UNIVERSITAT DE BARCELONA

Final Degree Project

Biomedical Engineering Degree

Development of an image-based Deep Learning model for *Schizosaccharomyces pombe* cell cycle phase classification

Barcelona, 06 June 2022

Author: Mireia Alibau

Tutor: Rosa Aligué

ABSTRACT

Despite medical advances, cancer remains a life-threatening disease, accounting for millions of deaths per year. Aiming to develop novel cancer treatments, understanding the nature of cancerous cells and mechanisms of activation and progression at the microscale are material of study of many investigators. Special interest has been given to the actin cytoskeleton, whose morphology is altered in many cell functions -such as cell division- under tumorous processes. However, observing and manually identifying cells according to their intracellular architecture might be time-consuming for the investigator.

In light of the above, this study intents to construct a Convolutional Neural Network -a type of Deep Learning model- for *Schizosaccharomyces pombe* -a common model of study- classification according to their actin cytoskeleton during the cell cycle. For this purpose, a dataset containing representative images of different actin phenotypes was used for the training and testing phases, as well as for the final validation of the constructed model.

The outcoming results demonstrate the successful learning capacity of the algorithm, whose evaluation metrics depicted a nearly perfect model. Nonetheless, when facing unseen data, its reliability is questioned, since it fails to correctly identify a considerable proportion of the introduced unseen images. For this reason, the algorithm prediction cannot be considered as the absolute truth but as a complementary tool. In order to improve the predictive ability of the model and seeking for a better performance, a future dataset reconstruction and expansion, with a subsequent validation, should be performed.

***Keywords**: Actin cytoskeleton, Classification model, Deep Learning*

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere appreciation to my project tutor Rosa Aligué for offering me the chance to collaborate together with the biomedical research group as well as taking part of the current study being carried on. I am also very thankful for all the people working in the group, for making me feel like one member more of the team.

Secondly, I would like to show my deepest gratitude to Maribel Marquina for her guidance on the biomedical field and supervising at the starting of the project. I am immensely thankful for her encouragement during all my final degree stay, always keeping an eye on me with her kind and supportive personality.

This project could not have been possible without my family and friends. Specially to my father, who encouraged me to temporally move to Barcelona during my last university year and for all those 'good morning' messages, day over day, that he send me as a way of showing me his support. Also, I am thankful for my grandparents that, whenever I went to visit them, welcomed me with open arms and make me disconnect from the student life.

Finally, special thanks to my partner, who has been by my side during all the journey, listening to my long chats about the project, cheering me up whenever I felt frustrated and giving me the extra stamina to fulfill my dream of becoming a biomedical engineer.

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application programming interface
BO	Bayesian Optimization
CLSM	Confocal laser scanning microscope
CNN or ConvNet	Convolutional Neural Networks
DL	Deep Learning
DNN	Deep Neural Networks
ERDF	European Regional Development Fund
FN	False Negative
FP	False Positive
GFP	Green Fluorescent Protein
HM	Harmonized mean
IDE	Integrated Development Environment
LC	Learning curve
ML	Machine Learning
MLP	Multi-Layer Perceptron
MM	Minimal Media

MSE	Mean Squared Error
O/N	Overnight
OD	Optical Density
OpenCV	Open Source Computer Vision Library
ReLU	Rectified Linear activation Unit
RNN	Recurrent Neural Networks
SBL	Soybean lectin
SWOT	Strengths, Weaknesses, Opportunities and Threats
TN	True Negative
TP	True Positive
TPR	True positive rate
WBS	Work Breakdown Structure

TABLE OF CONTENTS

1. INTRODUCTION

Cancer is one of the leading causes of death in the world: just in 2020 an estimated number of 19.3 million new cases were detected [1]. Although human efforts directed to find novel treatments to cure cancer, there is still a long pathway to eradicate such disease. The first step in this race against time consists in the comprehension of this illness at the molecular and cellular level.

The actin cytoskeleton regulation under stress conditions, with special focus on its behavior during cytokinesis, has been described as a relevant trigger of tumor generation. Due to its role in cell shape maintenance and repercussion in eukaryotic cells function, it has a significant impact in cancer progression.

Previous biomolecular studies conducted with *Schizosaccharomyces pombe* -a unicellular eukaryote organism- focused on whether cells could adapt or not in front of a stressed cytoskeleton based on their genotype. Next steps were directed to evaluate the morphological changes of the actin structure. Confocal microscopy images were taken, proving researchers' suspicions: phenotypical changes were detected. Even so, the potential of image analysis was not exploited at all.

Manual image analysis represents a horrendous task for many investigators: processing images, segmenting cells, detecting relevant features for classification and finally classifying them are tasks which require time and both patience and practice for part of the researcher. Nonetheless, part of the image analysis problem has been overcome thanks to the nowadays technological advances in the biomedical field. As biomedical engineers is our role to work hand in hand with researchers, scientists and medical professionals for the development of tools to ease their daily life.

1.1 DESCRIPTION OF THE PROJECT

The described problem underlines the need of an automatic system capable of analyze and classify cells eluding the time investment in front of the screen. Retrieved the project's staring point, this final degree work presents an alternative methodology for cell analysis which, in contrast of the burdensome manual modus operandi, aims to automatize the process. In this way, we present a Deep Learning (DL) algorithm to automatically classify *pombe* cells based on their actin cytoskeleton phenotype in brief time. Images will be processed by the model and attributed a label according to the features detected.

The project has been carried out with the *Cell signaling by protein kinases and cancer* research group in the Department of Biomedicine from the University of Barcelona. Data collection was performed in the Advanced Optical Microscopy Department located in the same building. A specific dataset was built for the project, making use of the images acquired with a confocal microscope for the current study taking place on the above-mentioned department.

1.2. OBJECTIVES

As already started, the current study seeks to develop a *pombe* cells classifier according to their actin architecture. In comparison to the manual methodology, applying an automatic classification will have a direct impact on the required time for the conduction of this task: the algorithm will be capable of mimicking human intelligence and solve the illustrated problem in a short period of time. For the purpose of accomplishing the main goal, different sub-objectives have been defined:

- Biomedicine practical sessions

  Aiming to gather information about the problem in question and develop a proper solution, understanding the project background and get familiarized with cell manipulation and sample preparation have been depicted as crucial activities. A whole week will be purposely dedicated to biomedical training under the tuition of a biomedical researcher, acquiring the knowledge necessary to reproduce the experimental protocol by oneself without help.

- Review and polish our skills in data manipulation and Machine Leaning

  Gather information on data handling techniques and Machine Learning (ML) models, along with their development and implementation. This sub-objective implies both theoretical and practical perspectives: knowledge acquisition together with programming and extrapolation in the current study.

- Data acquisition and management

  Construct a proper dataset to be feed into our ML algorithm with images specifically recorded for the project. The sample preparation and image acquisition together with the subsequent data processing will be in our charge.

  Bibliographic research will be done and several methodologies will be tested until adopting the final cell mounting approach. By means of trial and error, the optimal conditions for image acquisition are meant to be found, taking into account both the researcher and biomedical engineer student interests: on the one hand, capture images where relevant cell features for the investigators are visible and in the other hand, acquire pictures suitable for the analysis.

- Building a DL model

  Defined as the core of the project, this sub-objective aims to set the essential pillars of ML algorithms' development. The problem to solve will be described in detail and different solutions will be proposed. Next step will be to pass from paper into reality and start building, training and testing the algorithm. Finally, the performance of the model will be evaluated to address its functioning and behavior.

1.3. SCOPE AND LIMITATIONS

The project consists of a DL model development for *pombe* cell classification according to their actin architecture and later testing and validation. The Department of Biomedicine located at the University of Barcelona is the place where the project was embedded. The study's duration is approximately 18 weeks, from the end of January 2022 until the beginning of June 2022. This timeframe encompasses the completion of the complete project, including the definition and initial conception of the project, the biomedical practical sessions and far-reaching research on the subject through the model development and mandatory validation, and finishing with the writing of the final degree report.

Constrains are not exempted for the project and, when thinking about the main limiting factor, time is the first variable that pops up in our heads. The sample cultivation presented a mean time of three days, limiting the data acquisition at one time per week. Strictly talking, this image acquisition could only be done if, additionally to satisfy the first requirement, the microscopy service could be booked.

Manipulating the images is indispensable before the DL model construction, process among which one of the most critical steps is segmentation. Segmentation, considered to be one of the most non-trivial tasks in image processing, involves extracting objects from an image. The human brain is capable of rapidly splitting images into different regions of interest just by simply looking at them. Even though advances in computer intelligence have made possible to automatize and mimic many human labors in seconds, it often fails to reproduce the human visual system, resulting in an inaccurate segmentation. Although it may seem that we are shooting ourselves in the foot, decanting for a semi-automatic segmentation will be advantageous for the project, reducing the time needed to perform such labor. Last but not least, image resolution should be worth noting. Poor image quality has a negative impact on the segmentation outcome and therefore, reliable acquisition devices should be considered to mitigate its effect.

Finally, remark the importance of the learning curve during the final degree work, including the biomedical training phase for the proper comprehension of the project's framework as well as the knowledge acquisition in the ML field, both elements depicted as essential for the project's consolidation. On the one hand, learning how to properly manipulate cells accurately following the experimental protocol for the subsequent image acquisition was crucial: as any other project involving sample handing, even a small error -prior or in the acquisition moment- can rebound on the data outcome, making it useless. On the other hand, the ML knowledge was learnt from scratch by enrolling on online courses.

2. BACKGROUND

Section 2 is aimed to provide information of both parts concerning the current work: the biomedical part and the engineering/technological part. Reviewing the theoretical background allows to lay down and comprehend the project basics for the fulfillment of the stablished objectives.

2.1. STUDY MODEL

Schizosaccharomyces pombe is a specie of yeast, a eukaryotic unicellular fungus, traditionally used as a model organism for the study of cell cycle control, DNA replication and genome integrity [2]. In only three hours, it can divide either by bipartition or binary fission. Additionally to showing a cell cycle similar to human cells, genes involved in human disease have been identified in *S. pombe* [3], making it a great model system for the identification of key cellular pathways in cancer and other diseases [4].

2.1.1. CYTOSKELETON AND CELL CYCLE

The cell cytoskeleton consists of a network of filaments that runs through the cell, structure composed by three elements: microtubes, actin filaments and intermedium filaments together with their associated proteins. Rather than being just the cell backbone, it is in charge of the basic cell functions, such as morphogenesis, cell migration and cell cycle. Nonetheless, these functions usually fail and become abnormal in cancer cells.

In this current work, special focus is given to the cell cycle -the process by which cells grow and divide into two cell daughters- together with the cytoskeleton changes. In *S. pombe* cells, by means of fluorophore staining, the different filaments are visible, reflecting the phases of the cell cycle. For a better understanding of which will be the different labels of the DL algorithm -in other words, the possible actin cytoskeleton phenotypes-, we introduce a brief description of this biological process.

The cell cycle is composed of two main stages: interphase and mitosis or M phase. At the same time, the interphase can be divided into three ordered phases [5]:

- G1: The growth phase, in which the cell is preparing to divide. The RNA, proteins and all the elements required for the DNA replication are synthetized.
- S: In this phase, a copy of the DNA is generated, an extra set of the genetic material.
- G2: The genetic material is organized and condensed; step previous to the cell division. The RNA and all the essential proteins are finally synthesized, and the cell acquires the adequate size for the division.

Finalized the G2 phase, the mitosis begins. During the M phase, the original cell -commonly referred as mother cell- is divided into two daughter cells, each one with an exact copy of the genomic material. The mitosis consists of five stages [6]:

- Prophase: The genetic material is condensed. Next, the formation of the mitotic spindle takes place, structure in which the chromosomes get attached.

- Metaphase: The chromosomes, coupled to the microtubes, get aligned at the center of the cell thanks to the tension forces.
- Anaphase: The chromatids -the two identical halves of a chromosome- get separated to the extremes of the cell. Furthermore, the actomyosin ring is assembled.
- Telophase: The mitotic spindle gets disorganized and the actomyosin ring gets contracted, aiming to divide the cell.
- Cytokinesis: A septum -cell wall- is form, separating the mother cell into two.

Concluded the cell division, the described process re-starts. *Figure 1* offers a visual review of the cell cycle. The distinct phases are represented, being observable how different behave the components of the cytoskeleton in each stage. More information about the relevance of such phenomenon for our project can be found in *Section 2.2*.

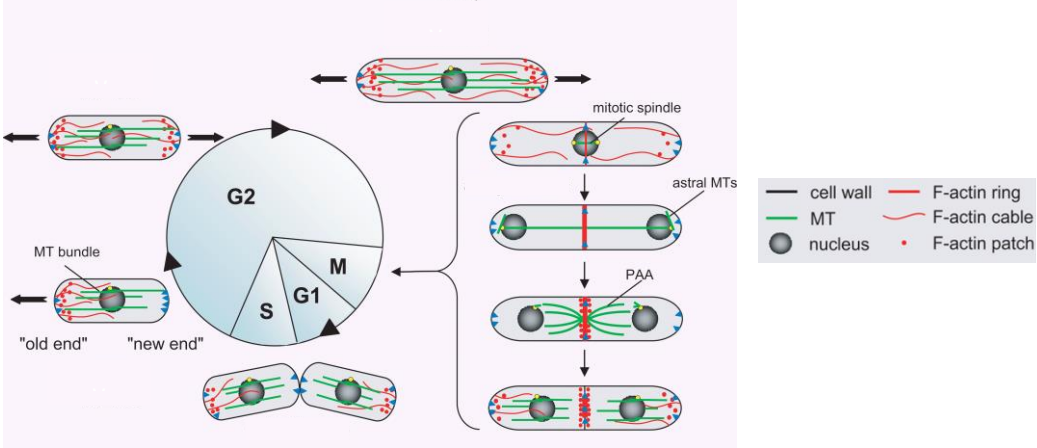


Figure 1. Fission yeast cell cycle, from birth to division and cell separation (left image) and image legend (right image) [7].

One might ask themselves how these intracellular changes will be visible. Actin across the cell can be observed by means of fluorescence or confocal laser scanning microscopy (CLSM). To do so, cells have been genetically manipulated and actin has been tagged with Green Fluorescent Protein (GFP).

2.1.2. REGULATORY ELEMENTS OF THE CELL CYCLE

The identification of cell cycle regulatory elements has become the target of many cancer studies, given that the disruption of the normal regulation of such process could be caused by gene mutations implicated in cancer. Once discovered, new therapeutic agents targeting such genes can be developed. However, this task is not straight-forward. To tackle this problem, different experiments have to be performed, where the cell undergoes environmental stressors,guio and its response to confront them is analyzed. Under normal conditions is expected that cells present an adaptive response to external stimuli whereas mutated cells present this via affected.

Underline the genes -and the subsequent transcription factors activated via cascade- playing a role in the cellular stress response is the current object of study in the Biomedicine Department of the University of Barcelona. The early study conducted by this research group in 2005 [8] was just the prelude to the analysis line regarding the CAMk family of proteins and its role in the reviewed cell

function. Special interest has been given to the Srk1 protein kinase, due to its role in cell stress response and its rebound effect on the cell cycle. In addition to Srk1, many are the experiments conducted under this research with other proteins which have been depicted to be related to this one, trying to underline their contribution to cell cycle (dys)regulation.

The Latrunculin B -or Lat B for short-, part of the family of marine toxins [9] produced by certain sponges -including *Latrunculia*, from which its name derives- has been widely used in many of the studies conducted. It inhibits the actin polymerization by disrupting the actin filament organization, causing the collapse of the actin cytoskeleton and subsequently mitotic delay [10]. Under non-lethal doses, wild type cells can overcome this situation and continue its cycle, whereas mutated cells, even though they can enter mitosis, show difficulties. Thanks to such discovery, scientists have been able to underline the key role of different proteins -including the Srk1- in response to actin cytoskeleton defects, given their role in the stability of the cell morphology during mitosis.

2.2. IMAGE CLASSIFICATION MODELS

Technological advances have positively impacted our lives: what seemed impossible in the past, is a nowadays a reality. Great improvements have been achieved in the biomedical field thanks to the implementation of engineering in the described area.

Image processing and analysis, along with recognition of biological samples are essential activities of the researcher's daily life, although they usually compromise the investigator to spend hours analyzing images. In this context, image classification models become handily: human tasks can be minutely reproduced by machines in less time. Those models are capable of categorizing and labeling sets of pixels -digital images regions- based on their most relevant features [11]. The algorithm is trained with an image dataset and, finished the learning phase, it will be capable to face future situations with new data and classify it.

Image classification models are englobed on the concept of Artificial Intelligence (AI), which is defined as technical discipline based on the use of computers to mimic tasks involving human intelligence [12]. Great benefits derive from the implementation of AI, mostly related to the overcoming of human-centered problems: error reduction, faster decision and processing of bigger data amount are just few examples of this revolutionary technology.

AI can be subdivided into different branches and sub-branches. Among them, we find Deep Learning (DL), the most popular Machine Learning (ML) sub-field [13].

2.2.1. DEEP LEARNING

DL, also known as Deep structured Learning, is described as a branch of ML inspired by human neural networks to interpret data using Multi-Layered Neural Networks [14] -characteristic from which the term "Deep" arises-. Raw data is processed through the multiple stages, automatically extracting features with an increasing level of complexity.

The learning landscape is very complex, englobing four large categories: supervised, unsupervised, semi-supervised and reinforcement learning [15]. Supervised and unsupervised models differ in the type of data training used. The first approach, based on an input-output pair relationship, requires labeled data, whereas the unsupervised approach can learn patterns with

unclassified data. Semi-supervised, as its name suggest, combines the two described methods. Finally, the reinforcement training methodology learns by interacting with the environment, aiming to follow those actions that will maximize the output.

Each class presents a wide range of algorithms: selecting the most optimal one represents a challenge for the programmer. Whether one category is chosen or not depends on different criteria, mostly focused on the kind of problem being faced and the kind of data that we are working with. Having considered the framework of the project and the type of data to be collected -digitalized images-, we decided for the implementation of supervised learning. Furthermore, owing the fact that previously labelling the data would ensure finding out those classes which we are interested in, this model is suitable for our goals.

2.2.2. SUPERVISED DEEP LEARNING ALGORITHMS

In this section we drill down into the three most spread DL supervised algorithms, gathering sufficient information to ease the decision making presented in *Section 4*.

2.2.2.1. MULTI-LAYER PERCEPTRON

The Multi-Layer Perceptron (MLP), considered to be the basic structure of Deep Neural Networks (DNNs), consists of an Artificial Neural Network (ANN) with a fully connected network. Every node in a layer is connected to each node of the following layer with its respective associated weight. It also presents a feed forward architecture [16]: connections are always directed from lower to upper layers, not allowing neurons of the same layer to be interconnected.

The model starts with an input layer, composed by *n* units, and finishes with an output layer of *e* units, responsible for taking a decision respect to the input data -attribute a certain class-. Between them, one or more successive layers of intermediate units are found: the hidden layers [17]. The hidden and output layers commonly use a non-linear activation function -modeling the human neurons behavior, deciding whether to fire a neuron or not-. Highlight that the input layer is fed up with a 1-D vector, aspect that rebounds in the data type which this model can deal with.

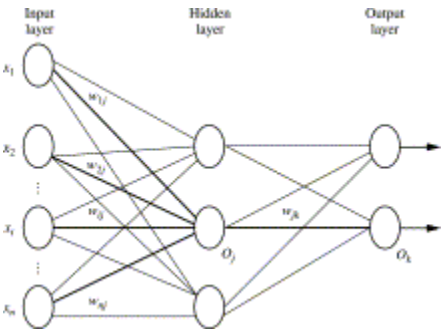


Figure 2. Multi-layer Perceptron with one hidden layer [18].

2.2.2.2. CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Networks (CNNs or ConvNets), a Multi-Layer Perceptron with special structure, are considered to be the most popular DL architecture, as there is no need of feature extraction [19]. In contrast to MLP, nodes do not need to be connected to every following node: layers are sparsely connected rather than fully connected. CNN also differs in the fact that, rather than the 1-D format of MLP, the input presents a 2-D/ 3-D- format [20], working well with data which presents a spatial relationship.

Analogous to MLP, CNN presents hidden layers, which can be split into convolutional and pooling layers. Convolutional layers -or detection layers, given that they function as feature detectors-, are

composed by kernels, which convolve with the input data and create feature maps -the output of that layer-. Pooling layers -or down-sampling layers- compress the previous feature map by computing a simple function over small regions of the image, usually computing the maximum [21]. In this way, the number of parameters and computations drastically decreases while keeping the most important information.

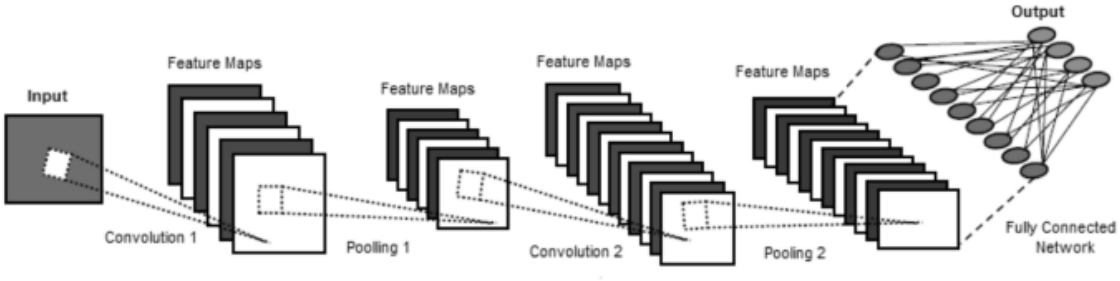


Figure 3. Convolutional Neural Network [19].

2.2.2.3. RECURRENT NEURAL NETWORK

Recurrent Neural Networks (RNNs) are a specific type of ANN designed to work with sequential or temporal data [22]. Contrary to the previous described neural networks, RNNs stand out for their 'memory': data from previous inputs can flow through hidden layers and rebound on the current input and subsequent output. The feedforward architecture is not present anymore: self-loops are now possible and hence, units can feed into themselves [23]. More parameters are present on RNN, directly result of the additional extra weights.

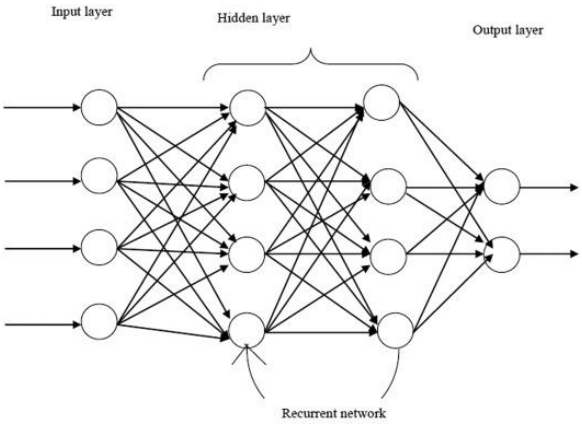


Figure 4. Recurrent Neural Network [24].

2.2.3. MODEL PERFORMANCE EVALUATION

Evaluating the performance once the model has been built and implemented is crucial: we have to ensure a correct predictive behavior. Different kinds of metrics and methods can help to determine the model reliability, pointing out whether the model is valid or not. In this current work we will make use of learning curves and confusion matrix, commonly used in the evaluation of image classification models.

2.2.3.1. LEARNING CURVE

A learning curve (LC) plots the model's execution on a given task as a function of the training time or number iterations/epochs for a given number of training examples [25]. In this sense, one can get acknowledge of which are the changes in the learning performance for those algorithms which learn over time in terms of varying amounts of learning effort.

Commonly, during the ML model training, dual LCs are generated: for each training step, the current state can be evaluated from the point of view of the training or the validation sets. And, according to the evaluation metric selected, multiple learning curves can be created, [26]:

- Optimization LCs: based on minimizing metrics by which the model's parameters are being optimized, such as loss or Mean Squared Error (MSE). Larger numbers reflect more learning.
- Performance LCs: based on maximizing metrics by which the model will be evaluated and selected, such as accuracy or recall. Smaller numbers are a sign of how perfect the training set has been learnt. As it can be seen on *Figure 5*, the accuracy is expected to increase over time, reflecting the successful training.

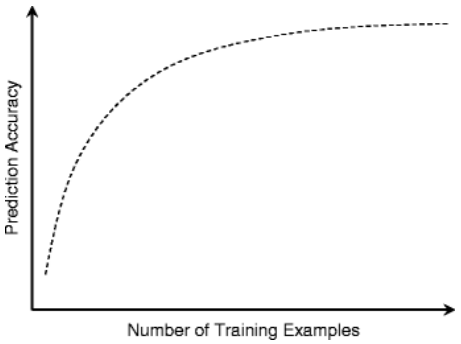


Figure 5. Performance Learning Curve [25].

For the current study, loss and accuracy were chosen for the construction of the LCs, which result to be the commonly used ones for DL model assessment [26].

The dynamics of the learning curve and its resulting shape enable to diagnose the ML algorithm behavior, suggesting which decisions could be taken in order to improve the model's performance. Different scenarios can be encountered [27]:

- Overfitting: The algorithm has captured too well the training data, including statistical noise or random fluctuations of the specific dataset. Therefore, it will be unable to generalize to new data, failing to accurately predict unseen data.
- Underfitting: Contrary to overfitting, the model is uncapable of learning from the training dataset. The model fails to capture the relationship between the input and the output.
- Good fitting: Ideally, one expects to observe a behavior between the above-described situations, resulting in a minimal error on both the training and test data.

LCs results, rather than only depicting the model problems, are useful for proposing further implementations and corrections for future improvements.

2.2.3.2. CONFUSION MATRIX

A confusion matrix is a 2x2 table that summarizes the number of known observations and the predicted outcomes, including both the correct and incorrect predictions. In this way, one can get an insight of not only the number of errors but also at their type.

As showed in *Table 1*, four different scores conforming the confusion matrix can be observed. "TP" stands for True Positive, the number of positive correctly predicted events whereas "FP", the False Positive value, the incorrectly predicted events values. On the other hand, we have "FN" or False Negative, the incorrectly predicted no-events, and "TN" or True Negative, the correctly predicted no-events.

	Negative Predicted	Positive Predicted
Actual Negative	TN	FP
Actual Positive	FN	TP

Table 1: Confusion matrix diagram

From the values present on the confusion matrix, different useful metrics can be computed for the model evaluation, which are accuracy, precision, sensitivity and F1-score [28].

The accuracy is calculated using the following formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Equation 1. Accuracy formula

As shown above, accuracy represents the ratio of number of correct predictions respect to the total number of predictions.

Precision is described as the ratio of correct positive predictions respect to the total predicted positives, that can be computed according to *Equation 2*.

$$Precision = \frac{TP}{TP + FP}$$

Equation 2. Precision formula

Another interesting parameter is the sensitivity or Recall or True Negative Rate (TNR), the ratio of correct positive predictions respect to the total positive examples.

$$Sensitivity = \frac{TP}{TP + FN}$$

Equation 3. Sensitivity formula

Finally, F1-Score or F-measure combines both precision and recall of a classifier into a single metric.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Equation 4. F1-Score formula

As seen in *Equation 4*, F1-score is computed as the harmonized mean (HM) between precision and recall [29], giving equal weight to both measures. The interesting point of this metric is its sensitivity to extreme values, strongly punishing them. This means that if precision and recall differ from each other, HM will penalize more the metric as compared to the arithmetic mean, providing more relevant information as regards to the model performance.

3. MARKET ANALYSIS

3.1. MARKET EVOLUTION

Advances in technology have made possible the widespread of Machine Learning algorithms in many industries, among which the biomedical sector has not been left out. Since the early usage of AI in 1961 [30], the usage of ML systems has exponentially increased in this field, thanks to its polyvalent nature. The applications of AI methods are wide and diverse: big data, diagnostic predictions, image segmenting…

As regards DL, it has experienced a boost in its popularity over the last 5 years, mostly due to its multidisciplinary character and efficiency. Among the areas in which this subfield of ML can be applied, feature representation learning, image segmentation, image registration and classification of anatomical structures have become the fastest-growing areas [31]. In contrast to the classical ML approaches which require hand-crafted features, DL can automatically extract them without difficulties even with large image datasets. Over 6000 studies regarding the use of DL for image classification have been published in PubMed from 2015 to nowadays, demonstrating its interdisciplinary character, efficiency and viability in a wide variety of studies.

After a bibliographic research looking for similar lines of work to the current study being developed, only a scientific article whose approach resembles ours was found, although some are the aspects in which they differ. Hence, given the retrieved information and the project framework, one would venture to say that the current study is one of the first to implement CNN for cell classification based on intracellular structures.

The research group composed by Oei, R. W et al [32] presents a supervised CNN model for human breast-derived cell lines classification using microscope images of intracellular actin networks. More specifically, by means of confocal immunofluorescence microscopy, actin-labeled cell images were captured. The network was fed up with images from three different cell lines -two cancerous cell lines and a normal one, previously immunostained with GFP-. Positive results were obtained, highlighting the potential of CNN to tackle these types of image classification problems, even outperforming human capacity in terms of accuracy.

3.2. FUTURE MARKET

The developed code is meant to be implemented in future studies which bear a resemblance to ours -in other words, to similar experiments carried out with *pombe* cells and the GFP-labelled actin cytoskeleton-. Moreover, it can be seen as the preamble of the extension of the use of CNN -or other Deep Learning classification algorithms- for the classification of cell images according to their fluorescence-labelled intracellular structures in contrast to the usual extracellular morphology-based systematization.

As mentioned in *Section 1*, the image dataset will not be as extensive as we would like, given the time constraints retrieved. The current database could be expanded by collecting more images in a near future and retrain the model, increasing its accuracy. Together with an improvement of the segmentation algorithm, these changes would boost up the model's performance. For this reason,

aiming to perform future improvements, the script and dataset will be publicly available for anyone who asks for it.

4. CONCEPTION ENGINEERING

Manual detection of relevant images features for a posterior classification represents a long-time consuming labor. Therefore, developing methodologies to automatize the process is a priority which has to be addressed. In this context is where Deep Learning shows up thanks to its capacity to extract high-level features from the input data and subsequently classification in an automatic way. Various are the available options and thus, aiming to ease the decision task, this following section is devoted to review the different options for the carry-on of the project as well as the criteria and discussion behind the final option selection.

4.1. PROPOSED SOLUTIONS

Prior to the model construction, one has to evaluate different aspects of the project development aiming to optimize its overall performance. Therefore, three main aspects will be examined: cell mounting approaches, programming software and DL supervised algorithms.

4.1.1. CELL MOUNTING

Since *pombe* cells are hard to trace during live cell imaging due to their non-adherent nature, they need to be immobilized. Different cell mounting techniques were proposed and tested during the project course, aiming to find the most appropriate and optimal methodology for a proper cell visualization and subsequent image acquisition.

In a first instance, the protocol proposed in *Microscopy of Fission Yeast Sexual Lifecycle* by Vjestica A. et al. [33] was tested. Easy to reproduce with minimum material, this method can be easily reproduced in minutes even without previous knowledge on cell mounting. Agarose is used as the supporting material for the cell immobilization. On top of a rectangular glass slide, a drop of 200 µL of liquid agarose is deposited. Deposited the melted agar, another slide is placed on top., which are separated with two spacers. After 10 seconds, the spacers are removed and the top slide is rotated and removed, leaving the substrate ready to be used. Furthermore, if more than one cell specimens are meant to be observed, the pad can be cut with a scalpel to make multiple minipads. For a proper understand of the reviewed protocol, *Figure 6* summarizes the process in a visual way.

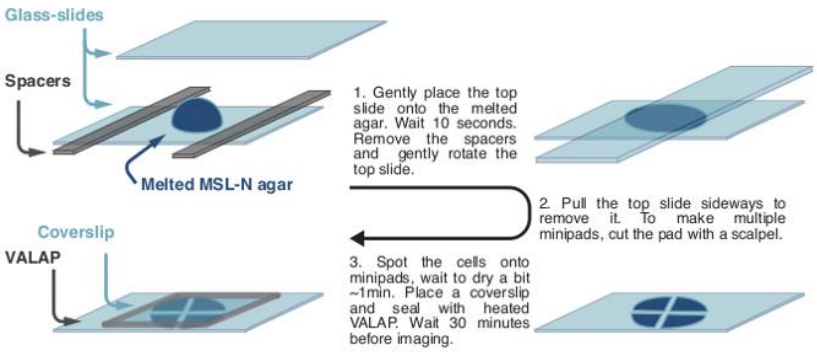


Figure 6. Schematic representation of the protocol proposed by A. Vjestica et al.

The next method tested was the one proposed by Asakawa H. [34], which suggests the use of aqueous concanavalin A or aqueous soybean lectin (SBL) as immobilizers. We decanted for the use of SBL-an antinutritional seed constituent- for the cell fixation owing the familiarity of the biomedical group with this substance. Round microscope slide glass cover slips were coated with 50 µL of 2 mg/mL aqueous SBL, which was uniformly spread all over the surface. After one minute, the excess of SBL was removed and left dry overnight. Once dried, the coated glass cover was ready to be used as the supporting material.

Finally, we thought out an upgraded version of the lectin approach. Instead of using a unique circular glass surface for each specimen, approach which requires preparing one cover slip for each specimen and hence, observing only one cell line at a time, the same procedure could be repeated for an eight chambered cell culture slides was chosen. Hence, the described drawback is overcome and different cells can be observed at the same time.

4.1.2. PROGRAMMING SOFTWARE

For the development of the DL model, many are the software available for conducting such hazardous task. The above options were proposed considering our previous skills and knowledge alongside the facilities offered by each one.

RStudio is an Integrated Development Environment (IDE) specifically designed for R [35], a programming language for statistical computing. The program offers the user the possibility of installing additional user-created packages which are not included in the basic installation, characteristic that extends the basic capabilities of R.

From the point of view of our project, it is object of interest the R Interface to Tensorflow [36]. Tensorflow is an end-to-end, open-source platform specifically designed for ML, with particular focus on DL. Together with the Keras R package [37] -a DL Application Programming Interface or API for fast computerization-, customizable layers and Deep Learning models can be built.

MATLAB [38] is a private open-source numeric computing system developed by MathWorks with its own programming language -M language-, especially designed for the development of algorithms, data analysis, visualization and numerical calculation. The user has available an extensive list of Toolboxes, collections of functions built on MATLAB which may result interesting for a specific task.

DL is also possible with MATLAB thanks to the Deep Learning Toolbox [39], allowing the user to design and implement DL for classification, regression… among other tasks. The examples and pretrained networks present in this toolbox make Deep Learning easy to implement, even without previous knowledge on this field. In addition to this package, the user can import -or even export- networks architectures from TensorFlow and other Deep Learning frameworks, advantageous characteristic since it allows importing pretrained models and weights from other frameworks.

Python is a free programming language, characterized for an easy to learn syntax [40]. Over the last decades, Python has gained popularity as a scientific programming language mostly due to its easy-to-handle nature. In addition, many are users who openly share their State of Art Image

Processing tools, allowing other researchers to implement the code or even adapting it to their own desires.

A vast list of packages and modules are supported by Python, list in which we can find relevant libraries for DL such as the `TensorFlow` and `Keras` libraries. As mentioned beforehand, these two tools make DL easy to handle. Nonetheless, additional libraries require to be installed, such as `Numpy, Scypi` or `Pandas`.

4.1.3. DEEP LEARNING MODEL

As previously mentioned in *Section 2.2.*, given the nature of the data, we decanted for apply supervised learning: the model will be trained with labeled data. Supervised learning algorithms -or any of the other three methods mentioned- can be divided into two types: classification and regression. Classification algorithms predict discrete variables -such as labels-, whereas regression algorithms predict a continuous one.

The model to develop aims to automatically classify *pombe* cells based on their actin cytoskeleton phenotype. Hence, we are facing a classification problem: we aim to obtain a discrete prediction, which is no more than the class to be attributed to each cell.

The models introduced in S*ection 2.2.2*. are all valid for classification problems, although each one presents differential characteristics that make them more suitable for a certain scenario. All of them were equally considered as candidates, and the constrains and advantages of each one were taken into consideration for the final decision, ensuring to choose the most appropriate algorithm.

4.2. FINAL SOLUTION

The above-mentioned candidates were analyzed and discussed from the point of view of our necessities, selecting the options more suitable for the course and development of the project.

Among all the cell mounting methods tried, placing lectin on top of the eight chambered cell culture dishes shown the most satisfactory results. In comparison to agarose, cells remained in place -or were less prone to move-, during image acquisition, with the additional fact that cells tended to overlap less between them, relevant aspect for the subsequent image processing. Hence, using lectin was indisputably the best option. When it comes to practical terms, the eight chambered method overcome one of the previous project limitations: time. Not only stands out for its simple reproduction in short time but also it allowed us to observe several cell strains at once.

Regarding the programming software of preference, although the three options reviewed were suitable for the carry-on of the project, Python was the final selection. The experience gathered with this free programming language during our biomedical engineering degree, together with its easy apprenticeship, made it be the perfect candidate. Some of the libraries needed for the project were already familiar to us since we had previously worked with them in some of our university subjects. Remark that Fiji, although not discussed on the list of proposed solutions, was needed to open the archive containing the cell observations and save the content in a *.png* format, compatible with Python.

CNN was chosen as the Deep Learning model meant to be implemented. Although MLP has been for many years the preferred option for image classification, is nowadays deemed insufficient for more advanced and complicated tasks. MLP has the characteristic of presenting a fully connected network, aspect that makes that the total numbers of weights needed for building the model exponentially growth for large images. Furthermore, its dense architecture makes that easy-to-handle tasks take long time to be solved. CNN can overcome these problems due to the convolutional operation, needing less parameters. Making much deeper networks it is possible owing the fact that layers are partially connected rather than fully connected, aspect that allows the extraction of more complex features for a most accurate classification.

To sum up, Python will be the programming language used for the development of the CNN model based on cell images acquired with an eight chambered cell culture dishes and lectin immobilizer.

5. DETAILED ENGINEERING

The current study was structured in different stages, thoroughly defined below. These tasks can be clustered in six phases, as shown in *Figure 7.*

Figure 7. Detailed engineering flowchart

This section describes in detail how was the proposed solution developed as well as the steps followed to implement it.

5.1. SAMPLE PREPARATION

As presented in the introduction, the student will be in charge of the sample preparation, starting from the cell strain growth and sample preparation to the cell mounting.

Three days were required for the total conduction of the experiment, days in which rigorously adjusting to the settled time was our main priority. The steps had to be accurately reproduced at exact hours for the continuity of the experiment. Otherwise, the procedure could not be conducted -in the *Appendix*, the original experimental design can be found, which was readapted and shortened for practical and temporal issues-.

This subsection briefly introduces the protocol followed, handed by the biomedical research group.

5.1.1. CELL CULTURE

Different *pombe* cells strains were cultured under the same conditions, as decided by the biomedical group, aiming to outline the dissimilarities among them in terms of the actin cytoskeleton behavior and hence, its rebound in the cell cycle.

Prior to the beginning of the experiment, given that cell lines were cryopreserved at -80 ºC, they needed to be thawed rapidly and plated in cell culture dishes with YES media. Cells were left drying at room temperature before introducing them in an incubator during 48-72 h at 37 ºC.

Once observed that cells started growing in the plates, we could proceed to follow the experimental protocol. The following steps were routinely repeated week by week:

- Day 1
    - 9:00 h: Seed cells in 5 ml of Complete Minimal Media (MM). Incubate overnight at 30 ºC in a shaker incubator.

- Day 2
    - 9:00 h: Measure Optical Density (OD). Dilute cultures at an OD of 0.35 in 5 ml of Complete MM. Incubate during 4-5 h at 30 ºC in a shaker incubator.

- 13:00-14:00 h: Measure ODs. Dilute cultures at an OD of 0.025 in 5 ml of Complete MM. Incubate overnight at 30 ºC in a shaker incubator.

- Day 3*:
  - 9:00 h: Measure ODs. If the OD is between 0.8-1, dilution is not necessary. For bigger ODs, dilute at an OD of 0.8.
  - 10:00 – 11:00h: Start preparing cell cultures for sample mounting

*Depending on the day arranged for the microscopic services, rather than the predefined three days, the experimental protocol could be extended until the sample visualization day. If this was the case, day 2 was repeated the day(s) prior to the cell observation -which would be set as the day 3-.

At the end of the week, aiming to preserve cells alive for further experiments, subculturing was required. A few *pombe* cells were transferred to a fresh growth YES medium and introduced in the incubator at 37º during the weekend. In this way, we enabled the further proliferation of the cell strain.

Nonetheless, it has to be said that, despite the experimental part being originally planned as one of the student duties, due to organizational problems and lack of time, the undergraduate could not be in charge of the reproduction of the experiment starting from the second week of February -beginning of the second university semester-. For this reason, a biomedical researcher took their role and accurately followed the described steps.

5.1.2. CELL MOUNTING

On the third day, once the measured OD presented a value between 0.8-1 -or the pertinent dilution was done in order to obtain such value-, cell cultures were ready for the sample mounting. As explained in *Section 4*, the eight chambered slides demonstrated the best results and thus, was selected as the optimal procedure and reproduced in the following days.

The day before the sample observation, lectin was deposited in each chamber following the reviewed protocol and left dry overnight. In each compartment, 100 µl of the cell culture were set down. After five minutes, the excess of liquid was removed and let dried at room temperature. Once the bottom of the chamber was completely dried, 400 µl of media containing LatB at the desired concentration was deposited on it.

Highlight that, as it was mentioned beforehand, the original experimental design underwent some changes as a way of reducing the workload and the time required. For this reason, the four original scenarios -control and three different concentrations of LatB- that were meant to be analyzed were simplified to one concentration. Furthermore, the LatB was deposited few minutes prior to the image acquisition rather than dropping the substance at appointed times.

5.2. DATA ACQUISITION

Owing the fact that the project took place at the Faculty of Medicine and Health Science of the University of Barcelona, we took advantage of the technology owned by the Department of Advanced Optic Microscopy for the present study. This department is ubicated in a different area of the university and therefore, we needed to move to another location. To conserve the

fluorescence of cells, samples were covered with aluminum foil. Otherwise, the actin filaments GFP-labeled could lose their fluorescence upon exposure to light.

Time-lapse images of the different cell strains were recorded during 2 to 3 hours with the confocal microscope Carl Zeiss LSM 880 Airyscan. Equipped with high technology and a multiline argon laser, it offers high sensitivity and resolution at high-speed [41][42]. Owing the fact that we do not have the skills required to manipulate the microscope and set the correct parameters for the data acquisition, a microscope technician was in charge of this task.

For the microscope configuration, the Zeiss ZEN [43] software was used. This user-friendly interface allows adjusting the image parameters to our requirements and preferences. Among the countless functions included, highlight the sample carrier option, characteristic that enabled us to observe the different cell lines at once and in an automatic mode. Additionally, the microscope can capture multiple positions sequentially. In this way, we were able to observe different spots for each chamber in one acquisition. The Zeiss microscope can work with a wealth number of fluorophores. Among them, the GFP fluorochrome, object of interest in our study, is included. For its observation, a specific designed filter for the GFP dye was selected.

Properly focusing the cells represented the most tedious task for the technician. The Z-position of the microscope had to be properly adjusted to the imaging plane of interest. Even though being placed on top of an anti-vibratory table and once configurated the microscope automatically performs the image acquisition, we had to observe that the cells do not move out of focus. Otherwise, the sample should have to be refocused.

Using a 63x oil immersion objective and with an image resolution of 512 x 512 pixels, images were acquired one by one. The resulting pictures were saved as stacks -one per spot- in a *czi format.* The same parameters were reproduced on all the microscopy appointments, whatever the cell mounting approach -as described in *Section 4*- was tested. The several trials performed prior to the final selection of the cell mounting methodology were not in vain. Being rejected as the optimal approach was not link to a direct exclusion from the project. Although it has been described in the previous section the problems observed when testing the methods and the practical motives that also influenced to their rejection, the obtained images presented a quality good enough for the processing and segmentation steps, being used for the dataset construction.

5.3. DATA PROCESSING

Raw images must be correctly processed in order to present to the DL model adequate data which it can work with. This section is intended to describe all the steps followed along the criteria behind each decision.

5.3.1. DATA CONVERSION

Even though Python can handle with (almost) any file formats -including *.czi* files-, we decanted for a more malleable format: *.png.* Raw image stacks were first converted to a *.png* format with Fiji [44], an open-source platform which is basically ImageJ with additional plugins such as Bio-Formats [45], an add-in for reading data in many life sciences file formats.

The fluorescence channel contained the information relevant for our project, which is no more than the actin cytoskeleton. Thus, the transmitted light one was disregarded.

5.3.2. IMAGE SEGMENTATION

Image segmentation, as it was introduced in *Section 1.1.3.,* is described as the process of separating an image into multiple objects -which, in our case, were the cells-. For this purpose, various approaches can be implemented: manual, semi-automatic or automatic segmentation.

It was previously advanced the problems of computer-based segmentation procedures, failing to accurately reproduce human-drawn segmentation. Despite that, studies have demonstrated the potential of semi-automated segmentation [46], significantly faster than manual segmentation and with comparable performance. Even so, constraints were still present, where algorithms often fail to accurately perform the segmentation duty.

Having in mind that time was against us, we decanted for the development of our own semi-automatic algorithm. It has to be remarked that, although the overall results were satisfactory, a subsequent manual check was necessary as a way of eluding the bad segmented images -and correct them if necessary or directly discharge them from the dataset-. All the following steps are gathered in *Image processing.ipynb*.

5.3.2.1. IMAGE MANIPULATION

Performing a prior image preprocessing is required for an enhancement of the segmentation procedure. Until the final code proposed on this project was redacted, different prototypes were redacted and tested. Many were the issues that arose from them but, by means of trial and error, problems were addressed one by one till we find, what we considered to be, an acceptable algorithm for image segmentation.

The algorithm parameters set -as it can be seen on the code attached on the *Appendix*- differed depending on the specific situation. Cell density and movement together with light intensity were the main variables changing among the different samples -or even between frames-, thus requiring the algorithm parameters to be readapted for each observed spot. Therefore, keeping them as separate stacks -or even as sub-stacks, given the stochasticity of the sample environment conditions-, rather than merging all the pictures into a single folder ensured a more accurate segmentation.

`Scikit-image` is a Python library specially designed for image processing [47]. The module `segmentation` contains a wide variety of functions useful for this procedure, list among which we can find the watershed algorithm, a region-based tool derived from mathematical morphology [48] commonly used for regions-of-interest close to each other. Inspired by ridges and valleys, images are treated as topographic reliefs [49].

The green channel was selected for a subsequent binarization, not only for being the less sensitive one to noise but also for containing the information in which we were interested in -the GFP signal-. Since inhomogeneous lighting was observed in images, global thresholding methods are not the best decision to take. Rather than selecting a unique threshold for the entire image, applying an

adaptive threshold for each pixel or region is more convenient. Thus, we applied the Sauvola threshold [50], a local thresholding approach.

Speckle-like noise was observed in all images, artifacts that highly influenced in the performance of the segmentation algorithm. On top of that, cells were holed, direct result of the image binarization process. These aspects were corrected with the `morphology` and `ndimage` modules from the `Scikit-image` library.

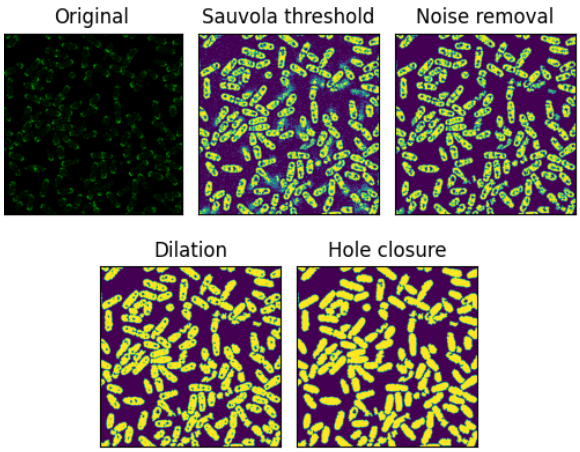


Figure 8. Image processing steps.

As for the markers, we selected the local minima -the valley points- of the segmented objects, from which basins are flooded. To find them, it was first computed the Euclidean distance transform between the cell pixels to the nearest background pixels. Subsequently, the maxima of the distance -which is no more than the minimum value of the contrary of the distance- was set as the algorithm markers.

Over-segmentation issues arose when directly applying the watershed algorithm: some cells presented more than one local minima, phenomena which was misinterpret by the algorithm and spitted them in multiple segments. The problem might be (partially) solved by merging incorrected spilt regions.

To solve this unwelcome situation, we figure out a solution based on Euclidean distances. We hypothesized that the uncorrected split cells, whose origin derives from the erroneously number of markers detected, could be corrected thanks to the information of the neighborhood pixels. Based on the retrieved information, the middle point between the points detected on the same cell would be computed and hence, the split fragments would be merged into one. The procedure applied was the following:

1. Find the centroids of the detected objects
2. Compute the Euclidean distance derived from all the possible combination of pair of centroids.
3. Calculate the Euclidean distance transform and find the maxima distance. Save the results as a coordinate list and as a Boolean mask.
4. From the results obtained on step 2, those points whose distance is smaller than a certain cutoff -value representing the minimum distance which can be considered that two points form part of the same cell- will be selected.
5. Create two subarrays of *n* x *n* pixels -where *n* should be specifically adjusted for the image being processed-, each one centered at one of the pair points from step 5, containing the Euclidean distance transform values.
6. Look at the pixel values. If there are zeros in the neighborhood pixels, the analyzed pair of points are independent. Otherwise, they are part of the same cell.

7. Finally, with those points that fulfill the non-zero condition, the middle point will be computed. The list of local peaks will be actualized: the middle point will replace the original points with which it has been computed.
8. The actualized list of markers will be introduced to the watershed algorithm for the final segmentation.

At this point, one may wonder about the origin of such condition. Imagine two points detected at a certain minimum distance. From them, a window of *n* x *n* pixels centered at each point is created, whose values are no other than the Euclidean distance transform. In this scenario, two phenomena are possible, as represented on *Figure 9*: two points belonging to the same cell -top image- or points from different entities -bottom image-. In case they are independent points, background pixels -valued with zero, black pixels- will be placed between them. Thus, the neighborhood pixels in the Euclidean distance transform array will acquire zero values. This condition is not met when points are part of the same unit, helping us to differentiate the two possible scenarios.

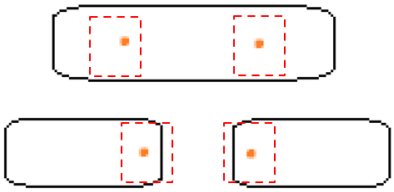


Figure 9. Schematic representation of the scenarios encountered on the over-segmentation proposed solution. Dependent (top image) and independent points (bottom image)

The list of corrected local peaks was therefore used for the final cell segmentation. Around the array returned by the algorithm -the segmented regions-, a rectangular area meant to be cropped was drawn, containing the individual cells. Results were saved on the desired directory as *.png* files.

5.3.2.2. DATA REMOVAL

The results were not exempted of problems, reason why filtering the obtained cropped images was required. During image acquisition, not all cells were captured in the elongated plane or did not remain in place, picturing them in an incorrect plane. Moreover, it was observed that the algorithm often failed to separate adjacent cells, which were mistaken as a single entity. The first situation was out of our control, this phenomenon occurred independently of the protocol followed. However, the second issue may be corrected by particularly addressing them with a subsequent segmentation. Whichever was the situation, these images had to be expressly treated and removed them -or corrected if possible-. The following criterion was applied:

- Scenario 1: Cluster of cells

  Rather than directly removing them, we opted for giving a second opportunity to those images containing cluster of cells as a way of not losing valuable information. The potential candidates for a second segmentation were those ones who meet the following requirements: number of bytes and pixels -height and width- larger than the most frequent values of the cells belonging to a same stack.

The prior segmentation algorithm was applied on them, although the scale difference required adapting the window size of the Sauvola threshold and the minimum Euclidean distance among the detected points to a lower value.

- Scenario 2: Bad captured cells + over-segmented cells

  It was observed that images containing cells in an incorrect plane were characterized for a lower image size -number of bytes-. Therefore, the easiest way to discharge them from the dataset was by stablishing a threshold and remove those ones which do not fulfill the stablished criteria.

  Over-segmented cells were also included in this context. Although optimal results were obtained with our suggested code, there were cases where it failed and thus, some cells remained over-segmented. Owing the fact that their information did not add any value, were not considered for the dataset assembly. The same criteria reviewed in this scenario was implemented on them.

Many were the cases where both reviewed scenarios were combined, implication discussed in *Section 6*. In view of the results and the issues depicted during the process, as a way of ensuring the consistency of the output, images were manually inspected looking for unprocessed or inadequate images -and thus, discharging them-. Furthermore, as required for the supervised approach, images needed to be labeled, selecting only those ones containing illustrative examples of the actin phenotypes of interest.

Finally, highlight that not all individual entities were used for the dataset construction. Working with time-lapse fluorescence microscopy, differences between frames may be insignificant. If one used all the retrieved information for the data frame construction, the algorithm would be feed with "duplicate" images: bias would be introduced in the dataset. Therefore, images obtained from successive frames were manually compared, looking for "identical" entities and discharging them.

By the end of the process, from the total original number of 6529 images, after the selection and exclusion stages, we ended up with 2117 individual cells.

5.3.2.3. IMAGE ENLARGING

As required by the neural network to work properly, images need to present identical dimensions. Therefore, they required to be resized while preserving the original aspect ratio, which represents a daunting challenge.

Two approaches can be implemented to address this problem: zero-padding or interpolation [51]. By means of the first technique, images are enlarged by adding borders of zero pixels around them until achieving the desired size. On the other hand, interpolation resize the image by adding new pixels whose intensity value is based on the surrounding pixels. Zero padding was used in our project, mainly due to two reasons: in comparison to scaling, the risk of deformation is null and furthermore, computationally speaking, is less expensive. Additionally, extra zero pixels do not

entail any kind of information relevant to distinguish between labels, as there is no relation between the extra black background and the bulk image

Taking into account the rule of thumb of Machine Learning, one could consider resizing images to 256x256 pixels, as most researchers do. Following the advice of Jeremy Howard, a data scientist founder of *fast.ai* [52], we rather decanted for smaller dimensions. Given the dimensionalities of the raw cropped images and aiming to minimize the padding area, we considered that an adequate ratio would be 150x150 pixels.

5.4. DATASET CONSTRUCTION

A well-structured dataset is essential for the correct Deep Learning model development. Its success strongly depends on the quality and size of the dataset. Therefore, the resulting processed images have to be properly manipulated and modified, structuring the data according to the algorithm requirements. This section includes the steps followed for the sake of properly constructing the dataset meant to be feed into the DL model, code which can be found in the *Dataset construction.ipynb* Python script.

5.4.1. DATA SPLIT

Prior to the train-test data splitting, as we decanted for a supervised learning, manually categorizing the segmented data was mandatory. Time-intensive, supervised Machine Learning can be a burden for many, requiring hours in front of the computer for the correct categorization of the images. Nonetheless, one will be future rewarded by the clarity of data and the specific definition of the classes to be obtained. Images were analyzed one by one, attributing them a label according to the observed actin arrangement, and stored on different folders according to the observed phenotype:

- Non-dividing: interphase cells -G1, S, and G2 phases-. Actin filaments are found to be dispersed and disorganized, forming an entangled network [53].
- Dividing: mitotic cells. Arrangement and organization of the actin filaments, where it can be observed the formation of the actin contractile ring [53].

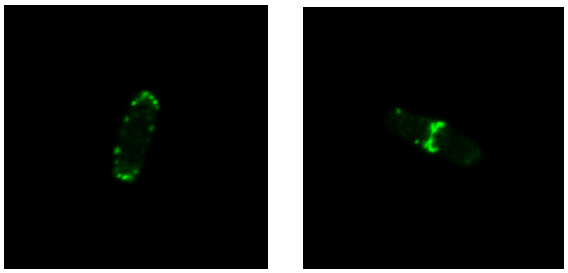


Figure 10: Examples of the data labeling. On the left, non-dividing cell. On the right, dividing cell.

Class imbalance [54] has been reviewed as one of the most common problems in Machine Learning, especially in classification problems. An imbalance occurs when one or more classes, the minority groups, present significantly fewer samples as comparison to the other classes, resulting in unbalanced proportions. The target classes probabilities are dissimilar and therefore, the majority group will be overclassified whereas the minority one, misclassified. In light of the above, with the main purpose of overcoming this issue, we ensured that both classes were equilibrized.

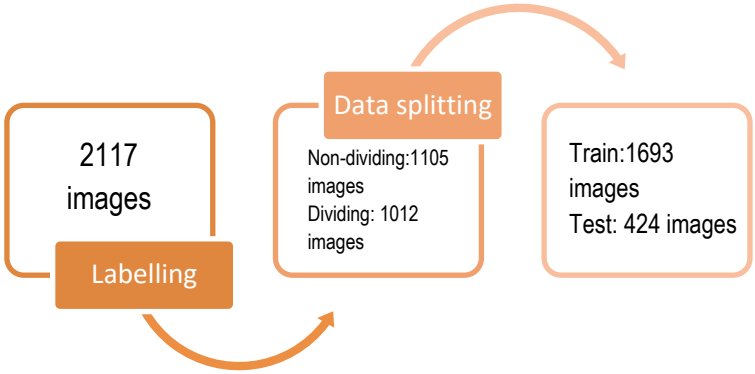


Figure 11. Schematics of the data before and after the train-test split.

Figure 11 shows the data splitting results, starting from the data labelling to the train-test split. Images were arbitrary distributed according to the traditional 80/20 rule: the first 80% of it into the training set and the remaining ones, to the test set.

5.4.2. DATA AUGMENTATION

Provided that neural networks are usually feed with thousands of images during the algorithm training -expectations that were not meet in our project-, more data was required to maximize the model performance. In this context, data augmentation, a set of techniques that artificially increase the amount of data, becomes handy. Different are the techniques available to deal with the lack of training data, including basic image manipulation, kernel filters, random erasing… [55]. Despite their origin, they all pursue the same objective: increase the diversity and number of the training set.

Prior to the augmentation phase, the data had to be properly manipulated. First of all, from each set, X and Y were defined, items corresponding to the input -the images or features- and output -the labels- variables, respectively. Once defined, the `ImageDataGenerator` [56] function -part of the `TensorFlow` library- was therefore used, generating bundles of synthetic tensor images in seconds. Each image was iterated several times, generating four copies of the original image randomly rotated, flipped or brightened. By the end of the process, the starting number of training images was incremented to 8465.

As required by the Keras framework, both the training and test sets were stored in a `NumPy` format, one of the foundational tools in ML [57]. Saving the information as arrays makes the subsequent computations involved in the CNN fast to compute, speeding up the process.

5.5. INITIAL MODEL DEVELOPMENT

After the data processing and dataset construction, the following steps focused on the DL model development. Although the data was ready to be used, there was a crucial aspect to be addressed: deciding the model architecture. This section gathers the course followed and the decisions taken for the final model structure to be implemented, information collected under the *Bayesian optimization.ipynb* script.

5.5.1. NORMALIZATION

As part of the Machine Learning data preparation -although not always mandatory- the dataset has to be translated into a common scale, generally ranging from 0 to 1: in other words, data needs to be normalized. Therefore, one can avoid disproportionate weight assessment to some parts of the system and, subsequently, incorrect predictions.

Normalization techniques are wide and diverse, all of them pursuing the same goal: transform data features to a similar scale. In our case, we rather decanted for the manual classical approach, which can be applied with *Equation 5.*

$$x_{norm} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

Equation 5: Min-max normalization formula

Where x is the image array, x_i corresponds to a particular pixel, min(x) and max(x) are the minimum and maximum values in the array and x_{norm} represents the resulting normalized pixel.

5.5.2. DATA SHUFFLING

Randomly mixing data up, the so-called data shuffling, was the final step of the dataset manipulation procedure. The training set was rearranged with the main of goal of reducing the risk of overfitting: if we were about to train the model with a dataset sorted beforehand with a certain order, the model may expect to see the same distribution of features in the test set, leading to a misclassification of the data.

Thanks to the function `shuffle` from the `Scikit-learn` library, the training set -both the features and labels, while keeping them correlated- was randomly shuffled.

5.5.3. HYPERPARAMETER OPTIMIZATION

Hyperparameters, contrary to model parameters, have to be manually determined and tuned before the model training. Consequently, the performance of the algorithm will be strongly rebounded by the decision taken.

As they cannot be estimated from the data, assessing which are the best values may be challenging. In the literature, different approaches have been proposed for hyperparameter optimization, list among which one can find the Bayesian optimization, procedure applied in our project, as described below.

5.5.3.1. CNN ARCHITECTURE

Prior to the model tuning, defining a function for the CNN structure was required, in which the arguments to be called are the hyperparameters to optimize. Regarding the current work, the dropout rate -referred as *drop*-, the number of output filters in the convolution -referred as *filters*- and the number of hidden layers -referred as *hidden layers*- were selected. As it has been reviewed in literature, special importance has been given to these hyperparameters, given their great influence on the NNs efficiency [58].

Regarding the architecture defined, we rather decanted for the typical structure, consisting of the input layer -with dimensions equal to the cell images- followed by several convolutional -3x3 kernel- and pooling -2x2 kernel- layers. Subsequently, we added a flatten and a densely-connected NN layer. To prevent overfitting, a prior dropout layer was added before the output layer.

As regards of the activation function in the hidden layers, whether we select one or another will control how well the model learns and performs. Rather than opting for the conventional sigmoid activation function, we went for the Rectified Linear activation Unit (ReLU) function, as is nowadays commonly applied [59]. Furthermore, there was another reason behind such decision: when facing a multilabel classification with mutually exclusive data, the sigmoid function often fails in, scenario in which the ReLU function works properly.

When it come to terms of number of epochs, we have to take in mind that a high number can lead to overfitting while too few can drive to the opposite scenario, underfitting. For the sake of avoiding such hazardous situations, the early stopping method was used to finish the training once the model performance stops improving on the validation set.

5.5.3.2. BAYESIAN OPTIMIZATION

Regarding the hyperparameter optimization, the Bayesian Optimization (BO) technique, an iterative global optimization approach, was implemented: it combines prior information about the unknown function to optimize with data from the sample, deducing the optimal value to enhance the model performance. This tool is commonly applied in Machine Learning algorithms, thanks to its capacity to speed up the model selection in comparison to other optimization algorithms [60].

As for the current work, hyperparameters were searched with the following settings of ranges: *drop* (0-0.9), *filters* (2-25) and *hidden layers* (0-6). Those ranks were chosen according to our own criteria and the observations during the student's programming self-learning, where we noticed that many were the toy examples whose optimal working conditions were set on those values.

As a way of getting a bit more clarity on the influence of the evaluated hyperparameters on the model performance and final accuracy, results were visually displayed as a pairplot, as shown in *Figure 12.* The non-diagonal figures -the 2D scatter plots-, show a bivariate relationship, whereas the diagonal ones, a univariate distribution, depicting the marginal arrangement of the data. There was not a trend when it comes to the hyperparameter: different accuracy could be obtained for the same -adjacent- number. That is to say that is the hyperparameter values combination which rebounds in the model's performance rather than a hyperparameter by itself.

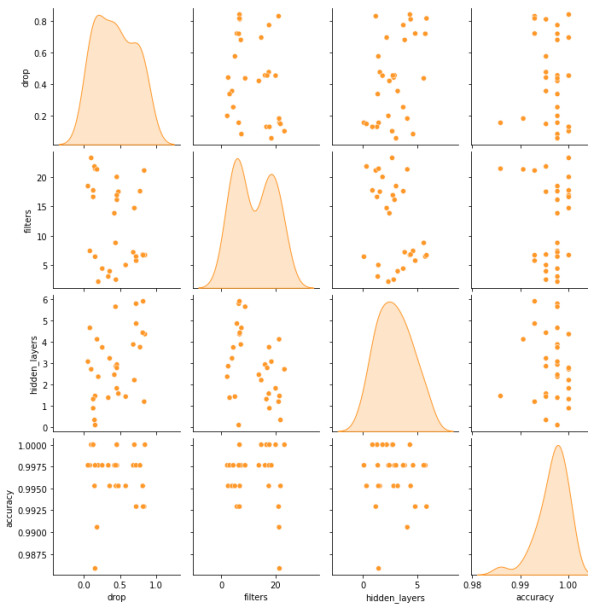


Figure 12. Hyperparameters pairplot.

Among the list of the parameters probed, the ones that maximize the model's performance were chosen, as it can be seen in *Table 2*. As for the filters and hidden layers hyperparameters, the values were rounded, since they require to be integer numbers.

	BEST COMBINATION	APPROXIMATED VALUES
Drop	0.84	0.84
Filters	6.7	7
Hidden layers	4.35	4

Table 2. BO optimal results. First column contains the best combination obtained. Second column contains the rounded values introduced on the model.

5.6. FUNCTIONAL MODEL

Once the optimal hyperparameters were found, we could proceed to the final model construction. The predefined CNN model was first adjusted with the best configuration obtained from the BO. After this, the already prepared training data -which was priorly normalized and shuffled, as it was introduced in the above section- was introduced to the model so it could learn from all the labelled images. In this way, the ConvNet algorithm would be able to extract the most relevant image features for a future classification. Subsequently to the model fitting, part of the training data -priorly selected via the hold-out method- was used to provide an unbiased evaluation of the model fit on the training set [61], estimating its skill and classification ability.

Finalized the training and validation steps, the model performance was checked with the test dataset, so it could predict the labels of each image inserted and compare its results with the actual labels. The resulting output together with the metrics that could be derived from such information, were therefore used for the final assessment of the model's predictive ability and generalization capability, as discussed in *Section 6.* Results are given with probabilities, the probability of belonging to a certain class. This means that two scores ranging from 0 to 1 will be predicted, one per each class. To ease the work, the function `argmax` from the `NumPy` library was implemented in the results, operation that finds the class with the largest predicted probability. In this sense, instead of being returned with a vector of probabilities, we are retrieved with a number -0 or 1, which corresponds to *Non-dividing* and *Dividing*, respectively-.

To conclude, the already functional model was saved for future implementations. The model architecture and weights were separated into two different files, whose formats differ according to the stored data: .j*son* and *.h5* formats, respectively.

5.7. PYTHON MODULES

In order to perform all the above-described steps, several Python modules have been used. One will notice that some of them were earlier introduced in the previous sections, given their relevance on the course of the code, while others, which were overlooked, will also be briefly described down below.

As for the initial image processing -and commonly used in the different python scripts-, the modules `cv2`, `itertools`, `numpy`, `math`, `matplotlib`, `os`, `glob`, `skimage`, `scipy` and `PIL` have been used. Regarding the `os` and `glob` modules, they allow the interaction with the operating system and thus, they are useful for file management The `numpy`, `matplotlib` and `PIL` packages were used to open and manipulate image as arrays. Together with the `scipy` and `skimage` modules, which contain mathematical and geometrical functions, images were processed. As for `cv2`, an OpenCV -Open Source Computer Vision Library- interface, its versatility allowed us from interacting with images to their automatically segmentation.

Concerning the DL model, the `keras` and `tensorflow` modules provided the framework required for the CNN construction as well as for the obtention of all the evaluation metrics. The Bayesian Optimization could have not been done if it was not for the `bayes-opt` package which, in addition to the global optimization function, contains several properties to make the hyperparameter optimization more malleable.

As regards `seaborn` and `pyplot` modules, they allowed a non-complex and instinctive data visualization for the results display and image observation.

6. RESULTS AND DISCUSSION

By the end of the above-mentioned methodology, we ended up with a model whose reliability needed to be evaluated in detail before giving by approved its final validity and prediction capacitance. Hence, *Section 6* includes the obtained results and evaluation metrics of the performed work, as well as a brief discussion of the outcome, including both the project's implications and limitations.

6.1. LEARNING CURVE ANALYSIS

Learning curves are visual tools that underline the learning progress of a ML model, exhibiting the progress overtime of a certain learning metric during the training and validation steps. Thus, they represent a powerful instrument for problem depiction and the optimization of the prediction performance. As it was previously introduced, the most popular evaluation metrics -accuracy and loss- were selected for the model evaluation.

6.1.1. OPTIMIZATION LEARNING CURVE

Accuracy, described as the number of correct predictions out of all the data points -usually in percentage-, has been, by further, the preferred metric for model validation due to its easy-to-understand nature and facile interpretation. Epoch over epoch, both training and validation accuracy values should increase, indicating that the model can capture the data complexity. By the end of the process, the accuracy value is expected to be as high as possible, close to 1 -scenario in which the model is able to correctly predict all the values-.

The Optimization LC of our model can be observed in *Figure 13*, highlighting the monitored accuracy during the training and validation phases. At a first glance, it can be perceived that there is not a no perfect fitting. Partially true, the training and validation accuracy are separated by a gap, phenomena by which one might hypothesize that we are facing a case of overfitting, implying that the model will perform poorly when predicting unseen data. Nonetheless, if we take a closer to the accuracy scale, it is noticeable that the existing gap is at the order of ~0.01. Hence, our speculations were neglected: it was concluded that the model successfully captured the data complexity, although further analysis are necessary prior to taking a final resolution.

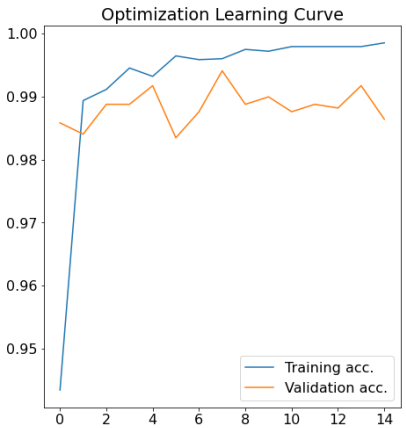


Figure 13. Optimization Learning Curve.

6.1.2. PERFORMANCE LEARNING CURVE

Together with accuracy, loss is the second most well-known metric in Machine Learning. Unlike the above-described metric, which was given in percentage, loss is a real value, the summation of the incorrectly number of predictions in the model. The retrieved information is slightly different in comparison to the previous metric: depict how poorly (or well) the model fits the training and unseen data. Contrary to accuracy, its interpretation is not straightforward: according to the problem being

faced, the loss tolerance might vary. Nonetheless, it is preferable that the loss value is minimized over the training experience.

In an ideal situation where the model generalization ability is high, the training and validation loss values are reduced until reaching a stability point with a minimum gap. Nevertheless, constructing a model which perfectly predicts new incoming data is in practice rare to be achieved. The model loss is usually well-nigh lower on the training set than the validation set, situation where is expected to be observed the generalization gap -space between the different loss LC-.

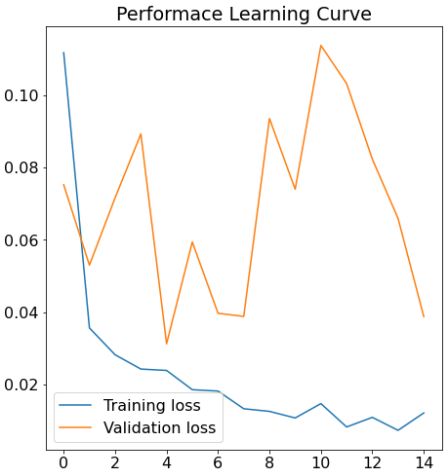


Figure 14. Performance Learning Curve.

As it can be observed in *Figure 15*, further from the desired situation, the retrieved loss values differ: the good fitting scenario is out of our reach. Despite these discouraging results, situation in which one would be prone to give up, there is still light at the end of the tunnel. The validation loss by itself cannot be generalized as the overall model's performance, given that, as it was recently stated, different interpretations can be given to results according to the point of view adopted. Hence, the loss metric should be validated together with another complementary metric, such as the confusion matrix. Furthermore, it has to be said that the observed spikes do not seem to be an insight of over-fitting: in comparison to the scale graph, they are not terribly large.

6.2. CONFUSION MATRIX ANALYSIS

The above-described metrics by themselves do not provide enough information for the final verdict of the constructed model. Commonly misbelieved, accuracy and loss are thought to be sufficient for the model robustness evaluation, despite they can give some clues of the model behavior, these metrics by themselves do not provide enough information for the algorithm assessment. Contrary, many are the variables that came into play in the model validation and thus, a deeper analysis of the resulting output is necessary.

Confusion matrices, a special type of contingency tables, result handily in this context: the predicted category labels versus the true labels are presented in a visual manner. As a way of expanding the algorithm performance analysis, the confusion matrix of the tested set was obtained, as shown in *Figure 15.* From the total 424 samples tested, only 6 out of the total number of images -where incorrectly predicted: *Dividing* samples were mistaken as *Non-dividing*. The algorithm incorrectly identified them and hence, failed to attribute the correct label. Finally, the evaluation metrics -accuracy, precision, recall and F1-score-, derived from the confusion matrix, are presented in *Table 3*.

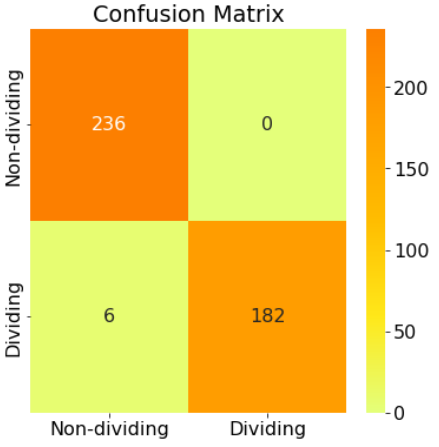


Figure 15. Confusion matrix of the test set.

As can be seen in the attached table, the resulting evaluation metrics shown that the model is close to the desired scenario: a perfect classification algorithm. Far from dictating its reliability in future case scenarios, the model is not yet exempted of problems. Obtaining such perfect results could be exclusively of the dataset used -in other words, the model works only well with the data frame used but not with new data- and thus, they cannot be extrapolated to general cases. Aiming to give the ultimate verdict, it was necessary to validate the DL model in a hypothetic genuine situation with unseen samples before deploying it.

Evaluation metric	Result
Accuracy	0.985
Precision	1.000
Sensitivity	0.968
F1-Score	0.983

Table 3. Model evaluation metrics

6.3. FINAL TESTING

The final aim of the developed algorithm is to predict the phenotype of the cells. Aiming to assess its generalization capacity, a set of images previously set apart from the training and test sets were introduced into the model. For the final validation, the model predictions were contrasted with the actual classes of the images, previously labeled by hand.

The confusion matrix obtained, as can be observed in *Figure 16,* demonstrates that the model's prediction capacity is not thoroughly precise, especially when it comes to identify the positive results -*Dividing* label-: 27 out of 261 samples were misclassified. As per the evaluation metrics derived from it, attached in *Table 4*, although the overall output has worsened, the sensitivity parametric has been the main rebounded -as it should be considering the model's difficulty to label *Dividing* entities-.

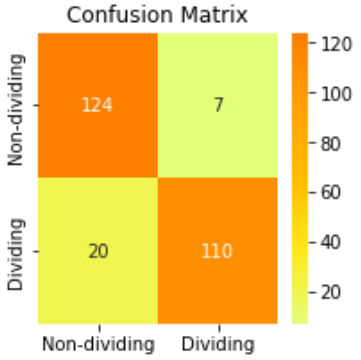


Figure 16. Final testing results: confusion matrix.

Evaluation metric	Result
Accuracy	0.897
Precision	0.940
Sensitivity	0.847
F1-Score	0.897

Table 4. Final testing results: evaluation metrics.

Given the nature of ML algorithms -referred as *black boxes* since one is unable to fully understand their complexity and how they work together with the interpretation of the results-, identifying the root of the problems that one may encounter is a tough task to address: their origins are immensely diverse.

In light of the retrieved information, this displeasing situation might be treated as a potential case of imperfect data [62]: the constructed dataset does not entirely capture the real scenario in which the algorithm is mean to be used. Enough number of perfect annotations might lack and hence, the model fails to label unseen data. First and foremost, we are concerned with the quality and selection of the retrieved data: focusing on carefully choosing representative images, we might unintentionally pick images whose features are similar among them and thus, shielding the variety of contractile ring. Furthermore, data could be biased in favor of a specific actin ring shape, underrepresenting the rest of architectures and failing in the classification task.

This section just presents a toy example putting in practice the developed algorithm, but it proves the potential of Deep Learning in the current image classification problem. Given the uncertainty of the constructed algorithm, although the overall outcome could be considered satisfactory, the resulting classification model cannot be taken as the absolute truth but as a complementary tool for investigators.

6.4. IMPLICATIONS

Regardless of the high quality of the captured data in terms of image resolution, and the great number of images taken, the following processing and selection significantly reduced the dataset and hence, decreasing the variability of images. The data variation is also restricted by the underlined problems during image acquisition, such as cells captured in an incorrect plane, clusters of cells… and thus loosing valuable information and underrepresenting the diversity of cell phenotypes. The simple but efficient segmentation algorithm did occasionally overlook some of the cells present on the picture, aspect which also rebounded in the total number of cropped cells. Nonetheless, we ensured selecting a representative dataset, where the phenotypes of interest were perfectly identifiable.

The high scores resulting from the training phase evidence the successful learning stage, where the model has been capable of extracting those features considered relevant for the classification. However, as the confusion matrix results point out that the model might be "so perfect" that fails to generalize. With the additional information extracted from the final testing stage, the constructed model was far from being trustworthy. For the sake of obtaining more reliable predictions and improving the model performance, more images should be included in the dataset, ensuring to cover the wide variety of cell phenotypes of the classes to be predicted. Even more, it would be interesting to reconstruct it and accurately select those images relevant for the model learning and discharge the ones not relivable enough for a robust data frame. To do so, it would also be interesting to partner with an expert on the field to supervise the work and ensure the consistency of the dataset.

Given the reasons described, the constructed algorithm needs to be considered as a potential tool with still room for improvement rather than taking for granted the results obtained.

6.5. LIMITATIONS

Project limitations are described as constraints and handicaps recognized during or even previously to its development, which cannot be covered since they are beyond our possibilities. Uncapable to confront these challenges, some of the original goals were impartially solved -especially when it comes to the dataset construction goal-. Owing the fact that the solution is dependent on data processing and model construction, it would be expected that most of the limitations arise from the initial data acquisition and following processing.

The introduced basic but efficient segmentation methodology, although originally thought to be automatic, ended up being a semi-manual approach. Images required to be treated as individual entities even though belonging to the same stack, requiring finding by hand the optimal code parameters that would achieve the best segmentation while minimizing the loss of information. If one was luckily enough, those parameters might be found in the first or subsequent trials, although

this scenario rarely occurred. Continuing in the same line, image acquisition was itself a challenge. As mentioned beforehand, the project participants mutually agreed when it comes to the image environmental parameters, seeking for the adequate conditions that would enable to fulfill their personal goals. Nonetheless, stochasticity was against our favor. Cells not properly fixed, in an inadequate plane or placed at the edge of the image, together with clusters of cells, were common issues detected in the captured images, rebounded in the total number of segmented cells and thus, reducing the dataset size.

Finally, not originally contemplated to be a restrictive factor for the project execution, being technology-limited constituted a stone on the road. When handling with large amount of data and facing Big Data analytics, a multicore processor CPU may be preferred due to its high processing speed, tangible out of our hands. As for the current project, using the technology available, the running time could range from minutes to hours depending on the code complexity, enlarging the software development and testing period above the initial planned time.

7. EXECUTION SCHEDULE

For the correct development of any kind of project, previously defining the tasks and milestones meant to be achieved for the accomplishment of the initial objectives in time are mandatory. The following section is devoted to cover this aspect, as a way of keeping track and structuring the different project phases.

7.1. WORK BREAKDOWN STRUCTURE

A Work Breakdown Structure (WBS) is a technique that 'deconstructs' a project in a hierarchical way. The project is broken down into smaller and manageable tasks -or work packages-, helping to determine the essential elements of the global project [63]. In this way, the project workflow can be managed appropriately, enabling a good organization and progress.

In our case, the final degree project WBS was constructed with three levels of detail:

- Top level: the overall scope of the project, which, in this case, is the development of an image-based Deep Learning model for *Schizosaccharomyces pombe* cell cycle phase classification.
- Second level: the main phases of the project, starting from a more a general view to a close-up focus on the project specifications.
- Final level: the work packages, obtained by fractioning the second layer blocks.

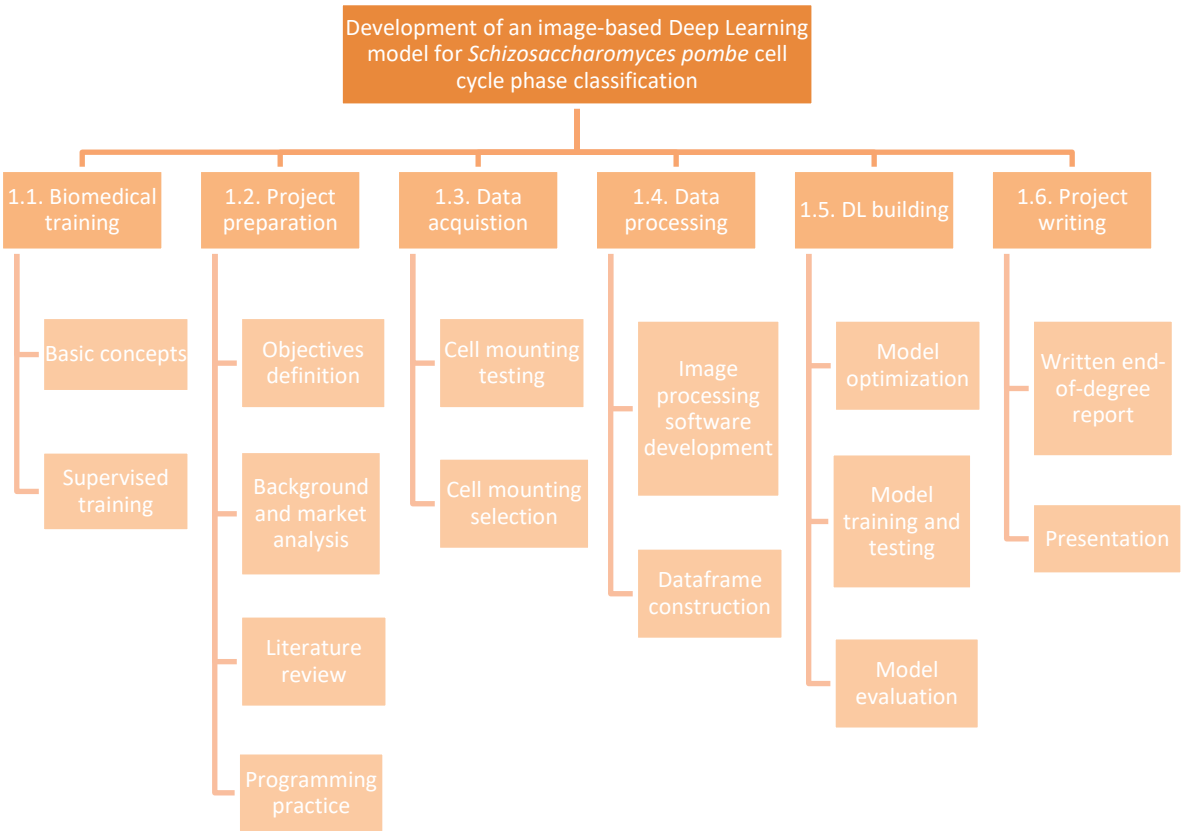


Figure 16: Work Breakdown Structure of the end-of-degree project.

7.1.1. WORK BREAKDOWN STRUCTURE DICTIONARY

A WBS dictionary includes a brief definition of all the project tasks and subtasks, as is presented below:

1. BIOMEDICAL TRAINING: Understanding the framework of the study with the aim of tackling the project's problem and think out the best approach to addressed it.

1.1. Basic concepts: Introduction on the subject and familiarization with the biomedical research field and basic concepts.

1.2. Supervised training: Practical sessions under the mentoring of a formal biomedical researcher. The student will learn to reproduce the experimental protocol by themselves.

2. PROJECT PREPARATION: Defining the basis of the project, previously reviewing all the basic elements and relevant aspects for the project execution.

2.1. Objectives definition: Description of the objectives to be achieved at the end of the project

2.2. Background and market analysis: Analysis of the project background, and the trends and environment of the market in which it is enrolled.

2.3. Literature review: Reading articles on both image processing techniques and Machine Learning models to gather information.

2.4. Programming practice: Putting in practice the reviewed concepts. Practice with Python and toy datasets, learning the concepts required for the development of our own data frame -including the prior image processing- and DL model.

3. DATA ACQUISITION: Observation of the *pombe* cells and image retrieving with a confocal microscope.

3.1. Cell mounting testing: Examination and testing of the proposed data acquisition methodologies.

3.2. Cell mounting selection: Final assessment of the optimal cell mounting method. Reproduction of the approach for the following data acquisition.

4. DATA PROCESSING: Analysis and processing of the digitalized images as well as properly manipulating them for a proper segmentation.

4.1. Image processing software development: Data examination and redaction of the image processing code -cell segmentation and filtering-. Testing and correction of the proposed code.

4.2. Dataset construction: Assembly of the image data frame meant to be feed into the model. Manual labelling and train-test split.

5. DL BUILDING: Construction of the cycle phase classification model for *Schizosaccharomyces pombe* cells.

5.1. Model optimization: Initial development of the DL algorithm structure. Implementation of Bayesian Optimization for the hyperparameter tuning.

5.2. Model training and testing: Learning phase with the training and hold-out validation datasets. Testing of the model with the test dataset.

5.3. Model evaluation: Obtention of the evaluation metrics and assessment of the model performance. Exemplification on unseen data for the final model's verdict. Results interpretation and drawing of conclusions.

6. PROJECT WRITING: Project wrap up. Redaction of the project' report and development of a brief presentation.

6.1. Written end-of-degree report: Writing of the final report of the project

6.2. Presentation: Realization of the project presentation

7.2. GANNT DIAGRAM

Once we had defined all the tasks and sub-tasks for the accomplishment of the initial goals, a timetable must be created as a way of assessing the deadline for each task and keeping track of the project's progress.

For this reason, a GANTT diagram or chart, a visual planning tool of displaying the timing of tasks [64], was created. As seen in *Figure 17,* the scale time is in weeks. Each task is represented with a bar: its position and length summarize the duration of such activity, including booth the starting and ending days.

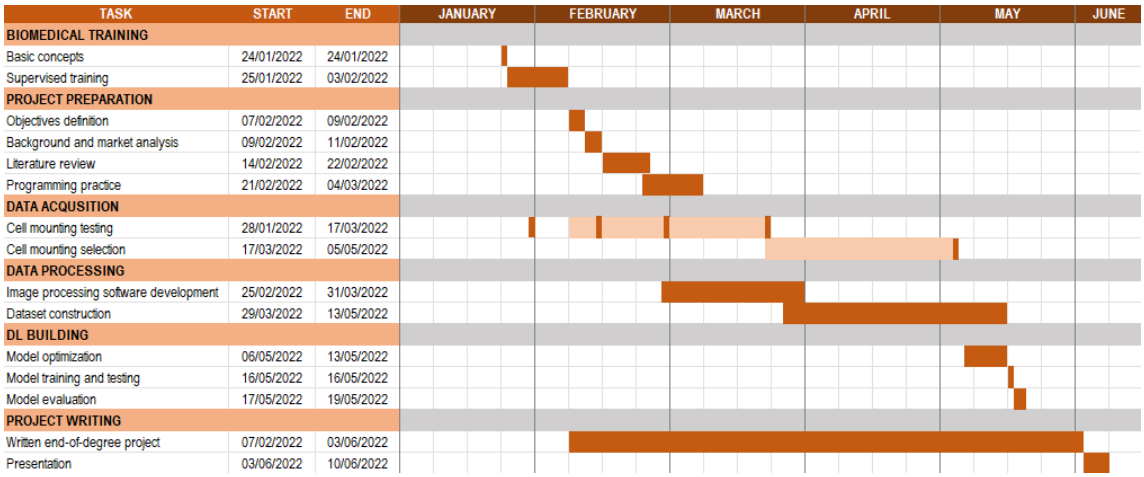


Figure 17. GANTT diagram.

In the GANTT chart, the extent per weeks of the different sub-tasks is embodied. The dark orange palette represents the activities performed by the student, whereas the light ones, the tasks where a biomedical researcher took their role for the carry-on of the experimental part.

Data was collected for 4 days -2-3 hours/day- spread over 12 weeks. Except for the first day, which was only an observation trial done with the fluorescence microscope owned by the Biomedicine Department, the reaming days were previously appointed with the Advanced Optical Department. Thus, the images acquired on the 28th of January were not included on the dataset -mainly to reasons of low image quality-. Over those weeks, both the biomedical researcher and the student worked together: while the first one was in charge of the following on of the experimental protocol, ensuring that the cells grew in the adequate conditions, the student performed the cell mounting and supervised the data acquisition.

8. TECHNICAL VIABILITY

A detailed analysis of the project technical feasibility is presented in this section. Both the internal and external aspects that might rebound in our work are reviewed, enabling us to determine the progress of the project together with its final success and detail those aspects which could be improved.

8.1. TECHNICAL CONSIDERATIONS

The final degree project, as it was mentioned in previous sections, was conducted at the Department of Biomedicine with place at the Medicine and Health Science Faculty of the University of Barcelona. Regarding the technological instruments for the image acquisition, we had to shift to the Advanced Optical Microscopy Department, ubicated on the same building. Except for the confocal microscope, all the necessary material, facilities and equipment for the carry-on of the experimental part were provided by the biomedical research team. In light of the above, the whole project was technically feasible thanks to the facilitations provided by the Biomedicine Department.

8.1.1. SWOT ANALYSIS

The SWOT analysis, one of the most well-known project management tools, enables to identify, from a positive and negative point of view, the internal -Strengths and Weaknesses- and external -Opportunities and Threads- factors affecting the project. Bearing this in mind, one can obtain a general overview of the solution but also identify those details that make it stand out together with its weak points.

	POSITIVE	NEGATIVE
INTERNAL	STRENGTHS Semi-automatic segmentation Previous Python experience High-tech microscopy	WEAKNESSES Time-consuming dataset construction No previous experience in DL modeling Only appointed days for image acquisition Segmentation problems
EXTERNAL	OPPORTUNITIES DL models not yet implemented in this area of work No similar studies have been reported	THREATS No similar studies to compare results with Database expansion depends on the number of similar future experiments

Figure 18. SWOT analysis.

- STRENGTHS

When it comes to the strong points of the project, both the student previous experience in Python and the high-tech microscope used for the image acquisition equally favor the programming of the semi-automatic segmentation code.

The development of a segmentation algorithm is not a straightforward task. Starting from a robust base, which is not more than our previous knowledge on programming and how to tackle segmentation problems, can help to address this issue. With the addition of the high-quality images, -which could not be obtained if it was not for the use of the high-tech microscope-, constructing a segmentation algorithm is now less challenging.

- WEAKNESSES

Prior to the project development, the student had never constructed a Deep Learning algorithm. Additionally, when it comes to terms of the project core elements, time constraints arose. On the one hand, data could only be acquired on prior pointed days. On the other hand, processing it and -partially- correcting the segmentation problems for the following dataset construction required hours in front of the computer to an extent of weeks for its final development.

- OPPORTUNITIES

The untapped potential of DL models in this area of work is suited to tackle the described project's problem. Automatically labelling individual cells helps researchers to save time, critical parameter in many investigation studies.

Furthermore, given that not similar studies have been identified, we were grated with the opportunity of becoming one of the firsts to open the door to further implementations of DL algorithms in similar experimental approaches and unravel its power.

- THREATS

Because not similar studies have been reported, the model outputs could not be compared and thus, results could not be compared for a more robust validation.

Finally, as described in previous sections, CNNs require a large image dataset, aspect not fulfill on our project due to time constraints and the image problems described in *Section 6*. As a way of increasing the model's accuracy, enlarging the image dataset is desirable, aspect dependent on future similar experiments carried out, either by the biomedical research group or by external entities.

9. ECONOMIC VIABILITY

Evaluating the economic impact that result from the development of a project is a crucial aspect which has to be addressed for its correct managing as well for delimiting the current work to an extension that we can economically shoulder. An estimation of the total cost and budget required is presented in the above section, considering both the direct and indirect expenses.

9.1. MATERIAL RESOURCES

9.1.1. HARDWARE

All the materials used for the experimental part were not specifically bought for the project as the biomedical group already had them in stock. In light of the above, their cost was neglected in the economical assessment.

The data collected for the project was acquired with the confocal microscope Carl Zeiss LSM 880, which belongs to the Advanced Optical Microscopy Department from the Medicine and Health Science Faculty of the University of Barcelona. Depending on an external entity for the cell observation and data acquisition, costs were derived for the microscopy service costs, presenting an economical value of 16.04€/hour*.

A computer was needed for the carry-on of the project. For practical reasons, a laptop was preferred, given that the project was not carried out in a fixed location.

9.1.2. SOFTWARE

The developed program -including the data processing and manipulation as well as the model building, training, testing and final evaluation- was accomplished with Python, an open-source software. Images were recorded with the Zeiss ZEN software, already installed in the computer attached to the confocal microscope Zeiss LSM 880. The time-lapse images stored in a *.czi* format were converted and saved as *.png* files with the usage of the free software Fiji. Last but not least, Microsoft Office® 365 services were used for the writing of the final report and project presentation.

9.2. HUMAN RESOURCES

When it comes to the human resources, two main participants have to be considered: the biomedical engineer student and the biomedical researcher. Additionally, we could include a third one: the microscope technician. Owing the fact that we did not have the required knowledge for the microscope manipulation, paying for the services of a microcopy operator was mandatory, resulting in an additional cost for the development of the project. As a way of estimating the human resources, and according to the previously detailed tasks, it was considered the time inverted by each participant in the project and the approximated salary per hour for each person.

The engineer student invested 21 hours weekly -5 hours/day, three days a week working in the Biomedicine Department and 3 hours/day, two days a week tele-working as a way of overcoming the organizational problems introduced in *Section 5*-, resulting in a total of 378 hours. As for the

other reviewed project contributors, an approximated number of 30** hours were invested by the biomedical researcher and 3,5 for the services of the microscope operator.

Weighting the work carried out by the undergraduate biomedical engineer, the salary was set at 10€ per hour. As per the biomedical researcher, according to the average wage for these professionals in Spain [65], the price per hour was settle at 22€. Finally, the microscope operator per hour worked was of 36.27€/hour.

9.3. TOTAL COST

All the costs previously described are sum up in *Table 5*, estimating the overall budget of the project.

	UNIT/HOUR PRICE (€)	UNIT/HOURS	TOTAL (€)
MATERIAL RESOURCES: HARDWARE			
Lenovo V15-IIL	649	1	649
Carl Zeiss LSM 880 Airyscan	16,04*	11	176,44
SUBTOTAL			***825,44***
MATERIAL RESOURCES: SOFTWARE			
Python software	-	1	-
Fiji software	-	1	-
Zeiss Zen	-	1	-
Microsoft Office® 365***	-	1	-
SUBTOTAL			-
HUMAN RESOURCES			
Biomedical engineering student	10	378	3.780
Biomedicine researcher	22 [65]	30**	660
Microscopy advanced technician	36,27*	3	108,81
SUBTOTAL			***4.548,81***
TOTAL			***5.374,25***

Table 5. Project cost table.

*Owing the fact that the microscope was financed by the European Regional Development Fund (ERDF), a discount of the 70% on the actual price (with IVA) was applied, valid for the forthcoming two years. Otherwise, the costs of the microscopy related services would be up to 44,20€ and 99,92€.

**The hours devoted to the undergraduate training were not borne in mind. Similar to any new employee on their first days of work, guidance is required by a former worker, thought to be part of the worker functions and not an extra service. Thus, only the hours where the researcher took the role of the student and performed the experimental part were taken into account.

***Free license as provided by the University of Barcelona

10. CONCLUSION AND FUTURE LINES

Underlying the cancer roots represents one of the major issues to tackle in the medical field. Aiming to eradicate and/or palliate this disease, studies have to be conducted to underline and understand the basic aspects behind cancer generation and progression. For this reason, biomedical researchers move their focus on the cellular and molecular level, looking for those biological triggers and genetics changes involved in tumor formation. The subcellular organization of cells and its spatial rearrangement throughout the cell cycle -either in normal conditions or stress situations- have been described as relevant precursors of malignant growth. Along with genotypical information retrieved from biomedical studies conducted with different mutated cells, the observation of cells under microscope and analysis of its actin cytoskeleton behavior during the cell cycle can help to understand how the cell response and readapts when facing stressful situations and thus, find out which are the genes' role and its repercussion in cancer formation.

The hazardous and time-consuming manual cell identification evidence the need of developing automatic classification methods for the discrimination of images according to a certain criterion which, in this present study, are no more than *pombe* cells in relation to their actin cytoskeleton during the cell cycle. In this context, DL techniques -whose popularity has been exponentially increasing in the biomedical field over the last years- become handy. Therefore, this project aims to construct a Deep Learning algorithm for the problem reviewed and prove its potential benefits as compared to the classic manual classification in this area of study.

In light of the obtained results, the model's successful learning of the introduced data has been proved and, what we considered more interesting, what seems to be a nearly perfect classifier since great were the results obtained after the testing phase, stage in which just a few samples were misclassified. Despite that, the consistency of the model ability differs from the reviewed scenario: issues regarding its classification robustness arose when feeding it with new data. The evaluation metrics worsened when testing the model with a toy dataset composed by images set apart from the original dataset. After reviewing the different evaluation parameters, it was concluded that the dataset was imperfect, failing to capture in its total extension the wide variety of actin structure phenotypes. Therefore, the constructed model, rather than being taken as the absolute truth, has to be seen as a complementary tool for investigators.

The uncertainty of the model's reliability shows that there is still room for improvement, especially when it comes to address the reviewed data frame problem. The dataset construction, considered to be one of the most critical and challenging parts in any Machine Learning project, could be improved and rectified by acquiring new and high-quality data and, more important, focusing on the avoidance of a possible case of data bias. To address this issue, one has to ensure the diverse representation in their data to be feed into the neural network, resulting in large and balanced dataset capable of capturing the diversity of real-world scenarios -the variability of actin phenotypes during the cell cycle-. Subsequently, the model must be trained, validated and tested with the constructed dataset, evaluating its performance and ensuring its generalization capability with a great number of unseen data before its deployment in real life.

Although not contemplated as the main project's objective, the retrieval of new data could be improved by the implementation of more complex but efficient approaches such as ANNs, algorithms which, after being feed with images and their respective target segmentation masks and following training, can automatically segment images. To do so, high-speed processors are indicated, given their great processing capacitance for this type of tasks. Decanting for this kind of technological equipment would also be beneficial for data handling and ML tasks, as some were the computational limitations observed when conducting those assignments with a uniprocessor system.

BIBLIOGRAPHY

1. Sung, H., Ferlay, J., Siegel, R. L., Laversanne, M., Soerjomataram, I., Jemal, A., & Bray, F. (2021). Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries. *CA: a cancer journal for clinicians, 71*(3), 209–249. https://doi.org/10.3322/caac.21660

2. *Schizosaccharomyces pombe.* Nature. https://www.nature.com/subjects/schizosaccharomyces-pombe

3. Hylton, H. M., Lucas, B. E., & Petreaca, R. C. (2020). *Schizosaccharomyces pombe* Assays to Study Mitotic Recombination Outcomes. *Genes, 11*(1), 79. https://doi.org/10.3390/genes11010079

4. Chung, K. S., Jang, Y. J., Kim, N. S., Park, S. Y., Choi, S. J., Kim, J. Y., Ahn, J. H., Lee, H. J., Lim, J. H., Song, J. H., Ji, J. H., Oh, J. H., Kyung Bin Song, Yoo, H. S., & Won, M. (2007). Rapid Screen of Human Genes for Relevance to Cancer Using Fission Yeast. *Journal of Biomolecular Screening, 12*(4), 568–577. https://doi.org/10.1177/1087057107301007

5. Sveiczer, A. (2004). Modelling the fission yeast cell cycle. *Briefings in Functional Genomics and Proteomics, 2*(4), 298–307. https://doi.org/10.1093/bfgp/2.4.298

6. Daga, R. R., & Chang, F. (2005). Dynamic positioning of the fission yeast cell division plane. *Proceedings of the National Academy of Sciences, 102*(23), 8228–8232. https://doi.org/10.1073/pnas.0409021102

7. Carbona, S., Goff, C., & Goff, X. (2006). Fission yeast cytoskeletons and cell polarity factors: connecting at the cortex. *Biology of the Cell, 98*(11), 619–631. https://doi.org/10.1042/bc20060048

8. López-Avilés, S., Grande, M., González, M., Helgesen, A. L., Alemany, V., Sanchez-Piris, M., Bachs, O., Millar, J. B., & Aligue, R. (2005). Inactivation of the Cdc25 Phosphatase by the Stress-Activated Srk1 Kinase in Fission Yeast. *Molecular Cell, 17*(1), 49–59. https://doi.org/10.1016/j.molcel.2004.11.043

9. *Latrunculin B (CAS 76343–94-7)*. SCBT - Santa Cruz Biotechnology. https://www.scbt.com/es/p/latrunculin-b-76343-94-7

10. Gachet, Y., Tournier, S., Millar, J. B. A., & Hyams, J. S. (2001). A MAP kinase-dependent actin checkpoint ensures proper spindle orientation in fission yeast. *Nature, 412*(6844), 352–355. https://doi.org/10.1038/35085604

11. Zhu, X., Jia, X., & Wong, K. Y. K. (2015). Structured forests for pixel-level hand detection and hand part labelling. *Computer Vision and Image Understanding, 141,* 95–107. https://doi.org/10.1016/j.cviu.2015.07.008

12. Amisha, Malik, P., Pathania, M., & Rathaur, V. (2019). Overview of artificial intelligence in medicine. *Journal of Family Medicine and Primary Care, 8*(7), 2328. https://doi.org/10.4103/jfmpc.jfmpc_440_19

13. Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets, 31*(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

14. Mao, S., Wang, B., Tang, Y., & Qian, F. (2019). Opportunities and Challenges of Artificial Intelligence for Green Manufacturing in the Process Industry. *Engineering, 5*(6), 995–1002. https://doi.org/10.1016/j.eng.2019.08.013

15. Pecht, M. G., & Kang, M. (2018). Machine Learning: Fundamentals. *Prognostics and Health Management of Electronics: Fundamentals, Machine Learning, and the Internet of Things* (pp 85-109). Wiley-IEEE Press.

16. Gupta, M. M., Sinha, N. K., & Zadeh, L. A. (2000). Neural Networks for Identification of Nonlinear Systems: An Overview. *Soft Computing and Intelligent Systems* (pp 337-356). Elsevier Gezondheidszorg.

17. Baum, E. B. (1988). On the capabilities of multilayer perceptrons. *Journal of Complexity, 4*(3), 193–215. https://doi.org/10.1016/0885-064x(88)90020-9

18. Han, J., Kamber, M., & Pei, J. (2011). 9 - Classification: Advanced Methods. *Data Mining: Concepts and Techniques (3rd ed.)* (pp 393-442). Morgan Kaufmann Publishers.

19. Sarker, I. H. (2021). Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Computer Science, 2*(6). https://doi.org/10.1007/s42979-021-00815-1

20. Botalb, A., Moinuddin, M., Al-Saggaf, U. M., & Ali, S. S. A. (2018). Contrasting Convolutional Neural Network (CNN) with Multi-Layer Perceptron (MLP) for Big Data Analysis. *2018 International Conference on Intelligent and Advanced System (ICIAS).* https://doi.org/10.1109/icias.2018.8540626

21. Grosse R. (2019, February 5). *Convolutional Networks.* http://www.cs.toronto.edu/~rgrosse/

22. Zhou, S. K., Rueckert, D., & Fichtinger, G. (2019*).* Deep learning: RNNs and LSTM. *Handbook of Medical Image Computing and Computer Assisted Intervention* (pp 503-519). Elsevier Gezondheidszorg.

23. Sanchez, E. N., & Alanis, A. Y. (2017). Mathematical Preliminaries. *Discrete-Time Neural Observers* (pp 9-22). Elsevier Gezondheidszorg.

24. Binu, D., & Rajakumar, B. R. (2021). Neural networks for data classification. *Artificial Intelligence in Data Mining* (pp 109-131). Elsevier Gezondheidszorg.

25. Webb, G. I., Sammut, C., Perlich, C., Horváth, T., Wrobel, S., Korb, K. B., Noble, W. S., Leslie, C., Lagoudakis, M. G., Quadrianto, N., Buntine, W. L., Quadrianto, N., Buntine, W. L., Getoor, L., Namata, G., Getoor, L., Han, X. J. J., Ting, J. A., Vijayakumar, S., . . . Raedt, L. D. (2011). Learning Curves in Machine Learning. *Encyclopedia of Machine Learning,* 577–580. https://doi.org/10.1007/978-0-387-30164-8_452

26. *Learning Curves in Machine Learning*. (2020, July 30). Baeldung on Computer Science. https://www.baeldung.com/cs/learning-curve-ml

27. Brownlee, J. (2019, August 6). *How to use Learning Curves to Diagnose Machine Learning Model Performance.* Machine Learning Mastery. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

28. Belavagi, M. C., & Muniyal, B. (2016). Performance Evaluation of Supervised Machine Learning Algorithms for Intrusion Detection. *Procedia Computer Science, 89*, 117–123. https://doi.org/10.1016/j.procs.2016.06.016

29. Batarseh, F. A., & Yang, R. (2020). *Foundations of data imbalance and solutions for a data democracy.* Data Democracy (pp 83-106). Elsevier Gezondheidszorg.

30. Lipkin, M. (1961). Digital Computer as Aid to Differential Diagnosis. *Archives of Internal Medicine, 108*(1), 56. https://doi.org/10.1001/archinte.1961.03620070058008

31. Kim, M., Yan, C., Yang, D., Wang, Q., Ma, J., & Wu, G. (2020). Deep learning in biomedical image analysis. *Biomedical Information Technology,* 239–263. https://doi.org/10.1016/b978-0-12-816034-3.00008-0

32. Oei, R. W., Hou, G., Liu, F., Zhong, J., Zhang, J., An, Z., Xu, L., & Yang, Y. (2019). Convolutional neural network for cell classification using microscope images of intracellular actin networks. *PLOS ONE, 14*(3), e0213626. https://doi.org/10.1371/journal.pone.0213626

33. Vjestica, A., Merlini, L., Dudin, O., Bendezu, F. O., & Martin, S. G. (2016). Microscopy of Fission Yeast Sexual Lifecycle. *Journal of Visualized Experiments*, 109. https://doi.org/10.3791/53801

34. Asakawa, H., Hiraoka, Y., & Haraguchi, T. (2012). Live CLEM imaging: an application for yeast cells.

35. *RStudio | Open source & professional software for data science teams*. RStudio. https://www.rstudio.com/

36. *R Interface to Tensorflow.* Tensorflow for R. https://tensorflow.rstudio.com/

37. *R Interface to Keras*. Keras. https://keras.rstudio.com/

38. *MATLAB.* MathWorks ®. https://www.mathworks.com/products/matlab.html

39. *Deep Learning Toolbox.* MathWorks ®. https://www.mathworks.com/products/deep-learning.html

40. *Python*. Python™. https://www.python.org/c

41. *ZEISS Introduces LSM 880 with Airyscan* (2014, July). ZEISS. https://www.zeiss.com/microscopy/int/products/confocal-microscopes/lsm-980.html?vaURL=www.zeiss.com/lsm880

42. Carl Zeiss Microscopy GmbH (2014). *LSM 880: Operating manual*. Jena, Germany: Author.

43. *ZEISS ZEN Microscope Software for Microscope Components.* ZEISS. https://www.zeiss.com/microscopy/int/products/microscope-software/zen.html

44. Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., Tinevez, J. Y., White, D. J., Hartenstein, V., Eliceiri, K., Tomancak, P., & Cardona, A. (2012). Fiji: an open-source platform for biological-image analysis. *Nature Methods, 9*(7), 676–682. https://doi.org/10.1038/nmeth.2019

45. Linkert, M., Rueden, C. T., Allan, C., Burel, J. M., Moore, W., Patterson, A., Loranger, B., Moore, J., Neves, C., MacDonald, D., Tarkowska, A., Sticco, C., Hill, E., Rossner, M., Eliceiri, K. W., & Swedlow, J. R. (2010). Metadata matters: access to image data in the real world. *Journal of Cell Biology, 189*(5), 777–782. https://doi.org/10.1083/jcb.201004104

46. McGrath, H., Li, P., Dorent, R., Bradford, R., Saeed, S., Bisdas, S., Ourselin, S., Shapey, J., & Vercauteren, T. (2020). Manual segmentation versus semi-automated segmentation for quantifying vestibular schwannoma volume on MRI. *International Journal of Computer Assisted Radiology and Surgery, 15*(9), 1445–1455. https://doi.org/10.1007/s11548-020-02222-y

47. van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., & Yu, T. (2014). scikit-image: image processing in Python. *PeerJ, 2*, e453. https://doi.org/10.7717/peerj.453

48. Preim, B., & Botha, C. (2014). Image Analysis for Medical Visualization. *Visual Computing for Medicine,* 111–175. https://doi.org/10.1016/b978-0-12-415873-3.00004-3

49. Bandara R. (2014). Image Segmentation using Unsupervised Watershed Algorithm with an Over-segmentation Reduction Technique.

50. Sauvola, J., & Pietikäinen, M. (2000). Adaptive document image binarization. *Pattern Recognition, 33*(2), 225–236. https://doi.org/10.1016/s0031-3203(99)00055-2

51. Hashemi, M. (2019). Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. *Journal of Big Data, 6*(1). https://doi.org/10.1186/s40537-019-0263-7

52. Fast.Ai. https://www.fast.ai

53. Gibieža, P., & Petrikaitė, V. (2021). The regulation of actin dynamics during cell division and malignancy. *American journal of cancer research, 11*(9), 4050–4069.

54. Guo, X., Yin, Y., Dong, C., Yang, G., & Zhou, G. (2008). On the Class Imbalance Problem. *2008 Fourth International Conference on Natural Computation*. https://doi.org/10.1109/icnc.2008.871

55. Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data, 6*(1). https://doi.org/10.1186/s40537-019-0197-0

56. *tf.keras.preprocessing.image.ImageDataGenerator.* TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

57. Albon, C. (2018). *Python Machine Learning Cookbook*. Van Duuren Media.

58. Gupta, M., Rajnish, K., & Bhattacharjee, V. (2021). Impact of Parameter Tuning for Optimizing Deep Neural Network Models for Predicting Software Faults. *Scientific Programming, 2021*, 1–17. https://doi.org/10.1155/2021/6662932

59. Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging, 9*(4), 611–629. https://doi.org/10.1007/s13244-018-0639-9

60. Wang, L., Dernoncourt, F., & Bui, T. (2020). Bayesian Optimization for Selecting Efficient Machine Learning Models. *ArXiv, abs/2008.00386*.

61. Tennenholtz, G., Zahavy, T., & Mannor, S. (2018). Train on Validation: Squeezing the Data Lemon. *ArXiv, abs/1802.05846.*

62. Wei Y., Zheng S., Cheng M., Zhao H., Wang L., Ding E., Yang Y, Torralba A., Liu T., Sun G. et al (2020). LID 2020: The Learning from Imperfect Data Challenge Results. https://doi.org/10.48550/arXiv.2010.11724

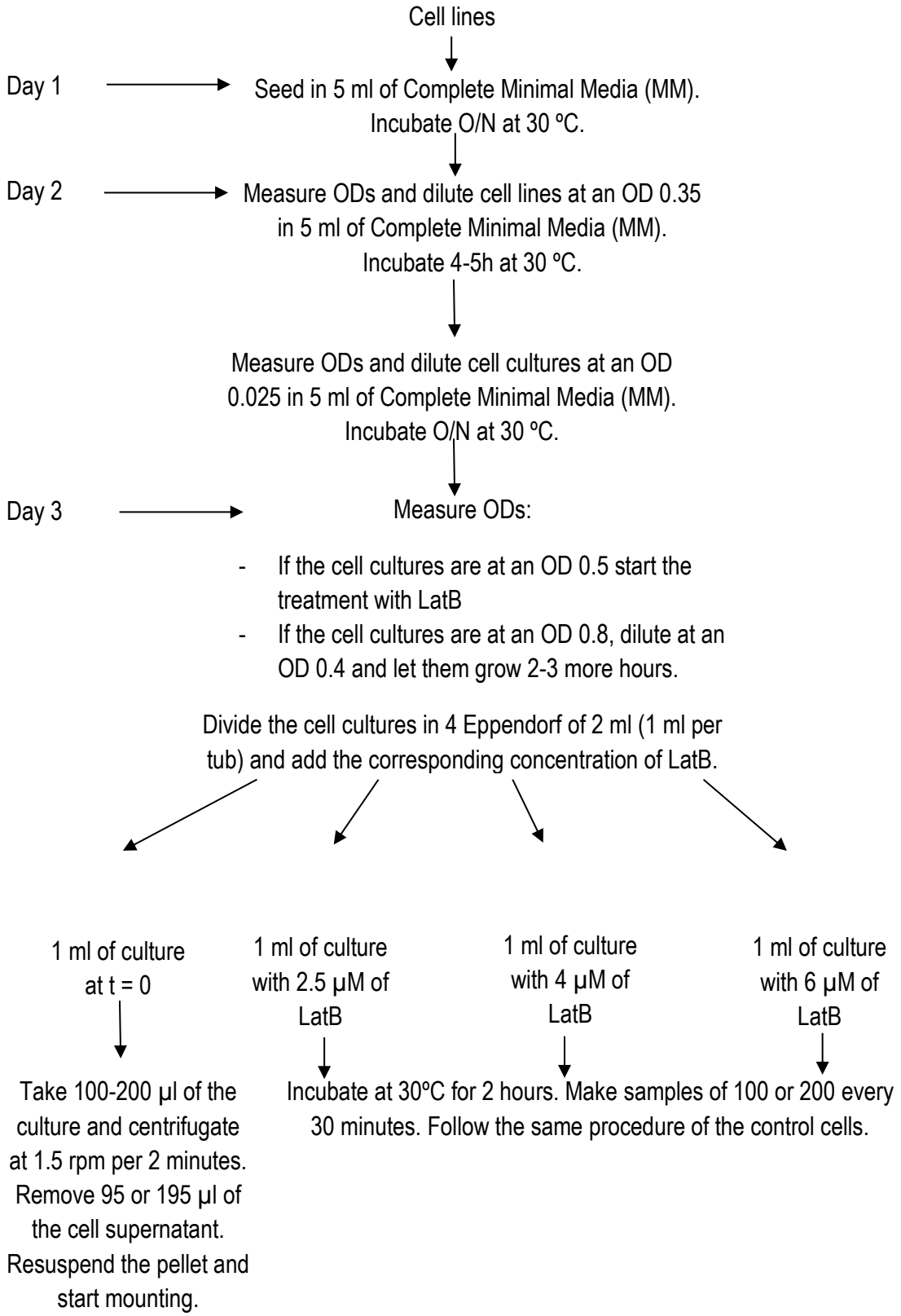
63. Smith, R. (2001). Planning and Scheduling Outages. *Plant Engineer's Handbook,* 889–914. https://doi.org/10.1016/b978-075067328-0/50053-7

64. Wilson, J. M. (2003). Gantt charts: A centenary appreciation. *European Journal of Operational Research, 149*(2), 430–437. https://doi.org/10.1016/s0377-2217(02)00769-5

65. ERI Economic Research Institute. *Salary Expert - Biomedical Scientist Salary Spain*. Salary Expert. https://www.salaryexpert.com/salary/job/biomedical-scientist/spain

APPENDIX

1. Experimental design



2. Code

Image processing.ipynb

This notebook is meant to prepare raw images for the subsequent analysis: data will be manipulated and processed, preparing it for a following segmentation. The raw cropped cells will be filtered for the dataset construction.

```
import cv2
import itertools
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib
import matplotlib.patches as mpatches
import os
import glob
import skimage.filters as filters
import skimage.morphology as morph
from skimage.feature import peak_local_max
from scipy import ndimage
from scipy.spatial import distance
from skimage.measure import label, regionprops
from PIL import Image
from skimage.segmentation import watershed
%matplotlib qt
```

1. Path configuration

1. **path**: directory with raw images to be processed.
2. **path2**: directory where raw individual cell images will be saved.
3. **path3**: directory where processed individual cell images will be saved.

```
path = r"C:\Users\mirei\OneDrive\Escritorio\4 Eng biomedica\TFG\DATA
TO ANALYZE\Raw"

path2 = r"C:\Users\mirei\OneDrive\Escritorio\4 Eng biomedica\TFG\DATA
TO ANALYZE\Crop"

path3 = r"C:\Users\mirei\OneDrive\Escritorio\4 Eng biomedica\TFG\DATA
TO ANALYZE\Corrected"
```

2. Image manipulation and segmentation

Raw images need to be segmented into individual cells. For each stack, parameters may change as regards to the current image requirements.

```
#%matplotlib qt

for file in glob.glob(path+"/*.png"):
    im = plt.imread(file)
```

```
    idx = file.rfind("\\")
    filename = file[idx+1:-4]

    # Green channel selection
    img = im[:, :, 1]

    # Image binarization
    th = filters.threshold_sauvola(img, window_size = 17, k = 0.5)
    imb = img > th

    # Correct images artifacts
    imb2 = morph.remove_small_objects(imb)
    imb3 = ndimage.binary_dilation(imb2, structure=np.ones((3,3)))
    imb4 = morph.remove_small_holes(imb3)

    # Pass image to uint8 format
    imb4 = np.array(imb4, dtype=np.uint8)

    # Automatic cell detection
    retval, labels, stats, centroids =
cv2.connectedComponentsWithStats(imb4)

    # Background is detected as a label. Therefore, it has to be
ignored
    centroids = np.delete(centroids,0,0)

    # Swap x and y
    centroids=centroids[:,::-1]

    min_dt = [] # Dictionary where the Euclidean distances will be
saved

    # Compute all possible combinations between the detected centroids
    # and calculate the Euclidean distance
    for subset in itertools.combinations(centroids, 2):
        min_dt.append(distance.euclidean(subset[0], subset[1]))

    # Width and minimum distance may need to be adjusted for the
specific case
    width = 7 # Number of neighborhood points

    # Prepare the markers for the watershed algorithm
    D = ndimage.distance_transform_edt(imb4)
    localMax = peak_local_max(D, indices = False, min_distance = 15,
labels = imb4)
    localMax1 = peak_local_max(D,  min_distance = 15, labels = imb4)

    # Over-segmentation corrective code
    for subset in itertools.combinations(localMax1, 2): # all the
combinations possible of pair of points
        if (distance.euclidean(subset[0], subset[1]) < 60):
            idx_left_1  = max(subset[0][1] - width, 0)
            idx_right_1 = min(subset[0][1] + width + 1, im.shape[0] -
1)
            if idx_right_1 == im.shape[0] - 1:  # If the last index in
the x axis is selected
                idx_right_1 = None
```

```
            idx_bot_1    = max(subset[0][0] - width, 0)
            idx_top_1    = min(subset[0][0] + width + 1, im.shape[0] -
1)
            if idx_top_1 == im.shape[0] - 1:    # If the last index in
the y axis is selected
                idx_top_1 = None
            idx_left_2   = max(subset[1][1] - width, 0)
            idx_right_2 = min(subset[1][1] + width + 1, im.shape[0] -
1)
            if idx_right_2 == im.shape[0] - 1:  # If the last index in
the x axis is selected
                idx_right_2 = None
            idx_bot_2    = max(subset[1][0] - width, 0)
            idx_top_2    = min(subset[1][0] + width + 1, im.shape[0] -
1)
            if idx_top_2 == im.shape[0] - 1:    # If the last index in
the y axis is sele1cted
                idx_top_2 = None

            # Create the neighborhood area, one for each of the pair
of points
            view1 = D[idx_bot_1:idx_top_1, idx_left_1:idx_right_1]
            view2 = D[idx_bot_2:idx_top_2, idx_left_2:idx_right_2]

            if 0 not in view1 or 0 not in view2:
                localMax[subset[0][0]][subset[0][1]] = False # Remove
the points
                localMax[subset[1][0]][subset[1][1]] = False
                new_pt = (subset[0]+subset[1])/2 # Compute the middle
point
                new_pt = [math.trunc(x) for x in new_pt]
                localMax[new_pt[0]][new_pt[1]] = True  # Add the new
point as a marker

    markers = ndimage.label(localMax, structure=np.ones((3, 3)))[0]
    labels = watershed(-D, markers, mask=imb4)

    #fig, ax = plt.subplots(figsize=(10, 6)) # In case of wanting to
observe the images, uncomment the lines related with matplotlib
    #ax.imshow(im)
    crop = []

    for region in regionprops(labels):
        # Take regions with large enough areas
        if region.area >= 100:
            # Draw a rectangle around the segmented cells
            minr, minc, maxr, maxc = region.bbox
            crop.append((minr, minc, maxr, maxc))
            #rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr
- minr,
                                      #fill=False, edgecolor='red',
linewidth=2)
            #ax.add_patch(rect)


    #ax.set_axis_off()
    #plt.tight_layout()
```

```
    # Save the segmented cells as individual images
    for num, crop in enumerate(crop):
        im_crop = im[crop[0]:crop[2],crop[1]:crop[3]]*255
        cv2.imwrite(path2+ "/" + filename+str(num)+".png",im_crop)
```

3. Image filtering

Even though image acquisition is performed meticulously, some situations were out of our hands: cells acquired in an incorrect plane. Furthermore, it was observed that the algorithm failed to segment clusters of cells, loosing valuable information.

Those problems could be partially solved applying the following criteria:

1. Perform a subsequent segmentation on those images whose dimensionalities are larger than the mean size of the images -cluster of cells-.
2. Set a threshold to remove images whose size is lower than a certain value and whose overall intensity does is below a certain threshold -remove cells out of plane + over-segmented cells-.

```
# Dictionaries to save the image number of bytes and the cells to be
removed
bts = []
remove_cells = []

for file in glob.glob(path2+"/*.png"):
    im = plt.imread(file)

    # Append the image shape and bytes in the corresponding images
    bts.append(os.path.getsize(file))

bts.sort()
bts_max = bts[-1]

for file in glob.glob(path2+"/*.png"):
    im = plt.imread(file)
    idx = file.rfind("\\")
    filename = file[idx+1:-4]

    # Green channel selection
    img = im[:,:,1]


    # Image binarization
    th = filters.threshold_sauvola(img, window_size = 29, k = 0.75)
    imb = img > th

    # Select those images big enough to be considered as clusters and
present a size bigger than a certain threshold
    if ((im.shape[0] >= (40)) or (im.shape[1] >= (40))) and
(os.path.getsize(file) >= (bts_max - 5000)):
        remove_cells.append(file)
```

```
        # Image processing
        imb2 = morph.remove_small_objects(imb)
        imb3 = ndimage.binary_dilation(imb2, structure=np.ones((3,3)))
        imb4 = morph.remove_small_holes(imb3)

        # Pass image to uint8  format
        imb4 = np.array(imb4, dtype=np.uint8)

        # Adjust width and min_distance according to the case
        width = 10

        # Markers for the watershed algorithm
        D = ndimage.distance_transform_edt(imb3)
        localMax = peak_local_max(D, indices = False, min_distance =
20, labels = imb4)
        localMax1 = peak_local_max(D,  min_distance = 20, labels =
imb4)

         # Cell clustering corrective code
        for subset in itertools.combinations(localMax1, 2):
            if distance.euclidean(subset[0], subset[1]) < 80:
                idx_left_1  = max(subset[0][1] - width, 0)
                idx_right_1 = min(subset[0][1] + width + 1,
im.shape[0] - 1)
                if idx_right_1 == im.shape[0] - 1:  # If the last
index in the x axis is selected
                    idx_right_1 = None
                idx_bot_1  = max(subset[0][0] - width, 0)
                idx_top_1   = min(subset[0][0] + width + 1,
im.shape[0] - 1)
                if idx_top_1 == im.shape[0] - 1:    # If the last
index in the y axis is selected
                    idx_top_1 = None
                idx_left_2  = max(subset[1][1] - width, 0)
                idx_right_2 = min(subset[1][1] + width + 1,
im.shape[0] - 1)
                if idx_right_2 == im.shape[0] - 1:  # If the last
index in the x axis is selected
                    idx_right_2 = None
                idx_bot_2   = max(subset[1][0] - width, 0)
                idx_top_2   = min(subset[1][0] + width + 1,
im.shape[0] - 1)
                if idx_top_2 == im.shape[0] - 1:    # If the last
index in the y axis is selected
                    idx_top_2 = None

                # Create the neighborhood area, one for each of the
pair of points
                view1 = D[idx_bot_1:idx_top_1, idx_left_1:idx_right_1]
                view2 = D[idx_bot_2:idx_top_2, idx_left_2:idx_right_2]

                if 0 not in view1 or 0 not in view2:
                    localMax[subset[0][0]][subset[0][1]] = False #
Remove the points
                    localMax[subset[1][0]][subset[1][1]] = False
                    new_pt = (subset[0]+subset[1])/2 # Compute the
middle point
```

```
                    new_pt = [math.trunc(x) for x in new_pt]
                    localMax[new_pt[0]][new_pt[1]] = True  # Add the
new point as a marker

        # Watershed algorithm
        markers = ndimage.label(localMax, structure=np.ones((3,
3)))[0]
        labels = watershed(-D, markers, mask=imb4)

        fig, ax = plt.subplots(figsize=(10, 6))
        ax.imshow(labels)

        crop = []
        for region in regionprops(labels):
        # Take with large enough areas
            if (region.area >= 100) and (region.area<=6000):

                # Draw a rectangle around the segmented cells
                minr, minc, maxr, maxc = region.bbox
                crop.append((minr, minc, maxr, maxc))

        # Save individual segmented images
        for num, crop in enumerate(crop):
            im_crop = im[crop[0]:crop[2],crop[1]:crop[3]]*255
            cv2.imwrite(path2+ "/" + filename+str(num)+".png",im_crop)

# Remove images
for file in remove_cells:
    os.remove(file)

# Dictionaries to save the image intensities and the cells to be
removed

its = []
remove_cells = []

for file in glob.glob(path2+"/*.png"):
    im = plt.imread(file)

    if (os.path.getsize(file) < 1490) and (np.sum(im) > 2000):
        remove_cells.append(file)

# Remove images
for file in remove_cells:
    os.remove(file)
```

4. Image padding

The resulting images present different dimensions. Therefore, they need to be rescaled to a common size while preserving the original aspect ratio. We decided to resize them to 150x150.

To do so, paddings can be added to the images to achieve the desired size: black borders will be added around them to achieve the expected dimensions.

```
files = 0
size = 150 # Wanted size dimensions
```

```
for file in glob.glob(path2+"/*.png"):
    files += 1
    im = cv2.imread(file)
    idx = file.rfind("\\")
    filename = file[idx+1:-4]
    img = Image.open(file)
    width, height = img.size  # Image shape

    pad = Image.new('RGB', (size, size), color='black') # Create a 8-
bit true color image with black background

    # Place the original image in the center
    x_center = (size - width) // 2
    y_center = (size - height) // 2
    pad.paste(img, (x_center, y_center, width + x_center, height +
y_center))
    pad = np.array(pad)

    # Save the output in the corresponding path
    cv2.imwrite(path3+"/"+filename+str(files)+".png",pad)
```

Dataset construction.ipynb

The model learning starts with the splitting of the dataset into two sets: train and test.

The train set, as the name suggests, will be used to train the model -for the learning of the image features and future prediction- whereas the test one, to evaluate the performance of the resulting model.

```
import glob
import numpy as np
import os
import shutil
import cv2
from keras.preprocessing.image import ImageDataGenerator
from sklearn.utils import shuffle
from keras.utils import np_utils
```

1. Path configuration

1. **path**: directory with images separated according to their label
2. **path2**: directory containing the train and test sets
3. **path3**: directory containing the data to input on the model

```
path =r"C:\Users\mirei\OneDrive\Escritorio\4 Eng biomedica\TFG\DATA TO
ANALYZE\Corrected"

path2 = r"C:\Users\mirei\OneDrive\Escritorio\4 Eng
biomedica\TFG\Datasets"

path3 = r"C:\Users\mirei\OneDrive\Escritorio\4 Eng biomedica\TFG\Data"
```

2. 80/20 split

The dataset will be split following the traditional 80/20 rule: 80% of images to the train set and 20% to the test.

```
# Cell classes and splitting ratio
labels = ["Non-dividing", "Dividing"]
test_ratio = 0.2

for cls in labels:
    files = os.listdir(path + "/" + cls)
    # Image shuffle
    np.random.shuffle(files)

    # 80/20 split
    train_files, test_files =
np.split(np.array(files),[int(len(files)* (1 - test_ratio))])

    # Save the corresponding files to the train and test directories
    train_files = [path + "/" + cls +'/'+ name for name in
train_files.tolist()]
    test_files = [path + "/" + cls +'/' + name for name in
test_files.tolist()]

    for name in train_files:
        shutil.copy(name, path2 +'/Train/' + cls)

    for name in test_files:
        shutil.copy(name, path2 +'/Test/' + cls)
```

3. Data augmentation

Data limited, more images were required for the correct training model. Rather than obtaining new images, the current dataset could be increased by means of data augmentation approaches: create modified data from the existing one.

```
# Data augmentation parameters
datagen = ImageDataGenerator(rotation_range = 20,
                             horizontal_flip=True,
                             vertical_flip=True,
                             brightness_range=[0.75,1.5])

X = [] # Images
Y = [] # Labels
for i, cls in enumerate(labels):
    files = glob.glob(path2 + "/Train/" + cls +"/*png")
    for id, file in enumerate(files):
        img = cv2.imread(file)
        img = cv2.resize(img,dsize=(150, 150))
        X.append(img)
        Y.append(i)


        # Convert 3d images to 4d array
```

```
        copy = img[None,...]

        # Prepare iterator
        it = datagen.flow(copy, batch_size = 1)

        # For each image, repeat the augmentation procedure by four
times
        for k in range(4):

            # Generate batches of images
            batches = it.next()
            batches = batches[0,...] # Restore 3D data from 4D
            batches = np.array(batches, dtype = np.uint8)
            X.append(batches)
            Y.append(i)


# Save the output as NumPy arrays
X = np.array(X)
Y = np.array(Y)
np.save(path3 +"/X_train.npy",X)
np.save(path3 +"/Y_train.npy",Y)
print("Training set has augmented from {} files to {}
files".format(sum([len(files) for r, d, files in os.walk(path2 +
"/Train")]), len(Y)))

X = []
Y = []
for i, cls in enumerate(labels):
    files = glob.glob(path2 + "/Test/"+ cls +"/*png")
    for id,file in enumerate(files):
        img = cv2.imread(file)
        img = cv2.resize(img,dsize=(150, 150))
        X.append(img)
        Y.append(i)

print("Test set contains {} files".format(sum([len(files) for r, d,
files in os.walk(path2 + "/Test")])))


# Save the test set as NumPy arrays
X = np.array(X)
Y = np.array(Y)
np.save(path3 +"/x_test.npy",X)
np.save(path3 +"/y_test.npy",Y)
```

Bayesian optimization.ipynb

Finding the correct hyperparameters for the DL model is not a straightforward duty. For this reason, as it is contained in this script, applying automatic methodologies -such as the Bayesian Optimization- can help to reduce the time required in comparison to the manual searching approach.

```
import numpy as np
```

```
import random
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from keras.utils import np_utils
from sklearn.utils import shuffle
from keras.models import Sequential
from keras.layers import Input, Conv2D,MaxPooling2D,Dense,
Flatten,Dropout
from tensorflow.keras.optimizers import RMSprop
from keras.callbacks import EarlyStopping
from bayes_opt import BayesianOptimization
from keras.models import Model
```

1. Path configuration

path: directory containing the train and test sets

```
path =r"C:\Users\mirei\OneDrive\Escritorio\4 Eng biomedica\TFG\Data"
```

2. Data normalization and shuffling

The goal of normalization is standardizing the data to a common scale without distorting the data information. Min-max normalization was the strategy applied on the data, scaling the pixels values into 0-1.

Finally, the train set was randomly mixed, reducing the variance.

```
# Load the train and test arrays
X_train = np.load(path +"/X_train.npy")
Y_train = np.load(path +"/Y_train.npy")

x_test = np.load(path  +"/x_test.npy")
y_test = np.load(path +"/y_test.npy")

# Image normalization
X_train = (X_train - np.min(X_train)) / (np.max(X_train) -
np.min(X_train))
x_test = (x_test - np.min(x_test)) / (np.max(x_test) - np.min(x_test))


# Convert array of labeled data to a vector
Y_train = np_utils.to_categorical(Y_train,3)
y_test = np_utils.to_categorical(y_test,3)


# Randomly mix the training set
X_train,Y_train = shuffle(X_train, Y_train)
```

3. CNN architecture

First of all, the CNN core structure has to be constructed. In this case, we decanted for the basic architecture -input layer, hidden layers (convolutional and pooling layers) and an output layer- plus some extra layers -flatten, dense and dropout-.

As for the hyperparameters to be optimized, we decanted for:

- Number of filters: dimensionality of the output space
- Drop: fraction of the input units to drop
- Number of hidden layers.

3.1. Defining CNN architecture function

```
def cnn_model_relu(filters, drop, hidden_layers):

    filters = int(filters)
    hidden_layers = int(hidden_layers)

    # Input layer
    inputs = Input(shape = X_train.shape[1:])

    x = Conv2D(filters, (3, 3), padding='same',
activation='relu')(inputs)
    x = MaxPooling2D((2, 2), padding='same')(x)

    # Hidden layers
    if hidden_layers !=0:
        for i in range(1,hidden_layers+1):
            x = Conv2D(filters*(2**i), (3, 3), padding='same',
activation='relu')(x)
            x = MaxPooling2D((2, 2), padding='same')(x)

    # Flatten, dense and dropout layers
    x = Flatten()(x)
    x = Dense(filters*(2**(hidden_layers+1)), activation='relu')(x)
    x = Dropout(drop)(x)

    # Output- predictions
    outputs = Dense(3, activation='softmax')(x)

    # Modeling
    model = Model(inputs = inputs, outputs = outputs)

    model.compile(optimizer = 'rmsprop',
                          loss = 'binary_crossentropy',
                          metrics = ['accuracy'])

    early_stopping = EarlyStopping(patience = 10, verbose = 1)

    # Learning
    history = model.fit(
        X_train, Y_train, validation_split = 0.2, epochs = 30,
batch_size = 20, callbacks = [early_stopping])
```

```
    # Evaluate the model with the test set
    score = model.evaluate(x_test, y_test)

    print('Test loss:', score[0])
    print('Test accuracy:', score[1])

    acc_relu.append(score[1])

    return score[1]
```

3.2. Bayesian Optimization

The optimal hyperparameters will be found by means of the Bayesian Optimization approach.

```
# Range of the possible values that the hyperparameter can take
def bayesian_opt_relu():
    parameters = {
        'drop' : (0,0.9),
        "filters" : (2,25),
        'hidden_layers' : (0,6)
    }

    # Define Bayesian Optimization
    optimizer = BayesianOptimization(f = cnn_model_relu, pbounds =
parameters)

    optimizer.maximize(init_points = 10, n_iter = 20)
    return optimizer

# Run the Bayesian Optimization and return the hyperparameters
result_relu = bayesian_opt_relu()
result_relu.res
result_relu.max
```

4. Hyperparameters plotting

Once the Bayesian Optimization has finished, we can plot the hyperparameters tried and observe how the accuracy behaves according to different hyperparameters combination.

```
relu = ['relu' for i in range(15)]

# Dictionaries containing the different hyperparameters tried
drop_relu = []
filters_relu = []
hidden_layers_relu = []
results_relu = result_relu.res

# Access the results obtained and saved the hyperparameters in their
corresponding dictionary
for pms in results_relu:
    params = pms["params"]
    drop_relu.append(params["drop"])
    filters_relu.append(params["filters"])
    hidden_layers_relu.append(params["hidden_layers"])
```

```
# Create a data frame containing the hyperparameters
df_relu = pd.DataFrame({'drop':drop_relu,
                    'filters':filters_relu,
                    'hidden_layers':hidden_layers_relu,
                     'accuracy': acc_relu,
                     'activation': relu })

df.reset_index(drop=True, inplace=True)

# Plot pairwise relationships in the data frame
sns.pairplot(df_rs,vars = df.columns[0:4], hue= 'activation',
diag_kind = 'kde')

plt.show()
```

Functional model.ipynb

This script contains the final model to be implemented with the hyperparameters properly adjusted.

The model evaluation metrics are also contained in the current script.

```
import numpy as np
import random
import seaborn as sns
import matplotlib.pyplot as plt
from keras.utils import np_utils
from sklearn.utils import shuffle
from keras.models import Sequential
from keras.layers import Input, Conv2D,MaxPooling2D,Dense,
Flatten,Dropout
from tensorflow.keras.optimizers import RMSprop
from keras.callbacks import EarlyStopping
from bayes_opt import BayesianOptimization
from keras.models import Model, load_model, model_from_json
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
f1_score, precision_score, recall_score, confusion_matrix,
accuracy_score
```

1. Path configuration

1. **path**: directory containing the train and test sets
2. **path2**: directory where the functional model will be saved

```
path =r"C:\Users\mirei\OneDrive\Escritorio\4 Eng biomedica\TFG\Data"

path2 = r"C:\Users\mirei\OneDrive\Escritorio\4 Eng
biomedica\TFG\Model"
```

2. Data normalzation and shuffling

Similar to the Bayesian Optimization.ipynb, data need first to be normalized and for the train set, shuffled.

```
# Load the train and test sets
X_train = np.load(path +"/X_train.npy")
Y_train = np.load(path +"/Y_train.npy")

x_test = np.load(path  +"/x_test.npy")
y_test = np.load(path +"/y_test.npy")

# Image normalization
X_train = (X_train - np.min(X_train)) / (np.max(X_train) -
np.min(X_train))
x_test = (x_test - np.min(x_test)) / (np.max(x_test) - np.min(x_test))


# Convert the array of labeled data to a vector
Y_train = np_utils.to_categorical(Y_train,3)
y_test = np_utils.to_categorical(y_test,3)


# Randomly mix the training set
X_train,Y_train = shuffle(X_train, Y_train)
x_test,y_test = shuffle(x_test, y_test)
```

3. Functional model construction: training and testing

Adjust the hyperparameters according to the values obtained from the Bayesian Optimization.

The model will be trained with the training dataset -and validated via the hold-out method-. After the testing of the built model, it will be saved future usage.

```
# Hyperparameters obtained from the Bayesian Optimization

drop = 0.68
filters = 13
hidden_layers = 2


# Input layer
inputs = Input(shape = X_train.shape[1:])

# Convolutional and pooling layers
x = Conv2D(filters, (3, 3), padding='same', activation='relu')(inputs)
x = MaxPooling2D((2, 2), padding='same')(x)

# Hidden layers
if hidden_layers !=0:
    for i in range(1,hidden_layers+1):
        x = Conv2D(filters*(2**i), (3, 3), padding='same',
activation='relu')(x)
        x = MaxPooling2D((2, 2), padding='same')(x)
```

```
# Flatten, dense and dropout layers
x = Flatten()(x)
x = Dense(filters*(2**(hidden_layers+1)), activation='relu')(x)
x = Dropout(drop)(x)

# Output - predictions
predictions = Dense(3, activation='softmax')(x)

# Modeling
model = Model(inputs = inputs, outputs = predictions)

model.compile(optimizer = 'rmsprop',
                        loss = 'binary_crossentropy',
                        metrics = ['accuracy'])

early_stopping = EarlyStopping(patience=10, verbose=1)

# Learning. The validation data comes from the train set
history = model.fit(
    X_train, Y_train, validation_split = 0.2, epochs = 100, batch_size
= 20, callbacks = [early_stopping])

# Evaluated accuracy for the test set
print(model.evaluate(x_test, y_test))

# Save the model for future usage

model_json = model.to_json() # Serialize model to JSON
with open(path2 + "/Actin model" +".json", "w") as json_file:
    json_file.write(model_json)

model.save_weights(path2 + "/Actin model"+".h5") # Serialize weights
to h5
```

4. Model evaluation

Assessing the model performance and accuracy is crucial to ensure that the built model properly predicts and labels new incoming data.

Here, we propose two metric evaluations:

- Learning curve -both optimization and performance LCs-.
- Confusion matrix -and related metrics-.

4.1. Learning Curve

The overall learning performance can be visually evaluated with the use of Learning Curves. Two different metrics will be evaluated: accuracy and loss.

```
# Load accuracy and loss metrics from the model history
accuracy = history.history['accuracy'] # Accuracy for the training
data
val_accuracy = history.history['val_accuracy'] # Accuracy for the
validation data
```

```
loss = history.history['loss'] # Loss for the training data
val_loss = history.history['val_loss'] # Loss for the validation data

# Create the Learning Curves
fig, (axP, axO) = plt.subplots(ncols=2,figsize=(16,8))
plt.rcParams["font.size"] = 16
axP.plot(accuracy,label='Training acc.')
axR.plot(val_accuracy,label='Validation acc.')
axR.set_title('Optimization Learning Curve')
axR.legend(loc='best')
axO.plot(loss,label='Training loss')
axO.plot(val_loss,label='Validation loss')
axO.set_title('Performace Learning Curve')
axO.legend(loc='best')
plt.show()
```

4.2. Confusion matrix

The confusion matrix is a tool that allows to observe how well does the model predict the data, since it directly correlates the expected results with the actual output.

Further information can be extracted from the confusion matrix, giving a final verdict: precision, recall, f1-score and accuracy values.

```
# Classes to be predicted
labels = ["Non-dividing", "Dividing"]

# Predictions from the constructed model
y_predict = model.predict(x_test)
y_true = y_test
y_predict = np.argmax(y_predict, axis = -1)

# Create the confusion matrix
fig, ax = plt.subplots(figsize=(12, 12))
cm = confusion_matrix(y_true,y_predict)
sns.heatmap(cm, annot = True, fmt = 'g', cmap = 'Wistia', xticklabels
= labels,  yticklabels = labels)
ax.set_title("Confusion Matrix")
plt.show()

# Print accuracy, f1, precision and recall scores
print(accuracy_score(y_true, y_predict))
print(precision_score(y_true, y_predict))
print(recall_score(y_true, y_predict))
print(f1_score(y_true, y_predict))
```