall concavity of a particle. It is defined as $solidity = \frac{area}{convex\ area}$. The area and the convex hull area approach each other for the solid particles (resulting a value close to 1).

On account of the fact that these are the best features we repeated the study with different groups of patches to check the results. The neurons of the set of images were split into two subsets. The first subset (g0_n) consists of the patches classified as neurons, however they have noise or their contrast was lower and the process of binarization was not clear (in this group there are 146 patches). The second (g1_n)) contains the neurons which are clear and they do not generate any problem to analyse (the number of patches is 263). On the other hand, other two subsets were created for the patches of the background. The first contains of the patches classified as background by the expert (they are 1497 patches and it is called bg_gs) and the second contains patches which are background but in the previous process were considered as neurons (bg_FP). We decided to work with these groups because from them we could observe what happens with the patches which are not clear.

Table 5.5 shows the recall and precision values for the different combinations of the groups when the clustering algorithm (k-means) is run on them.

Afterwards seeing and analysing these results, we conclude that when the neuron is well-defined, that is to say, when the neuron is clear in the image, there are not noise or artefacts on it, to classify the patches computing these features based on the binarization of the patch is possible with a clustering technique. However, it is not an easy problem when the image has noise or has not contrast.

In view of these results, we observed these features are not always enough. Using the intensity features implies to loss information in most cases. To segment or, for instance, apply the morphological operations requires to binarize the image which results in losing information.

	bp+cc+solidity	
Sets of patches	recall	precision
gs	0.7213	0.7995
gs+bg_FP	0.6626	0.8187
g0_n+bg_gs	0.5548	0.2547
g1_n+bg_gs	1	1
g0_n+bg_FP	0	0
g1_n+bg_FP	0.9886	0.8125

TABLE 5.5: Recall and precision values for the different combinations of the subsets of patches.

Thus, we decided to study other features related with other aspects of the image and not only with the value of the intensity; for instance, the texture features, which are based on the location of a pixel respect to its neighbours.

5.4 Methodology Used Over Textures Features

The work explained in this section has been presented and published in the IEEE 13th International Symposium on Biomedical Imaging (ISBI) held in Prague, Czech Republic, 2016. It was presented as a poster and published as a short-paper under the title: "Automatic detection of neurons in high-content microscope images using machine learning approaches", see Mata, Radojevic, Smal, Morales, Meijering, and Rubio, 2016.

5.4.1 Introduction

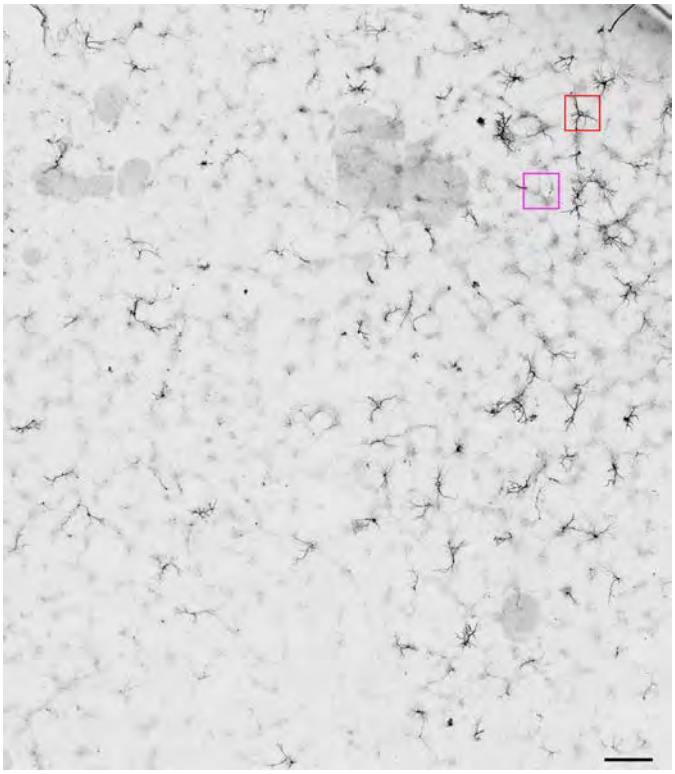
Before going any further, we explain a term which is used in analysis of large images and is the core of the next approach.

An exciting recent development in neuroscience is the use of high-content analysis (HCA). HCA generally refers to the combination of automated acquisition and analysis of large microscopic image data sets for biological discovery and is often employed by pharmaceutical and biotechnology companies but increasingly also in academia and research institutes (see Xia and Wong, 2012). In view of the vast amounts of data and the desire to eliminate possible human bias, automation is a key requirement in HCA, and thus the analysis methods must be highly robust and reliable. Although challenging, HCA is now used also in basic neuroscience research (see Dragunow, 2008; Anderl, Redpath, and Ball, 2009; Radio, 2012), and various image analysis pipelines have already been developed for neuron quantification in high-content image data (see Vallotton, Lagerstrom, Sun, Buckley, Wang, DeSilva, Tan, and Gunnersen, 2007; Zhang, Zhou, Degterev, Lipinski, Adjeroh, Yuan, and Wong, 2007; Wu, Schulte, Sepp, Littleton, and Hong, 2010; Charoenkwan, Hwang, Cutler, Lee, Ko, Huang, and Ho, 2013).

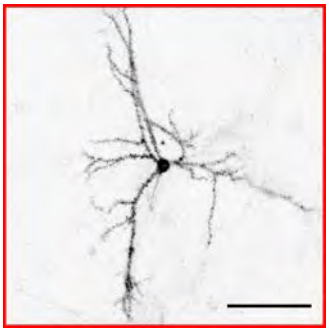
The first step in an HCA pipeline for neuron screening is to detect image regions of interest containing neurons as opposed to background or irrelevant structures (see Figure 5.13). Commonly this is done by image prefiltering (denoising, illumination correction, contrast enhancement) followed by some form of intensity-based thresholding. However, the images often contain debris and other artefacts that are larger or more complex than standard prefiltering techniques can eliminate, and thus more sophisticated solutions are needed.

Therefore, we studied the potential of machine learning based approaches for automatic detection of neurons in microscopic images for HCA. In Section 5.2 we studied an intensity-based detection method, using expert manual annotation as the gold standard (see SubSection 5.2.4). In this section, we implement and compare two approaches based on different feature sets and classifiers, and compare their performance to each other.

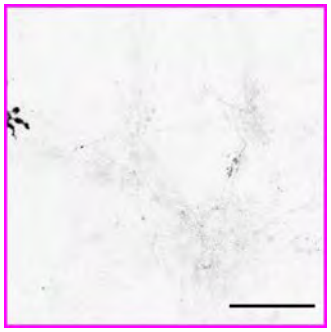
We want to demonstrate that with the right feature set and training procedure, machine learning can indeed improve neuron detection.



(a) Example high-content image. Scale bar: 600μm.



(b) Neuron region. Scale bar: 100μm.



(c) Background region. Scale bar: 100μm.

FIGURE 5.13: Example of a high-content microscope image (a) and regions containing a neuron (b) and background (c). Image intensities are inverted here for displaying purposes.

5.4.2 Context

As an alternative to intensity based neuron detection (Section 5.2) and previous section (Section 5.3), we considered two machine learning based approaches. Detection was achieved by classification of image patches as positive (containing neuron structure) or negative (containing background) using features computed from these patches.

WND-CHARM tool

The first approach was based on computing an extensive list of image features using the WND-CHARM library (Orlov, Shamir, Macura, Johnston, Eckley, and Goldberg, 2008).

This software is an open source designed to be used for biological image analysis, in fact, this tool can be used for a wide range of biological data sets. This software has two steps: firstly, it extracts image content descriptors from the raw image, image transforms, and compound image transforms. Secondly, the most relevant features are selected and the feature vector of each image is used to obtain a model and to classify new images.

The type of features used by this software can be classified in four categories (Orlov, Shamir, Macura, Johnston, Eckley, and Goldberg, 2008): different types of polynomial decompositions, high contrast features, pixel statistics, and texture descriptors. Polynomial decomposition is generated to approximate the image to some fidelity, and its coefficients are used as descriptors for the image content. Other features are based on high contrast, such as edges and objects, size, shape, spatial distribution, etc. Pixel statistics are based on the intensity within the image and use histograms and moments. The last category is texture features which denotes the pixel variation in intensity for different directions and resolutions.

These features are computed for raw images and in grayscale. In addition, an image is subjected to three standard transforms (Fourier, wavelet and Chebyshev) and some transform combinations.

To compute the information for each image, *wnd-charm* uses the following algorithms, described more thoroughly in Orlov, Shamir, Macura, Johnston, Eckley, and Goldberg, 2008:

- Radon transform features are computed from a projection of pixel intensities onto a radial line from the image center at these angles: $0°$, $45°$, $90°$, and $135°$. (Lim, 1990),
- Chebyshev Statistics and Chebyshev-Fourier features are computed from Chebyshev polynomials (Gradshtein and Ryzhik, 1994),
- Gabor filters are based on Gabor wavelets (Gabor, 1946) to construct spectral filters which search for image texture and periodicity descriptors,
- Multi-scale histograms are computed with different number of bins (3, 5, 7 and 9) as defined in Hadjidementriou, Grossberg, and Nayar, 2001,
- The first four Moments which are mean, variance, skewness and kurtosis are also computed in four different directions: $0°$, $90°$, $+45°$, and $-45°$,
- Tamura texture features measure the scale of the texture (*coarseness*), estimate the dynamic range of the pixel intensities (*contrast*), and indicate if the image favours a certain direction (*directionality*)(Tamura, Mori, and Yamavaki, 1978),

- Edge Statistics features are calculated on the image's Prewitt (Prewitt, 1970),
- Object Statistics are computed from a binary mask obtained to apply to the image the Otsu global threshold (Otsu, 1979),
- Zernike and Haralick features. First ones are the coefficients of the Zernike polynomial approximation of the image and the others are calculated on the image's co-occurrence matrix. Both algorithms to calculate these features are described in (Murphy, 2001).

All these features are computed for all patches of images of the experiment. The total number of descriptors which are calculated is 1059 for each image. They are stored in a sig-file. However not all features may be equally relevant, in the training step a Fisher discriminant score (Bishop, 2006) is computed for each, which allowed building a ranked preference list of features. In the testing phase of the experiments we used the top-15% features according this score. Classification of patches based on these features was done using a weighted neighbour distances (WND) classifier (Orlov, Shamir, Macura, Johnston, Eckley, and Goldberg, 2008).

SIFT-features

The second approach was based on computing the scale-invariant feature transform (SIFT) described in Lowe, 2004. This approach consists of computing features which are invariant to image scale and rotation, and partially invariant to change in illumination and 3D viewpoint.

To this aim, the method described by Lowe, 2004 is made up of four main steps.

The first is scale-space extreme detection which searches over different scales and image locations to identify interest points. The next step consists of selecting some points based on measures of stability. These points are called keypoints and this step is recognised as keypoint localization. In the third step, the local image gradient directions are used to assign one or more orientations to each keypoint location. In the final step, a representation is obtained from the local image gradients. As a consequence, it is possible to study levels of local shape distortion and change in illumination.

Finally, each keypoint describes the local image region using a vector as a descriptor, which is composed of 128 elements with the values of the orientation histogram entries. Additionally, this vector is normalized to unit length, hence the effects of illumination change are reduced and the contrast changes are canceled by this vector normalization. In addition, a brightness change does not affect the values because they have been calculated from different pixels. Therefore, the descriptor is invariant to these kinds of changes in the image.

In order to compare different images described with SIFT-features, it is necessary to normalize the data since the descriptors for each image do not have to be an identical number. The “Bag of Words" model (BoW model, Sivic, 2009) is a popular approach to achieve this normalization. This algorithm consists of obtaining a “codeword", which is considered the base for generating the histogram of features. This codeword is obtained using the method called k-means clustering (MacQueen, 1967) on all the descriptors. Finally, an image can be represented by a histogram which indicates how many of its descriptors are in each cluster.

Classification of patches based on these features was done using the Naive-Bayes (NB) classifier from the WEKA toolkit (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten, 2009).

5.4.3 Methodology

The performance of the methods was initially evaluated directly on the expert annotated neuron patches (positives) and sampled background patches (negatives) as described in Section 5.2.4. A random selection of 75% of the patches from each class was used for training of the machine learning based detectors while the remaining 25% was used for testing. Given the classification output and the annotation, the number of true positives (TP), false positives (FP), and false negatives (FN) could automatically be determined for each method, from which the classification performance was quantified in terms of recall and precision.

In order to use these machine learning approaches, we used the same gold standard obtained for the before section. About 1,000 patches (of 500×500 pixels as motivated above) were sampled uniformly from the image, which suffices to capture the true neuron regions (typically only several dozens).

The locations of the considered patches typically did not match exactly with those of the annotated patches. Therefore patches classified as positive were further examined: if a positive patch overlapped less than 20% with any annotated neuron patch, it was declared a false positive, and all positive patches overlapping 20% or more with the same annotated neuron patch were considered a unique (single) true positive match. The 20% threshold may seem rather conservative but additional experiments showed us that higher percentages did not consistently improve the results.

Due to the sampling, many patches in the second experiment potentially contained small portions of neuron structures, which to some degree caused a mismatch with the initial training set. Therefore we expected the performance of the methods to be lower in the second experiment as compared to the first. To improve performance we implemented a bootstrapping approach, where the classifiers were retrained in a second round using as negative examples only the false positives from the first round, while continuing using the true positive examples from the expert annotation. The machine learning detectors without (versus with) using bootstrapping are referred to as WND-CHARM-A and NB-SIFT-A (versus WND-CHARM-B and NB-SIFT-B).

5.4.4 Experimental Results

We compared the results obtained to apply the methods of Section 5.4 with the first approach, also known as LocationJ (described in Section 5.2). Therefore, we observe better the difference among all methods.

The results of the first experiment are presented in Table 5.6. Although WND-CHARM-A showed perfect recall, its precision was lower than LocationJ, indicating that it produced more false positives and the training set did not well-reflect possible background variability. Precision was drastically improved by using the bootstrapped variant, WND-CHARM-B, with still near-perfect recall. Bootstrapping improved the performance of NB-SIFT in terms of both measures, resulting in a better recall than with LocationJ but a slightly lower precision. The negative effect of patch sampling on the performance of all methods is seen from the results of the second experiment in Table 5.7. As anticipated, in virtually all cases both the recall and the precision was considerably lower than in the first experiment. The differences between the two measures are also much higher, with the precision being substantially

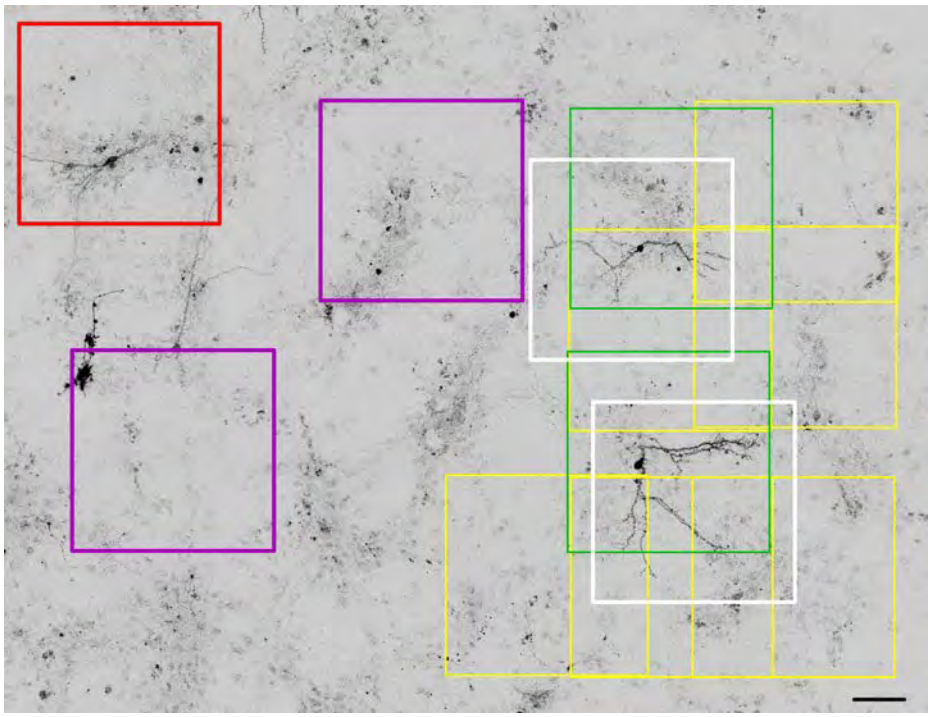


FIGURE 5.14: Example detection result using NB-SIFT-B. Shown are part of a high-content microscope image of neurons and the borders of various patches overlaid in colour coding: an expert annotated neuron region missed by the detector (red), two annotated regions found by the detector (white), true-positive patches (yellow) ignored by the detector in favor of the best overlapping patch (green), and false-positive patches (magenta). Image inverted. Scale bar: 100μm).

lower than the recall in all cases, indicating an increase of false positives. Of the machine learning approaches only NB-SIFT-B performed better than LocationJ in terms of both measures in these experiments. Example detection results from NB-SIFT-B are shown in Figure 5.14.

Method	Recall	Precision
LocationJ	0.80	0.86
WND-CHARM-A	1.00	0.62
WND-CHARM-B	0.97	0.82
NB-SIFT-A	0.74	0.46
NB-SIFT-B	0.88	0.83

TABLE 5.6: Results of the first experiment.

Method	Recall	Precision
LocationJ	0.64	0.50
WND-CHARM-A	0.89	0.19
WND-CHARM-B	0.66	0.44
NB-SIFT-A	0.60	0.26
NB-SIFT-B	0.94	0.57

TABLE 5.7: Results of the second experiment.

5.5 Discussion

Up till now, in this chapter we have investigated the problem of neuron detection in high-content microscope images from screening experiments in neuroscience. We have evaluated the performance of two machine learning based approaches using different feature sets and classifiers in comparison with a method based on hysteresis thresholding of intensity information only. The results showed that machine learning approaches may perform superiorly. However this improvement does not come lightly and requires careful consideration of the ingredients. Of the two machine learning based approaches considered, the best results were obtained with NB-SIFT in combination with a bootstrapping procedure for the training stage, even though WND-CHARM takes into account a much wider variety of image features. This suggests that selection of the right feature set as well as the right training set is crucial to achieve good detection performance.

5.6 Deeper Study about Features of Images and Machine Learning Algorithms

The work explained in this section is still in progress and it is in collaboration with Erik Meijering, PhD and his group, from the Biomedical Imaging Group Rotterdam of the Erasmus University Medical Center of Rotterdam in the Netherlands, and Carlos Fernández-Lozano, PhD, from the Instituto de Investigación Biomédica de A Coruña of the Complejo Hospitalario Universitario de A Coruña in Spain.

5.6.1 Introduction

Afterwards the results previously obtained, we aim to perform a more detailed study of the effects of different types of features in combination with different types of classifiers and training procedures. This work is described in the following subsections.

5.6.2 Methodology

For the following study we also used the images explained in the previous section and their experimental process described in Appendix B. It is important to highlight that each image was approximately 10,000 $\times$ 12,000 pixels (covering approximately 70 mm^2 of the culture dish) and contained an average of 40 transfected (with the fluorescent protein GFP) neurons (see Figure 5.13 (a)). Specimens usually have about 100 neurons but more than half are not imaged as they are in different optical planes or close to the borders of the dish. This is a reason to take into account the HCA techniques.

The image set was obtained in two steps. The first step, to obtain the *gold standard* and the second, to generate the patches for the study.

Firstly, the gold standard was obtained by the same method which was explained previously, however the following is a reminder of the aforementioned method. The expert marked the neurons with different-sized regions, fixing each region to the size of the neuron. The majority of neurons were fully contained in regions of a determined size — in this case, 500x500 pixels. Afterwards, the expert marked the neurons again using this same size to mark the regions, where the neurons are located. On the other hand, the areas of the image which did not contain a neuron were declared non-neuron regions.

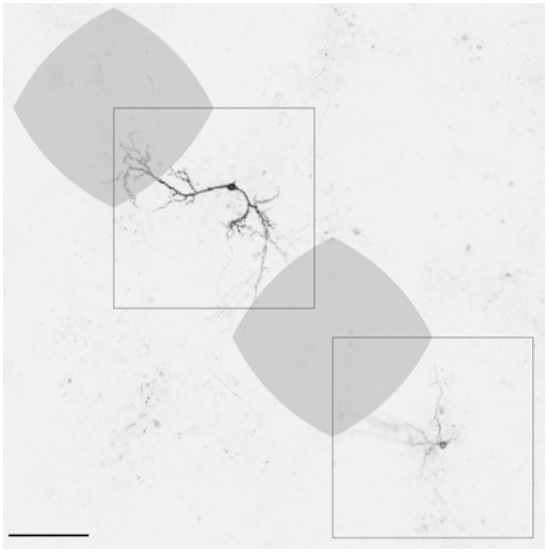


FIGURE 5.15: Example of two neurons. The gray regions represent all possible patches with 50% or more of their area overlapping with the neuron marked by the expert (square). They consist of all the upper-left corners of these patches. Scale bar: 200μm. Image inverted.

The next step consists of obtaining the patches for the study. To this aim, each mosaic is divided using a grid where each region is the same size (a square of 500 pixels of side), and the patches do not overlap but are side-by-side.

In order to be able to use these patches in our study with machine learning approaches, it is necessary to tag the patches. To this end, any patch which overlapped with less than 50% of an annotated neuron patch from the gold standard, was declared a background or non-neuron patch. On the contrary, all patches which overlapped with 50% or more of a positive patch from the gold standard, are considered a neuron or positive patch. As it was mentioned, the expert found approximately 400 neurons in the set of mosaic and the remaining approximately 4,800 patches in the image were negative, resulting in an imbalanced set.

On the other hand, data sets in machine learning techniques can be balanced or imbalanced. Both kinds of sets are suitable, however imbalanced sets usually offer misleading results since the class of interest is much smaller or rarer than the other classes. This fact is common in a lot of the problems analysed with machine learning methods, however, there are different techniques to convert sets from imbalanced to balanced (see Batista, Prati, and Monard, 2004).

We used a balanced set and it was obtained using an oversampling method — adding instances from the under-represented class.

Since a balanced set has to have approximately the same number of samples for both classes. In our case, the negative patches were extracted using the same method to obtain the patch sampling. However, as the positive patches are the class with less samples, it was necessary to extract more patches. To this aim, a number of patches meeting the criteria of positive patches, were extracted for each neuron marked by the expert.

They were obtained randomly from among all the patches which overlapped with more than 50% with the neurons from the gold standard (see Figure 5.15).

This fixed number is calculated by adding up the negative patches of all the mosaics which belong to the set, and dividing this number by the total number of neurons (which belong to the gold standard), see Figure 5.16.

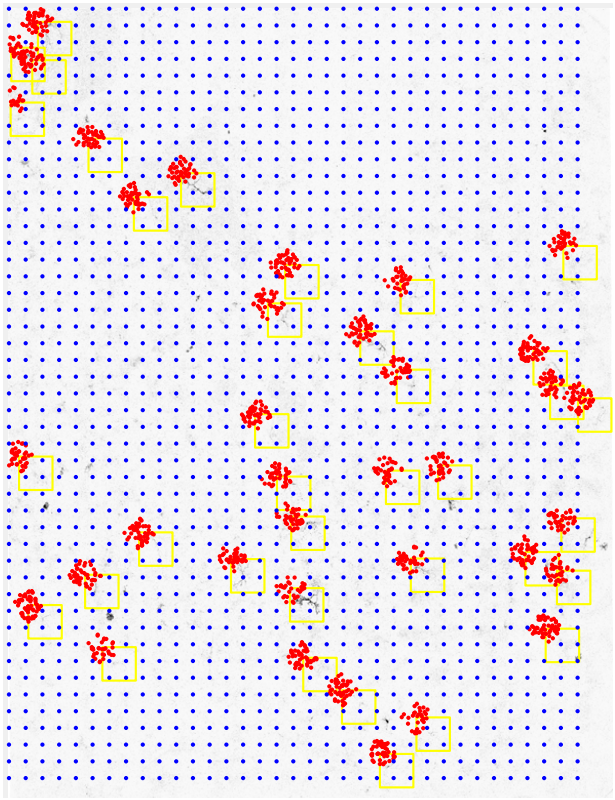


FIGURE 5.16: Example of a mosaic sampling. The yellow square are the neurons marked by the expert and the red dots are the upper-left corner of the patches selected as neuron. Image inverted.

We considered several machine learning techniques based on two ways to extract the features from the images. The first approach was based on computing an extensive list of image features using the *WND-CHARM* library (Shamir, Orlov, Eckley, Macura, Johnston, and Goldberg, 2008). The second one was based on computing the scale-invariant feature transform (SIFT) with the method proposed by Lowe, 2004. Both approaches were explained in detail in the previous section, Section 5.4.2. Although in this case, we only used the first approach, the WND-CHARM tool, to compute the features of the images. We did not use the classifier offered by this software.

The method proposed to extract the set of features in each patch is explained in the following subsection.

Implementation Details

Firstly, the background set of the data set was obtained from creating a grid in the mosaics. The patches which met the criteria to be considered background (their overlapped area with a real neuron is less than 50%) were stored.

Next, the neuron set is created. The neuron patches are chosen randomly from amongst all the possible patches which are classified as neurons. These patches are all of the ones in which 50% of the area or more overlaps with a neuron from the gold standard (see Figure 5.15). However, not all the patches have the same relevance. The higher the percentage of overlapping, the higher the quantity neuron present in the patch.

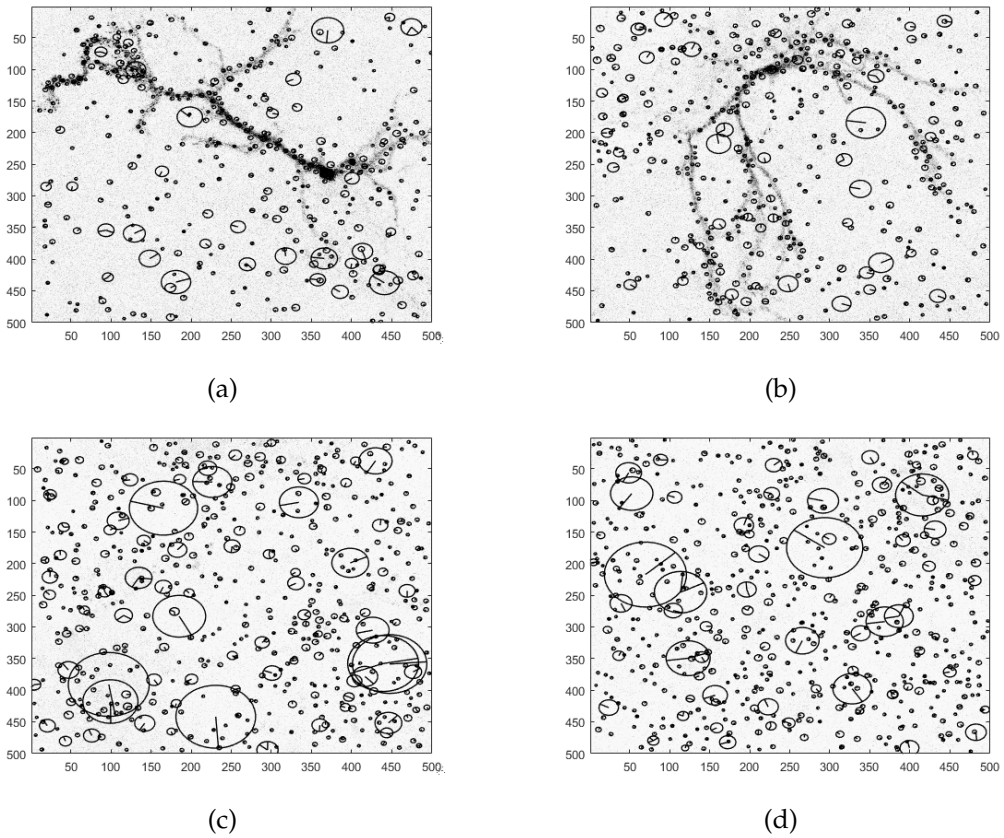


FIGURE 5.17: Example of neuron patches, (a) and (b), and background patches, (c) and (d), with the SIFT-features computed. Images inverted.

Based on this idea, the patches which can be classified as neurons have an associated weight. Taking this into account, the patches which balance the data set are chosen randomly. Once the data set is balanced (that is, there is a similar number of background and neuron patches), the SIFT and CHARM-features are calculated for each of the patches.

The WND-CHARM programme is used to obtain the CHARM-features as it was explained before. The programme returns a file for each patch, which are processed to create a new file with all patches and their features together.

Meanwhile the SIFT-features are calculated too. In this case, to obtain these features we used MATLAB (MathWorks, 2016) and the library called VLFeat (Vedaldi and Fulkerson, 2008), see an example of the SIFT-features extracted in Figure 5.17. Using these tools, the descriptors from each patch are computed, though there does not have to be an identical number for all of them. The algorithm called "Bag of Words" model (BoW model, Sivic, 2009), is used to work with this kind of data and, it explained in the previous section. To obtain the histograms for each patch, different number of clusters, such as 20, 40, 60, 80, 100, 150, 200 and 230, have been studied to this end. As mentioned BoW model computes the centers of the clusters (the centroids) and each descriptor of a patch is classified based on the proximity to the centroids (using k-means algorithm). This process generates a new descriptor for each patch based on the histogram (in this particular case, the histograms have 20, 40, 60, 80, 100, 150, 200 and 230 bins).

		SIFT-features							
data sets	CHARM features	20	40	60	80	100	150	200	230
01	x								
02		x							
03			x						
04				x					
05					x				
06						x			
07							x		
08								x	
09									x
10	x	x							
11	x		x						
12	x			x					
13	x				x				
14	x					x			
15	x						x		
16	x							x	
17	x								x

TABLE 5.8: Features in each data set.

Data sets Details

After obtaining the patches and computing their features, the data sets were created as following. First, the features were considered separately. Consequently, different files were obtained, one of them with only the CHARM-features; and others with only SIFT-features, one data set for each number of cluster. Later, other data sets were created mixing the features - adding the CHARM-features to each combination of the SIFT-features. As a result, there are seventeen data sets, which are in the Table 5.8.

We are currently working on the experimental step of this study. This phase consists of evaluating the data sets using different machine learning algorithms which is partially explained in the subsection below.

5.6.3 Experimental Results

This step is split into four phases:

1. choose the data sets which obtain the best results,
2. evaluate the algorithms with specific data sets to determine the best performance,
3. ascertain which features are the most relevant in this study, and
4. from an experimental analysis determine the best model according to a null hypothesis test.

In order to asses the data sets, four state-of-the-art machine learning algorithms were used: Support Vector Machines (SVM) (Boser, Guyon, and Vapnik, 1992; Vapnik, 1999), Elastic Net (GLMNET) (Zou and Hastiel, 2005), Random Forest (RF)

(Breiman, 2001) and k-Nearest Neighbour (KNN) (Hechenbichler and Schliep, 2006). Those algorithms have different hyperparameters that need to be tuned in order to get the best performance results. In order to assess that the tuning process does not affect to the performance of the models (avoid overfitting) and to ensure that we honestly compare all the algorithms under exactly the same conditions and with the same samples within different experiments and folds, we included in this experimental design a nested resampling.

In order to find the best tuning hyperparameters in the inner resampling loop we perform these experiments following a holdout resampling (two-thirds of the images for training and one-third for testing). In the outer resampling loop of a 10-fold approach, we have ten pairs of training/test sets. On each of these outer training sets in order to perform the parameter tuning we executed the inner resampling loop. Generally speaking, we get one set of selected hyperparameters for each outer training set. Afterwards, the learner is fitted on each outer training set using the best hyperparameters and its performance is evaluated on the outer unknown test sets.

We used the Area Under the Receiver Operating Characteristic curve (AUROC) measure to compare the performance of the algorithms, the tables of the results are in Appendix E and we used R-programme to obtain these results (R Core Team, 2016).

We started our experiments using seventeen different data sets (see Table 5.8) which tried to find which data set works better on different conditions and to observe the behaviour of these data sets on different methods.

The next phase of the experiment consists of a deeper study of the methods of machine learning dealing with smaller data sets.

Based on the results obtained in the previous step (see Appendix E), it is appropriate to study the data set that have only CHARM-features. Besides, it is important to choose the cluster that contains the correct number of SIFT-features in order to choose a data set to continue. Regarding the results, a valid option for the number of clusters is 230. Taking this into account, the data set with SIFT-features grouped by 230 clusters and the data set which contains both kinds of features (CHARM and SIFT-features) are selected too to continue to work with them.

At this point we have three different data sets (CHARM, CHARM_SIFT_230 and SIFT_230) and four machine learning algorithms (SVM, RF, GLMNET and KNN) following our experimental design with three repetitions of an outer 10-fold cross-validation loop and an inner loop for tuning the hyperparameters.

As shown in Figure 5.18, we achieved the best results using SVM and RF with the following data sets SIFT_230 and CHARM_SIFT_230. As mentioned before the number of features for each data set is the following.

	Number of Features
CHARM	1059
CHARM_SIFT_230	1289
SIFT_230	230

TABLE 5.9: Number of features for each data set.

The data set called CHARM_SIFT_230 is the result of the combination of the other two, CHARM and SIFT_230, thus it is of relevance to notice that we achieved the best result in AUROC with the data sets that had a higher number of features, but the data set with the lower number is very close to both of them. At this point, we studied the influence of the features following a Feature Selection (FS) approach.

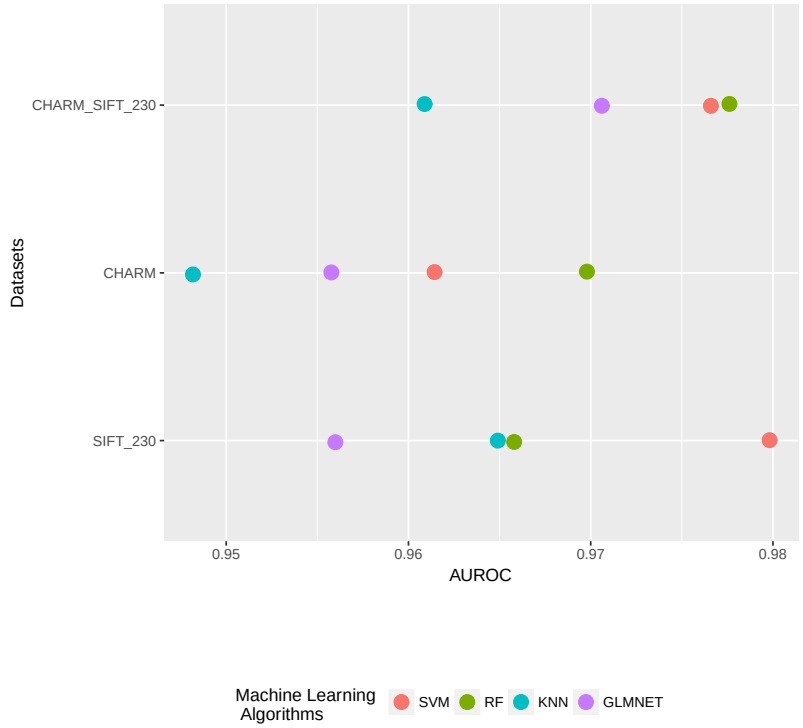


FIGURE 5.18: Mean aggregated error of the algorithms with each one of the data sets.

The third phase of this study was to determine which features are the most relevant. Which of them are necessary for obtaining good results and which features are noise.

To this aim, we analyzed the subsets of features and we realized that there are no CHARM features in the 25 and 100 subsets of features. Furthermore, in the 200 and 600 subsets the number of SIFT features is always higher and with more relevance than the CHARM ones. Each subset of features belongs to the subset with a higher number of features, it means that $subset_{25} \subset subset_{100} \subset subset_{200} \subset subset_{600}$. The features selected for the subsets with 25 and 100 features are only compound for features which belong to the CHARM-features. However, for the other subsets (of 200 and 600 features) have a higher percentage of the SIFT-features, see Table 5.10.

In addition, the features which are representation in the subsets are those which are related with the First 4 Moments (also called Comb Moments in Tables 5.10 and 5.11), however the number of these features regarding to the total of features computed of this kind is lower than others, for instance the features related with the compute in a binary image, Object Statistics (also called Otsu Object Features) are a low percentage in the subset, but three subsets have more than 20% of these features.

Other relevant features are the Haralick Textures which study the co-occurrence matrix and the features which study different scale of histograms (also called Multi-scale Histograms in Tables 5.10 and 5.11).

On the other hand, the SIFT-features are not in the lowest subsets, however, this kind of features is in the other subsets — 200 and 600 — and its presence is high. In fact, the SIFT-features have the highest percentage of features, and the highest total.

	Subsets			
Features	25	100	200	600
Radon Coefficients	12	13	6.5	3.83
Chebyshev Coefficients	4	5	2.5	3.5
Gabor Filters	0	0	0	0
Multiscale Histograms	16	13	11.5	10.67
Comb Moments	28	29	20	28
Tamura Textures	12	4	4.5	2.83
Edge Features	0	4	3	1.5
Otsu Object Features	8	15	9.5	6.5
Zernike Coefficients	0	0	1.5	6.33
Haralick Textures	20	15	8.5	9.5
Chebyshev-Fourier Coeff.	0	2	1.5	3
SIFT-features	0	0	31	24.33

TABLE 5.10: Percentage of the features selected for each subset.

	Subsets			
Features	25	100	200	600
Radon Coefficients	6.25	27.08	27.08	47.92
Chebyshev Coefficients	1.56	7.81	7.81	32.81
Gabor Filters	0	0	0	0
Multiscale Histograms	2.78	9.03	15.97	44.44
Comb Moments	2.43	10.07	13.89	58.33
Tamura Textures	8.33	11.11	25	47.22
Edge Features	0	14.29	21.43	32.14
Otsu Object Features	2.94	22.06	27.94	57.35
Zernike Coefficients	0	0	2.08	26.39
Haralick Textures	2.98	8.93	10.12	33.93
Chebyshev-Fourier Coeff.	0	3.13	4.69	28.13
SIFT-features	0	0	26.96	63.48

TABLE 5.11: Percentage of the features used to the selection regarding to the total of features computed for each kind of features.

The last phase of this experiment consists of selecting the best model according to the a null hypothesis test.

5.6.4 Discussion and Current Work

We are currently working on this last phase. The results show that in the first step, Random Forest (RF) with only SIFT-features using either low or high numbers of words gives the best AUROC and accuracy results. In addition, SIFT-features with the rest of the methods give better results than the others features. In fact, our intuition about that the combination of CHARM and SIFT-features would improve the classification is not true, since only SIFT-features obtained the best results in the first phase.

Additionally, looking at both the AUROC, of all classifiers and all possible data sets, SVM with only SIFT (using the highest numbers of words) obtain the overall best results ($AUC > 0.98$).

However, it is worth remarking that CHARM-features alone never performs best, while SIFT-features alone or the combination formed by CHARM and SIFT-features do, although this depends of the classifier. In fact CHARM-features alone does perform second-best for the Random Forest method.

Therefore, we decided to fix the number of words for the SIFT-features and to analyse better the behavior of the methods and study the data sets called SIFT_230, CHARM and the combination of both, CHARM_SIFT_230-features.

The difference in performance between all four classifiers is smaller with CHARM_SIFT_230 than with SIFT_230 alone or CHARM alone. In order to use this last alone generally does not work as well. However, SIFT_230-features gives better performance, so it was not a surprise to check that CHARM_SIFT_230 obtains better results than CHARM-features alone. SIFT_230 features helps to improve the classifications.

The experimental analysis and the extraction of information obtainable is ongoing.

Chapter 6

Conclusions

This thesis has presented automatic and semi-automatic methods for analysing biomedical images, in particular, images of fluorescence acquired with a confocal microscope.

To begin with, one of the semi-automatic methods was explained in Chapter 2. SynapCountJ is a plug-in for Fiji or ImageJ — platforms for processing images — which allows us to quantify synapses in an efficient and objective way. Given that the images are not always acquired using the same procedures, the quality of the samples may vary due to the intensity of the lasers, the amount of noise, and other factors. This means that in order to control the process, users have to be aware of the optimal settings for the analysis of the images. Once these settings are known, it is possible for the users to automatise the process by fixing the input parameters.

This procedure uses the mathematical concept of connected components. We though the problem about knowing the synaptic density as a problem about counting connected components, and then the process was validated through an indirect approach.

In addition, based on this validation, we established a way to verify code developed in Java and to be used in Fiji and ImageJ. This work was explained in the second part of Chapter 2. While this task was a challenge, there can be no doubt about the significance of the work given the importance of being able to verify that a programme is what it says and does what it is meant to.

Chapter 3 then went on to explain another plug-in used in the analysis of images. However, in this case, the aim was to determine the number of neurons in a mosaic image. This procedure was developed using intensity and geometrical criteria. NucleusJ ascertains the number of total cells and neurons found in an image. Furthermore, the plug-in allows users to edit the result from their own experience.

The work explained in Chapter 4 is a clear example of how there are mathematical concepts which can be applied to the analysis of images. Both methods, based on the persistence of the objects in the image, offer alternatives to recognise the structure of a cell — in particular, neuronal structures. The images studied were acquired in different planes of the Z-plane.

In the case of the homology persistence, the procedure studies the filtrations obtained as result of all the planes. In the second approach, based on Zigzag theory, the structure is obtained studying the "life" of the connected components throughout all planes.

The methods previously explained also offer a solution to the problem of distinguishing parts of cells which have crossed over with different cells; it can then be difficult to differentiate them when using techniques such as segmentation.

In order to complete this study, we reviewed how machine learning techniques help in the analysis of images. In Chapter 5, we described the natural process which

we followed to finish our study using algorithms of machine learning to localise neurons in mosaic images.

The first approach was to use the methods explained in the previous chapters. This problem was dealt with using intensity and geometrical criteria. Although these techniques came close to obtaining a good result, we observed that it was possible to describe patches of the image with features which explained what was happening in each case, for instance, whether a patch contains neuronal structure.

Therefore, we focused on finding the specific features which are the best algorithms to classify these patches.

We are currently working on the final step of this investigation in order to submit it for publication.

It is clear that the ability to process and analyse images is becoming more and more necessary. It is not enough to merely observe - we must be capable of understanding their content.

The rapid advancement of technology today means that analysis techniques should advance accordingly, though techniques of processing images must be verified to assure a correct functionality.

Images show moments which are represented by means of matrices of numbers and there are abundant mathematical algorithms available to help process this information, not to mention mathematical concepts which can be applied to study them.

Moreover, these problems have final users who require tools to facilitate their tasks, making the computer science necessary to convert these ideas to tools.

We believe that this field (the field of biomedical images) is a multidisciplinary area by necessity — no progress would be made otherwise.

Chapter 7

Future Work

The methods presented in this work are approaches to solving some of the problems of the field of biomedical image analysis. However, we want to emphasise that these are not the only solutions. As mentioned during this work, various types of biological samples and microscopy techniques were used to acquire the images of these samples, and thus, this particular field requires adaptability in the methods used to deal with the aforementioned problems.

Further work on this project would include improving the usability of the plug-ins. In addition, particulary for the plug-in explained in Chapter 2, it would be useful to include a post-processing tool to manually edit the obtained results.

In the second part of Chapter 2, we explained some basic notions to be able to verify the code produced, however it is possible to improve the method described. For instance, the transformation from real Java code to Krakatoa could be automated (the steps outlined in Algorithm 4 could be understood as a list of requirements with this aim). Furthermore, a formal study of this transformation could be undertaken to increase the reliability of the method. As for applications, more verification is needed to obtain a certified version of a plug-in such as SynapCountJ. However, these preliminary results allow us to be reasonably optimistic with respect to the feasibility of this objective.

In order to continue the performance of the method described in Chapter 3 must be determined. To this aim, it is necessary to obtain experimental results and to compare them with the data manually obtained until now.

Additionally, we want to test whether the numerical methods used in Diffusion Tensor Imaging for fiber tracking (Dellani and Glaser, 2007; Mori and Zijl, 2002; Weeden and Wang, 2008) could be used to improve the algorithms explained in Chapter 4. On top of that, we plan to automate the heuristics extracted from the experimental study in subsection 4.3.3 in Chapter 4.

However, the current work, and thus the immediate future, consists of continuing the study described in the final paragraphs of Chapter 5, finding the best features for describing neurons in a determined kind of image. To this end it is important to study the combinations of features and compare them with the use of different methods. Determining which neuron detection methods are reliable would greatly improve the efficiency of the biological experiments and this highlights the importance of reducing false positive detections.

Finally, the next step necessary in the biological process is the accurate neuron reconstruction, and the analysis of the complexity of the dendritic arborizations and spine properties.

The final aim is the complete automation of the whole process — using all these methods to obtain a procedure which automatically detects the neuronal morphology and its functionality.

Appendix A

Definitions

Adjacency: two elements A and B are adjacent if the distance between them is less or equal than 1. In the particular case of the pixels: two pixels (x_0, y_0),(x_1, y_1) are adjacent if $|x_0 - x_1| \leq 1 \wedge |y_0 - y_1| \leq 1$.

Artefact: part of the contents of an image that does not have a counterpart in the physical object being imaged; false structures which can appear occasionally in images acquired by microscope (e.g. aliasing artefact, beam hardening artefact, motion artefact, partial volume artefact).

Aspect Ratio: the ratio of the height to the width of a rectangle. It can also be defined as the ratio of the major axis to the minor axis of an ellipse. Objects with a high aspect ratio are elongated.

Centroid: it is the center point of the region which is the average of the x and y coordinates of all of the pixels in the region. In the binary regions, the centroid match the center of mass -first order spatial moments.

Confusion Matrix: it is a special kind of contingency table. It is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class (or vice versa).

Connected component: to understand this term which is typical in Topology, it is necessary to be familiar with the terms explained below (Kaczynski, Mischaikow, and Mrozek, 2004):

Definition A.0.1. Let X be a topological space. X is connected if the only subsets of X that are both open and closed are $\emptyset$ and X. If X is not connected, then it is disconnected.

Definition A.0.2. Definition: A topological space X is a path connected if for every two points x, $y \in X$ there exists a continuous map $f : [a, b] \to X$ such that $f(a) = x$ and $f(b) = y$. Such a map is called a path joining x to y. Any interval is obviously path-connected. A special case of a path connected set is a convex set.

Theorem A.0.1.
Any path-connected space is connected.

Also, intuitively, a connected component is a set made by only one piece, which can not split. Two elements (e.g. pixels) are “connected” if they are adjacent.

DAPI (or 4',6-diamidino-2-phenylindole): a fluorescent stain that binds strongly to Adenine and Thymine rich regions in DNA. It is used extensively in fluorescence microscopy. As DAPI can pass through an intact cell membrane, it can be used to stain both live and fixed cells.

Dendrite: the branched projections of a neuron which propagates the electrochemical stimulation received from other neurons (see Figure A.1).

Feret's diameter: the longest distance between any two points along the selected boundary, also known as maximum caliper.

GLMNET: *Elsatic Net* is a regularization method (Zou and Hastiel, 2005) based on the lasso (penalized least squares method) developed in order to solve its limitations (Tibshirani, 1996). Similar to the lasso, this algorithm does a continuous shrinkage and it could get advantage of groups of correlated features. It linearly combines different penalties of the lasso and ridge methods. Particularly, we fit a generalized linear model via penalized maximum likelihood and using those penalties at a grid of values for regularization parameter (Friedman, Hastie, and Tibshirani, 2010).

Gold Standard: refers to the group of samples which have been evaluated by an expert under reasonable conditions. It is usually used in medicine and statistics. This set of data is used to calibrate and validate a model, algorithm, procedure, etc.

High-Content Analysis (HCA): also known as High-Content Screening (HCS) is the automated extraction and analysis of cellular images taken during high-resolution light microscopy.

Homology group: to understand well what is an homology group, firstly there has to be some mathematical concepts related to it.

Definition A.0.3. A *chain complex* C_* is a family of pairs $(C_n, d_n)_{n\in\mathbb{Z}}$ where $(C_n)_{n\in\mathbb{Z}}$ is a graded module and $(d_n)_{n\in\mathbb{Z}}$ is a differential of C_*.

The module C_n is called the module of *n-chains*. The image $B_n = \operatorname{Im} d_{n+1} \subseteq C_n$ is the (sub)module of *n-boundaries*. The kernel $Z_n = \operatorname{Ker} d_n \subseteq C_n$ is the (sub)module of *n-cycles*.

In many situations the ring R is the integer ring, $R = \mathbb{Z}$. In this case, a chain complex C_* is given by a graded abelian group $(C_n)_{n\in\mathbb{Z}}$ and a graded group morphism of degree -1, $(d_n : C_n \to C_{n-1})_{n\in\mathbb{Z}}$, satisfying $d_{n-1} \circ d_n = 0$ for all n. From now on in this memoir, we will work with $R = \mathbb{Z}$. In fact, one of the most important invariants used in Homological Algebra is the following. Given a chain complex $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$, the identities $d_{n-1} \circ d_n = 0$ mean the inclusion relations $B_n \subseteq Z_n$: every boundary is a cycle (the converse in general is not true). Thus the next definition makes sense.

Definition A.0.4. Let $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$ be a chain complex of R-modules. For each degree $n \in \mathbb{Z}$, the *n-homology module* of C_* is defined as the quotient module

$$H_n(C_*) = \frac{Z_n}{B_n}$$

It is worth noting that the homology groups of a space X are the ones of its associated chain complex $C_*(X)$; the way of constructing the chain complex associated with a space X is explained, for instance, in Maunder, 1996.

In an intuitive sense, homology groups measure "n-dimensional holes" in topological spaces. H_0 measures the number of connected components of a space and H_1 measures the number of the holes of a space. The homology groups H_n measure higher dimensional connectedness. For instance, the n-sphere, S^n, has exactly one n-dimensional hole and no m-dimensional holes if $m \neq n$.

Moreover, it is worth noting that homology groups are an *invariant*, see Maunder, 1996. That is to say, if two topological spaces are homeomorphic, then their homology groups are isomorphic.

Immunostaining: a general term in biochemistry that applies to any use of an antibody-based method to detect a specific protein in a sample (first time described in Coons, Creech, and Jones, 1941). However, immunostaining now encompasses a broad range of techniques used in histology, cell biology, and molecular biology that utilise antibody-based staining methods.

k-Means: popular method for clustering. Its aim is the partition of n observations into k clusters. Each observation forms part of a cluster with the nearest mean or centroid. In order to compute the distance among values to centroids is possible to use different methods of distance. The most popular distance is the *Euclidean distance*.

k-Nearest Neighbours (KNN): it is a non-parametric method used for classification and regression, in pattern recognition. A main property of this algorithm is based on assigning to an unclassified image or datum the classification of the class with highest frequency form the k-most similar images or data, more detailed in Hechenbichler and Schliep, 2006.

Kurtosis: the fourth order spatial moment describes the flatness of a distribution. The order spatial moments are powerful way to describe the spatial distribution of values. The interpretation of this spatial moment is the following.

If its value is:

$= 0$: there is a Gaussian (or 'normal') distribution

< 0 : the distribution is flatter than 'normal'.

> 0 : the distribution is more peaked than 'normal'.

< 1.2 : there is a bimodal (or multimodal) distribution.

Machine Learning: it is a special kind of artificial intelligence (AI) that as Arthur Samuel in 1959 said: "it provides computers with the ability to learn without being explicitly programmed". It explores the study and construction of algorithms which can learn from and make predictions on data. The machine learning task are mainly classified in two categories: Supervised and Unsupervised learning, depending on the nature of the learning.

Making an image binary with ImageJ: it uses a variation of the IsoData algorithm, also known as iterative intermeans (Ridler and Calvard, 1978). This method divides the image into object and background with an initial threshold. The averages of the pixels below and above the threshold are computed. The average of these values are also computed. The threshold is incremented and the process is repeated until the value of threshold is higher than the composite average.

MAP2: it belongs to the Microtubule Associated Proteins (MAPs) family. This protein is expressed only in neuronal cells, therefore the antibodies to MAP2 are excellent markers on this particular kind of cell.

Maximum: the maximum value within the selection.

Mean: the average values. This is the sum of the values of all the pixels in the selection divided by the number of pixels.

Median: the median value of the pixels in the image or selection.

Median Filter: a nonlinear digital filtering. It is normally used to reduce noise in an image. It often does a better job than the mean filter of preserving useful detail in the image. This filter considers each pixel around a main pixel (neighbourhood of a pixel). These neighbours depends of a radio, for instance, if the radio is 1, the neighbourhood is formed by 8 pixels and, if the radio is 2, there are 24 pixels around of the main one. The median is calculated by first sorting all the pixel values from the surrounding neighbourhood into numerical order and then replacing the pixel being considered with the middle pixel value. If the neighbourhood under consideration contains an even number of pixels, the average of the two middle pixel values is used.

Morphological operations: a set of image processing operations that do so based on shapes. In a morphological operation, the value of a pixel in the input image is based on an operation of the corresponding pixel in the input image of its neighbourhood. The most basic morphological operations are erosion and dilation. *Dilating* an object of the image consists of adding pixels to the boundaries of the object while *Eroding* is to remove pixels from object boundaries.

Neuron: a specialized, impulse-conducting cell that is the functional unit of the nervous system. It is consisting of the cell body and its processes, the axon and dendrites (see Figure A.1).

NMDA: (N-Methyl- D- aspartic Acid). Activator of the neuronal NMDA receptor, a subtype of glutamate receptors. Its activation has been implicated in neuronal death by a process call excitoxicity. Its activation is also important in the process of memory consolidation.

Noise: a random variation of brightness or colour information in images, usually an aspect of electronic noise. It is not present in the object imaged. It can be produced by the sensors and circuitry of a digital camera. Image noise is an undesirable by product of image capturing that adds spurious and extraneous information.

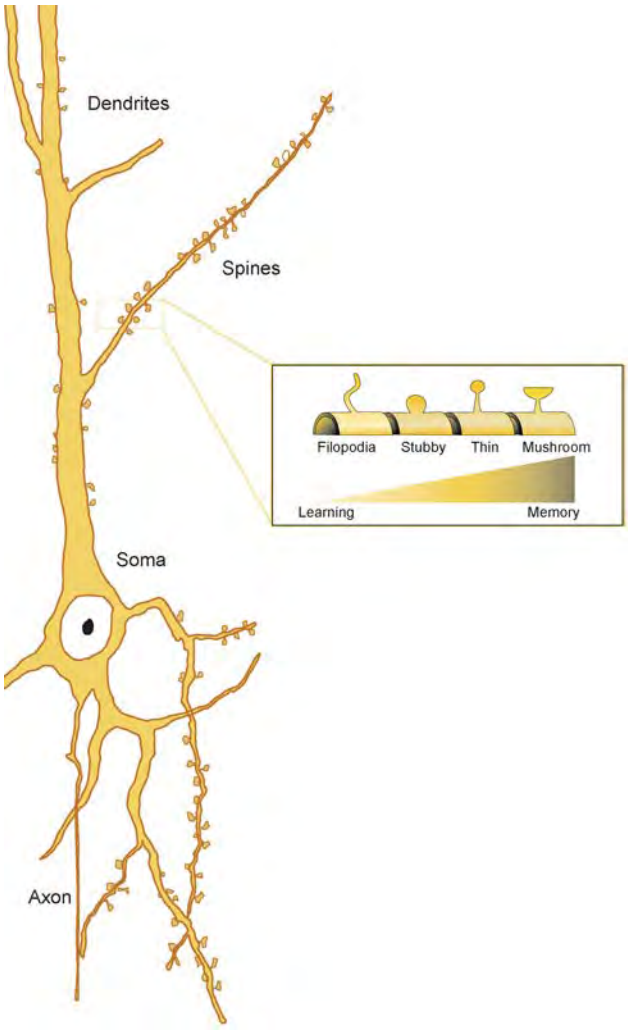


FIGURE A.1: Structure of a pyramidal hippocampal neuron.

PI3K: *Phopho Inositol 3 Kinase*. A kinase which is an enzyme whose activation involves an important physiological regulation in many tissues. In mammals, PI3k is very important for the development of the nervous system and inducing neuronal survival.

Precision: also known as Positive Predictive Value (PPV) is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved. It is obtained from this formula $precision = PPV = \frac{TP}{TP+FP}$, TP and FP being the values of a confusion matrix .

PTD4-PI3K: it is a transduction peptide which can control of PI3K activity. It is used to induce synaptogenesis and spinogeneis in hippocampal neurons. (Cuesto and Enriquez-Barreto, 2011).

Random Forests (RF): Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. More detailed in Breiman, 2001.

Recall: also known as sensitivity or True Positive Rate is the fraction of relevant instances that have been retrieved over total relevant instances in the image. It is obtained through of the values of a confusion matrix using this formula $recall = TPR = \frac{TP}{TP+FN} = \frac{TP}{P}$.

ROC space: a receiver operating characteristics (ROC) graph is a technique for visualizing, organizing and selecting classifiers based on their performance. The graph shows the 'true positive rate' (TPR, recall or sensitivity) against the 'false positive rate' (FPR, fall-out or probability of false alarm). The ROC space is defined by FPR and TPR as x and y axes respectively. Each prediction result or instance of a confusion matrix represents one point in the graph. The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, that means, there are not false positives and not false negatives. This point is also called a 'perfect classification'.

ROI: a region of interest (abbreviated ROI), a selection or subset of a sample within a data set which has a particular purpose.

Skewness: the third order spatial moment measures the symmetry of a distribution. The order spatial moments are a powerful way to describe the spatial distribution of values. The interpretation of this spatial moment is the following.

If its value is:

$= 0$: there is a symmetric distribution

< 0 : the distribution is asymmetric to the left (the tail of the graph extends left of centre of mass).

> 0 : the distribution is asymmetric to the right (the tail of the graph extends right of centre of mass).

Soma: or "cell body" is the bulbous, non-process portion of a neuron or other brain cell type, containing the cell nucleus (see Figure A.1).

Solidity: the measurement of the overall concavity of an object. It is defined as $solidity = \frac{area}{convex\ area}$. The area and the convex hull area approach each other for the solid objects (resulting a value close to one).

Standard Deviation: a measure used to quantify the amount of variation or dispersion of values. If the value of standard deviation is low, it means that the data tend to be close to the mean of the set. On the other hand, a high standard deviation indicates that the data are spread out over a wider range of values.

Supervised Learning: it is a kind of machine learning algorithm that uses a known data set (called the training data set and it usually based on a ground truth) to make predictions. The training data set includes input data and response values. From it, the supervised learning algorithm seeks to build a model that can make predictions of the response values for a new data set. A test data set is often used to validate the model. Using larger training data sets often yield models with higher predictive power that can generalize well for new data sets. There are two categories of Supervised Learning methods: Classification and Regression. Some of the common classification algorithms are the following:

Classification:

- Support Vector Machine (SVM)
- Naïve Bayes
- Decision Trees

Regression:

- Linear regression
- Nonlinear regression

Support Vector Machine (SVM): it is an algorithm which solves problems in classification, regression, and novelty detection. The goal of a SVM is to find the optimal separating hyperplane which maximises the margin of the training data. An important advantage of SVM is that the determination of the model parameters corresponds to a convex optimization problem, and so any local solution is also a global optimum (see Boser, Guyon, and Vapnik, 1992; Vapnik, 1999).

Unsupervised Learning: it is a type of machine learning algorithm. It based on obtaining inferences from data sets consisting of input data without labeled responses. The most common unsupervised learning method is "cluster analysis" and among all algorithms, the most known is k-Means Clustering.

Z-stack: The output of the confocal microscope typically is a stack of images acquired at different locations along the optical axis. In microscopy, this axis is generally labeled as the Z-axis (with the image planes, which are transverse to the optical axis, denoted as X,Y planes). Thus the individual images are said to be taken at different Z-levels, and the collection of such images from a given acquisition is called a *Z-stack* Pawley, 2006.

Appendix B

Biology and Acquisition

B.1 Experimental Phase

Neurons often have elaborate axonal and dendritic arbours which importance in brain function has been recognized since the pioneer work of Cajal, 1998. Dendrites are the neuronal features where most of synapse lay and where synaptic transmission take place and, therefore, the morphology of dendrites reflects connectivity between neurons. Hence, the complexity of neuron morphology contributes to the formation of selective connections within the brain circuits and, consequently, to the establishment of memories. Abnormal dendritic arborization and synaptic density are a recurrent theme in several autistic syndrome disorders, schizophrenia and Alzheimer disease among other neurodegenerative diseases (Enríquez-Barreto and Morales, 2016).

In the context of Alzheimer's disease, hippocampal loss of synaptic connectivity has been well described at early stages of the pathology both in humans and in rodent models of the disease (Selkoe, 2002; Selkoe and Hardy, 2016). In fact, it has been proposed that the distinctive memories losses, during Alzheimer development is a consequence of this defect in synaptic contacts. Therefore the study of the signaling pathways involved in synaptic formation and regulation has therapeutical implications. Previous work of my research group has demonstrated that PI3K activation induce the formation of synapses both in vitro and in vivo (Cuesto, Carames, Cantarero, Gasull, Acebes, and Morales, 2011; Enríquez-Barreto, Cuesto, Domíguez-Iturza, Gavilán, Ruano, Sandi, Fernández-Ruiz, Martín-Vázquez, and Morales, 2014; Cuesto, Jordán-Álvarez, and Enriquez-Barreto, 2015). Therefore, we have proposed that the regulated activation of PI3K can be a putative target for Alzheimer treatment.

Modem confocal microscopy and the wide use of digital acquisition of microscopy images has sparked the use of imaging techniques in biology sciences. One of the aims of this work was to provide to the science community with a set of tools for the acquisition and analysis of neuronal features, helping in the development of therapeutical approaches to treating Alzheimer or other neurodegenerative diseases.

B.1.1 Experimental Methods

The main experimental system employed along this thesis was the in vitro cultures of hippocampal neurons (Kaech and Banker, 2006). In this conditions, hippocampal neurons become appropriately polarised, develop extensive axonal and dendritic arbours and form numerous, functional synaptic connections with one another. Additionally, hippocampal cultures have been used widely for visualising the subcellular localisation of endogenous or expressed proteins, for imaging protein trafficking and

for defining the molecular mechanisms underlying the development of neuronal polarity, dendritic growth and synapse formation.

Therefore all the images employed in this thesis were obtained from cultures of rat hippocampal neurons in culture.

B.1.2 Primary Neuronal Cultures

Primary hippocampal cultures were obtained from P0 rat pups (Sprague-Dawley, strain, Harlan Laboratories Models SL, France). Animals were anesthetized by hypothermia in paper-lined towel over crushed-ice surface during 2-4 minutes and euthanized by decapitation. Animals were handled and maintained in accordance with the Council Directive guidelines 2010/63EU of the European Parliament. Briefly, glass coverslips (12 mm in diameter) were coated with poly-L-lysine and laminin, 100 and 4 μg/ml respectively. Neurons at a 10×104 neurons/cm^2 density were seeded and grown in Neurobasal (Invitrogen, USA) culture medium was supplemented with glutamine 0.5 mM, 50 mg/ml penicillin, 50 units/ml streptomycin, 4% FBS and 4% B27 (Invitrogen, CA, USA), as described before in Cuesto and Enriquez-Barreto, 2011. At days 4, 7 and 14 in culture a 20% of culture medium was replace by fresh medium. Cytosine-D-arabinofuranoside (4 μM) was added to prevent overgrowth of glial cells (day 4) — process as described in Morales, Colicos, and Goda, 2000.

B.1.3 Immunocytochemistry of neurons in cultures

Synaptic Density

Synaptic density on hippocampal cultures was identified as previously described in Cuesto and Enriquez-Barreto, 2011. In short, cultures were rinsed in phosphate buffer saline (PBS) and fixed for 30 min in 4% paraformaldehyde-PBS. Coverslips were incubated overnight in blocking solution with the following antibodies: anti-Bassoon monoclonal mouse antibody (ref. VAM-PS003, Stress Gen, USA) and rabbit polyclonal sera against Synapsin (ref. 2312, Cell Signaling, USA). Samples were incubated with a fluorescence-conjugated secondary antibody in PBS for 30 min. After that, coverslips were washed three times in PBS and mounted using mowiol (all secondary antibodies from Molecular Probes-Invitrogen, USA).

Percentage of synaptic change is the average of different cultures under the same experimental conditions. As a control, we used sister untreated cultures growing in the same 24 well multi plate. Coverslips were stored a 4°C in a dark place until the moment of use.

Dendritic Morphology

Fixation of tissue was performed as described (see above). To visualise dendritic structure a polyclonal antibody against MAP2B (protein specific of dendrites; Cell Signaling reference 4542) was employed. Coverslips were incubated overnight in the antibody solution. Subsequently, samples were washed three times with PBS, and incubated for 30 min in PBS solution containing the fluorescence-conjugated secondary antibodies, washed five times with PBS and mounted in mowiol. Coverslips were stored a 4°C in a dark place until the moment of use.

GFP-transfected Neurons

In this section primary cultures were prepared as described, although, in order to visualise neuronal morphology, neurons were transfected with a plasmid encoding the fluorescence protein GFP (Green Fluorescence Protein), fused with chicken b-actin. The vector expression was under the control of the platelet-derived growth factor promoter region, a neuronal promoter to ensure that Actin-GFP expression was always under the physiological range (Morales, Colicos, and Goda, 2000). Electroporation was performed before plating using a BioRad Cell electroporator system following manufacturer instructions. Briefly, Approximately 4x106 cells and 10 micrograms of plasmid were mixed in BioRad electroporation buffer (BioRad). An exponential discharge protocol with the following parameters was employed: 220 V, 950 mF and resistance fixed to infinite. Neurons were plated immediately after electroporation. At days 4, 7 and 14 in culture, a 20% of culture medium was replaced by fresh medium. Cytosine-D-arabinofuranoside (4 μM) was added to prevent overgrowth of glial cells. After three weeks in vitro, the percentage of transfection varied from 10 to 20%, allowing and easy identification of individual neurons.

Bioloistic Stainning

This section described the experimental procedure employed in Chapter 4. Brain sections came from hippocampus of wild type mice. After dissection and brain fixation, hippocampal neurons from the CA1 region were stained following a biolistic labelling protocol (Enríquez-Barreto, Cuesto, Domíguez-Iturza, Gavilán, Ruano, Sandi, Fernández-Ruiz, Martín-Vázquez, and Morales, 2014). Briefly, tissue slices of 400 microns thick were shot with a blend of tungsten particles (1.7 microns of diameter, Bio-Rad) impregnated in DiI or DiO (lipophilic dies from Molecular Probes, Invitrogen). Shooting was performed using a Helios Gene Gun System (Bio-Rad, USA) following the instructions of the manufacturer. The mix of tungsten bullet randomly impacted over the pyramidal neurons, when in contact with the cellular membrane the lipophilic dye spread along the extracellular membranes, allowing an easy identification of individual neurons by using a fluorescence microscope.

B.2 Acquisition of Images

For the work developed here, we used a Leica SP5 automated confocal microscope for acquiring the images. However, different conditions of acquisition were necessary for each one.

B.2.1 Images for the Analysis of Synaptic Density

The images used in Chapter 2 are stack images which contain the following features:

- pixel size: 0.09 microns
- Z step: 0.5 μm
- the objective was a 40x lens with a 1.3 NA (Numerical Aperture)

Figure B.1 is an example of such an image.

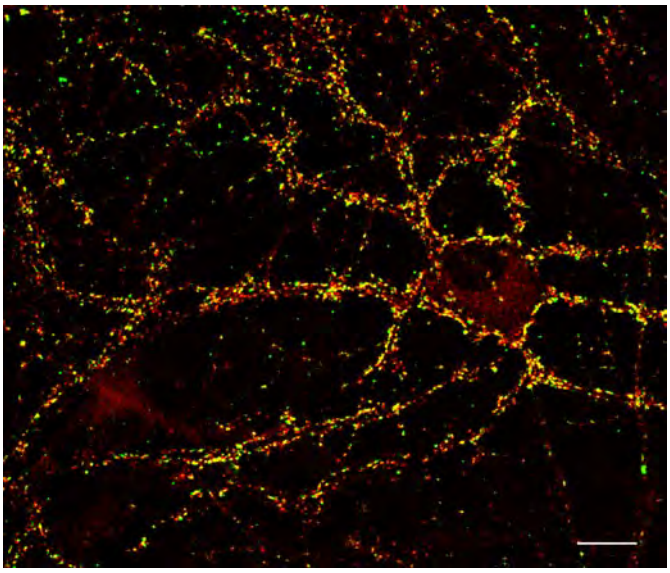


FIGURE B.1: Example of image used for study, described in Chapter 2. This picture shows a neuron marked with the bassoon antibody marker (green channel) and the synapsin antibody marker (red channel). Scale bar: 50μm.

B.2.2 Images for the Analysis of the Immunocytochemistry of Neurons in Cultures

The images used in the study described in Chapter 3 were obtained using the Matrix modules of Leica's microscope - Figure B.2. The acquisition protocol was divided into two parts: an initial pre-autofocus test, followed by the acquisition itself.

- Autofocus job: Contrast Based Method 1, format is 512x512px, speed of 40 Hz bidirectional and a capture length of 30μm; number of steps: 12 steps.
- Sequential job: two channels (DAPI and MAP2B), images format: 2048x2048px with a zoom of 1.7 and speed of 700 Hz and Airy1 of pinhole.

Each image was approximately 14,000 $\times$ 9,000 pixels (covering $\approx 8\,\text{mm}^2$ of the culture dish). These images were acquired with the following settings:

- pixel size: 0.07 microns
- the objective was 20x dry

B.2.3 Images for the Analysis of the Structure of GFP-Transfected Neurons

Two kinds of images were used in Chapter 4 and although they came from two different samples, the acquisition was the same. The images consist of Z-stacks, with the number of slices depending on the thickness of the sample. These images were acquired using:

- pixel size: 0.06 microns
- Z step: 1.01 μm

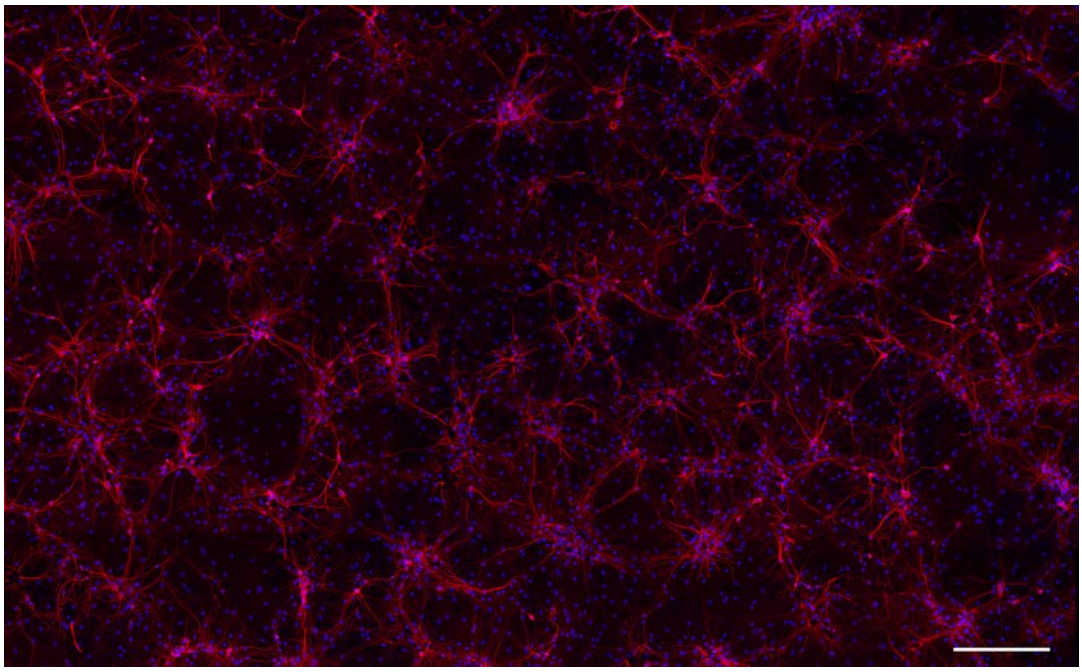
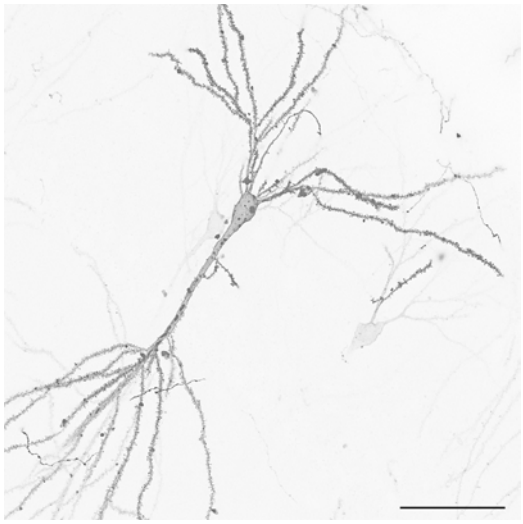


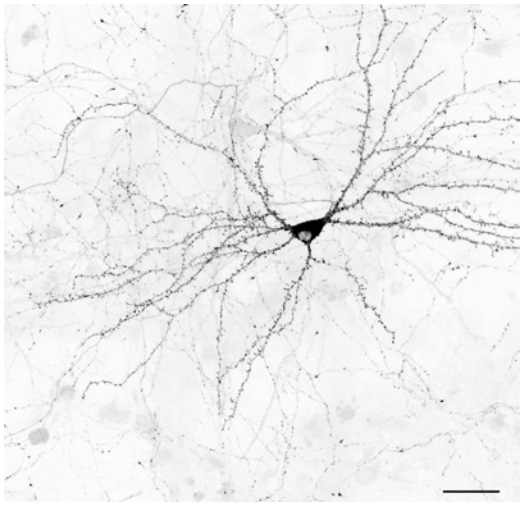
FIGURE B.2: Example of image used for study, described in Chapter 3. This picture shows a whole mosaic where the neurons are in the red channel and the nuclei are in the blue channel. Scale bar: 200μm.

- the objective was a 63x oil-immersion

For instance, Figure B.3 shows the maximal intensity projection of a Z-stack composed of 15 optical planes, each one 1.79 microns thick. This neuron was stained with a DiI biolistic protocol.



(a) Sample neuron stained with a DiI biolistic protocol.



(b) Sample of a transfected neuron with GFP.

FIGURE B.3: Example of images used to the study described in the Chapter 4. Scale bar: 50μm. Images inverted.

B.2.4 Images for the Analysis of the GFP-Transfected Neurons

In this case, the sample for acquiring the pictures is the same as the type of image used to study the structure in the Chapter 4 - transfected neurons with GFP. However, the study described in the Chapter 5 involved images acquired using the Matrix modules. Each image was approximately 10,000 $\times$ 12,000 pixels (covering $\approx 70\,\text{mm}^2$ of the culture dish) and contained an average of 40 transfected neurons (using the fluorescent protein, GFP), see Figure B.4.

These images were acquired with the following settings:

- pixel size: 0.07 microns
- the objective was 20x dry

FIGURE B.4: Example of a whole mosaic used for study, described in the Chapter 5. Image with transfected neurons. Scale bar: 500μm. Image inverted.

Appendix C

Technology

- ACL2

A Computational Logic for Applicative Common Lisp (ACL2) is a logic and programming language with which is possible to model computer systems and prove the properties of those models. It is designed to support automated reasoning in inductive logical theories, mostly for the purpose of software and hardware verification. ACL2 is free and open source software under a BSD license.
See Kaufmann and Moore, 2012.

- Coq

It is an interactive theorem prover. If it is seen as a programming language, then implements a dependently typed functional programming language, while if it is seen as a logical system, it implements a higher-order type theory. The development of Coq has been supported by INRIA with the collaboration of others (for instance, École Polytechnique, CNRS, ...).
See COQ development team, 2012.

- ImageJ

A public domain Java image processing program inspired by National Institutes of Health. NIH and ImageJ have been pioneers as open tools for the analysis of scientific images. It runs, either as an online applet or as a downloadable application, on any computer with a Java 1.4 or later virtual machine. Downloadable distributions are available for Windows, Mac OS, Mac OS X and Linux. ImageJ was designed with an open architecture that provides extensibility via Java plug-ins and recordable macros.

These features make this programme is a useful tool for biological image processing and analysis.
See Schneider, Rasband, and Eliceiri, 2012.

- Fiji

Fiji Is Just ImageJ is an open source image processing package based on ImageJ. in fact, it provides a distribution of ImageJ with many bundled plug-ins. Fiji is maintained by Curtis Rueden and the ImageJ development team at the Laboratory for Optical and Computational Instrumentation (LOCI) at the University of Wisconsin-Madison. It is licensed under the GNU General Public License.
See Schindelin, Arganda-Carreras, and Frise, 2012.

- Kenzo

A Common Lisp programme (Graham, 1996) devoted to Symbolic Computation in Algebraic Topology that was developed by Francis Sergeraert and some coworkers. Algebraic Topology is a vast and complex subject, in particular mixing Algebra and (combinatorial) Topology. The fundamental idea of the Kenzo system is the notion of object with effective homology combined with functional programming.
See Dousson, Rubio, Sergeraert, and Siret, 1999.

- Krakatoa

A verification tool for Java programs. The Java Modeling Language (JML) has been designed to formally specify the behaviour of Java programmes. This tool allows one to certify that JML-annotated method of a Java programme meets its specifications (especially, pre/post-conditions and class invariants). Krakatoa involves distinct components: the *Why* tool, which computes proof obligations for a core imperative language annotated with pre and post conditions, and *Coq* proof assistant for modelisation of specifications and development of proofs.
See Filliâtre and Marché, 2007.

- Matlab

MATrix LABoratory is a proprietary programming language developed by MathWorks Inc. (American privately held corporation). MATLAB allows matrix manipulations, implementations of algorithms among other functionalities. It is popular amongst scientists involved in image processing and in machine learning recently.
See MathWorks, 2016.

- Weka

A collection of machine learning algorithms for data mining tasks, developed at the University of Waikato, New Zealand. Weka is licensed under the GNU General Public License. It is implemented in the Java programming language and thus, it runs on almost any computing platform.
See Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten, 2009.

- Why

A software verification platform which contains several tools, as Krakatoa which is for the verification of Java programmes. This platform is used as a back-end by other verification tools although, it can also used directly to verify programmes. One of the main features of Why is to be integrated with existing provers (such as Coq, CVC3 and more)
See Bobot, Filliâtre, Marché, Melquiond, and Paskevich, 2015.

- WND-CHARM

A generalized pattern recognition system for images developed by the Goldberg group at the NIH/NIA. This tool extracts a large set of image features including polynomial decompositions, high contrast features, pixel statistics, and textures. The feature values are then used to classify test images into a set of pre-defined image classes. The classifier, called Weighted Neighbour Distances (WND), was tested on several different problems including biological image classification and face recognition.
See Shamir, Orlov, Eckley, Macura, Johnston, and Goldberg, 2008.

- R

An open source programming language and software environment for statistical computing and graphics. It is supported by the R Foundation for Statistical Computing. This language is common among statisticians and data miners for developing statistical software and data analysis, for instance data analysis which were obtained using algorithms of machine learning. R is a GNU package.
See R Core Team, 2016.

Appendix D

Validation process of the plug-in NeuronZigzagJ

In order to validate the plug-in NeuronZigzagJ (Chapter 4, Section 4.3, we have undertaken two experimental studies. In the first experience, the aim was to investigate the role of the maximal projection ($\sqcup$) and the union ($\cup$) strategies (with the corresponding filter and thresholding methods). In the second experiment, a more systematic study was carried out to estimate the accuracy of our program with respect to the observations of some human experts. The files containing the images we have worked with are available at `http://www.unirioja.es/cu/gamata/repository-of-images.html`, in order to allow other researchers to reproduce our programming experiments.

D.1 First experiment

First of all, we have considered a series of actual neurophysiology images and we have applied the plug-in using both the *maximal projection* and the *union* strategies (with the corresponding filter and thresholding methods). Concretely, we have applied our plug-in starting from 1146 slices (coming from 12 GFP-Actin stacks of images, and 11 DiI stacks).

The results of the experimental study are collected in the table of Table D.1. The description of the columns in the table is as follows. Column 1 contains an identification number, and column 2 is the name of the file containing each stack. Column 3 indicates the number of slices in each stack. Data of column 4 are provided by a human, who inspected the relevant number of connected components (i.e., significant fragments of dendrites) in each stack; it is the column of quality control of the table. Then, each row is divided into two sub-rows, depending on the kind of operation used to compute the zigzag persistence: the union $\cup$ or the maximal projection $\sqcup$ (let us stress that each operation comes with its own filtering and thresholding methods, as explained before). Finally, the last column indicates the slices from which it is necessary start from to get some of the components (in brackets the number of connected components reached from each slice).

The conclusions that we can extract from these experimental data are the following.

1. An examination of table in Table D.1 shows the "right" slice to start from is not always located at the same place in the stack (in principle, our conjecture was that the intermediate slice would contain most of the relevant information); this justifies our decision of letting the user choose the initial slice.

2. In DiI images, in 40% of cases there is not a unique starting slice allowing recovering all the information (because dendrites can sprout at distant layers in the stack).

3. Our decision of treating differently GFP-Actin and DiI images is supported by the experiments, since in the former case in a 33% of cases no information is obtained with the union $\cup$ operation, while in the latter case both operations behave similarly, and $\cup$ is preferred because it is cheaper to be computed.

4. To reinforce the previous conclusion, it was checked (but we are not able to reflect it in the table), that the chosen combination filtering-operation-thresholding produced a more accurate segmentation (other combinations were tested, but getting always worst results; in any case, let us recall that all these parameters can be fixed in the user interface, easing the reproduction of our computer experiments).

5. Information in the graphical and barcode outputs is complementary, without obtaining one as a by-product of the other one.

The last observation is specially interesting, because our preconception was that the barcode information would be simply a visual aid to a better understanding of the graphical output. This impression was biased because we underestimate the importance of the choosing of the starting slice. These experiments suggest a way of working with our plug-in:

1. First, focus on the barcode diagram, looking for long bars and also considering when two longs bars are not intersecting in the same column (in this second case, it implies that no starting slice could produce all the relevant homological information).

2. Second, produce the corresponding graphical outputs starting from one or from several slices, determined by the barcode examination.

D.2 Second experiment

In a second experiment, we have selected 60 images (30 GFP images and 30 Dil images). They has been randomly divided into three blocks of 30 images, and each block has been assigned to a researcher to make a manual analysis. In that way, we got that some images were analyzed by more than one human, trying to measure the influence of subjective behaviour in the study. Two of the observers were biologists, and the third one a computer scientist (to also control in this way possible biases with respect to the initial training of the testers).

For each image, the human observer had to annotate the number of crossings, the number of connected components, and also the exact location of each crossing and of each dendrite. In addition, a free text box was included where the observer could write some comments about his/her interpretation (quality of the image, ambiguities and so on). Tables D.2 and D.3 include a summary of the results of the three observers. We also include in Tables D.4, D.5 and D.6 the individual studies.

The result of the crossover study was clear: there was not any relevant discrepancy among the interpretations of the three observers, when looking at a same image. Even the free comments were quite similar in each case. This increases the reliability of the aggregated results obtained in the final table.

Actin-GFP	Title Image	No of Slices	Main Components	Operation	Slices which show components
1	Actin01	4	1	$\bigcup$	None
				$\bigsqcup$	all(1)
2	Actin02	6	1	$\bigcup$	None
				$\bigsqcup$	5,6(1)
3	Actin03	4	2	$\bigcup$	3,4(2)
				$\bigsqcup$	3,4(2)
4	Actin04	5	1	$\bigcup$	4,5(1)
				$\bigsqcup$	all(1)
5	Actin05	8	1	$\bigcup$	all(1)
				$\bigsqcup$	all(1)
6	Actin06	4	2	$\bigcup$	all(2)
				$\bigsqcup$	all(2)
7	Actin07	4	1	$\bigcup$	2(1)
				$\bigsqcup$	all(1)
8	Actin08	5	1	$\bigcup$	all(1)
				$\bigsqcup$	all(1)
9	Actin09	3	1	$\bigcup$	all(1)
				$\bigsqcup$	all(1)
10	Actin10	5	1	$\bigcup$	None
				$\bigsqcup$	1(1)
11	Actin11	5	1	$\bigcup$	all(1)
				$\bigsqcup$	all(1)
12	Actin12	5	2	$\bigcup$	4,5(2)
				$\bigsqcup$	4,5(2)
DiI	Title Image	No of Slices	Main Components	Operation	Slices which show components
1	R3N2S2C3	26	2	$\bigcup$	7-12(1)/14-23(1)
				$\bigsqcup$	7-12(1)/14-23(1)
2	R3N2S2C5	26	2	$\bigcup$	4-14(1)/17-26(1)
				$\bigsqcup$	4-14(1)/17-26(1)
3	R3PbR1N1C3	40	3	$\bigcup$	1-9(2)/14(1)
				$\bigsqcup$	1-8(2)/None
4	R3PbR1N1C4	59	3	$\bigcup$	4-9(2)/11-23(1)/24-31(1)/35-59(1)
				$\bigsqcup$	4-11(2)/18-23(1)/24-31(1)/35-59(1)
5	R3PbR1N1C5	59	3	$\bigcup$	11,12(3)
				$\bigsqcup$	11,12,13(3)
6	R3PbR1N1C6	59	1	$\bigcup$	18-43(1)
				$\bigsqcup$	18-43(1)
7	R3PbR1N1C7	59	2	$\bigcup$	36-44(2)
				$\bigsqcup$	39-47(2)
8	R3PbR1N1C8	59	4	$\bigcup$	17-18(3)/ 30-33(1)
				$\bigsqcup$	16-17(3)/27-32(1)
9	R1S8C5	44	2	$\bigcup$	36(2)
				$\bigsqcup$	35,36(2)
10	R1S8C2	40	1	$\bigcup$	11-34(1)
				$\bigsqcup$	11-34(1)
11	R1S8C4	44	2	$\bigcup$	28-38(2)
				$\bigsqcup$	27(2)

TABLE D.1: Results of first experiment.

The success of the plug-in was remarkable with respect to GFP images (90% of hits) and reasonable with respect to Dil images (77.6% of hits). Here *hit* means that the plug-in found the same results than the human observers, up to some small ambiguity present in the image. If we consider *full hits*, that is to say, exact equality with respect to the four measured features (number of crossing, number of dendrites, and location of both), the figures are 86% for GFP images, and 66% for Dil images. The poorer performance for Dil images is explained because each image has around 45 slices; then the human eye did not perceive all the intricacies contained in the image.

Title Image	Tester 1	Tester 2	Tester 3	Conclusion
DiI01	OK	OK	OK	1
DiI02	OK		OK	1
DiI03			OK	1
DiI04	It does not fit all			0
DiI05	It does not fit all	There are more dendrites in the PI because they are in some slice with low intensity		0
DiI06	OK			1
DiI07	It counts one dendrite as 2 but is just a ramification	OK		1
DiI08	There are too many dendrites superpossed, it is very difficult even by eye	There are more dendrites in the PI because they are in some slice with low intensity		0
DiI09	There is one piece that corresponds to a dendrite after the crossing but it is recognized as part of the other one			0
DiI10	OK	OK	OK	1
DiI11			The plug-in finds one dendrite that the human does not	1
DiI12	OK		The plug-in finds one dendrite that the human does not	1
DiI13	OK	OK		1
DiI14	Very difficult to say as there are a lot and they all cross (not even sure that there are 18) The program counts a lot of them as the same in white			0
DiI15			OK	1
DiI16	OK			1
DiI17	OK	OK		1
DiI18		It does not recognize one dendrite		0
DiI19		OK		1
DiI20		OK		1
DiI21		OK		1
DiI22			OK	1
DiI23		There are more dendrites in the PI because they are in some slice with low intensity		0
DiI24			OK	1
DiI25			OK	1
DiI26			The plug-in detects a crossing that seems to be real although it is not seen in the maximal projection	1
DiI27			OK	1
DiI28			The plug-in detects a crossing that seems to be real although it is not seen in the maximal projection	1
DiI29			OK	1
DiI30			OK, it is an axon	1
			Total	23

TABLE D.2: Summary of second experiment - DiI images.

Title Image	Tester 1	Tester 2	Tester 3	Conclusion
GFP01	OK, but a piece of dendrite from one of them is missing in the plugging result	Loses half dendrite of one of them		1
GFP02			OK	1
GFP03	OK			1
GFP04	OK	OK		1
GFP05	They all came from the same soma, so it finds them persistent from there but they are 3 diferent dendrites		OK	1
GFP06	OK	OK	OK	1
GFP07	OK		OK	1
GFP08		OK		1
GFP09	It has desconnected one dendrite	Loses half dendrite (it has a disconection point)	The plug-in sees a dendrite which does not exist	0
GFP10	It loses the smallest dendrite		OK	1
GFP11		OK		1
GFP12	OK, one of them is noise			1
GFP13		OK (this image has a soma, and it is recognized too)		1
GFP14	OK	OK	OK	1
GFP15		OK		1
GFP16	OK, but they are probably a bit connected but they are 2 I would say	OK	OK, there are three of them but they are in the same cell, perfect	1
GFP17	It misses some pices		It loses one dendrite	0
GFP18	Join both dendrites			1
GFP19	OK	OK (a few noise but it is ok)	OK	1
GFP20	OK			1
GFP21		OK		1
GFP22			OK	1
GFP23		OK		1
GFP24			OK	1
GFP25			OK	1
GFP26		There are more dendrites in the PI because they are in some slice with low intensity		0
GFP27			OK	1
GFP28		OK		1
GFP29		OK	OK	1
GFP30		OK		1
			Total	27

TABLE D.3: Summary of second experiment - GFP images.

	Kind of stained		Original image(OI)		Processed image(PI)		Comparative OI vs. PI	
Title Image	DiI	GFP	Dendrites	Crossings	Dendrites	Crossings	Dendrites which coincide	Crossings which coincide
DiI01	1		2	1	2	1	2	1
DiI02	1		2	0	2	0	2	0
DiI04	1		5	3	3	3	2	3
DiI05	1		4	4	3	2	3	2
DiI06	1		3	0	3	0	3	0
DiI07	1		4	1	5	3	4	2
DiI08	1		10	3	8	2	8	2
DiI09	1		2	1	2	1	2	1
DiI10	1		2	1	2	1	2	1
DiI12	1		1	0	1	0	1	0
DiI13	1		2	0	2	0	2	0
DiI14	1		18	5	10	2	10	2
DiI16	1		2	0	2	0	2	0
DiI17	1		1	0	1	0	1	0
GFP01		1	2	0	2	0	2	0
GFP03		1	1	0	1	0	1	0
GFP04		1	2	0	2	0	2	0
GFP05		1	3	1	1	0	1	0
GFP06		1	1	0	1	0	1	0
GFP07		1	1	0	1	0	1	0
GFP09		1	1	0	2	0	1	0
GFP10		1	2	0	2	0	1	0
GFP12		1	1	0	2	0	1	0
GFP14		1	2	0	2	0	2	0
GFP16		1	2	0	1	0	1	0
GFP17		1	1	0	1	0	1	0
GFP18		1	2	1	1	0	1	0
GFP19		1	2	0	2	0	2	0
GFP20		1	1	0	1	0	1	0

TABLE D.4: Results of second experiment - first observer.

	Kind of stained		Original image(OI)		Processed image(PI)		Comparative OI vs. PI	
Title Image	DiI	GFP	Dendrites	Crossings	Dendrites	Crossings	Dendrites which coincide	Crossings which coincide
DiI01	1		2	1	2	1	2	1
DiI05	1		2	1	4	2	2	1
DiI07	1		3	0	3	0	3	0
DiI08	1		4	0	6	0	6	0
DiI10	1		2	1	2	1	2	1
DiI13	1		2	0	2	0	2	0
DiI17	1		2	0	2	0	2	0
DiI18	1		3	0	2	0	2	0
DiI19	1		1	0	1	0	1	0
DiI20	1		2	0	2	0	2	0
DiI21	1		2	0	2	0	2	0
DiI23	1		1	0	2	0	1	0
GFP01		1	2	0	2	0	2	0
GFP04		1	2	0	2	0	2	0
GFP06		1	1	0	1	0	1	0
GFP08		1	1	0	1	0	1	0
GFP09		1	1	0	1	0	1	0
GFP11		1	1	0	1	0	1	0
GFP13		1	2	0	2	0	2	0
GFP14		1	2	0	2	0	2	0
GFP15		1	2	0	2	0	2	0
GFP16		1	2	0	2	0	2	0
GFP19		1	1	0	1	0	1	0
GFP21		1	1	0	1	0	1	0
GFP23		1	2	0	2	0	2	0
GFP26		1	1	0	2	0	1	0
GFP28		1	2	0	2	0	2	0
GFP29		1	3	0	3	0	3	0
GFP30		1	1	0	1	0	1	0

TABLE D.5: Results of second experiment - second observer.

	Kind of stained		Original image(OI)		Processed image(PI)		Comparative OI vs. PI	
Title Image	DiI	GFP	Dendrites	Crossings	Dendrites	Crossings	Dendrites which coincide	Crossings which coincide
DiI01	1		3	1	3	1	3	1
DiI02	1		2	0	2	0	2	0
DiI03	1		3	1	3	1	3	1
DiI10	1		2	1	2	1	2	1
DiI11	1		1	0	1	0	1	0
DiI12	1		1	0	2	1	1	0
DiI15	1		1	0	1	0	1	0
DiI22	1		2	0	2	0	2	0
DiI24	1		2	0	2	0	2	0
DiI25	1		1	0	1	0	1	0
DiI26	1		4	0	5	1	4	0
DiI27	1		2	0	2	0	2	0
DiI28	1		2	0	3	0	2	0
DiI29	1		2	0	2	0	2	0
DiI30	1		2	1	3	1	2	1
GFP02		1	3	0	3	0	3	0
GFP05		1	3	0	3	0	3	0
GFP06		1	1	0	1	0	1	0
GFP07		1	3	1	3	1	3	1
GFP09		1	1	0	2	0	1	0
GFP10		1	2	0	2	0	2	0
GFP14		1	2	0	2	0	2	0
GFP16		1	3	0	3	0	3	0
GFP17		1	2	0	1	0	1	0
GFP19		1	1	0	1	0	1	0
GFP22		1	1	0	1	0	1	0
GFP24		1	2	0	2	0	2	0
GFP25		1	1	0	1	0	1	0
GFP27		1	1	0	1	0	1	0
GFP29		1	3	0	3	0	3	0

TABLE D.6: Results of second experiment - third observer.

Appendix E

Results of Machine Learning Experiments

The following tables are the results obtained after running the methods explained in Subsection 5.6.3 in Chapter 5.

There are two tables for each algorithm (GLMNET, KNN, RF and SVM) and they show the results according the AUROC obtained using the R-package, see R Core Team, 2016.

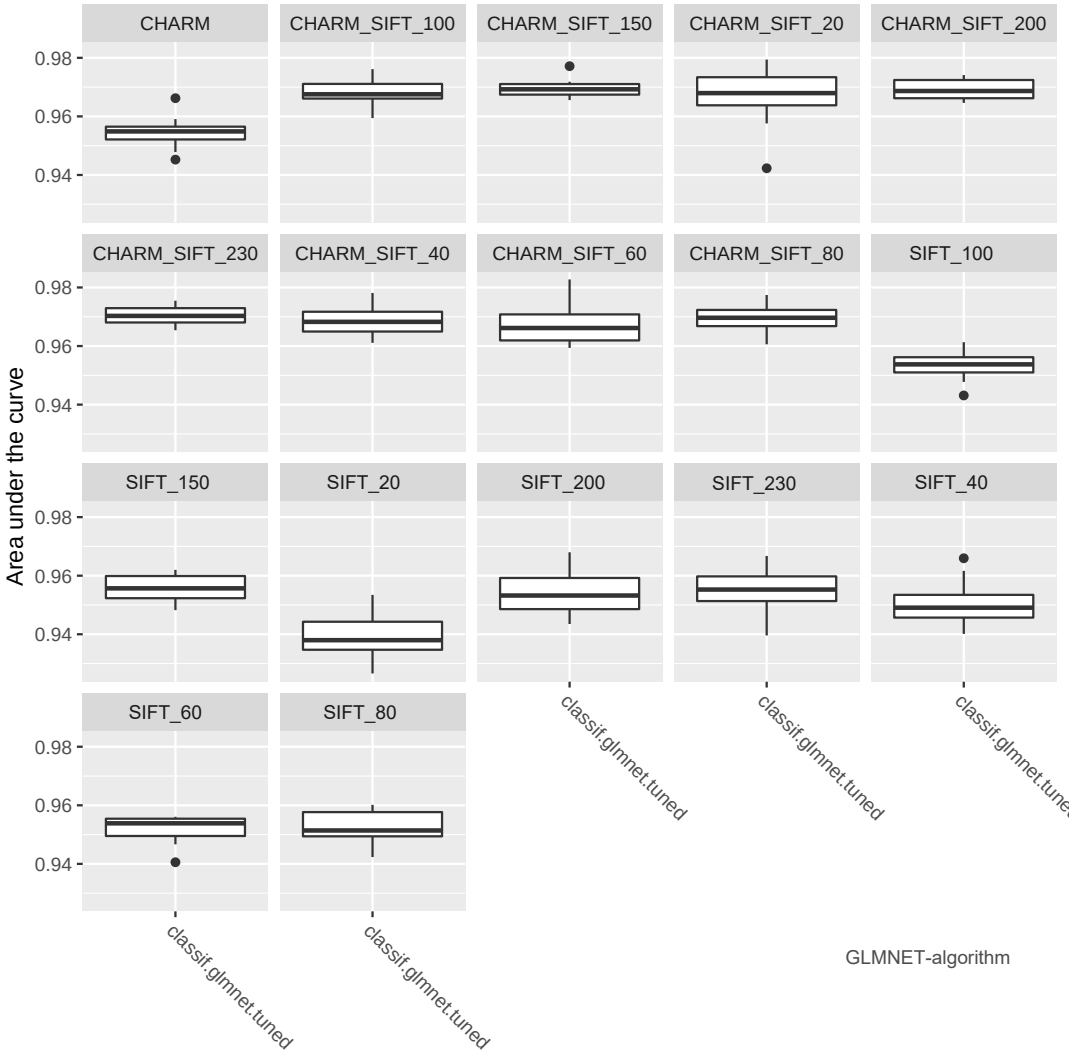


TABLE E.1: Table of GLMNET results.

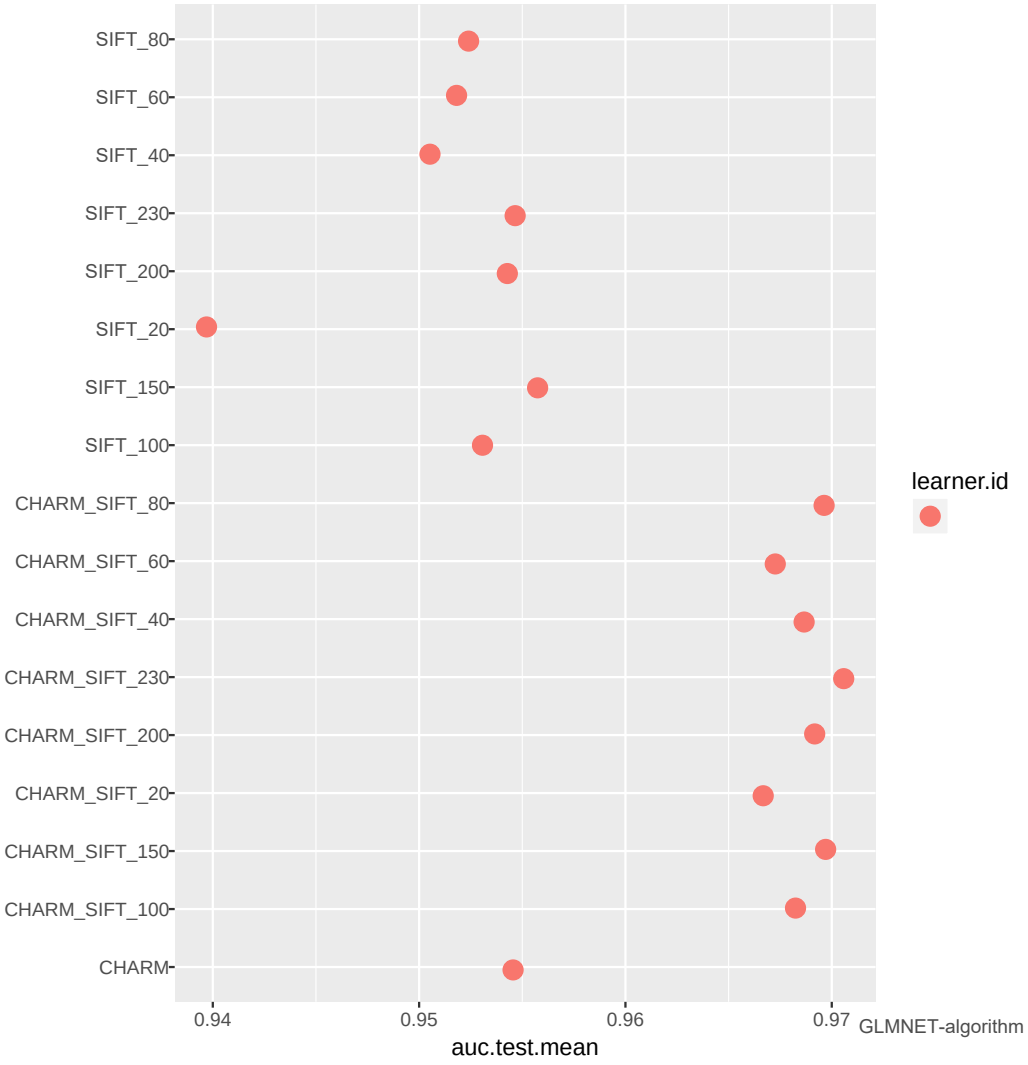


TABLE E.2: Table of GLMNET results.

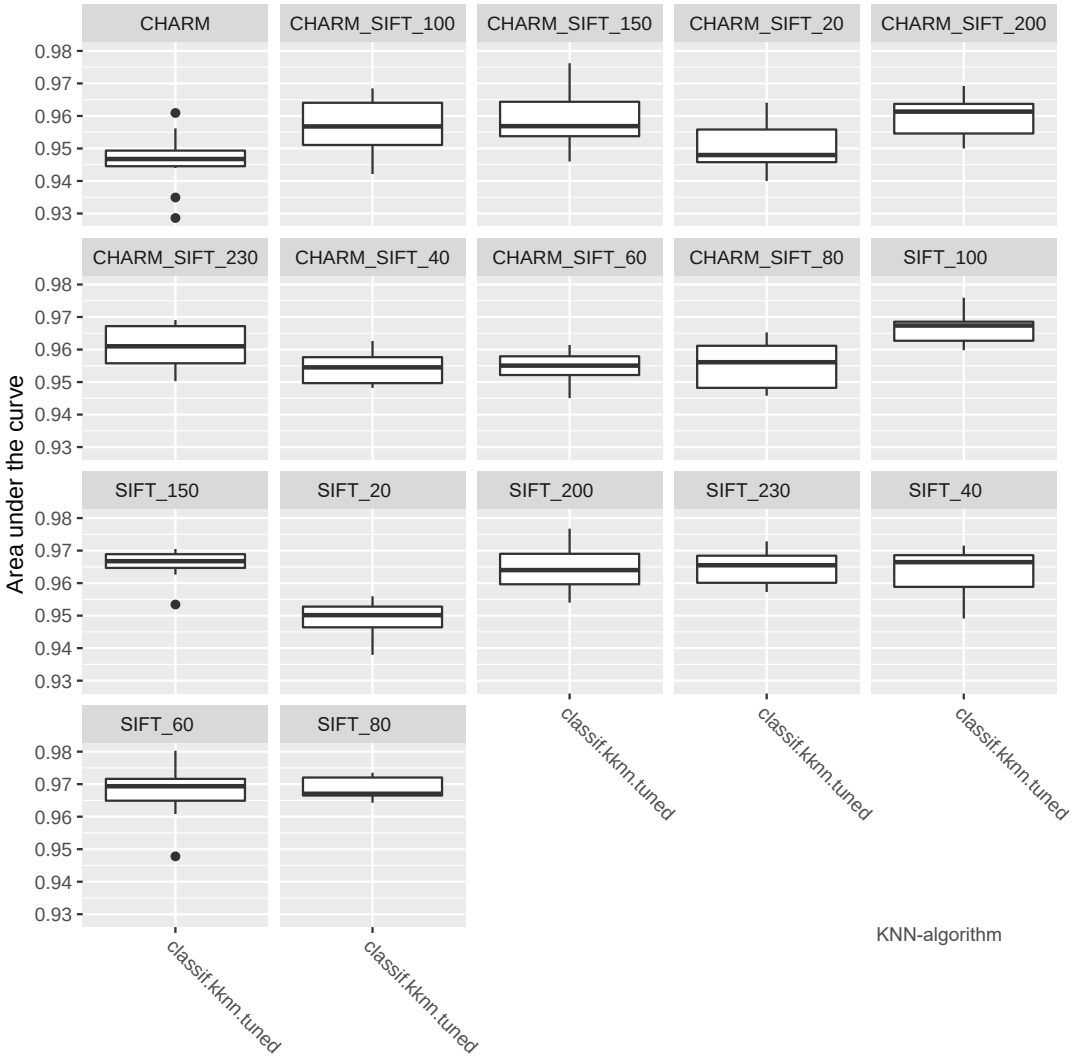


TABLE E.3: Table of KNN results.

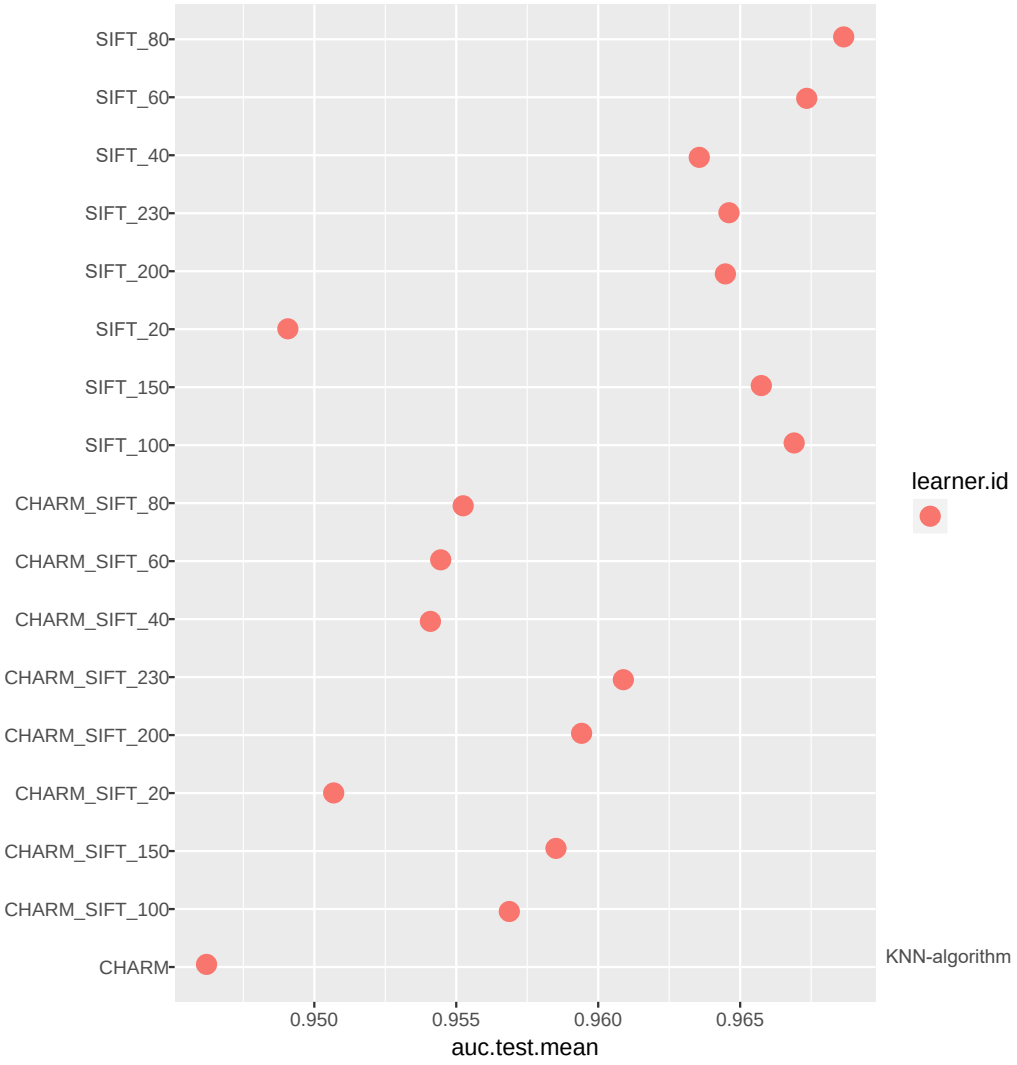


TABLE E.4: Table of KNN results.

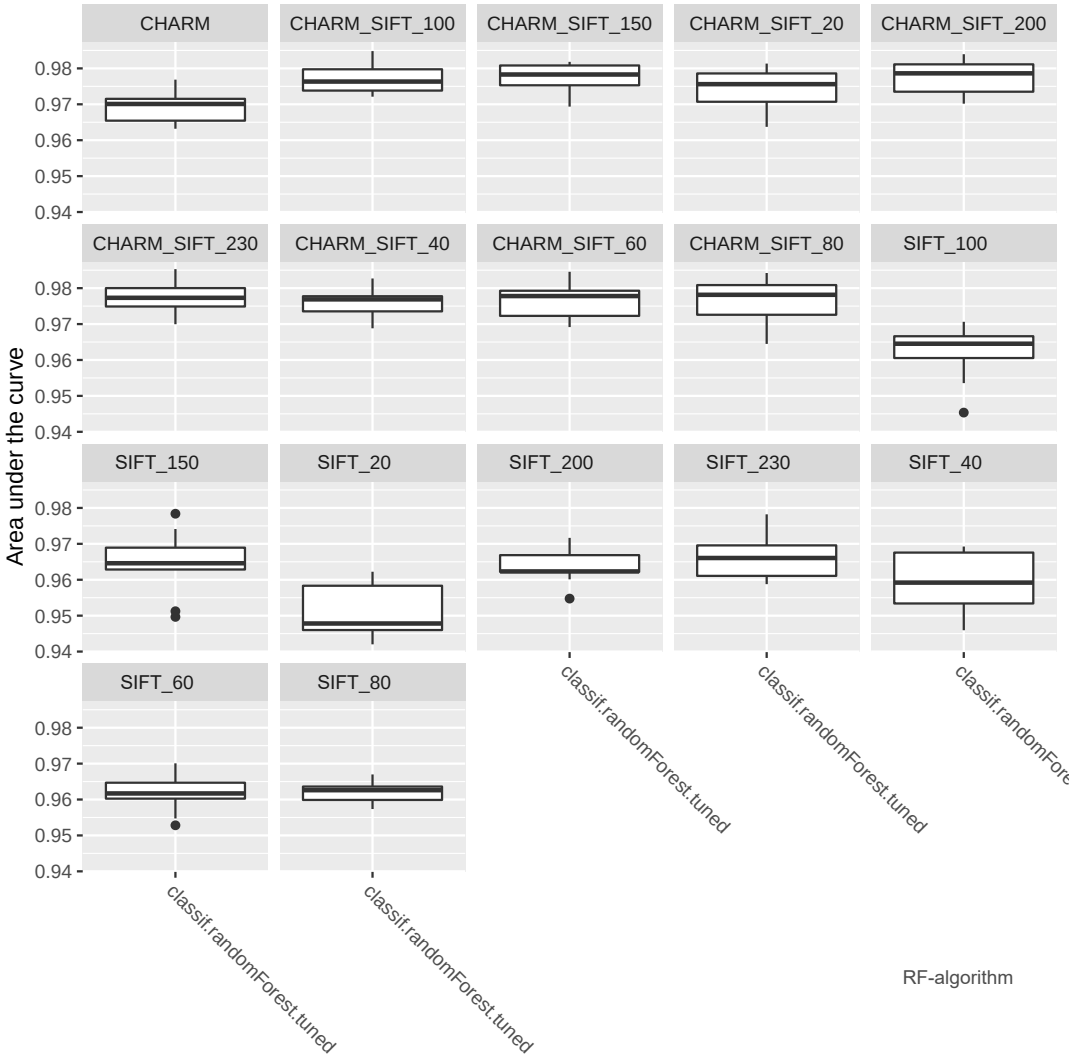


TABLE E.5: Table of RF results.

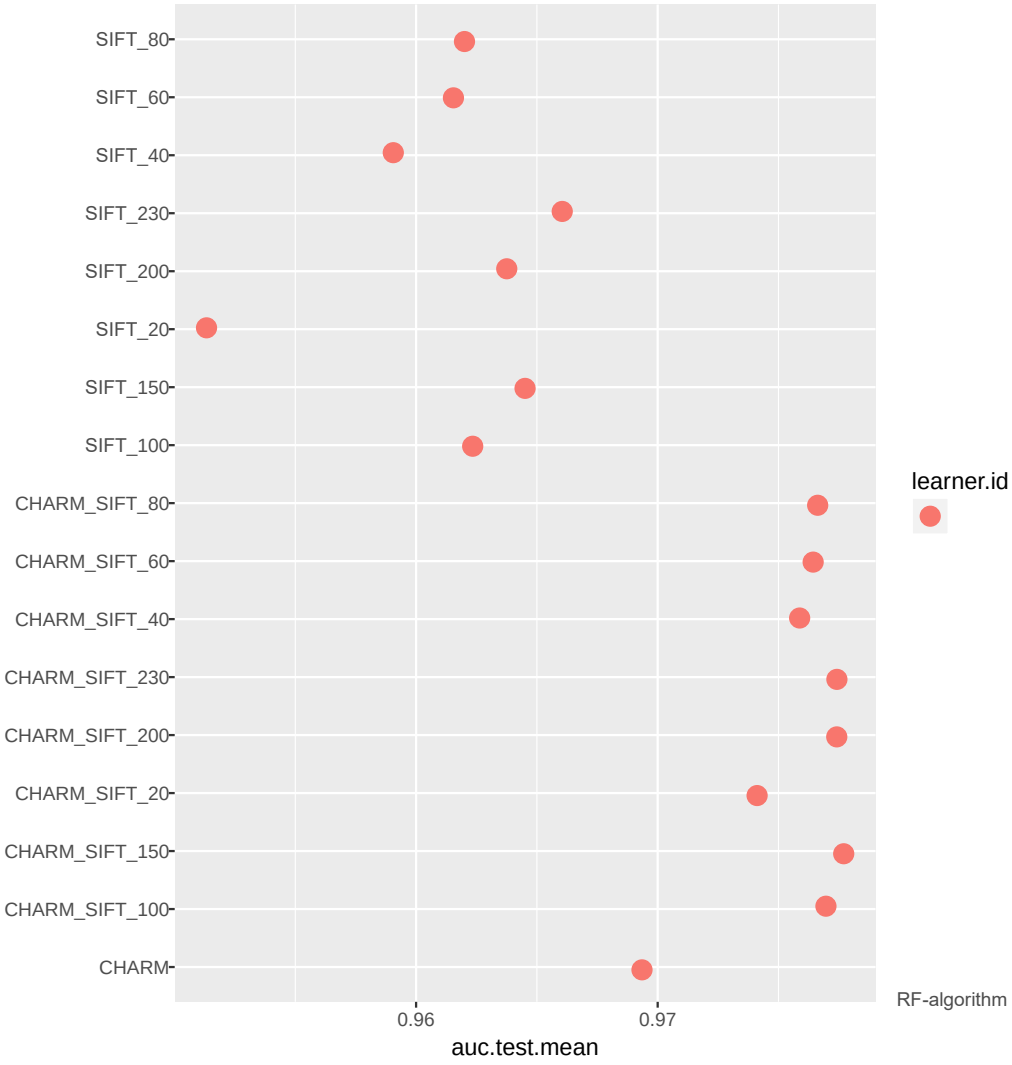


TABLE E.6: Table of RF results.

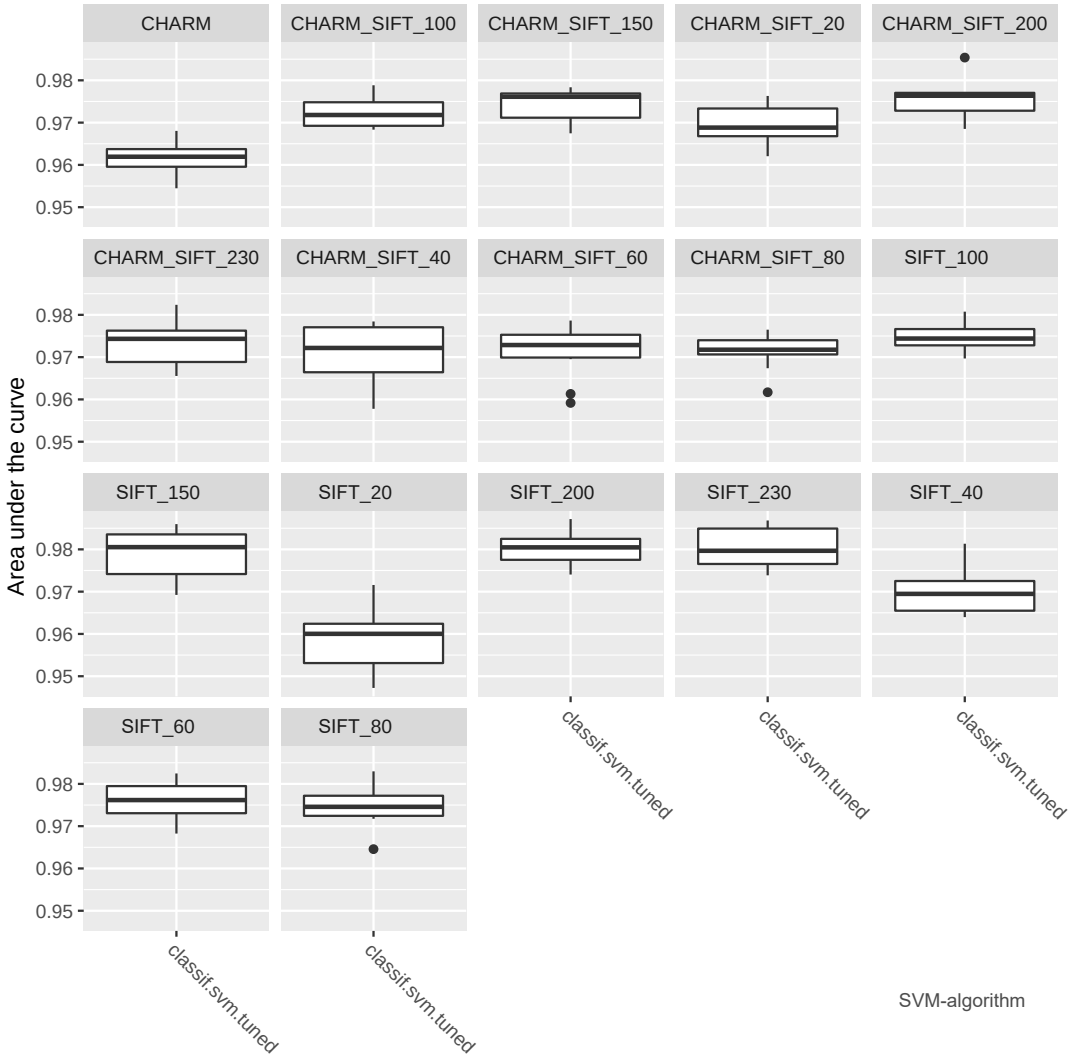


TABLE E.7: Table of SVM results.

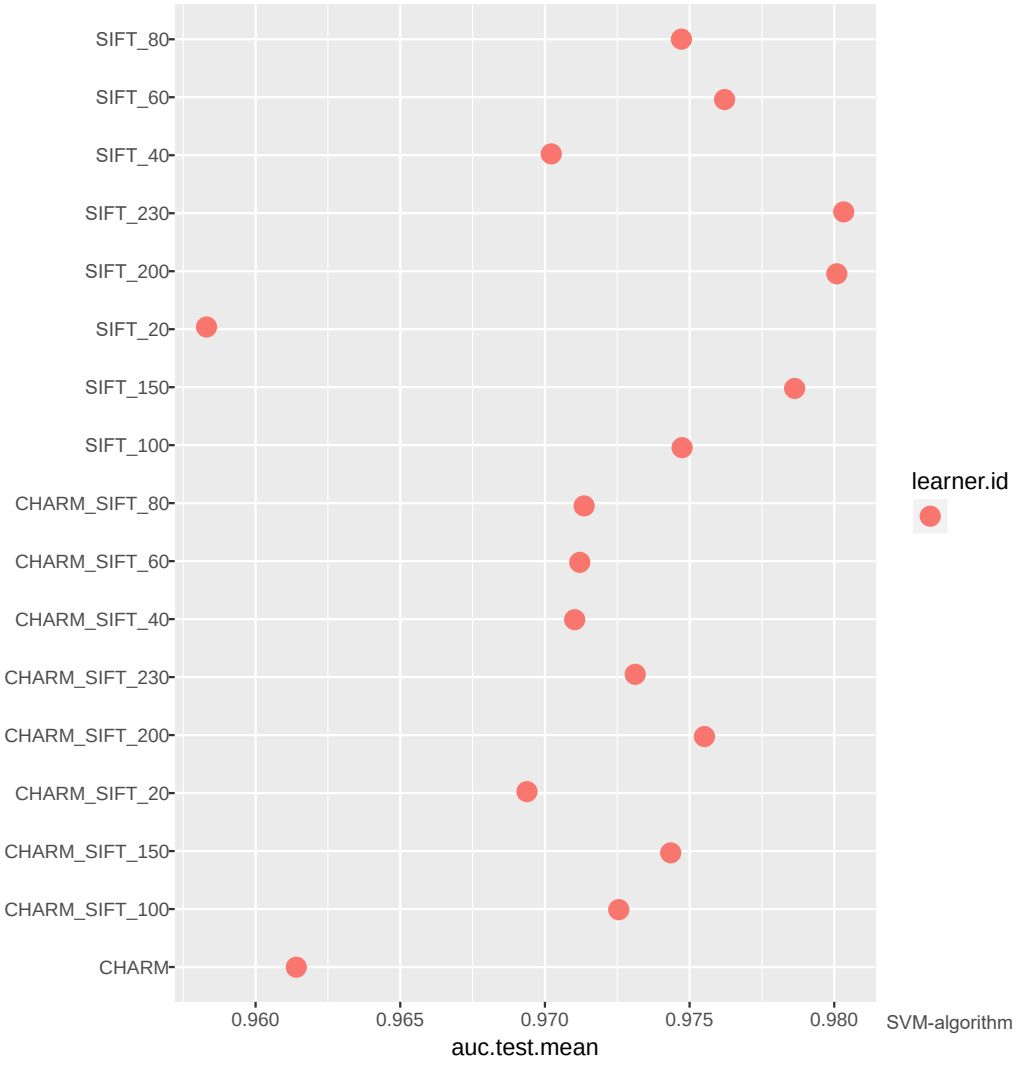


TABLE E.8: Table of SVM results.

Bibliography

Al-Kofahi, K. A., Lasek, S., Szarowski, D. H., Pace, C. J., Nagy, G., Turner, J. N., and Roysam, B. (2002). "Rapid automated three-dimensional tracing of neurons from confocal image stacks". In: *IEEE Transactions on Information Technology in Biomedicine* 6.2, pp. 171–187.

Amorim, A., Collins, N., DeHon, A., Hritcu, C., Pichardie, D., Pierce, B. C., Pollack, R., and Tolmach, A. (2014). "A Verified Information-Flow Architecture". In: *41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'14)*.

Anderl, J. L., Redpath, S., and Ball, A. J. (2009). "A neuronal and astrocyte co-culture assay for high content analysis of neurotoxicity". In: *J. Vis. Exp.* 5.27, p. 1173.

Aransay, J., Ballarin, C., and Rubio, J. (2008). "A Mechanized Proof of the Basic Perturbation Lemma". In: *Journal of Automated Reasoning* 40.4, pp. 271–292.

Aransay, J., Divasón, J., Heras, J., Lambán, L., Pascual, P., Rubio, A. L., and Rubio, J. (2012). *A report on an experiment in porting formal theories from Isabelle/HOL to Ecore and ACL2*. Tech. rep. URL: http://wiki.portal.chalmers.se/cse/uploads/ForMath/isabelle_acl2_report.

Ayala, R., Domínguez, E., Francés, A. R., and Quintero, A. (2003). "Homotopy in digital spaces". In: *Discrete Applied Mathematics* 125, pp. 3–24.

Ballesteros-Yáñez, I., Benavides-Piccione, R., Elston, G.N., Yuste, R., and DeFelipe, J. (2006). "Density and morphology of dendritic spines in mouse neocortex". In: *Neuroscience* 138.2, pp. 403–409.

Barrett, C. and Tinelli, C. (2007). "CVC3". In: *19th International Conference on Computer Aided Verification (CAV'07)*. Vol. 4590. LNCS, pp. 298–302.

Barthe, G., Pointcheval, D., and Zanella-Béguelin, S. (2012). "Verified Security of Redundancy-Free Encryption from Rabin and RSA". In: *Proceedings 19th ACM Conference on Computer and Communications Security (CCS'12)*, pp. 724–735.

Batista, Gustavo E. A. P. A., Prati, Ronaldo C., and Monard, Maria Carolina (2004). "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data". In: *SIGKDD Explor. Newsl.* 6.1, pp. 20–29.

Benton, N. (2006). *Machine Obstructed Proof: How many months can it take to verify 30 assembly instructions?*

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York, NY: Springer.

Bobot, F., Conchon, S., Contejean, E., Iguernelala, M., Lescuyer, S., and Mebsout, A. (2008). *The Alt-Ergo automated theorem prover*. URL: http://alt-ergo.lri.fr/.

Bobot, F., Filliâtre, JC., Marché, C., Melquiond, G., and Paskevich, A. (2015). *The Why3 platform, version 0.86.1*. version 0.86.1.

Boldo, S., Lelay, C., and Melquiond, G. (2013). *Formalization of Real Analysis: A Survey of Proof Assistants and Libraries*. Tech. rep.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*. ACM Press, pp. 144–152.

Breiman, L. (2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32.

Burdy, L. et al. (2005). "An overview of JML tools and applications". In: *International Journal on Software Tools for Technology Transf* 7.3, pp. 212–232.

Cajal, S. R. (1889). "Conexión general de los elementos nerviosos". In: *La Medicina Práctica* 2, pp. 341–346.

— (1917). *Recuerdos de mi vida*. Vol. 2. Imprenta y Librería de Nicolás Moya.

— (1998). *La textura del sistema nervioso del hombre y los vertebrados (1899 reprint)*. Springer Verlag.

Calderón de Anda, F. et al. (2012). "Autism spectrum disorder susceptibility gene TAOK2 affects basal dendrite formation in the neocortex". In: *Natural Neuroscience* 15.7, pp. 1022–1031.

Canny, J. (1986). "A Computational Approach to Edge Detection". In: *IEEE Pattern Anal. Mach. Intell.* 8.6, pp. 679–698.

Carlsson, G. and DeSilva, V. (2010). "Zigzag persistence". In: *Foundations of Computational Mathematics* 10.4, pp. 367–405.

Castleman, K. (1996). *Digital Image Processing*. Prentice-Hall.

Charoenkwan, P., Hwang, E., Cutler, R. W., Lee, H.-C., Ko, L.-W., Huang, H.-L., and Ho, S.-Y. (2013). "HCS-Neurons: identifying phenotypic changes in multi-neuron images upon drug treatments of high-content screening". In: *BMC Bioinform.* 14.S16, S12.

Codescu, M. et al. (2012). "Towards Logical Frameworks in the Heterogeneous Tool Set Hets". In: *Post-Proceedings 20th International Workshop on Recent Trends in Algebraic Development Techniques (WADT'10)*. Vol. 7137. LNCS, pp. 139–159.

Coons, A. H., Creech, H. J., and Jones, R. N. (1941). "Immunological Properties of an Antibody Containing a Fluorescent Group". In: *Proceedings of the Society for Experimental Biology and Medicine* 47.2, pp. 200–202.

COQ development team (2012). *The* COQ *Proof Assistant, version 8.4*. Tech. rep.

Cuesto, G., Carames, C., Cantarero, M., Gasull, X., Acebes, A., and Morales, M. (2011). "PI3K activation modulates synaptogenesis and spinogenesis in a hippocampal culture model". In: *Journal of Neuroscience* 31.8, pp. 2721–2733.

Cuesto, G., Enriquez-Barreto, L., et al. (2011). "Phosphoinositide-3-kinase activation controls synaptogenesis and spinogenesis in hippocampal neurons". In: *Journal of Neuroscience* 31.8, pp. 2721–2733.

Cuesto, G., Jordán-Álvarez, S., Enriquez-Barreto, L., et al. (2015). "GSK3β inhibition Promotes Synaptogenesis in Drosophila and Mammalian Neurons". In: *PlosOne* 10.3.

Danielson, E. and Lee, S. H. (2014). "SynPAnal: Software for Rapid Quantification of the Density and Intensity of Protein Puncta from Fluorescence Microscopy Images of Neurons". In: *PLoS ONE* 9.12.

Dellani, P. R., Glaser, M., et al. (2007). "White matter fiber tracking computation based on Diffusion Tensor Imaging for clinical applications". In: *Journal of Digital Imaging* 20.1, pp. 88–97.

Denney, E. (2000). "A Prototype Proof Translator from HOL to Coq". In: *13th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'00)*. Vol. 1869. LNCS, pp. 108–125.

Díaz de Greñu de Pedro, J. (2014). *Análisis Matemático de rutinas de procesamiento de imágenes digitales en Fiji/ImageJ*. Tech. rep. Universidad de La Rioja.

Domínguez, C. and Rubio, J. (2011). "Effective homology of bicomplexes, formalized in Coq". In: *Theoretical Computer Science* 412.11, pp. 962–970.

Donohue, D. E. and Ascoli, G. A. (2011). "Automated reconstruction of neuronal morphology: an overview". In: *Brain Research Reviews* 67.1–2, pp. 94–102.

Dousson, X., Rubio, J., Sergeraert, F., and Siret, Y. (1999). *The Kenzo program*. Institut Fourier, Grenoble.

Dragunow, M. (2008). "High-content analysis in neuroscience". In: *Nat. Rev. Neurosci.* 9.10, pp. 779–788.

Edelsbrunner, H., Letscher, D., and Zomorodian, A. (2002). "Topological persistence and simplification". In: *Discrete Computional Geometry* 28, pp. 511–533.

Elston, G. N. and DeFelipe, J. (2002). "Spine distribution in cortical pyramidal cells: a common organizational principle across species". In: *Progress in Brain Research* 136, pp. 109–133.

Enríquez-Barreto, L., Cuesto, G., Domíguez-Iturza, N., Gavilán, E., Ruano, D., Sandi, C., Fernández-Ruiz, A., Martín-Vázquez, G., and Morales, M. (2014). "Learning improvement after PI3K activation correlates with de novo formation of functional small spines". In: *Frontiers in Molecular Neuroscience* 6.54.

Enríquez-Barreto, L. and Morales, M. (2016). "The PI3K signaling pathway as a pharmacological target in Autism related disorders and Schizophrenia". In: *Molecular and Cellular Therapies* 4.2.

Fawcett, T. (2006). "An introduction to {ROC} analysis". In: *Pattern Recognition Letters* 27.8, pp. 861 –874.

Filliâtre, J. and Marché, C. (2007). "The Why/Krakatoa/Caduceus Platform for Deductive Program Verification". In: *19th International Conference on Computer Aided Verification (CAV'07)*. Vol. 4590. LNCS, pp. 173–177.

ForMath (2010–2013). *ForMath: Formalisation of Mathematics, European Project*. URL: http://wiki.portal.chalmers.se/cse/pmwiki.php/ForMath/ForMath.

Friedman, Jerome, Hastie, Trevor, and Tibshirani, Robert (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1, pp. 1–22.

Gabor, D. (1946). "Theory of communication". In: *Journal of IEEE* 93, pp. 429–457.

Gamboa, R. and Kaufmann, M. (2001). "Non-Standard Analysis in ACL2". In: *Journal of Automated Reasoning* 27.4, pp. 323–351.

Goedert, M. and Spillantini, M. G. (2006). "A Century of Alzheimer's Disease". In: *Science* 314, p. 777.

Gogolla, N., Galimberti, I., Deguchi, Y., and Caroni, P. (2009). "Wnt Signaling Mediates Experience-Related Regulation of Synapse Numbers and Mossy Fiber Connectivities in the Adult Hippocampus". In: *Neuron* 62.4, pp. 510 –525.

Gonthier, G. et al. (2013). "A Machine-Checked Proof of the Odd Order Theorem". In: *Proceedings 4th Conference on Interactive Theorem Proving (ITP'13)*. LNCS.

González-Díaz, R. and Real, P. (2005). "On the Cohomology of 3D Digital Images". In: *Discrete Applied Mathematics* 147.2–3, pp. 245–263.

Gordon, M. J. C., Kaufmann, M., and Ray, S. (2011). "The Right Tools for the Job: Correctness of Cone of Influence Reduction Proved Using ACL2 and HOL4". In: *Journal of Automated Reasoning*, pp. 1–16.

Govindarajan, A. et al. (2011). "The dendritic branch is the preferred integrative unit for protein synthesis-dependent LTP". In: *Neuron* 69.1, pp. 132–146.

Gradshtein, I. and Ryzhik, I. (1994). "Table of integrals, series and products". In: *Academic Press* 5, p. 1054.

Graham, P. (1996). *ANSI Common Lisp*. Prentice Hall.

Gurari, E (1999). "Backtracking Algorithms CIS 680: Data Structures: Chapter 19: Backtracking Algorithms". In: *Ohio State University* 23.08.

Hadjidementriou, E., Grossberg, M., and Nayar, S. (2001). "Spatial information in multiresolution histograms". In: *IEEE Conference on Computer Vision and Pattern Recognition* 1, p. 702.

Hales, T. (2005). *The Flyspeck Project fact sheet*. Project description available at http://code.google.com/p/flyspeck/.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I.H. (2009). "The WEKA data mining software: an update". In: *SIGKDD Expl.* 11.1, pp. 10–18.

Hardin, D. (2010). *Design and Verification of Microprocessor Systems for High-Assurance Applications*. Springer.

Hechenbichler, K. and Schliep, K. (2006). "Weighted k-nearest-neighbor techniques and ordinal classification". In: *Discussion Paper 399, SFB 386*.

Heck, N. et al. (2012). "A deconvolution method to improve automated 3D-analysis of dendritic spines: application to a mouse model of Huntington's disease". In: *Brain Structure and Function* 217.2, pp. 421–434.

Heras, J., Coquand, T., Mörtberg, A., and Siles, V. (2013). "Computing Persistent Homology within Coq/SSReflect". In: *To appear in ACM Transactions on Computational Logic*.

Heras, J., Dénès, M., Mata, G., Mörtberg, A., Poza, M., and Siles, V. (2012). "Towards a Certified Computation of Homology Groups for Digital Images". In: *Computational Topology in Image Context: 4th International Workshop, CTIC 2012, Bertinoro, Italy, May 28-30, 2012. Proceedings*. Springer Berlin Heidelberg, pp. 49–57.

Heras, J., Mata, G., Romero, A., Rubio, J., and Sáenz, R. (2013). "Verifying a Plaftorm for Digital Imaging: A Multi-tool Strategy". In: *Intelligent Computer Mathematics: MKM, Calculemus, DML, and Systems and Projects 2013, Held as Part of CICM 2013, Bath, UK. Proceedings*. Springer Berlin Heidelberg, pp. 66–81.

Heras, J., Pascual, V., and Rubio, J. (2012). "A Certified Module to Study Digital Images with the Kenzo system". In: *Proceedings of the 13th International Conference on Computer Aided Systems Theory (EUROCAST'11)*. Vol. 6927. LNCS, pp. 113–120.

Heras, J., Poza, M., and Rubio, J. (2012). "Verifying an Algorithm Computing Discrete Vector Fields for Digital Imaging". In: *AISC/MKM/Calculemus (CICM'12)*. Vol. 7362. LNCS, pp. 216–230.

Huang, L-K. and Wang, M-J. J. (1995). "Image thresholding by minimizing the measure of fuzziness". In: *Pattern Recognition* 28.1, pp. 41–51.

Ireland, A. and Stark, J. (1997). *On the Automatic Discovery of Loop Invariants*.

Jacquel, M., Berkani, K., Delahaye, D, and Dubois, C. (2011). "Verifying B Proof Rules Using Deep Embedding and Automated Theorem Proving". In: *Proceedings 9th International Conference on Software Engineering and Formal Methods (SEFM'11)*. Vol. 7041. LNCS, pp. 253–268.

Kaczynski, T., Mischaikow, K., and Mrozek, M. (2004). *Computational Homology*. Vol. 157, pp. 411–414.

Kaech, S. and Banker, G. (2006). "Culturing hippocampal neurons". In: *Nature Protocols* 1.5, pp. 2406–2415.

Kaufmann, M. and Moore, J. S. (2012). *ACL2 Version 6.0*. URL: http://www.cs.utexas.edu/users/moore/acl2/.

Keller, C. and Werner, B. (2011). "Importing HOL Light into Coq". In: *Proceedings 1st International Conference on Interactive Theorem Proving (ITP'11)*. Vol. 6172. LNCS, pp. 307–322.

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc. Chap. 6.1. ISBN: 0-201-89683-4.

Kozlov, D. N. (2008). *Combinatorial Algebraic Topology*. Vol. 21. Springer.

Krakatoa. *Krakatoa*. http://krakatoa.lri.fr.

Lambán, L., Martín-Mateos, F. J., Rubio, J., and Ruiz-Reina, J. L. (2013). "Verifying the bridge between Simplicial Topology and Algebra: the Eilenberg-Zilber algorithm". In: *Logic Journal of the IGPL* 22.1, pp. 39–65.

Landmesser, L. (1994). "Axonal outgrowth and pathfinding". In: *Progress in Brain Research* 103, pp. 67–73.

Lee, D. R. et al. (2007). "FPGA based connected component labeling". In: *Proceedings International Conference on Control, Automation and System*, pp. 2313–2317.

Lim, J. S. (1990). "Two-Dimensional Signal and Image Processing". In: *Prentice Hall*, pp. 42–45.

Linkert, M., Rueden, C. T., Allan, C., et al. (2010). "Metadata matters: access to image data in the real world". In: *The Journal of Cell Biology* 189.5, pp. 777–782.

Liu, H. and S., Moore J. (2004). "Java Program Verification via a JVM Deep Embedding in ACL2". In: *Proceedings 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'04)*. Vol. 3223. LNCS, pp. 184–200.

Lloyd, S. P. (1982). "Least squares quantization in PCM". In: *IEEE Trans. Information Theory* 28.2, pp. 129–136.

Lowe, D. G. (2004). "Distinctive image features from scale-invariant keypoints". In: *Int. J. Comput. Vis.* 60.2, pp. 91–110.

MacQueen, J. (1967). "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, C.: University of California Press, pp. 281–297. URL: http://projecteuclid.org/euclid.bsmsp/1200512992.

Mata, G. *NeuronPersistentJ*. http://imagejdocu.tudor.lu/doku.php?id=plugin:utilities:neuronpersistentj:start.

— *NeuronzigzagJ*. http://spineup.jimdo.com/downloads/.

— *NucleusJ*. https://spineup.jimdo.com/downloads/.

— *SynapCountJ*. http://imagejdocu.tudor.lu/doku.php?id=plugin:utilities:synapsescountj:start.

Mata, G., Cuesto, G., Heras, J., Morales, M., Romero, A., and Rubio, J. (2017). "SynapCountJ: A Validated Tool for Analyzing Synaptic Densities in Neurons". In: *Biomedical Engineering Systems and Technologies: 9th International Joint Conference, BIOSTEC 2016, Rome, Italy, Revised Selected Papers*. Springer International Publishing, pp. 41–55.

Mata, G., Morales, M., Romero, A., and Rubio, J. (2015). "Zigzag persistent homology for processing neuronal images". In: *Pattern Recognition Letters* 62.1, pp. 55–60.

Mata, G., Radojevic, M., Smal, I., Morales, M., Meijering, E., and Rubio, J. (2016). "Automatic Detection of Neurons in High-Content Microscope Images Using Machine Learning Approaches". In: *Proceedings of the 13th IEEE International Symposium on Biomedical Imaging (ISBI'2016)*. IEEE Xplore, pp. 330–333.

MathWorks (2016). *version 9.0.0.341360 (R2016a)*. The MathWorks Inc.

Maunder, C. R. F. (1996). *Algebraic Topology*. Dover.

Meijering, E. (2010). "Neuron tracing in perspective". In: *Cytometry Part A* 77.7, pp. 693–704.

Meijering, E., Jacob, M., Sarria, J. C. F., et al. (2004). "Design and Validation of a Tool for Neurite Tracing and Analysis in Fluorescence Microscopy Images". In: *Cytometry Part A* 58.2, pp. 167–176.

Miura, K. et al. (2016). *Bioimage Data Analysis*. Wiley-VCH.

Molecular Devices (2015). *MetaMorph Research Imaging*.

Morales, M., Colicos, M. A., and Goda, Y. (2000). "Actin-dependent regulation of neurotransmitter release at central synapses". In: *Neuron* 27.3, pp. 539–550.

Mori, S. and Zijl, P. C. M. van (2002). "Fiber tracking: principles and strategies – a technical review". In: *NMR Biomedicine* 15, pp. 468–480.

Munkres, J. R. (1984). *Elements of Algebraic Topology*. Addison-Wesley.

Murphy, K. (2001). "The Bayes net toolbox for Mat-lab". In: *Computing Science and Statistics* 33, pp. 1–20.

Obua, S. and Skalberg, S. (2006). "Importing HOL into Isabelle/HOL". In: *3rd International Joint Conference on Automated Reasoning (IJCAR'06)*. Vol. 4130. LNCS, pp. 298–302.

Orlov, N., Shamir, L., Macura, T., Johnston, J., Eckley, D. M., and Goldberg, I. G. (2008). "WND-CHARM: multi-purpose image classification using compound image transforms". In: *Pattern Recognit. Lett.* 29.11, pp. 1684–1693.

Otsu, N. (1979). "A threshold selection method from gray level histograms". In: *IEEE Transactions On Systems, Man and Cybernetics* 9, pp. 62–66.

Pawley, J. B., ed. (2006). *Handbook of Biological Confocal Microscopy*. Springer.

Pham, T.D., Crane, T.D., et al. (2004). "Extraction of fluorescent cell puncta by adative fuzzy segmentation". In: *Bioinformatics* 20, 2189 – 2196.

Poza, M., Domínguez, C., Heras, J., and Rubio, J. (2014). "A certified reduction strategy for homological image processing". In: *ACM Transactions on Computational Logic* 15.3.

Prewitt, J. and Mendelsohn, M. L. (1966). "The analysis of cell images". In: *Annals of the New York Academy of Sciences* 128.1, pp. 1035–1053.

Prewitt, J. M. (1970). "Object enhancement and extraction Picture Processing and Psychopictoris". In: *Lipkin BS, Rosenfeld A, editors*, pp. 75–149.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. URL: https://www.R-project.org/.

Radio, N. M. (2012). "Neurite Outgrowth Assessment Using High Content Analysis Methodology". In: *Meth. Mol. Biol.* 846, pp. 247–260.

Rakhmadi, A. et al. (2010). "Connected Component Labeling Using Components Neighbors-Scan Labeling Approach". In: *Journal of Computer Science* 6.10, pp. 1096–1104.

Ramón-Moliner, E. (1970). "The Golgi-Cox technique". In: *Contemporary Methods in Neuroanatomy*, pp. 32–55.

Ridler, T. W. and Calvard, S. (1978). "Picture Thresholding Using an Iterative Selection Method". In: *IEEE Transactions on Systems, Man, and Cybernetics* 8.8, pp. 630–632.

Rivest, R. L. et al. (2008). *The MD6 hash function A proposal to NIST for SHA-3*. Tech. rep.

Roberts, T. F., Tschida, K. A., E., Klein M., and R., Mooney (2010). "Rapid spine stabilization and synaptic enhancement at the onset of behavioural learning". In: *Nature* 463, pp. 948 –952.

Roerdink, J. B. T. M. and Meijster, A. (2000). "The Watershed Transform: Definitions, Algorithms and Parallelization Strategies". In: *Fundam. Inf.* 41.1,2, pp. 187–228.

Romero, A., Heras, J., Rubio, J., and Sergeraert, F. (2014). "Defining and computing persistent Z-homology in the general case". In: *CoRR* abs/1403.7086.

Romero, A. and Rubio, J. (2013). "Homotopy groups of suspended classifying spaces: An experimental approach". In: *Mathematics of Computation* 82, pp. 2237–2244.

Romero, A. and Sergeraert, F. (2010). *Discrete Vector Fields and Fundamental Algebraic Topology*.

Rosenfeld, A. (1974). "Adjacency in digital pictures". In: *Information and Control* 26.1, pp. 24 –33.

Schindelin, J., Arganda-Carreras, I., Frise, E., et al. (2012). "Fiji: an open-source platform for biological-image analysis". In: *Nature methods* 9.7, pp. 676–682.

Schmitz, S. K., Johannes Hjorth, J. J., Joemail, R. M. S., et al. (2011). "Automated analysis of neuronal morphology, synapse number and synaptic recruitment". In: *Journal of Neuroscience Methods* 195.2, pp. 185–193.

Schneider, C. A., Rasband, W. S., and Eliceiri, K. W. (2012). "NIH Image to ImageJ: 25 years of image analysis". In: *Nat. Meth.* 9.7, pp. 671–675.

Ségonne, F., Grimson, E., and Fischl, B. (2003). "Topological Correction of Subcortical Segmentation". In: *Proceedings of the 6th International conference on Medical Image Computing and Computer Assisted Intervention (MICCAI'03)*. Vol. 2879. LNCS, pp. 695–702.

Selkoe, D. J. (2002). "Alzheimer's diseases is a synaptic failure". In: *Science* 298.5594, pp. 789–791.

Selkoe, D. J. and Hardy, J. (2016). "The amyloid hypothesis of Alzheimer's disease at 25 years". In: *EMBO Molecular Medicine* 8.6, pp. 595–608.

Sergeraert, F. (1992). *Effective homology, a survey*. Tech. rep. Institut Fourier.

Shamir, L., Orlov, N., Eckley, D. M., Macura, T., Johnston, J., and Goldberg, I. G. (2008). "Wndchrm – an open source utility for biological image analysis". In: *Source Code Biol. Med.* 3.13, pp. 1–13.

Shapiro, L. and Stockman, G. (2002). *Computer Vision*. Prentice Hall.

Shiwarski, D. J., Dagda, R. D., and T., Chu. C. (2014). *Green and Red Puncta Colocalization*.

Sivic, J. (2009). "Efficient visual search of videos cast as text retrieval". In: *IEEE Transactions On Pattern Analysis and Machine Intelligence* 31.4, pp. 591–605.

Skiena, S. S. (2008). "Sorting and Searching". In: *The Algorithm Design Manual*. London: Springer London, pp. 103–144.

SpineUp (2014). *SpineUp.es*. http://spineup.es.

Tamura, H., Mori, S., and Yamavaki, T. (1978). "Textural features corresponding to visual perception". In: *IEEE Transactions On Systems, Man and Cybernetics* 8, pp. 460–472.

Tarjan, R. (1972). "Depth first search and linear graph algorithms". In: *Siam Journal on computing* 1.2.

Tessier-Lavigne, M. and Goodman, C. S. (1996). "The molecular biology of axon guidance". In: *Science* 274.5290, pp. 1123–1133.

Tibshirani, Robert (1996). "Regression Shrinkage and Selection Via the Lasso". In: *Journal of the Royal Statistical Society, Series B* 58, pp. 267–288.

Tschida, K. A. and Mooney, R. (2012). "Deafening Drives Cell-Type-Specific Changes to Dendritic Spines in a Sensorimotor Nucleus Important to Learned Vocalizations". In: *Neuron* 73.5, pp. 1028 –1039.

Vallotton, P., Lagerstrom, R., Sun, C., Buckley, M., Wang, D., DeSilva, M., Tan, S.-S., and Gunnersen, J. M. (2007). "Automated analysis of neurite branching in cultured cortical neurons using HCA-Vision". In: *Cytom. A* 71.10, pp. 889–895.

Vapnik, V. N. (1999). *The Nature of Statistical Learning Theory*.

Vedaldi, A. and Fulkerson, B. (2008). *VLFeat: An Open and Portable Library of Computer Vision Algorithms*. http://www.vlfeat.org/.

Velez-Pardo, C. et al. (2004). "CA1 hippocampal neuronal loss in familial Alzheimer's disease presenilin-1 E280A mutation is related to epilepsy". In: *Epilepsia* 45.7, pp. 751–756.

Vincent, L. and Soille, P. (1991). "Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 13.6, pp. 583–598.

Wark, B. (2013). *Puncta Analyzer v2.0*. URL: https://github.com/physion/puncta-analyzer.

Weeden, V. J., Wang, R. P., et al. (2008). "Diffusion spectrum magnetic resonance imaging (DSI) tractography of crossing fibers". In: *NeuroImage* 41, pp. 1267–1277.

Wu, C., Schulte, J., Sepp, K. J., Littleton, J. T., and Hong, P. (2010). "Automatic robust neurite detection and morphological analysis of neuronal cell cultures in high-content screening". In: *Neuroinform.* 8.2, pp. 83–100.

Xia, X. and Wong, S. T. C. (2012). "Concise review: a high-content screening approach to stem cell research and drug discovery". In: *Stem Cells* 30.9, pp. 1800–1807.

Xu, T., Yu, X., Perlik, A. J., Tobin, W. F., Zweig, J. A., Tennant, K., Jones, T., and Zuo, Y. (2009). "Rapid formation and selective stabilization of synapses for enduring motor memories". In: *Nature* 462, pp. 915 –919.

Zhang, Y., Zhou, X., Degterev, A., Lipinski, M., Adjeroh, D., Yuan, J., and Wong, S. T. C. (2007). "A novel tracing algorithm for high throughput imaging: screening of neuron-based assays". In: *J. Neurosci. Meth.* 160.1, pp. 149–162.

Zhou, X., Wang, X., Dougherty, ER., Russ, D., and Suh, E. (2004). "Gene Clustering Based on Clusterwide Mutual Information". In: *Journal of Computational Biology* 11.1, pp. 147 –161.

Zomorodian, A. (2001). "Computing and Comprehending Topology: Persistence and Hierarchical Morse Complexes". PhD thesis. University of Illionois at Urbana-Champaign.

Zou, H. and Hastiel, T. (2005). "Regularization and variable selection via the Elastic Net". In: *Journal of the Royal Statistical Society, Series B* 67, pp. 301–320.