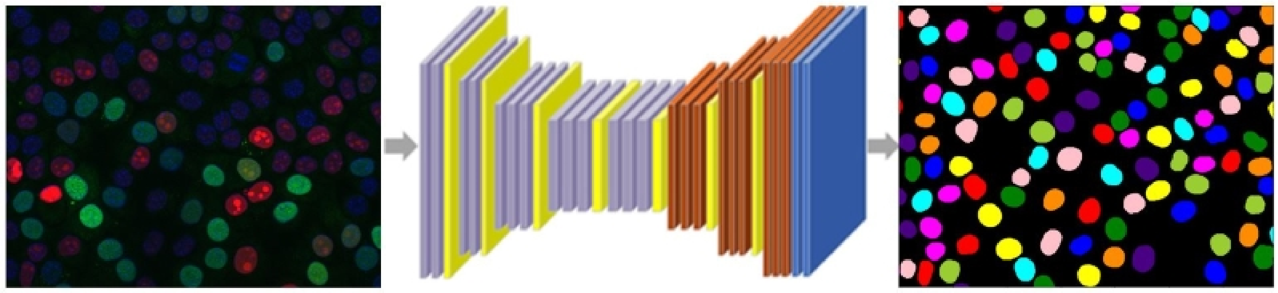


# Deep Learning Based Cell Nuclei Segmentation: An Application for Cell Cycle Staging

**Hemaxi Narotamo**

Thesis to obtain the Master of Science Degree in

**Biomedical Engineering**

Supervisors: Prof. Maria Margarida Campos da Silveira
Prof. João Miguel Raposo Sanches

**Examination Committee**

Chairperson: Prof. Cláudia Alexandra Martins Lobato da Silva
Supervisor: Prof. Maria Margarida Campos da Silveira
Member of the Committee: Prof. Jorge Dos Santos Salvador Marques

**October 2019**

## Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

# Preface

The work presented in this thesis was performed at the Institute for Systems and Robotics of Instituto Superior Técnico (Lisbon, Portugal), during the period February-October 2019, under the supervision of Prof. João Sanches, Prof. Margarida Silveira, Doctor Raquel Seruca and Doctor Sofia Fernandes, and within the frame of the FCT (Fundação para a Ciência e Tecnologia) through the projects TRACE (PTDC/BBB-IMG/0283/2014) and reference UID/EEA/50009/2019. Additionally, the author of this thesis wants to acknowledge the EPIC (Epithelial Interactions in Cancer) from i3S/Ipatimup for the biological support of this research. In particular for providing the dataset analyzed in this work, and for the new experiments that were done in order to obtain the gastric cancer cell line dataset.

# Acknowledgments

First and foremost, I want to thank my Mom, my Dad and my Brother for their love, support and for everything they have done for me throughout all my life. And a special thank you to my brother Nichal for playing video games with me during the small work breaks.

A special and sincere thanks to Professor João Sanches for trusting in my abilities to carry out this project, for all the help and availability, and for all the motivation and support that he gave me during this work.

A deep thanks to Professor Margarida Silveira for her guidance, support and availability to clarify my doubts. I learned a lot about deep learning from her due to her amazing teaching skills.

A very strong thanks to Doctor Raquel Seruca for being such an extraordinary person, for all the meetings we had, for all the motivation and advices she gave me, and for clarifying all the biological questions underlying this project.

Moreover, a special thanks to professors João, Margarida and Raquel for giving me the opportunity to attend a conference that helped me grow as a professional and as a person.

I would also like to thank the Ipatimup researchers with whom I had the opportunity to have a meeting. Especially, I want to thank Sofia Fernandes for all the work she has done in acquiring some of the images analyzed in this work and for all the biological support.

Additionally, I want to thank all my family members and my neighbours (Raj and Prachi and their family) for always supporting me and for being part of my life.

Last but not the least, thanks to my friend Catarina Pinto who supported me unconditionally during these 5 years, to João Freitas for all the help provided at the beginning of this project, to my friend Saber for all the help given during this period of 8 months; and, to my friends Miguel Joaquim, Catarina Gomes, Simão Pires, Luís Castro, Miguel Alves, Bárbara Costa, Miguel Tenreiro, Tiago Pires, and others whose names I didn't mention but they know who they are, for all the support, love and for all the moments of fun we spent together.

# Abstract

Cell nuclei segmentation is crucial in several bioimaging tasks such as cell counting and tracking, analysis of cell morphology and quantification of molecular expression. Accurate automatic nuclei segmentation is of special interest in high-throughput applications of microscopic images of cells or tissues. Cell nuclei segmentation is an open problem and an active field of research in the image processing community.

In this work, a novel deep learning based approach for high-throughput nuclei instance segmentation is proposed. It combines the detection capability of Fast YOLO with the segmentation ability of U-Net. With this method significant improvements on the processing time (approximately 9 times less) are achieved in comparison with the Mask R-CNN, without compromising the F1 score of the results.

Furthermore, the masks obtained with the proposed segmentation method were used to address the important problem of cell cycle staging from DAPI stained nuclei. A (supervised) SVM based approach is proposed, where the input features and ground truth labels used to train the classifier were automatically computed from nuclei stained with DAPI and FUCCI, respectively.

Results leading to F1 score of almost 90% (F1 score: $0.877 \pm 0.010$) were obtained, where in 21 out of the 130 test images a F1 score of 100% was achieved. Testing the classifier on images of cells from a different cell line from the one used to train the classifier lead to the impressive recall value of 93%, which demonstrates the robustness of the proposed approach for cell cycle staging.

# Keywords

Deep learning, Nuclei segmentation, Cell imaging, Cell cycle staging

# Resumo

A segmentação dos núcleos celulares é importante em várias tarefas de bioimagiologia, como contagem de células, análise da morfologia celular e quantificação da expressão molecular. A segmentação nuclear automática é de especial interesse para aplicações de alto rendimento de imagens microscópicas de células ou tecidos, e corresponde a uma área de investigação ativa na comunidade de processamento de imagem.

Neste trabalho é proposta uma nova abordagem, baseada em aprendizagem profunda, para a segmentação de núcleos celulares. Esta abordagem combina a capacidade de deteção da Fast YOLO com a capacidade de segmentação da U-Net. Com o método proposto, verificou-se que, sem comprometer o F1 score dos resultados, o tempo de processamento é cerca de 9 vezes menor em comparação com o Mask R-CNN.

Adicionalmente, as máscaras obtidas com o método de segmentação proposto foram usadas para abordar o importante problema do estadiamento do ciclo celular a partir de núcleos corados com DAPI. Uma abordagem supervisionada baseada no SVM é proposta, onde o input e o output usados para treinar o classificador foram automaticamente calculados a partir dos núcleos corados com DAPI e com FUCCI, respetivamente.

Obtiveram-se resultados com F1 score de quase 90% (F1 score: $0.877 \pm 0.010$), onde em 21 das 130 imagens de teste obteve-se F1 score de 100%. O classificador também foi testado em imagens de células de uma linha celular diferente da que foi usada para treiná-lo, atingindo o valor impressionante de recall de 93%, o que mostra a robustez do método proposto para o estadiamento do ciclo celular.

# Palavras Chave

Aprendizagem profunda, Segmentação de núcleos, Imagiologia celular, Estadiamento do ciclo celular

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Listings

# Acronyms

| | |
|---|---|
| **AdaDelta** | Adaptive Delta |
| **AdaGrad** | Adaptive Gradient |
| **ADAM** | ADAptive Moment Estimation |
| **AJI** | Aggregated Jaccard Index |
| **ANNs** | Artificial Neural Networks |
| **API** | application programming interface |
| **AT** | adenine-thymine |
| **BCE** | Binary Cross Entropy |
| **CNNs** | Convolutional Neural Networks |
| **CNN** | Convolutional Neural Network |
| **CNN2** | Two-Class Convolutional Neural Network |
| **CNN3** | Three-Class Convolutional Neural Network |
| **DAPI** | 4’,6-diamidino-2-phenylindole |
| **DC** | Dice Coefficient |
| **DNA** | Deoxyribonucleic acid |
| **DSC** | Dice similarity coefficient |
| **ELU** | Exponential Linear Unit |
| **FC** | Fully Connected |
| **FCL** | Fully Connected Layer |

| | |
|---|---|
| **FCN** | Fully Convolutional Network |
| **FCIS** | Fully Convolutional Instance-aware Semantic Segmentation |
| **FPN** | Feature Pyramid Network |
| **FUCCI** | Fluorescence ubiquitination cell cycle indicator |
| **GFP** | green fluorescent protein |
| **GIMP** | GNU Image Manipulation Program |
| **GPU** | Graphics Processing Unit |
| **HD** | Hausdorff distance |
| **HE** | Hematoxylin-and-Eosin |
| **IFC** | imaging flow cytometry |
| **IQR** | interquantile range |
| **IoU** | intersection over union |
| **LeakyReLU** | Leaky Rectified Linear Unit |
| **MAD** | Mean absolute distance |
| **mAP** | mean average precision |
| **Mask R-CNN** | mask regional convolutional neural network |
| **MATLAB** | MATrix LABoratory |
| **MNIST** | Modified National Institute of Standards and Technology |
| **Q1** | First Quartile |
| **Q2** | Second Quartile |
| **Q3** | Third Quartile |
| **ReLU** | Rectified Linear Unit |
| **RFP** | red fluorescent protein |
| **RGB** | Red-Green-Blue |
| **R-CNN** | Region-CNN |

| | |
|---|---|
| **RoI** | Region of Interest |
| **RoIs** | Regions of Interest |
| **RPN** | region proposal network |
| **SGD** | Stochastic Gradient Descent |
| **SSD** | Single Shot MultiBox Detector |
| **SVM** | Support Vector Machine |
| **Tanh** | Hyperbolic Tangent |
| **UV** | ultraviolet |
| **XML** | Extensible Markup Language |
| **YOLO** | You Only Look Once |
| **2D** | two dimensional |
| **3D** | three dimensional |

# 1 Introduction

## Contents

## 1.1 Motivation

Cell nuclei segmentation is important from the biological point of view. For instance, it provides valuable information about the Deoxyribonucleic acid (DNA) content [1], chromatin condensation [2] and nuclei morphology [3]. This information can be used, for example, to study the cell cycle [1] or to study mutations in proteins associated with cancer [4]. Cell overlapping, image noise and non-uniform acquisition and preparation parameters make the nuclei segmentation procedure a challenging task [2, 3].

Manual nuclei segmentation depends on the clinician's experience, it is time consuming and subjected to human error [3, 5, 6]. Consequently, to overcome subjective interpretation and reduce human's workload, automatic image analysis methods have been developed [2, 3, 6]. A lot of classical methods have been proposed for nuclei detection and segmentation. The most common traditional method is based on the combination of a thresholding method followed by watershed algorithm [2]. However, there are a lot of complications associated with traditional nuclei segmentation methods. For instance, they depend on manual tuning of parameters, they become very specific for a given type of image and their performance is degraded with the presence of noise [3]. Recently, deep learning is becoming state-of-the-art, since it leads to enhanced performance in many medical applications [7]. Deep learning based approaches present a good generalization capacity, they are able to automatically extract meaningful features from the image and can be more robust to noise when compared to traditional approaches for nuclei segmentation [8–10]. Hence, throughout the years several methods based on deep learning have been proposed for nuclei segmentation [11]. Nevertheless, the task of nuclei segmentation is still an active field of research [2, 6].

Moreover, nuclei segmentation is important for several applications, for instance, it can be used to address the important problem of cell cycle staging. Cell cycle is a fundamental mechanism of living organisms. The main goal is to ensure that when a cell divides it passes correctly the genetic information to the next generation. Additionally, it plays an important role in many diseases. For instance, dysregulation of the cell cycle is at the origin of many diseases, such as cancer, ischemia, neurodegenerative disorders and infection. Therefore, by studying and controlling the cell cycle one can understand the mechanisms of various diseases, and consequently find ways to tackle them. Furthermore, cell cycle control and manipulation can have numerous applications in regenerative medicine [12].

## 1.2 Objective and Original Contributions

This work focuses on the development of an automatic tool based on deep learning for cell nuclei instance segmentation in fluorescence microscopy images. After obtaining this tool the main goal is to study the cell cycle phase of each nucleus. The goal is to infer the cell cycle stage from the information contained in microscopic images of cells stained with 4',6-diamidino-2-phenylindole (DAPI). A supervised machine learning classifier (Support Vector Machine (SVM)) will be used for nuclei classification. To summarize, the main goal of this project is twofold (as shown in figure 1.1):

1. Development of an automatic tool for cell nuclei instance segmentation based on deep learning.
2. Development of an automatic tool for cell cycle staging from DAPI images.

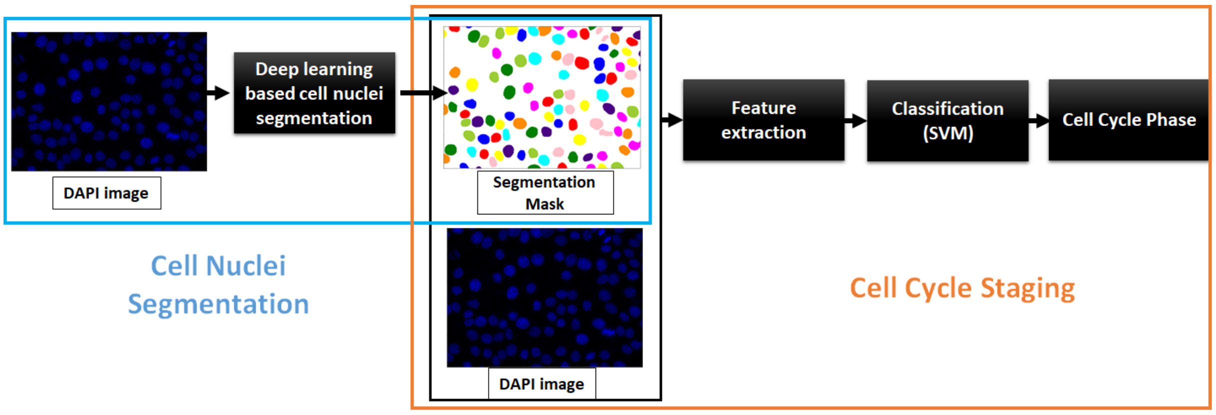


**Figure 1.1:** Schematic representation of this project's goal. The first and main goal of this work is to develop a deep learning based approach for cell nuclei instance segmentation in microscopy images (represented within the blue box). Thereafter, the goal is to develop a tool for cell cycle staging based on nuclei features extracted from the DAPI image, according to the corresponding segmentation mask (as shown inside the orange box); thus, the step of nuclei segmentation is really important for cell cycle staging.

In this work a new approach for cell nuclei instance segmentation based on deep learning is presented. It combines the Fast You Only Look Once (YOLO) and the U-Net, for object detection and segmentation, respectively. Moreover, a SVM based approach that can be applied for cell cycle staging of different cell lines is presented. In fact, this is the first supervised approach for interphase cell cycle staging of adherent cells from DAPI images. This approach extracts nuclei features from images of cells stained with DAPI and returns a label for each nucleus regarding its cell cycle phase.

## 1.3 Thesis Outline

In chapter 2 the background regarding deep learning is introduced. It starts with an introduction to computer vision and machine learning. Afterwards, a detailed explanation about Convolutional Neural Networks (CNNs) and the training of deep learning models is provided. This is followed by an explanation about the tasks of image classification, object detection and segmentation. Finally, in the last part of this chapter several works that have been developed for cell nuclei segmentation are described. In chapter 3 biological background is presented, it starts with an introduction to the cell, cell cycle and fluorescence microscopy. Finally, state-of-the-art regarding methods for cell cycle staging from microscopy images is presented. Chapter 4 starts with the explanation of the overall pipeline for cell nuclei segmentation and cell cycle staging. Thereafter, the proposed approach for cell nuclei segmentation is presented. The performance of this approach was compared with the performance of other methods, these methods are described in detail. Afterwards, the steps followed to perform cell cycle staging are explained in detail. Section 5 presents the experimental results and discussion regarding nuclei segmentation and cell cycle staging. Finally, in section 6 conclusions and topics requiring future studies are presented.

# 2

# Deep Learning Background

**Contents**

In this chapter the background regarding deep learning is introduced. This chapter starts with an introduction to computer vision and machine learning. Afterwards, a detailed explanation about CNNs and the training of deep learning models is provided. This is followed by an explanation about the tasks of image classification, object detection and segmentation. Finally, in the last part of this chapter several works that have been developed for cell nuclei segmentation are described.

## 2.1 Relevance of Computer Vision and Machine Learning

Computer vision studies visual data and it is focused mainly on complex image processing techniques. The main goal is to enable computers to view, process, analyze and understand digital images or videos [13]. Machine learning provides algorithms that can be used by computers to perform tasks such as time series analysis, speech recognition and medical image analysis [8]. With the increasing availability of labeled data on the Internet, machine learning techniques are being applied to solve computer vision tasks [9]. Recently, deep learning, which is a type of machine learning [9], is becoming state-of-the-art in many computer vision tasks, such as object detection, image classification and segmentation [8]. Deep learning is a machine learning algorithm which is composed of several layers that extract meaningful features from the input data to perform a task. For instance, when the input is an image, the first layers of the deep learning architecture extract simple features, whereas higher levels extract high level and more abstract features. These features allow to perform, for example, image classification, object detection and segmentation [8]. Throughout the years, several deep learning based algorithms and architectures have been developed [8]. For instance, the Convolutional Neural Network (CNN) is a deep learning architecture that was initially developed for image classification, and was quickly adapted for image segmentation [14]. Due to their importance in this work, in the next section a detailed explanation about CNNs is provided.

## 2.2 Convolutional Neural Networks

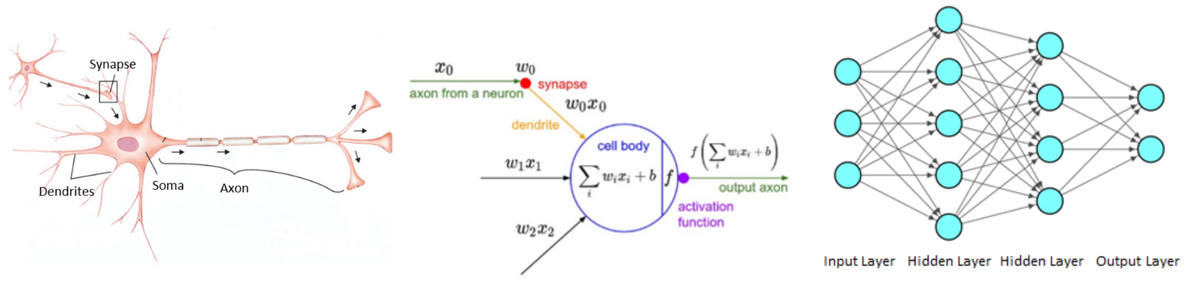
### 2.2.1 From Artificial Neural Networks to Convolutional Neural Networks

Artificial Neural Networks (ANNs) were inspired by the biological behavior of neurons [9]. A neuron (figure 2.1(a)) can be modeled as a function, which receives inputs (from the axons of other neurons) and retrieves an output, (figure 2.1(b)). First, the inputs are weighted according to a set of weights $\{\omega\}$. Then the processing unit will sum these weighted inputs and pass them through an activation function. Thus, the final output is given by:

$$y = f\left(\sum_i \omega_i x_i + b\right) \tag{2.1}$$

where $f(\cdot)$ denotes the non-linear activation function, $\omega_i$ represents the weight, $x_i$ denotes the input to the neuron, and b represents the bias term which is a threshold [15].

By connecting neurons together, an artificial neural network can be obtained (figure 2.1(c)) [15].



**(a)** **(b)** **(c)**

**Figure 2.1:** Biological inspiration of ANNs. a) Example of a biological neuron. Image adapted from [16]. b) Mathematical model of a neuron. Image retrieved from [17]. c) Example of an artificial neural network.

A neural network receives as input a vector, which will be processed along the hidden layer(s). Each hidden layer contains a set of neurons that are completely connected to the neurons of the previous layer but are not connected between them. The strength of the connection between two neurons is specified by the weight. The information propagates sequentially from the input layer to the last layer, which is called the output layer [15]. In a classification task the output of the network corresponds to the probabilities of each class. For example, in figure 2.2 a neural network is represented for Iris flower dataset [18, 19] classification. This dataset has three classes (flower species): setosa, versicolor and virginica. For each sample, four features were measured: sepal length, sepal width, petal length and petal width. For the example in question, the input vector is $\begin{bmatrix}4.5 & 2.3 & 1.3 & 0.3\end{bmatrix}$, after forward passing this input vector through the neural network, the class with highest probability (80 %) is setosa. Therefore, the sample with feature vector $\begin{bmatrix}4.5 & 2.3 & 1.3 & 0.3\end{bmatrix}$ is classified as belonging to setosa.

In the previous example, the input was a vector of size 4. What will happen if the input is an image? If the image has low resolution, for example $16 \times 16 \times 3$, then each neuron in the first fully connected layer would have $16 \times 16 \times 3 + 1 = 769$ weights. For an image of this size the fully connected network might work fine. However, for a high resolution Red-Green-Blue (RGB) image, for example $512 \times 512 \times 3$, the number of weights for each neuron becomes $512 \times 512 \times 3 + 1 = 786,433$ (see figure 2.3(a)). This is a significantly high number which would not only lead to overfitting but also create a computationally expensive neural network [15]. Therefore, based on the idea of neural networks, CNNs were designed to solve this problem. They take into account the idea that different pixels in an image aren't completely

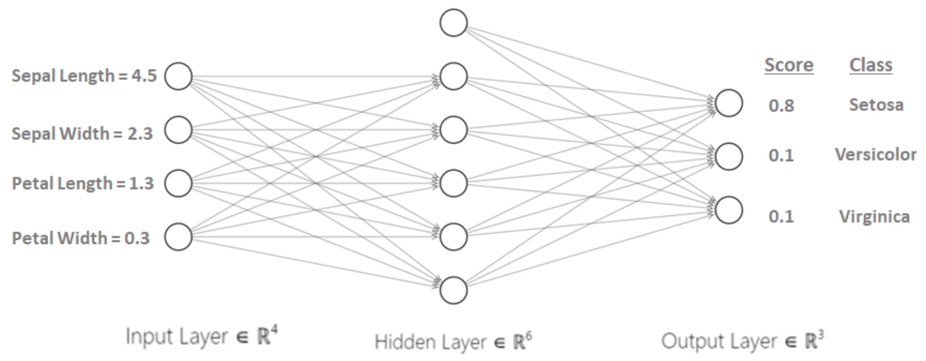


**Figure 2.2:** Classification example for Iris flower dataset using a neural network. The input vector has size 4 (sepal width, sepal length, petal length, petal width). The output layer provides a classification score for each class.

independent, therefore, each neuron looks at a region in the previous layer. The CNN architecture is based on the functioning of the brain. In the brain, each neuron responds to stimuli in a restricted region of the visual cortex, the receptive field of that neuron [20]. Similarly, in the context of CNNs receptive field is the area in the image at which the neuron is looking at. A CNN has several layers, and in each layer the neurons are organized into multiple feature maps. The neurons belonging to the same feature map share the same weight parameters. This weight sharing allows to reduce the number of learnable parameters [15]. Therefore, regardless the size of the input image, side by side neurons in a feature map, which observe a region of size $(n, m, d)$ and share the same weights, only need to learn $n \times m \times d + 1$ parameters. Typically they need to learn $3 \times 3 \times d + 1$ or $5 \times 5 \times d + 1$ parameters, where d is the depth of the input image or the number of feature maps in the last layer and 1 stands for the bias term. For instance, in figure 2.3(b) the neurons in the first layer are organized in three feature maps (green, orange and blue). As explained before, regardless the size of the input image the green neurons only need to learn $3 \times 3 \times 3 + 1 = 28$ parameters, because each green neuron looks at a region of size $3 \times 3 \times 3$ in the input image. By the same logic, the orange neurons need to learn $5 \times 5 \times 3 + 1 = 76$ parameters and the blue neurons also need to learn 76 parameters. The weight parameters are also known as the filter/kernel of the convolution operation which is explained in detail in the next subsection.

### 2.2.2 Convolutional Layer

The main building block of a CNN is a convolutional layer. Each convolutional layer has one or more filters. The filter is the kernel of the convolution operation. Therefore, first the input image will be convolved by a filter, i.e., the kernel is slid across the image computing dot products at each location. For every pixel in the input image, the filter is placed on top of that pixel and dot product is computed, this will return one value that is placed in the filtered image [15]. In figure 2.4, a schematic representation of the convolution operation is presented, note that the input image has size $6 \times 6$ and the output filtered

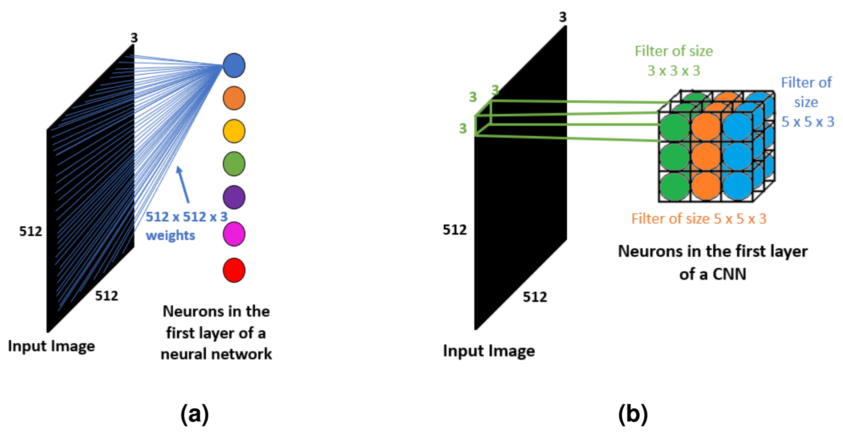


**Figure 2.3:** Illustrative representation of the neurons in the first layer of a neural network (**a)**) and the neurons in the first layer of a CNN (**b)**) for an input image of size $512 \times 512 \times 3$.

image has size $4 \times 4$. This happens because the center of the kernel can't be placed on top of the pixel located at the left top corner. In order to solve this problem, the original image can be padded by values. For instance, zero padding sets pixels outside the original image to zero [13]. The number of pixels to pad in each direction is calculated as:

$$p = \frac{f-1}{2} \tag{2.2}$$

where $f \times f$ is the size of the convolutional kernel, and p is the number of pixels to pad in each direction [15]. That is, padding an image with p zeros changes its size from $(h, w)$ to $(h + 2p, w + 2p)$ [21]. Figure 2.5 represents the result of the convolution operation where the original image is padded with zeros.

Another important concept is the stride. In the convolution operation depicted in figure 2.4, the filter shifts one pixel at a time, therefore the stride in that example is one. The stride represents the number of units/pixels by which the filter shifts at a time while being convolved with the image or with the feature map [22]. If an image of size $(h, w, d)$, is convolved with N filters of size $f \times f \times d$, considering that the image is padded with zeros according to the formula represented in equation 2.2, and considering that the stride of the convolution operation is $s$, the size of the output filtered image(s) is given by:

$$H = \frac{h + 2p - f}{s} + 1, \quad W = \frac{w + 2p - f}{s} + 1, \quad D = N \tag{2.3}$$

where (H, W) is the size of the filtered image(s) and D is the number of filtered image(s) [15].

Each filter in the convolutional layer will produce a filtered image. In figure 2.6, each number in the filtered image is the result of taking a dot product between the filter and a small volume $5 \times 5 \times 3$ of the image. In fact it is a 75-dimensional dot product plus a bias term ( $\omega^{\intercal}\mathbf{x} + b$, where $\omega$ denotes the weights of the filter, $x$ represents the values of the image in a window of size $5 \times 5 \times 3$ and b represents the bias term).

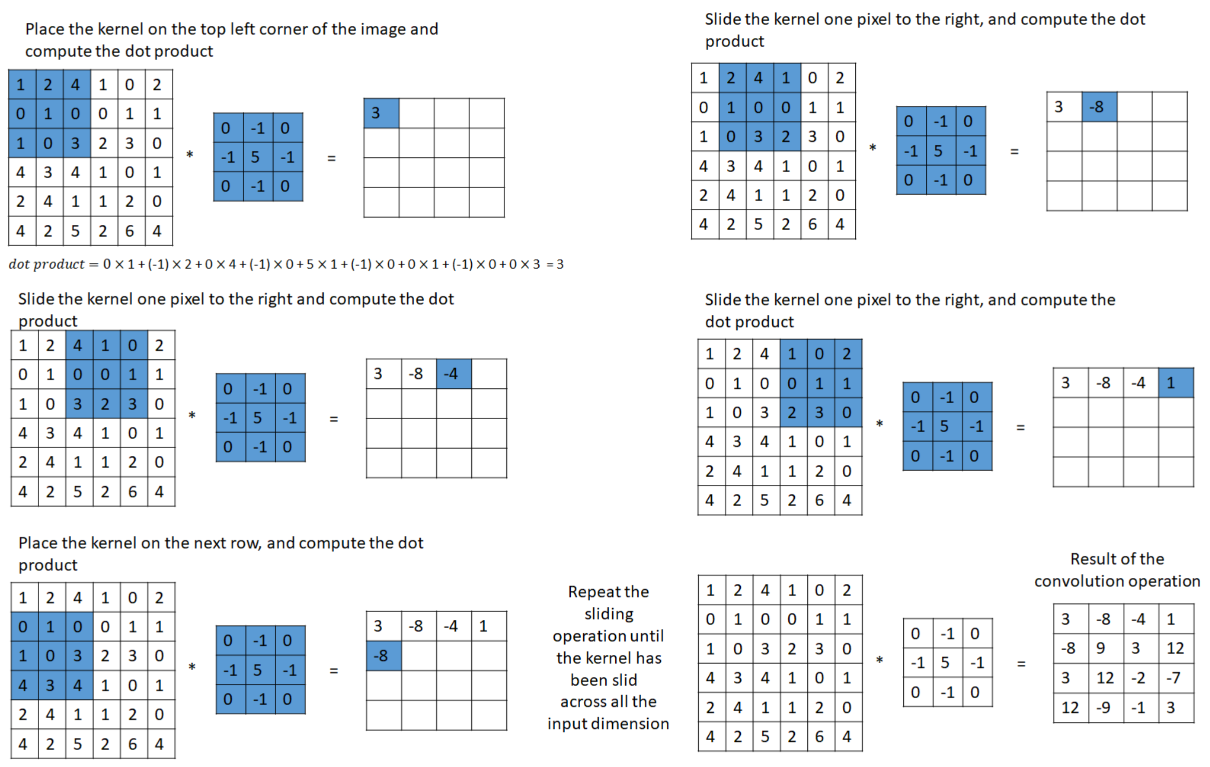

Place the kernel on the top left corner of the image and compute the dot product
dot product = 0 × 1 + (-1) × 2 + 0 × 4 + (-1) × 0 + 5 × 1 + (-1) × 0 + 0 × 1 + (-1) × 0 + 0 × 3 = 3
Slide the kernel one pixel to the right, and compute the dot product
Slide the kernel one pixel to the right and compute the dot product
Slide the kernel one pixel to the right, and compute the dot product
Place the kernel on the next row, and compute the dot product
Repeat the sliding operation until the kernel has been slid across all the input dimension
Result of the convolution operation

**Figure 2.4:** Illustration of the convolution operation.

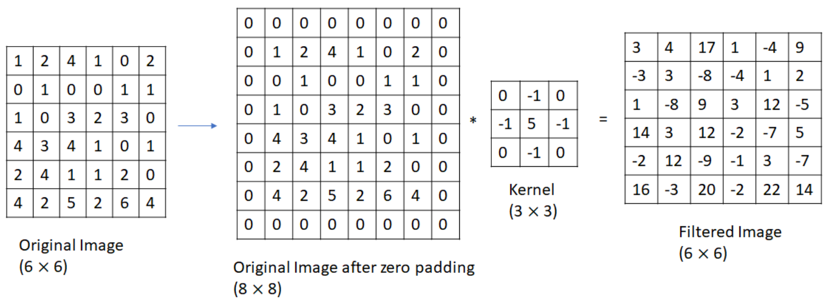


**Figure 2.5:** Result of a convolution operation after zero padding the original image.

Convolution is a linear operation, however the solution to most of the problems correspond to a non-linear mapping between the input and output. Therefore, non-linear activation functions are applied to the filtered image(s). This will allow the network to learn non-linear decision boundaries. Without activation functions the network would be a linear mapping between the input and the output [15].

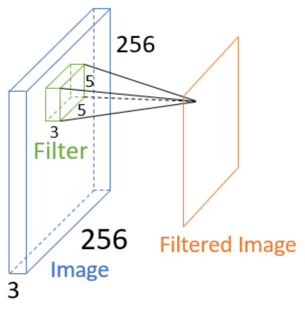
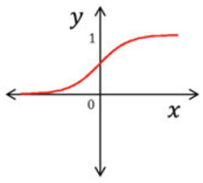
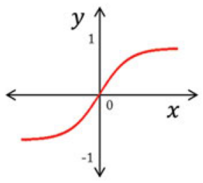
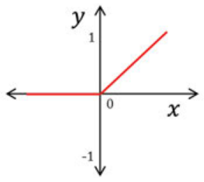
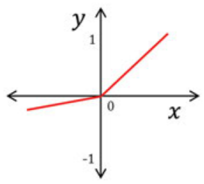


**Figure 2.6:** Result of taking a dot product between the filter and a small (5 x 5 x 3) patch of the image (i.e. 75 dimensional dot product + bias).

### 2.2.3 Activation Functions

The non-linear activation function can be interpreted as a selection mechanism that decides if a certain neuron, given its inputs, will be firing or not. The activation functions used in deep learning are differentiable in order to allow the backpropagation optimization [15], which will be introduced in section 2.3. Table 2.1 summarizes the most common activation functions used in deep learning, its advantages and disadvantages.

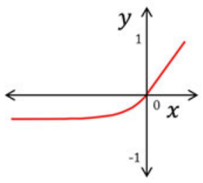
**Table 2.1:** Commonly used activation functions in deep learning, mathematical formula, plot, advantages and disadvantages. Images retrieved from [15].

| Activation Function | Plot | Advantages | Disadvantages |
|---|---|---|---|
| Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ |  | • Squashes the number to the interval $[0, 1]$; • Interpreted as a saturating "firing rate" of a neuron. [23] | • Saturated neurons/units can "kill" [1] the gradients; • Output isn't zero centered; • Exponential is computationally expensive. [23] |
| Hyperbolic Tangent (Tanh) $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ |  | • Zero centered. [23] | • When saturated "kills" the gradients. [23] |
| Rectified Linear Unit (ReLU) $relu(x) = max(0, x)$ |  | • In the positive region (i.e. $x > 0$) it doesn't saturate; • Computationally efficient; • Converges faster than sigmoid/Tanh. [23] | • The output isn't zero centered. • In the negative region (i.e. $x < 0$), it can result in dead neurons, which will never activate and/or update. [23] |
| Leaky Rectified Linear Unit (LeakyReLU) $leakyrelu(x) = max(0.01x, x)$ |  | • Doesn't saturate; • Computationally efficient; • Doesn't result in dead neurons. [23] | • There isn't a significant improvement in the performance when compared to ReLU. [23] |

Continued on next page

[1] If a neuron is saturated it will generate a small local gradient. "Killing" the gradient means that it will be so small that there will be no signal flowing through the saturated neuron. [9].

**Table 2.1 – continued from previous page**

| Activation Function | Plot | Advantages | Disadvantages |
|---|---|---|---|
| Exponential Linear Unit (ELU) $elu(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$ |  | • Mean activations are pushed towards zero. [23] | • Requires the calculation of the exponential which is computationally expensive; • Doesn't center the values at zero. [23] |

## 2.2.4 Pooling Layer

The pooling operation down-samples the feature map (reduces its height and width). This operation allows not only to reduce the required amount of computation [15], but also allows to obtain a feature representation which is invariant to small translations in the input [9]. For a pooled region of size $f \times f$ and stride $s$, the size of the output activation map is given by:

$$H = \frac{h - f + s}{s}, \; W = \frac{w - f + s}{s} \tag{2.4}$$

where $(h, w)$ and $(H, W)$ are the height and the width of the input feature map and of the result of applying the pooling operation to this feature map, respectively [15]. For an input volume of size $(h, w, d)$ the pooling layer produces an output of size $(H, W, d)$, where the relation between $(H, W)$ and $(h, w)$ is represented in equation 2.4. The most common pooling operations are the max and average pooling [14].

Max pooling slides a window across the feature map, and at each position the output corresponds to the maximum value within that window. This operation is usually applied to non-overlapping regions of the feature map, therefore the window size is typically $2\times2$ and the stride is generally $2$, this will give an output where the height and the width are halved [22]. Figure 2.7 presents an example of a max pooling operation, with window size $2 \times 2$ and stride $2$.

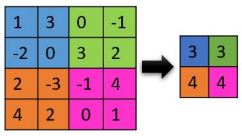


**Figure 2.7:** Example of a max pooling operation applied to a feature map of size $4 \times 4$ (represented on the left) and the corresponding output after applying the max pooling operation (represented on the right), which has size $2 \times 2$.

In summary, the main building block of a CNN (represented in figure 2.8) is composed of a convolution operation which given an image ( $I$ ) and N filters ( $\{\omega_1, ..., \omega_N\}$ ), generates N filtered images ( $\{I * \omega_1, ..., I * \omega_N\}$, where $*$ denotes the convolution operation). Thereafter, an activation function ($f(\cdot)$) is applied to these filtered images, which generates N activation/feature maps ( $\{f(I * \omega_1), ..., f(I * \omega_N)\}$ ) . Finally, an optional pooling operation is applied to these N activation maps, which generates N downsampled activation maps ( $\{pool(f(I * \omega_1)), ..., pool(f(I * \omega_N))\}$ ).

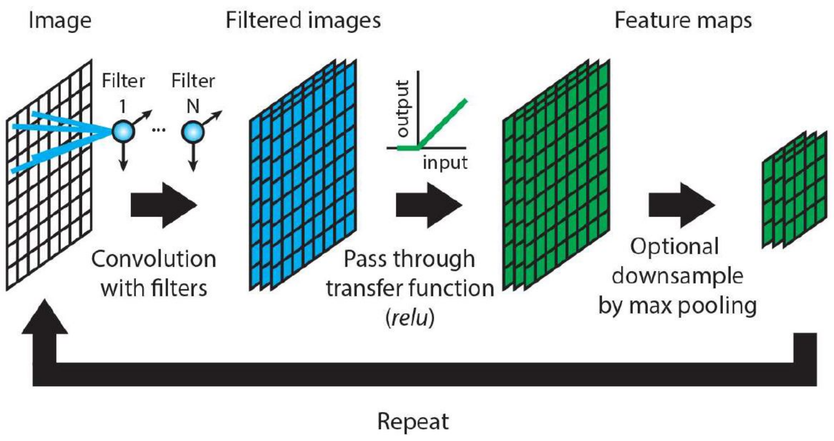


**Figure 2.8:** Main building block of a CNN. Image retrieved from [24].

### 2.2.5 Fully Connected Layer

As stated in the last subsection, CNNs are mainly composed of stacks of convolution and pooling layers, these layers extract features from the input image [22]. If a CNN is designed to solve the problem of image classification, these layers are followed by one or more fully connected layers. The last fully connected layer generates the class label, it outputs a M dimensional vector, where M is the number of classes of the classification problem [14]. The image classification task is explained in detail in subsection 2.4. Each neuron in the Fully Connected Layer (FCL) is connected to all the units of the previous layer [15]. For instance, if the last output volume of the convolutional and pooling layers has size $(h, w, d)$, each unit of the first FCL will have $h \times w \times d + 1$ weights/connections (see figure 2.9). The first FCL will look at the high level features, extracted by the convolutional and pooling layers, and will learn the correlation between these features and a particular class [25]. In the following FCL each unit will have N connections, where N is the number of units in the last FCL. As stated before, the last FCL will output the probabilities of each class. For the example represented in figure 2.9, the output has two units, since it is a binary classification problem.

## 2.3 Training

The CNN has the capability of learning automatically the free parameters (i.e. the biases and the weights) [26]. In this way, a CNN has the ability to learn features, it starts by learning simple features (for example, edges) and then aggregates them into progressively more complex features (for example, corners) [8].

The training of a network, which is focused on the estimation of the weights, is converted into an

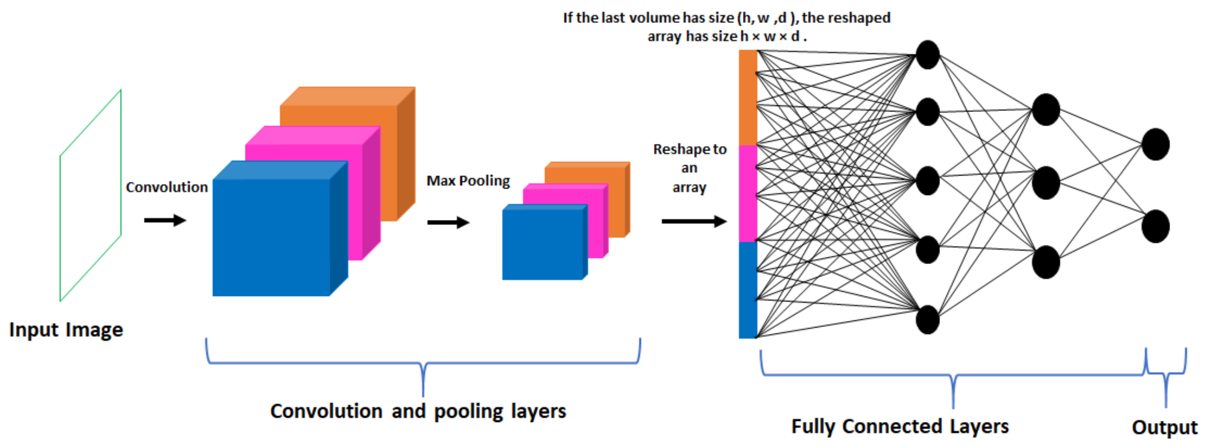

If the last volume has size (h, w ,d ), the reshaped array has size h × w × d .
Convolution
Max Pooling
Reshape to an array
Input Image
Convolution and pooling layers
Fully Connected Layers
Output

**Figure 2.9:** Example of a CNN to tackle the problem of image classification, where the number of classes is two.

optimization problem. CNNs learn the parameters from a training set, just like supervised machine learning methods. The training set consists of manually annotated datasets. For example, to perform the image classification task, the training set is composed of images and the respective labels/classes, these labels are the ground truth. During the training procedure the main goal is to minimize a loss function, which typically measures the difference/error between the network's output and the ground truth data [15]. The model's parameters are learned by solving an optimization problem, which is defined as follows:

$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}} \quad L(\omega) \tag{2.5}$$

where $\omega$ denotes the model's parameters and $L(\omega)$ represents the loss function. To minimize the loss function, numerical optimization methods are used [15]. The optimizer will define how the parameters are going to be updated. Commonly used optimizers are Stochastic Gradient Descent (SGD), SGD with momentum, SGD with nesterov momentum, Adaptive Gradient (AdaGrad), Adaptive Delta (AdaDelta), RMSprop, ADAptive Moment Estimation (ADAM) [15]. Figure 2.10 shows the training loss as a function of the training iterations, to solve a handwritten digit classification problem using the Modified National Institute of Standards and Technology (MNIST) dataset [2], in order to compare the convergence performance of different optimizers. This plot shows that ADAM is the best optimizer in this example, since it allows the network to achieve the smallest loss in comparison with the loss obtained by using the other optimizers. In fact, the ADAM optimizer is the one that is commonly used in deep learning [15]. Despite this fact, any of the above mentioned optimizers can be used to train a deep learning model. For instance, the minimization of the loss function can be attained by using the gradient algorithm:

$$\omega_i(t+1) = \omega_i(t) + \Delta\omega_i(t), \quad \Delta\omega_i(t) = -\eta \frac{\partial L}{\partial w_i}\Big|_{\omega(t)} \tag{2.6}$$

[2] MNIST dataset is a database of handwritten digits, which is composed of 60,000 training images and 10,000 test images [27].

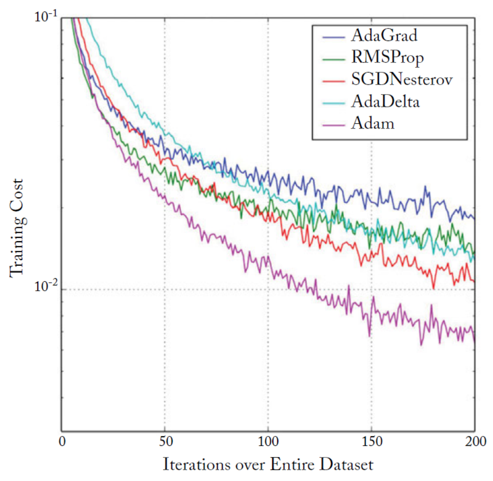


**Figure 2.10:** Training loss as a function of the training iterations, to solve the classification problem in the MNIST dataset. Image retrieved from [15].

where the $\eta$ denotes the learning rate, $\omega_i\,(t+1)$ represents the weight $\omega_i$ at iteration $t+1$, and $\omega_i\,(t)$ represents the weight $\omega_i$ at iteration $t$. This algorithm defines how the weights must be adjusted to minimize the loss function. In order to calculate the gradient $\Delta\omega_i(t)$ (represented in equation 2.6), backpropagation algorithm is commonly used. First, the input is forward passed through the network and the corresponding output is obtained. The loss value can be calculated by comparing this output with the corresponding ground truth data. Taking into account this loss value, the backpropagation algorithm goes backward (from the last layers to the first layers), and applies the chain rule to compute each parameter's contribution in the loss [22]. Thus, the backpropagation algorithm allows to calculate $\Delta\omega_i(t)$ and the weights are updated in each iteration according to the optimizer chosen to solve the problem.

In summary, the training procedure corresponds to the tuning of the model's parameters (weights and biases) in order to minimize a loss function, this is done by solving an optimization problem. The optimizer defines how the parameters are updated, and typically it requires the calculation of the gradient of the loss function with respect to the parameters, which is calculated using the backpropagation algorithm. Finally, the model's parameters are adjusted according to the formula defined by the optimizer method. This procedure is repeated until some stopping criterion is verified.

### 2.3.1 Training Modes

An epoch is when all the training examples have been used once to update the parameters. Depending on the number of examples from the training set that are used by the optimization algorithm to compute the gradient in each iteration, three types of training modes can be defined [9]:

1. Batch - all examples from the training set are used, in this case one iteration is equivalent to one

epoch;

2. Minibatch - uses a subset of the training set, however, the size of a minibatch is commonly described by the term batch size; in this case, if a dataset of N examples is divided into M batches, then $\frac{N}{M}$ iterations complete one epoch;

3. Online / Real time - uses one training example, if the training set has N examples, then N iterations complete one epoch.

### 2.3.2 Weights Initialization

Initially, some values need to be assigned to the weights of the network. So, in the first iteration after forward passing the input through the network a loss value can be calculated, which will be high in the beginning since the weights were initialized randomly [22]. It has been shown that certain initializations can make the training procedure more stable. Therefore, several methods for weights initialization have been proposed, such as the Gaussian random initialization, uniform random initialization and Xavier initialization [15].

### 2.3.3 Training, Validation and Test Sets

As explained in the beginning of section 2.3 a training set is necessary during the training stage, this set will allow the network to learn the weights by solving an optimization problem. However, the question that arises is: ”when should the training process be stopped?”. To answer this question the concept of validation set is essential. Validation set is a set that is independent of the training set, i.e., it contains examples that aren't present in the training set. The main goal after training the network is to obtain a model that generalizes well to new unseen data, which the network has never seen during the training process. Therefore, the network is trained using the training set, and its performance is evaluated on the validation set [28].

A common way to stop the training process is based on early stopping. This is a method to stop the training process when the validation loss starts to increase, this is referred as overfitting. Overfitting happens when the model's performance on the training set is high but its performance on the validation set is bad, this means that the model's generalization capability is bad, since it performs poorly on unseen data [15]. Figure 2.11 illustrates an early stopping approach during the training process, the training is stopped at iteration t, because from this iteration the validation error starts increasing. Another way to stop the training procedure is when the validation loss is small and smooth during several epochs. Thus, the set of weights that are kept are based on the validation set, therefore, this set doesn't provide an unbiased estimation of the model's performance. To solve this problem, the test set, an independent set from the training and validation set, is used to obtain a final unbiased estimation of the model's

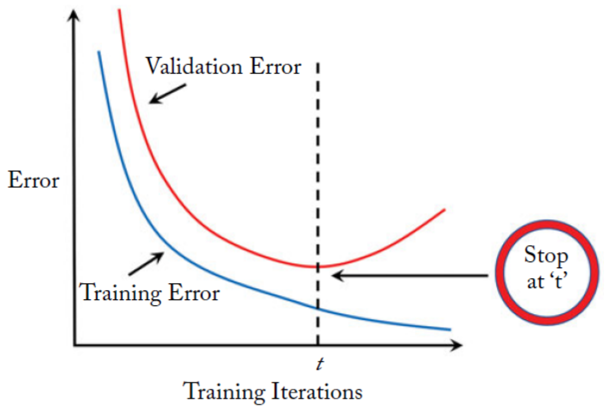


**Figure 2.11:** Training and validation loss as function of the training iterations, and illustration of an early stopping approach during the training process. Image adapted from [15].

performance [28]. To obtain the training, validation and test sets, the dataset is split into 3 independent sets. The most typical split consists in considering 60 % of the dataset as training, 20 % as validation and the other 20 % as test set [9].

## 2.4 Image Classification, Object Detection and Segmentation

In this section four computer vision tasks are presented: image classification, object detection, semantic segmentation and instance segmentation.

In the image classification task, the main goal is to predict which is the object in the image or, if the image contains more than one object, to predict the list of objects [29] (figure 2.12(a)). In object detection the goal is to provide a bounding box and a class for each object in an image [29], as illustrated in figure 2.12(b). If there are two or more instances of the same object, the output of the object detection task will provide a bounding box and classification score for each instance.

Image segmentation corresponds to the process of dividing an image into multiple parts, regions or sets of pixels [13]. Two types of segmentation can be performed: semantic segmentation and instance segmentation. In semantic segmentation the main goal is to make pixel wise predictions. Each pixel is classified as belonging to a certain class [29]. In the output of a semantic segmentation problem, two instances of the same object have the same label. For instance, in figure 2.12(c), the three cubes are labeled with purple color. In this type of segmentation the two touching cubes are identified as being a single object.

Finally, in instance segmentation the goal is to make pixel wise predictions and to identify each object individually. In this case, the output of the segmentation can discriminate different instances of the same object [29]. As illustrated in figure 2.12(d), each cube has a different label. The task of instance segmentation can be viewed as a combination of object detection and semantic segmentation.

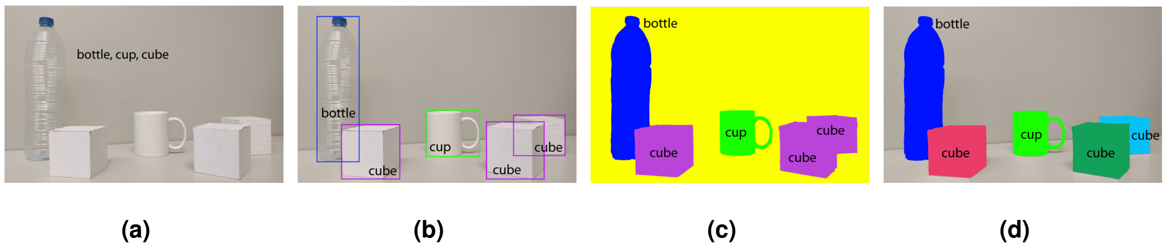


**Figure 2.12:** Illustration of the outputs of four computer vision tasks: a) Image classification, b) Object detection, c) Semantic segmentation and d) Instance segmentation. Images retrieved from [29].

## 2.5 State-of-the-Art

The structure of the nucleus plays a key role in the diagnosis of cancer or in the cell cycle staging, as mentioned in section 1.1. Therefore, several methods have been developed in order to allow its segmentation and later extraction of features to study its structure. In this section, state-of-the-art regarding nuclei segmentation will be presented.

### 2.5.1 Nuclei Segmentation Challenge

Segmentation of cell nuclei is important in several applications, as mentioned before. However there are a lot of challenges and difficulties related to this task, namely associated with:

- Image acquisition: presence of noise, background clutter [30], blurriness [31];
- Biological data: nucleus occlusion [30], touching or overlapping nuclei [30], variations in shape [31] and texture (differences in chromatin distributions) [32], differences in nuclear appearance in different pathologies [33];
- Experimental variations: preparation of the samples isn't uniform [34], variations due to different illumination conditions [24], use of different staining methods [33].

Although it is important to pay attention to all these difficulties, the one that has been object of study for several years is the presence of touching and overlapping nuclei. If the main purpose after nuclei segmentation is the extraction of features from each individual nucleus, it is important to perform nuclei segmentation at instance level. However, the presence of these overlapping nuclei is the main obstacle towards nuclei instance segmentation [2, 3].

Nuclei segmentation performed by a human operator is time-consuming and subjective. The final outcome depends strongly on the clinician's experience, however even a segmentation performed by an experienced pathologist can be subjected to human error. Moreover, it isn't a practical approach in high-throughput applications. Hence, with the increasing need to perform accurate and objective nuclei segmentation and to alleviate the workload of the pathologist, several automatic approaches have been proposed to tackle the problem of nuclei segmentation [2].

### 2.5.2 Classical Approaches for Nuclei Segmentation

Traditional techniques for nuclei segmentation, also known as handcrafted feature based approaches, include a combination of methods for nuclei detection, which can provide a seed for each nucleus or the area of nuclei, and nuclei splitting. The most common and popular method applied for nuclei detection is thresholding based in Otsu's method. [2, 3] Other approaches include, for example K-means clustering, graph cuts based methods, H-maxima transform and combination of Laplacian of Gaussian with Euclidean distance map. [2, 3],

However, cell overlapping commonly occurs in biopsy images. Therefore, if nuclei detection step generates the area of nuclei, after this stage many overlapping nuclei appear [35]. In order to split them, several methods can be applied, for example: least square ellipse fitting, concavity detection, edge path selection, Fourier shape descriptor [2]. On the other hand, if the seed of each nucleus has been obtained in the nuclei detection step, region growing [32], or marker-controlled watershed [32, 36] can be applied to obtain the contour of each nucleus.

Nonetheless, several issues are associated with these traditional techniques. In some cases manual tuning of parameters is necessary [3], in other cases the methods developed can be very specific to a given type of tissue or image [29]. Recently, the deep learning based approaches, which have been successfully applied in several tasks of computer vision, like image classification and object detection [8], are attracting the attention of researchers [3]. These methods have the ability to automatically extract important features from the image and present a good generalization capability, hence their superiority when compared to traditional techniques mentioned above [29].

### 2.5.3 CNNs for Nuclei Segmentation

Several deep learning models have been proposed for cell nuclei segmentation, the first approaches are based on CNNs. CNNs present state-of-the-art performance in many computer vision tasks. For instance, Song et al. (2014) [37] proposed a method based on CNN for the segmentation of cervical nuclei and cytoplasm. They applied a CNN for nuclei detection and then performed coarse segmentation based on sobel edge operator, morphological operations and thresholding. They obtained 94.50 % accuracy for nuclei detection, and regarding segmentation, they achieved a 91.43 % and a 87.26 %, precision and recall, respectively. Recently, Xing et al. (2016) [30] applied a Two-Class Convolutional Neural Network (CNN2) to digitized histopathology images, in order to generate probability maps for nuclei. Additionally, in order to solve the problem of overlapping nuclei, not only a robust shape model (dictionary of nuclei shapes) was constructed, but also a repulsive deformable model at local level was applied. In other words, alternatively the shape deformation is performed based on the deformable model and shape inference is made based on the dictionary containing nuclei shapes. On the other hand, Kumar et al. (2017) [38] proposed a model for nuclei segmentation based on CNNs, that predicts not only the nuclei but also the boundary of each nucleus, as represented in figure 2.13. Their results show that a Three-Class Convolutional Neural Network (CNN3) provides significantly better results in comparison with a CNN2. This is due to the fact that unlike CNN2, which only predicts nuclei and background, in CNN3 an additional class is added in order to predict nuclear boundary pixels. However, the post-processing step proposed in this method is time consuming. In this step, first the seeds of the nuclei are obtained by applying a threshold to the probability map. Then, an iterative procedure is followed to grow each seeded nucleus until a stopping criterion is verified, which is a slow procedure. Additionally, it is important to mention that the probability maps generated by CNNs are obtained by processing patches obtained from an image. For instance, CNN2 classifies the central pixel of each patch as belonging to a nucleus or to the background. It works in a sliding window based approach, that is, for each pixel in the image a patch of a given size is extracted and passed through a network, which classifies the central pixel of the patch. The network will scan the entire image providing an output segmentation mask where each pixel has a class (for example, foreground or background). This approach is computationally expensive, making the segmentation of an entire image very slow. For example, for an image of size $1024 \times 1024$, $1,048,576$ patches must be generated and the CNN has to classify each of them. So, CNNs are based on a sliding window approach, a better approach consists of taking as input the image and providing as output its segmentation mask on a single pass. An example of a model based on this idea is the Fully Convolutional Network (FCN).

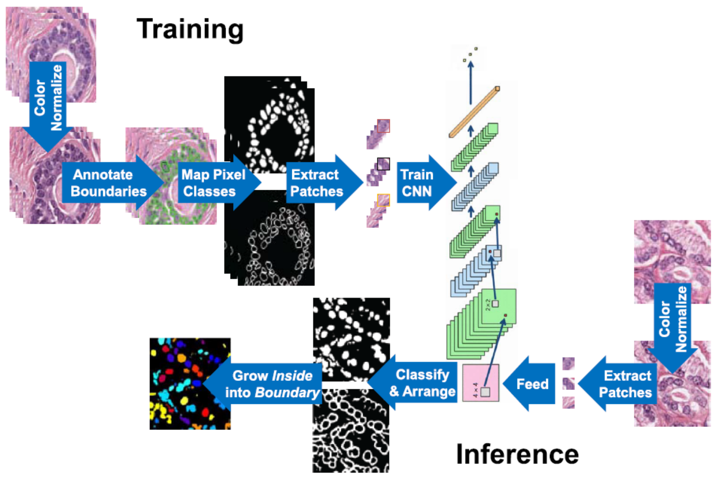
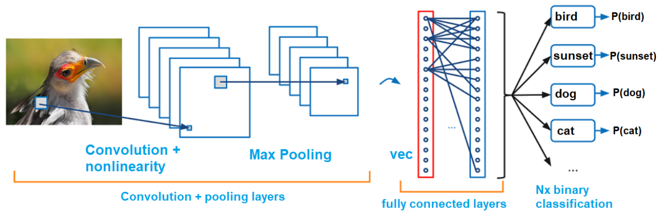


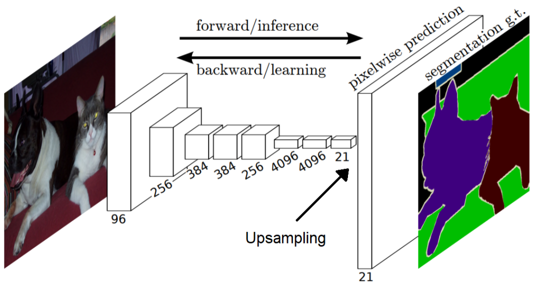
**Figure 2.13:** Schematic representation of the training and inference proposed in [38]. Image retrieved from [38].

### 2.5.4 FCN for Nuclei Segmentation

The first FCN for semantic segmentation was presented by Long et al. (2015) [39]. Their results showed that the FCN can achieve state-of-the-art performance in terms of segmentation. Furthermore, the inference step associated with this method is significantly faster, it takes less than 12 s per image to obtain the corresponding segmentation mask. It is inspired by the deep learning architectures designed for classification, where an image first goes through convolutional layers and then through Fully Connected (FC) layers to obtain an output which corresponds to one classification label for the entire input image (as illustrated in image 2.14(a)). For example, if the image contains a bird, the label for that image will be bird. In the case of the FCN there is an additional upsampling layer (represented in figure 2.14(b)) that allows to perform pixel-wise classification for the task of segmentation. Therefore, the final output is a segmentation mask for the input image instead of a single label. In other words, the main goal of a FCN is to obtain a classification label for each pixel in the input image.



(a)

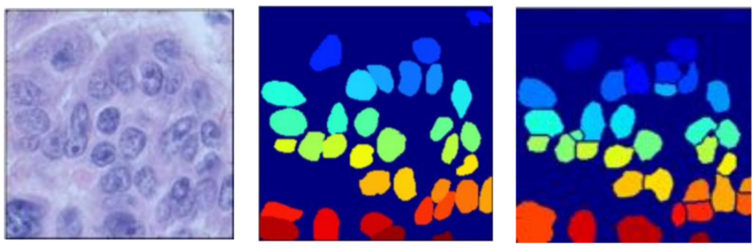


(b)

**Figure 2.14:** Comparison between a deep learning architecture used for classification and the FCN. In the architecture of FCN an additional upsampling layer is present, therefore the output is a segmentation mask of the input image. **a)** Example of an architecture used in classification problems. Figure adapted from [25]. **b)** Architecture of FCN presented in [39]. Image retrieved from [39].

In order to perform nuclei segmentation in histopathology images, Naylor et al. (2017) [40] used the FCN to obtain the nuclei probability map, afterwards watershed method was applied in order to split the touching nuclei. Although it achieved an accuracy of approximately 94 %, the final segmentation mask shows that nuclei boundaries predicted by this method aren't that accurate, as depicted in figure 2.15(c), in comparison with the ground truth image represented in figure 2.15(b).

Recently, in 2018, Höfener et al. [33] presented a study where they used a FCN for nuclei detection in Hematoxylin-and-Eosin (HE) stained images. They evaluated different configurations and parameters for nuclei detection. Additionally, the authors proposed a post-processing step for the probability maps generated by the FCN, which is designed to attenuate the effect of noise and outliers. This step corresponds to the application of the median filter and then the Gaussian filter. Finally, their results show that the proposed post-processing step is important to achieve good results, obtaining an F1 score of approximately 81 %.



(a) (b) (c)

**Figure 2.15:** Example of a result obtained by applying the method described in [40]. Images retrieved from [40]. **a)** Original Image (RGB), **b)** Ground Truth, **c)** Prediction.

## 2.5.5 U-Net and Nuclei Segmentation

Investigation in the area of deep learning is increasing rapidly, hence new architectures are being developed at a significantly fast speed. Additionally, taking into account the importance of cell nuclei segmentation, there are a number of approaches that have been presented to solve this problem, most of which are based on a simple architecture: U-Net [41]. U-Net is the most common architecture used for medical image segmentation. Specially designed for biomedical image segmentation, this architecture won the Cell Tracking Challenge in 2015 [41]. U-Net is inspired by FCN, however it has more up-sampling layers than FCN, making it symmetric, as illustrated in figure 2.16. In fact, the U-Net name derives from the symmetric U-shape of the architecture.

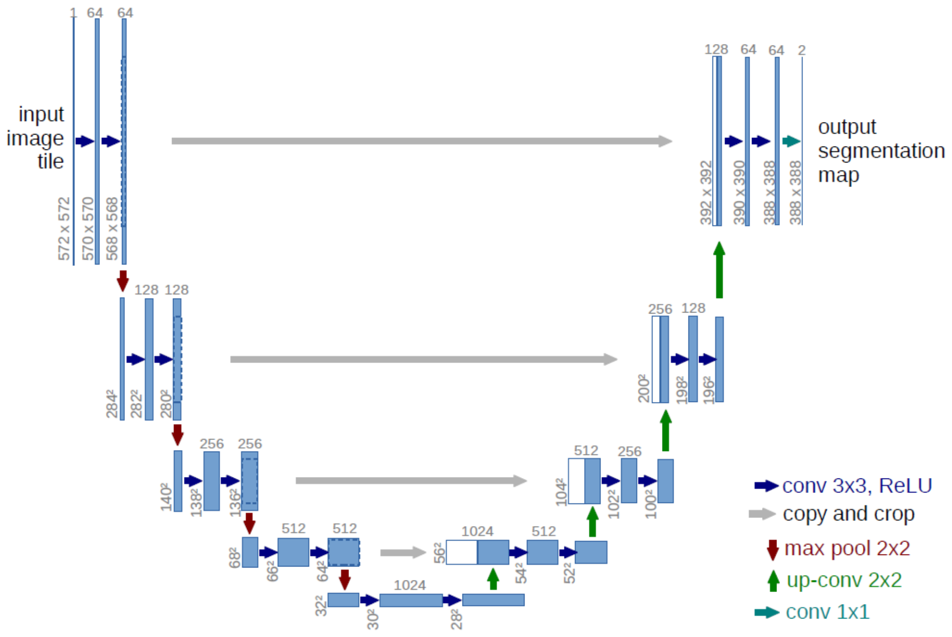


**Figure 2.16:** Schematic representation of the U-Net architecture proposed in [41]. Image retrieved from [41]

The U-Net architecture is composed of two paths [41]. The first path is the down sampling path, this is a contracting path which is also known as an encoder. The encoder is mainly composed of convolution and pooling layers, which allow to extract high level features from the image. In the encoder, the size of the image decreases, whereas the depth increases. While the spatial information is decreasing the receptive field is increasing, due to max pooling operations. After each layer the receptive field is increasing because the filters can capture larger areas, since after the max pooling operations the less important pixels are removed. The encoder generates feature maps which are low resolution representations of the input image [29]. The second path is the up-sampling path, an expanding path also known as a decoder. This path allows to convert the low-resolution images into a high-resolution image that represents pixel wise segmentation of the original image [29]. Several methods have been proposed for up-sampling, for example bilinear interpolation, nearest-neighbor interpolation and bicubic interpolation [42]. The decoder of the U-Net is composed of transposed convolutions, which is an image up-sampling technique with learnable parameters. Transpose convolution is also known as fractionally-strided convolution [15]. Figure 2.17 depicts a transposed convolution operation. Each value in the input gives a multiplication factor for the kernel. For instance, first the kernel is multiplied with the first value of the input (zero) and outputs a $3 \times 3$ matrix with zeros represented in blue, next the kernel is multiplied with 1 and outputs the values represented in the green matrix, the same is done for the other two numbers in the input. Finally, the positions where the output overlaps are summed. The weights of the kernels in the transposed convolution layers are learnable, just as the weights of the kernels in the convolutional layers. Thus, they will be adjusted during the training process.

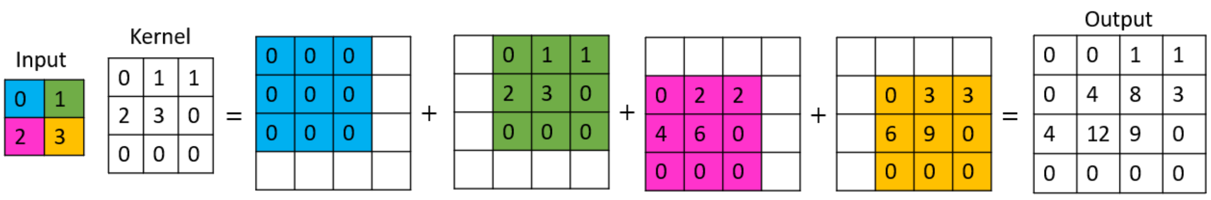


**Figure 2.17:** Illustration of the transposed convolution operation. The input is a $2 \times 2$ matrix, the kernel is a $3 \times 3$ matrix, and the output is a $4 \times 4$ matrix.

While in the convolution operation multiple activation units produce one activation in the output feature map, in the transpose convolution operation each unit in the activation map generates multiple activation units in the output (see figure 2.18).

For each layer in the expanding path the width and height of the image are doubled and the depth (number of channels) is halved. Additionally, in the decoder there is a concatenation operation where the information from the encoder is added, this allows for better spatial localization. In this step, the spatial information from the contracting path is included in the expanding path, this operation is represented by the horizontal gray arrows in figure 2.16.

U-Net is a semantic segmentation architecture, therefore if two or more nuclei are touching they will

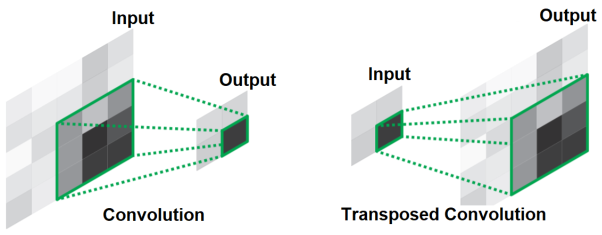


**Figure 2.18:** Illustration of the convolution and transposed convolution operations. Image adapted from [43].

be identified as a single object. Since in nuclei segmentation task each nucleus needs to be identified separately, several authors proposed methods, based on U-net, to address this difficulty. For instance, in the original U-Net paper [41] a weighted cross entropy loss function was presented in order to give higher weights to pixels that are closer or belong to touching borders of nuclei. By doing this the model is able to learn to classify these pixels as belonging to the background, so that it doesn't join two or more close instances into a single object. Since then, several approaches based on U-Net have been proposed in order to tackle the problem of nuclei segmentation. Cui et al. (2018) [32] have proposed a method, inspired by U-Net, to predict nuclei and their contours at the same time, in HE stained images. By predicting the contour of each nucleus and applying a sophisticated weight map in the loss function they were able to split touching and overlapping nuclei accurately, consequently the post-processing step consisted of a simple and parameter free procedure. Based on a similar idea, Caicedo et al. (2019) [44] trained an U-Net model in order to predict the nuclei and their boundaries, giving 10 times more weight in the loss function to the boundary class. A comparison with existing nuclei segmentation methods, namely DeepCell [24], was made, the results clearly showed that U-Net provides better results.

Other studies have shown modified versions of the U-Net or U-Net based architectures to perform nuclei segmentation. Khoshdeli et al. (2018) [34] proposed ENet. In order to solve the problem of touching nuclei their approach is based on the combination of three ENets: one for region-based segmentation, another for boundary-based segmentation and finally the outputs of these two networks are fused in a third ENet. Nevertheless, in this method additional post-processing steps are required since not all overlapping nuclei are separated.

Iglovikov et al. (2018) [45] proposed an approach to separate close instances of the same object (in their case building). They used this approach with the proposed architecture, the TernausNetV2, which was inspired in the U-Net. However, the presented approach can be used with any network that allows to do semantic segmentation. The basic idea to tackle the problem of instance segmentation relies on an additional channel to predict touching areas between objects or areas where objects are very close. Although this work was focused on satellite images, a similar approach was presented by the winning solution of Kaggle data competition 2018, which consisted on the task of instance segmentation of nuclei across different type of scans [46, 47]. The winners used an encoder-decoder type architecture based on the U-Net, initializing the encoders with pretrained weights. Additionally, in order to separate touching or overlapping nuclei, they added a third channel containing the touching borders between nuclei, converting the two-channel masks into three-channel masks. In this way the masks contain three classes: background, nuclei and touching borders. For the post processing step a combination of watershed and morphological operations was applied. On the other hand, with a similar idea in the mind, Guerrero-Peña et al. (2018) [48] proposed a new loss function to tackle the problem of instance segmentation of T-cells. They converted the binary masks into ternary masks, by adding a third class representing touching borders between cells (similar approach to the one presented in [47]). Additionally, they proposed a new loss function to give higher weights to underrepresented parts/classes of the image based on two weights maps which take into account the shape of the cells. In conclusion, the combination of a new loss function with the addition of a third class allows for better performance results in comparison with the results obtained by using the method described in [41]. However, additional post-processing steps are required in order to get the final segmentation masks.

Recently, Zeng et al. (2019) [49] proposed RIC-Unet, a network that is able to learn not only how to segment each nucleus but also its contour, based on multi-task learning. This architecture contains RI-Blocks with residual and inception modules, instead of the typical convolutional layers that U-Net has. However, an additional post-processing step is necessary in order to split touching nuclei, this step takes into account the information provided by both outputs of the network (segmentation mask and contour mask); nonetheless, the total test time for an image of size 1000 $\times$ 1000 is less than 1 second. Finally, a comparison with the methods described in [30], [38] and [41] regarding average Aggregated Jaccard Index (AJI), Dice and F1 Score was made, and it was shown that the method proposed in [49] outperforms the other methods, despite the fact that not all overlapping nuclei are split by applying this method.

US-Net was presented by Xu et al. (2019) [50] for nuclei instance segmentation. As depicted in figure 2.19, this architecture has two branches, one for segmentation and another for detection, furthermore they share the same backbone network. To achieve the instance segmentation goal, the authors combined the segmentation capability of U-Net with good object classification and detection ability of Single Shot MultiBox Detector (SSD) [51]. In a post processing step, for each detection made by the SSD, a refinement network is applied to it, this network consists of an U-Net designed to refine the detected regions. Finally, it was shown that the proposed network is more efficient than the result obtained by each branch individually.

### 2.5.6 Object Detection

As mentioned before, classical techniques for segmenting nuclei present two important steps: detection and segmentation. Throughout the years several deep learning architectures for object detection have

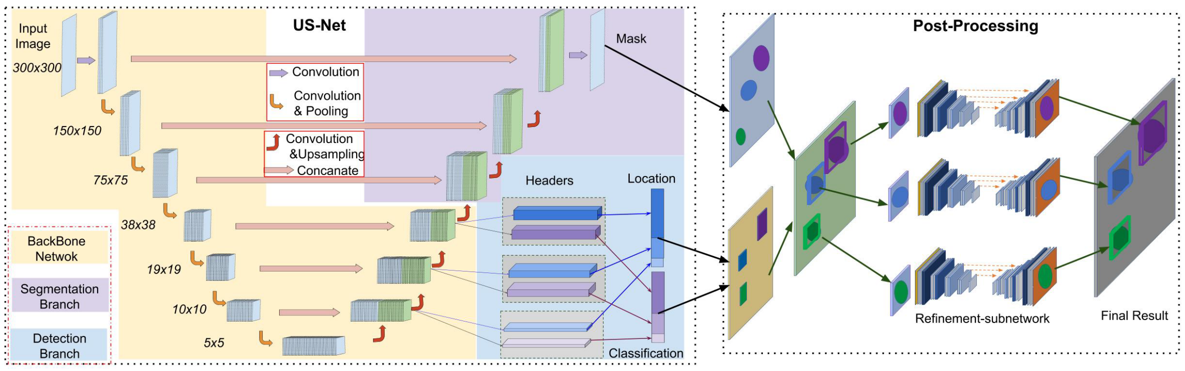


**Figure 2.19:** Schematic representation of the training and inference proposed in [50]. Image retrieved from [50].

been proposed. For instance, Girshick et al. (2014) [52] presented Region-CNN (R-CNN). Given an image, R-CNN first proposes about 2000 regions of interest, then it warps each image region and passes it through a convolutional network which gives as output the classification score for that region (as represented in figure 2.20(a)). However, the training of R-CNN is slow, and it requires a lot of disk space. Therefore, improved versions of R-CNN were developed, namely Fast R-CNN (2015) [53] and Faster R-CNN (2015) [54]. Fast R-CNN takes as input an image and various Regions of Interest (RoIs), firstly they pass through a convolutional network, then each Region of Interest (RoI) is reshaped into a fixed size through a RoI pooling layer. Finally, it is fed to a fully connected layer, which gives as output a bounding box and softmax probabilities for each class (as shown in figure 2.20(b)). The training time associated with this architecture is lower in comparison with R-CNN, however in both, region proposals are obtained through a selective search which is time consuming and computationally expensive. Thus, Faster R-CNN [54] was developed with the ability of letting the network learn the region proposals, represented by the region proposal network (RPN) block in figure 2.20(c). Compared with its predecessors, its speed is significantly higher to the extent that it can be used for real-time object detection [54].

Other state-of-the-art architectures for object detection include YOLO [56] and SSD [51], both proposed in 2016. These models are faster than region proposal based methods, however there are some disadvantages associated with them. For instance, on one hand, YOLO struggles when new objects or objects in different configurations are presented to it, because the bounding boxes are learned from the data. Additionally, it doesn't perform well on small objects that are in groups. On the other hand, SSD has a difficulty in dealing with small objects [57].

#### YOLO and YOLOv2

Given its importance for this work, a detailed explanation regarding YOLO is provided below.

YOLO is an architecture designed for object detection and classification. It is able to process images in real time at approximately 45 frames per second. YOLO is faster than region proposal based methods.

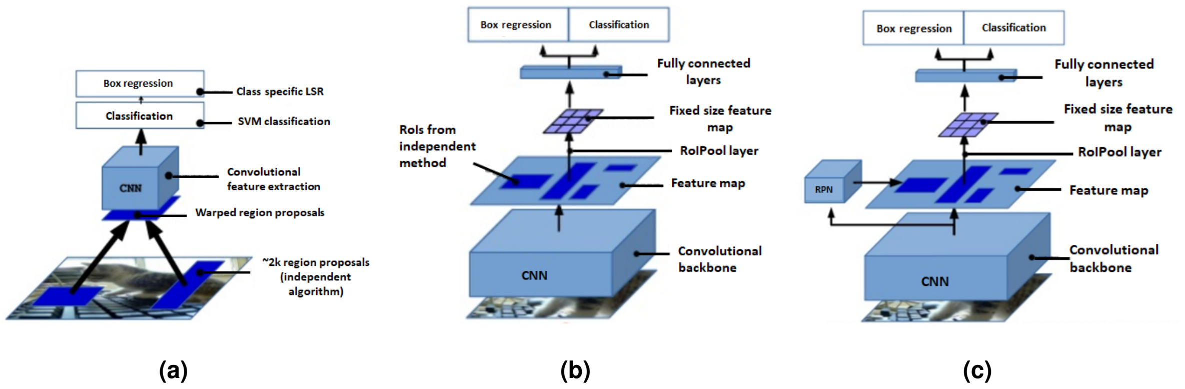


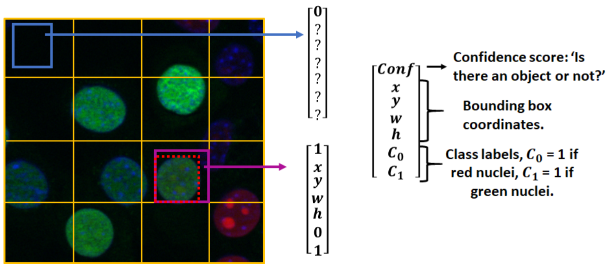
**Figure 2.20:** Evolution from R-CNN to Faster R-CNN. Schematic representation of the architectures used in [52], [53] and [54], respectively. Images retrieved from: [55]. **a)** R-CNN, **b)** Fast R-CNN, **c)** Faster R-CNN.

For instance, Faster R-CNN's region proposal network is based on a sliding window approach, which is slow and computationally expensive. R-CNN proposes regions of interest and then for each proposed region a classifier assigns a class to each of the boxes. Meanwhile, in YOLO, given an image, there is just one single neural network that, in a single pass, predicts the bounding boxes and its respective class probabilities. Thus, it not only predicts which objects are present in the image, but also predicts the localization of each object. YOLO solves the object detection and classification task as a regression problem. It is a mapping between the input image pixels and the coordinates of the bounding boxes and the respective class probabilities.

**Architecture**

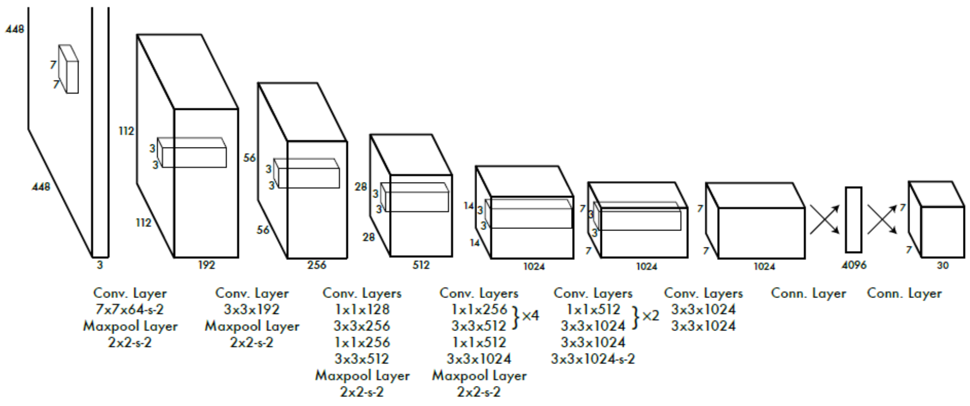
First the input image is divided into an $M \times M$ grid. An object is detected by a given grid cell if its center falls in that grid cell. A grid cell predicts N bounding boxes and a confidence score for each of them. The confidence score gives an idea about the model's level of confidence that a given bounding box contains an object. When the box doesn't contain an object, this score should be zero. Therefore, each box will predict 5 numbers: a confidence score and 4 coordinates of the bounding box ((x, y, w, h), where the x and y denote the coordinates of the center of the bounding box and the w, h denote the width and height of the bounding box, respectively). Moreover, each grid cell predicts a class probability as well. In fact, it predicts the conditional class probabilities, i.e., $P(class_i|object)$. This probability can be interpreted as: 'given an object enclosed in a bounding box what is the probability of this object belonging to class i?'. Figure 2.21 shows an example of what YOLO outputs for a grid cell that doesn't contain an object and for a grid cell that contains an object. The bounding box represented in blue doesn't have an object therefore its confidence score is zero. The bounding box represented in pink has a green object, therefore the class label for the grid cell where it belongs to will be [green, red] = [1, 0], and the confidence score is one. The bounding box represented in red dashed line is the ground truth bounding box.

Hence, the predictions made by YOLO are given by a tensor of size $M \times M \times (N \times 5 + C)$, (where C is the number of classes in the problem). YOLO is an architecture composed of 24 convolutional layers and 2 fully connected layers (figure 2.22). In the same paper, the authors proposed the Fast YOLO, which is a faster version of YOLO. It has fewer convolutional layers (9 instead of 24) and the number of filters is lower in comparison with the number of filters in YOLO.



0 ? ? ? ? ? ?
1 x y w h 0 1
Conf x y w h C0 C1
Confidence score: 'Is there an object or not?'
Bounding box coordinates.
Class labels, C0 = 1 if red nuclei, C1 = 1 if green nuclei.

**Figure 2.21:** Example of the output of YOLO for two grid cells, in a 2 class classification problem.



448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

**Figure 2.22:** Schematic representation of the YOLO architecture proposed in [56]. Image retrieved from [56].

### Loss Function

YOLO is trained to minimize the loss function represented in equation 2.7.

$$
\begin{aligned}
Loss\ function = \lambda_{coord} \sum_{i=0}^{M^2} \sum_{j=0}^{N} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2] \\
+\lambda_{coord} \sum_{i=0}^{M^2} \sum_{j=0}^{N} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w_i}})^2 + (\sqrt{h_i} - \sqrt{\hat{h_i}})^2] \\
+ \sum_{i=0}^{M^2} \sum_{j=0}^{N} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
+\lambda_{noobj} \sum_{i=0}^{M^2} \sum_{j=0}^{N} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
+ \sum_{i=0}^{M^2} \mathbb{1}_{i}^{obj} \sum_{c\, \in\, classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{2.7}
$$

where:

- $x_i, y_i$ denote the center of the ground truth bounding box, $\hat{x_i}, \hat{y_i}$ denote the center of the predicted bounding box;
- $\lambda_{coord}$ represents the weight given to the term of the loss function which compares the coordinates of the predicted bounding box and ground truth bounding box;
- $w_i, h_i$ represent the width and height of the ground truth bounding box and $\hat{w_i}, \hat{h_i}$ denote the width and height of the predicted bounding box;
- $C_i$ represents the ground truth confidence (i.e., intersection over union (IoU) between the ground truth and predicted bounding boxes) for grid cell i and $\hat{C_i}$ denotes the confidence score predicted by the model, for each grid cell i;
- $\mathbb{1}_{ij}^{obj}$ is one if cell i contains an object, and $\mathbb{1}_{ij}^{noobj}$ is one if cell i doesn't contain an object;
- $\lambda_{noobj}$ is the weight given to the term where it is applied to, basically it is a low value, because the term corresponds to the correct detection of the background, and typically, background is more frequent than foreground;
- finally, $p_i(c)$ and $\hat{p_i}(c)$ denote the ground truth and predicted conditional probability for class c in the grid cell i, respectively.

The first two terms represent the localization loss, i.e., they measure the error between the predicted bounding box $(\hat{x_i}, \hat{y_i}, \hat{w_i}, \hat{h_i})$ and the ground truth bounding box $(x_i, y_i, w_i, h_i)$. More specifically, the first term compares the localization of the bounding boxes and the second term compares the size of the bounding boxes. The third and fourth terms represent the confidence loss ('objectness' of the box), they are associated with the confidence score, (i.e., 'is there an object in the bounding box or not?'). Finally, the last term is the classification loss, this term compares the predicted and ground truth class for the object in grid cell i.

**Inference: Non-maximal suppression**

The non-maximal suppression is performed to remove duplicate detections. For instance, sometimes one object is detected multiple times, at close locations. To avoid this situation, non-maximal suppression can be applied. It selects boxes with high confidence scores and suppresses the boxes which have IoU above some threshold (T) with the selected boxes.

**Summary**

Figure 2.23 sums up YOLO's functioning. First the input image will be divided into regions. Then, for each region, the network outputs a confidence score and 4 coordinates of the bounding box. Moreover, for each grid cell it outputs a conditional class probability. However, one object can be detected by multiple bounding boxes. In order to avoid this, non-maximal suppression is performed to keep only the boxes with high confidence score (last image in figure 2.23).

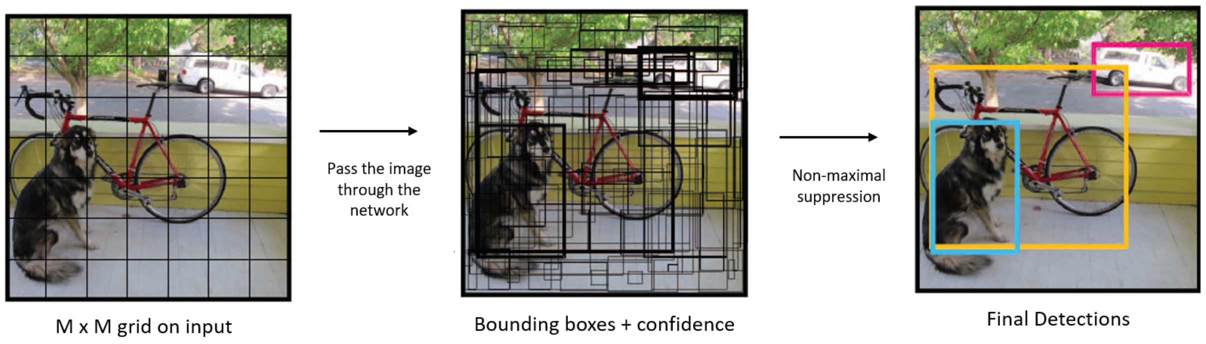

M x M grid on input
Pass the image through the network
Bounding boxes + confidence
Non-maximal suppression
Final Detections

**Figure 2.23:** Overview of YOLO. Image adapted from [56].

**YOLOv2**

YOLOv2 is the second version of YOLO, designed to improve the accuracy and speed of YOLO [58]. The changes relative to YOLO are [58]:

1. Batch Normalization

Large values can cause instability on the neural networks. To deal with this difficulty batch normalization can be applied. It consists of normalizing the mean and the variance of the CNN's outputs to follow a normal distribution. It is implemented as a layer and applied to the output of the convolutional layer before passing it through the activation function [15]. It was shown that the implementation of batch normalization in YOLOv2 improved the convergence and increased the mean average precision (mAP) by 2 %, compared to YOLO.

2. Anchors

Instead of predicting bounding box's coordinates directly from the image, YOLOv2 predicts offsets to pre-defined anchors. Anchors are obtained by applying k-means in the training data, which allows to define the sizes of the most common objects. Figure 2.24(a) illustrates 5 pre-defined anchors. The offsets relative to the anchors are predicted at each position of the grid cell. These predictions are made by a convolutional layer. That is, the fully connected layers of YOLO are replaced by a convolutional layer in YOLOv2, which makes the final prediction. Figure 2.24(b) shows an example of how the offsets are applied to the anchors. The pre-defined anchor is represented in dashed black line and the bounding box predicted by the network is represented in blue. This bounding box is obtained from the pre-defined anchor by applying the predicted offsets $(t_x, t_y, p_w, p_h)$, according to the formulas presented in the figure 2.24(b), ($(b_x, b_y, b_w, b_h)$ denote the coordinates of the predicted bounding box). Furthermore, $c_x, c_y$ denote the coordinates of the upper left corner of the grid cell for which the YOLOv2 predicted the blue bounding box. A sigmoid function is applied to $t_x, t_y$ to guarantee that the center of the predicted bounding box falls inside the grid cell.

3. Class probabilities per bounding box

In YOLO each grid cell had one conditional class probability, therefore, each grid cell was respon-

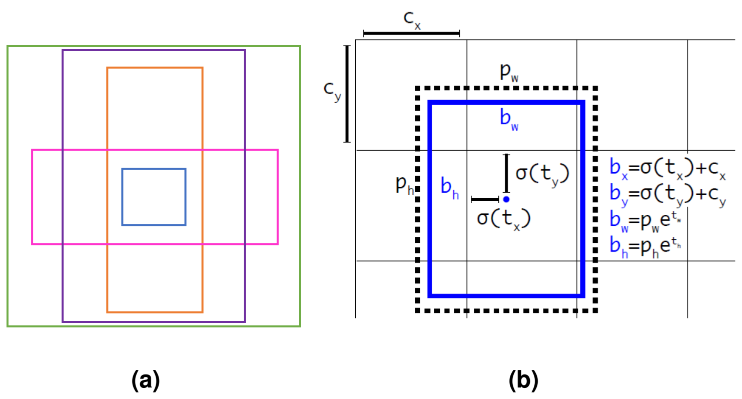


**Figure 2.24:** Anchors. **a**) Example of 5 pre-defined anchors. **b**) Illustration of a pre-defined anchor (represented in dashed black line) and the bounding box predicted by the network (represented in blue). Image retrieved from [58].

sible for detecting only one object. In YOLOv2, each bounding box has a conditional class probability. Considering that there are A pre-defined anchors for each grid cell, then each grid cell can detect a maximum of A objects, which can belong to different classes. So, for each grid cell, A bounding boxes are predicted, and for each bounding box 4 coordinates, 1 confidence score and C conditional class probabilities are predicted, where C is the number of classes in the problem.

### 2.5.7 Architectures for Instance Segmentation

Regarding instance segmentation, several approaches have been proposed. For instance, Dai et al. (2016) [59] presented a three-stage network for instance segmentation, the stages share the convolutional features and there is a causal dependency between them. Each stage is designed in order to perform a given task, stage 1 provides bounding boxes (RoIs) for each object detected in the input image, stage 2 generates masks for each object detected by the previous stage, and finally stage 3 classifies each mask. A schematic representation of this network is presented in figure 2.25. The test time associated with this method is about 360 ms, which is significantly fast. Additionally, it doesn't need an external region proposal method.

More recently, in 2017, Fully Convolutional Instance-aware Semantic Segmentation (FCIS) was proposed [60], which according to Li et al. is considered as "the first fully convolutional end-to-end solution for instance-aware semantic segmentation task" ( [60], p.1). In this approach object classes and masks are predicted simultaneously, as illustrated in figure 2.26, making the procedure fast. However, this approach fails in the segmentation of overlapping instances, which is the main difficulty in instance segmentation.

Based on the architecture of Faster R-CNN, He et al. (2017) [61] recently proposed mask regional

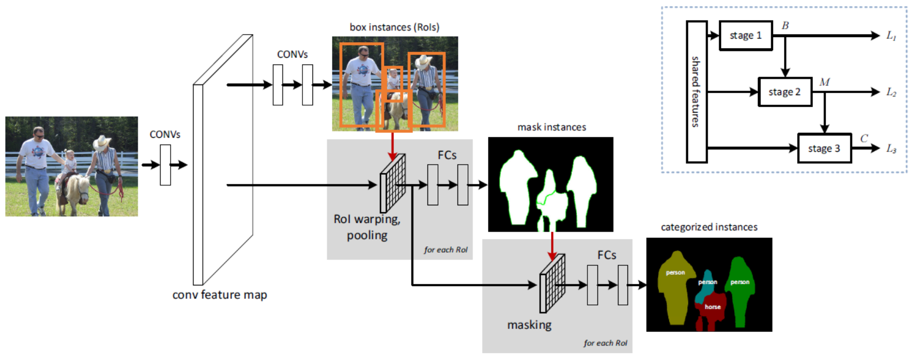


**Figure 2.25:** Schematic representation of the architecture proposed in [59]. In the top right corner a more simplified illustration of the network is presented. Image retrieved from [59].

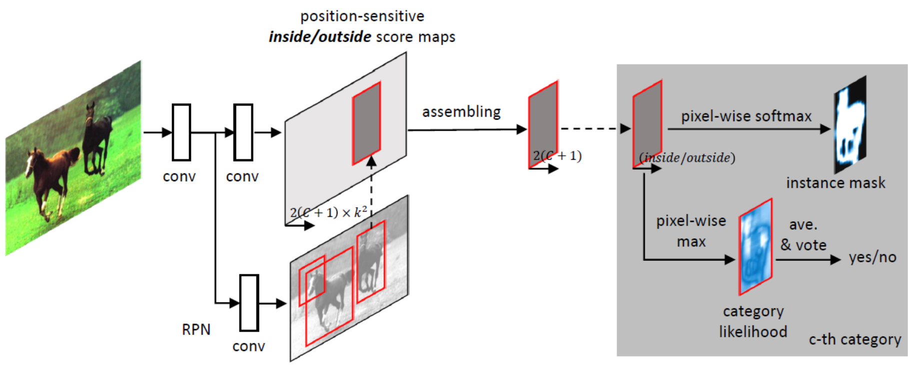


**Figure 2.26:** Schematic representation of the architecture proposed in [60]. Image retrieved from [60].

convolutional neural network (Mask R-CNN), an architecture designed for instance segmentation. As illustrated in figure 2.27, when compared to Faster R-CNN (figure 2.20(c)), Mask R-CNN has an additional branch for mask prediction making it suitable for instance segmentation. In fact, Mask R-CNN is the state-of-the-art architecture for instance segmentation. Compared to the method proposed in [59], Mask R-CNN has the advantage of predicting masks and classification labels in parallel, thus being not only simpler but also more flexible [61].

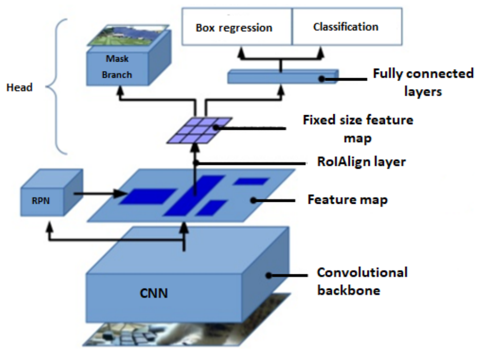


**Figure 2.27:** Schematic representation of the architecture proposed in [61]. Image retrieved from [55].

Mask R-CNN has essentially two stages, which share the convolutional backbone. The convolutional backbone serves as a feature extractor, typically this backbone is a ResNet50 or ResNet101. The first layers extract simple features (edges and corners) and the later layers extract high level features. In [61] the authors state that this backbone can be improved by using a Feature Pyramid Network (FPN), which can represent better the objects at different scales. The first stage of the Mask R-CNN is a RPN, which is based on a sliding window approach. That is, for each pixel in the feature map generated by the convolutional backbone, the RPN proposes k bounding boxes and a score that tells if the bounding box contains an object or not. At each position, a total of k proposals (called anchors) are possible. An anchor is centered at the sliding window and a scale and an aspect ratio are associated with it. In [61] five scales and three aspect ratios are used, which result in k = 15 possible anchors at each position of the sliding window. For each anchor, the RPN predicts not only the anchor class (foreground or background), but also refines it, that is, it estimates a change in the height and width so that the anchor box fits the object better.

So, the first stage proposes RoIs with different sizes. The final proposals are obtained after the non-max suppression step, in this step the overlapping RoIs are removed. In the second stage, the different sized CNN feature maps are resized so that they have the same size. Afterwards, classification and bounding box regression is performed for each fixed size feature map. On one hand, the RoI classifier outputs the class of the object in the RoI. Unlike the classification generated by the RPN (which is binary), this network has the ability to classify regions into specific classes (car, person, tree, etc.). On the other hand, the bounding box regression refines the bounding box to enclose the object better, this step is similar to the anchor refinement made by the RPN. Finally, in this second stage there is also a mask branch, which generates a mask for all the regions selected as positives by the RoI classifier.

### 2.5.8 Mask R-CNN for Nuclei Segmentation

Although Mask R-CNN was developed recently, there are already several works where it was used in order to tackle the problem of cell nuclei segmentation. For instance, Johnson (2018) [62] applied the Mask R-CNN model to detect nuclei in microscopy images, showing that an architecture that was originally designed for the instance segmentation of objects in common images (for example, day-to-day scenarios), has applications in the segmentation of nuclei in microscopy images. On the other hand, Liu et al. (2018) [31] proposed a new approach for cervical nuclei segmentation. This approach is based on a combination of Mask R-CNN with an additional step to refine the boundaries provided by Mask R-CNN. Their results showed that the proposed approach is better than some state-of-the-art methods, they achieved a 96 % average precision and recall. However, the authors didn't compare the performance of their approach with Mask R-CNN, which would allow to check if the additional step changes significantly the performance.

Recently, in 2019, a comparison between U-Net, Mask-R-CNN and an ensemble between these two models was presented by Vuola et al. (2019) [63]. As mentioned before, U-Net is designed for semantic segmentation and Mask R-CNN for instance segmentation; therefore, in order for the comparison between U-Net and Mask R-CNN to be fair, for the U-Net model the output was changed to predict not only the nuclei but also the borders between close nuclei, a similar approach to [45, 47], with an additional post-processing step based on watershed. Their results show that although Mask R-CNN performs better in the nuclei detection task, its segmentation masks are worse when compared to the ones provided by U-Net. In order to get the best of both models, they trained an ensemble model combining the outputs of both Mask R-CNN and U-Net networks. This model presented better mAP in both fluorescence and histology images. Finally, it is noteworthy to mention that the third place solution of Kaggle competition 2018 [46] was based on the Mask R-CNN model.

### 2.5.9 Conclusion

According to this review, the two architectures that have been most commonly used in order to tackle the problem of cell nuclei segmentation are the U-Net and the Mask R-CNN. In summary, U-Net is designed for semantic segmentation, so it doesn't have the ability to distinguish different instances of the same object. Mask R-CNN is designed for instance segmentation, however its architecture is more complex than that of the U-Net, usually requiring more computational resources. On one hand, the approaches based on U-Net require additional methods to separate touching nuclei and solve the problem as an instance segmentation problem. On the other hand, although Mask R-CNN was designed for instance segmentation and provides good segmentation results in the nuclei segmentation task, it isn't computationally efficient. And if the method is going to be implemented in the clinical routine, speed is an important factor to take into consideration. Therefore, research is needed in the area of nuclei instance segmentation, in order to develop an instance segmentation approach based on deep learning, which provides good segmentation results and is computationally efficient.

Finally, the different works that were cited in this brief review are summarized in table A.1.

# 3

# Biological Background

Contents

In this chapter the biological background that is important to understand this work is presented. It starts with an introduction to the cell, cell cycle and fluorescence microscopy. Finally, state-of-the-art regarding cell cycle staging from microscopy images is presented.

## 3.1 Cell

Cell is the basic and fundamental unit of life [12, 64]. Human body is composed of billions or trillions of cells which are organized into more complex structures. Just as the human being, the cells can grow, reproduce and interact with the surrounding environment by processing information and reacting to stimuli from this environment. An eukaryotic cell has a plasma membrane, a cytoplasm, organelles and a well-defined nucleus (figure 3.1). Plasma membrane is the membrane that surrounds the cell. The cytoplasm is the region between the nucleus and the plasma membrane. The cytoplasm contains organelles (Golgi vesicles, lysosomes, mitochondria, etc) which are surrounded by their own membrane [64]. The nucleus is an important component of the cell. It contains genetic material, the chromosomes (DNA + proteins), which contain information regarding cell's functioning. Every single cell has a copy of the genetic information, however depending on the cell's function some genes are turned off and others are turned on. This explains the diversity of the cells that compose the human body [64].

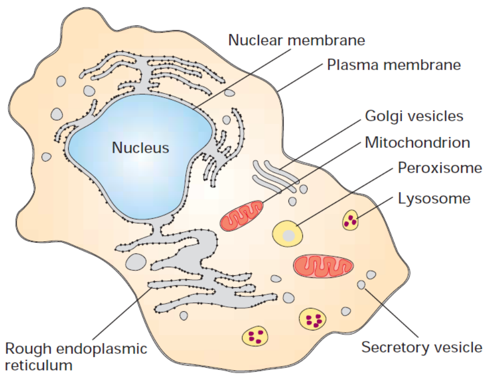


**Figure 3.1:** Structure of the eukaryotic cell. Image retrieved from [64].

## 3.2 Cell Cycle

Cell cycle is a timely coordinated sequence of events that occur in a cell leading to its growth, correct DNA replication, division of the nucleus and cytoplasm and, consequently, generation of two genetically identical cells [65]. The cell cycle consists of two fundamental parts: interphase and mitosis. Interphase is the period between the end of a cell division and the beginning of the next cell division. It is the longest phase of the cell cycle and consists of three stages [64, 65]:

- **G1**: During this period the cell grows and prepares for a division. Proteins are synthesized at the cytoplasm from the information codified by the DNA, inside the nucleus.
- **S**: DNA replication and chromosome duplication occurs during this phase. Each DNA molecule gives rise to two similar daughter molecules. In the beginning of this phase, each chromosome has one chromatid (indicated inside the green box in figure 3.2). For instance, the human cell has 46 chromosomes, and during the G1 phase each of these chromosomes has one chromatid. During the S phase, the cell makes a copy of each of its chromosomes. When this phase finishes each chromosome will have two sister chromatids. They are identical to each other and they are attached at the centromere (indicated inside the pink box in figure 3.2). The centromere is an

important region that will allow the separation of the chromatids during the mitotic (M) phase. After the S phase, a human cell contains 46 chromosomes and 92 chromatids.

- **G2**: This phase occurs after DNA replication and before nuclear division. During this period more proteins are synthesized, the cell continues to grow, and membrane structures are produced to form the daughter cells. These structures are generated from the molecules synthesized in the G1 phase.

After G2 phase, mitosis begins which consists of four phases: prophase, metaphase, anaphase and telophase. During mitotic (M) phase the nucleus undergoes several transformations that will lead to its division. In this phase the nuclear envelope breaks down and the mitotic spindle forms. The centromere will allow the chromosomes to attach to the mitotic spindle. Then, as the mitotic spindle elongates it pulls apart the sister chromatids to opposite poles (as shown inside the yellow box in figure 3.2). Thus, in this phase the genetic information is partitioned evenly to two nuclei. Afterwards, cytokinesis (division of the cytoplasm) occurs in order to obtain two genetically identical cells, each containing one nucleus [65]. These daughter cells don't need to continue cell division straightaway, they can quit the cell cycle and enter the G0 phase (resting/quiescent state), in this phase the cells remain metabolically active but they don't divide. However, under certain conditions, cells in G0 phase can reenter the cell cycle [64,65]. For instance, skin fibroblasts and liver, lung and kidney cells remain in the G0 phase under normal conditions, and they only divide to replace injured or dead cells [65]. In its turn, rapidly dividing cells enter the G1 phase after cytokinesis, and the whole process of cell growth, DNA replication and cell division is repeated [65].

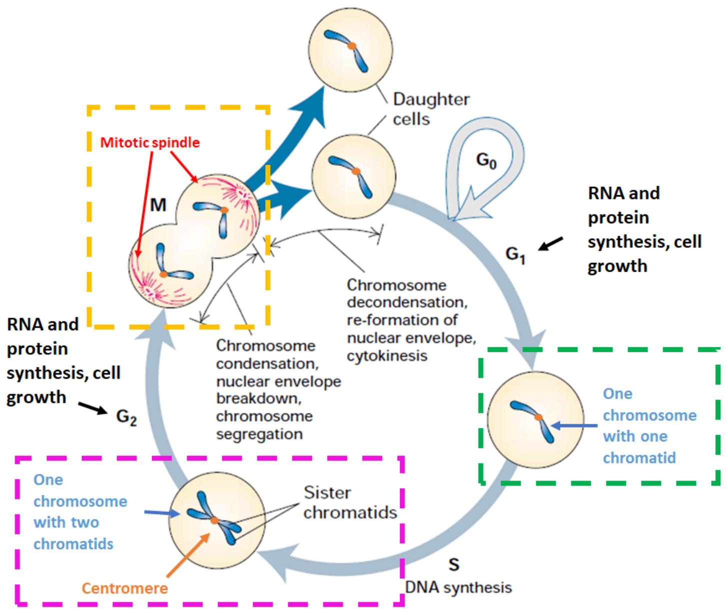

Daughter cells
Mitotic spindle
M
$G_0$
RNA and protein synthesis, cell growth
$G_1$
Chromosome decondensation, re-formation of nuclear envelope, cytokinesis
RNA and protein synthesis, cell growth
$G_2$
Chromosome condensation, nuclear envelope breakdown, chromosome segregation
One chromosome with one chromatid
One chromosome with two chromatids
Sister chromatids
Centromere
S
DNA synthesis

**Figure 3.2:** Diagram of the phases of the cell cycle. Image adapted from [64].

Cells at different stages of the cell cycle can be distinguished based on their DNA content, as observed in figure 3.2 [12]. Cells in the G1 phase are diploid, their DNA content is 2N, (the cell has 2 copies of each chromosome, and each chromosome has one chromatid). In turn, cells in the S phase have DNA content ranging from 2N to 4N. Finally, the DNA content of cells in the G2 phase is 4N (the cell has 2 copies of each chromosome, and each chromosome has 2 chromatids), and it decreases to half (2N) after cytokinesis. The DNA content can be studied by staining the cells with fluorescent molecules that bind to the DNA and, afterwards, by analyzing the fluorescence intensity of individual cells [65].

## 3.3 Fluorescence Microscopy

Fluorescence microscopy allows to visualize the physiology of cells. The sample is typically stained with fluorescent molecules that bind to certain regions of the cell (for example, a molecule that binds to the DNA contained in the nucleus). Thereafter, it is excited by a light of a given wavelength ( $\lambda_1$). The fluorescent molecules will absorb this light and emit a light with wavelength $\lambda_2$ a few nanoseconds later, where $\lambda_2 > \lambda_1$. That is, some energy is lost in this process, therefore the emitted light has less energy than the absorbed light [66].

DAPI is a commonly used fluorescent dye which binds to the DNA, specially to the adenine-thymine (AT) rich regions of the DNA [67,68]. When bound to the DNA, it absorbs ultraviolet (UV) light and emits blue light. Thus, in fluorescence microscopy it is excited with UV light and detected by using a blue filter. So, DAPI is a nuclear stain that gives blue color to the nuclei in fluorescence microscopy images [69] (figure 3.3). Since DAPI binds stoichiometrically to the DNA it is possible to quantify its amount from intensity based features computed from the blue plane of the DAPI stained microscopy image [1].

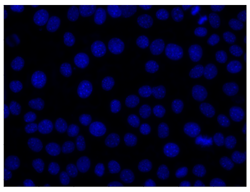


**Figure 3.3:** Example of a DAPI stained fluorescence microscopy image.

Fluorescence ubiquitination cell cycle indicator (FUCCI) is a system used to study the cell cycle. It is a fluorescent protein based sensor where the green fluorescent protein (GFP) and the red fluorescent protein (RFP) are fused to two cell cycle regulators, the geminin and the cdt1, respectively [70, 71]. On one hand, geminin is degraded in the G1 phase, therefore nuclei in G1 phase only have cdt1-RFP. Thus, nuclei in phase G1 present red color. On the other hand, cdt1 is degraded in S/G2/M phases, therefore only geminin-GFP is present. Hence, nuclei in S/G2/M phases are green [70–72]. Therefore, FUCCI labels the nuclei. The ones stained with red are in the G1 phase, whereas the ones stained with green are in the S/G2/M phases [70, 71], (see figure 3.4). Furthermore, cells in G1/S transition are yellow, because they co-express both the cdt1-RFP and geminin-GFP [70] (figure 3.4).

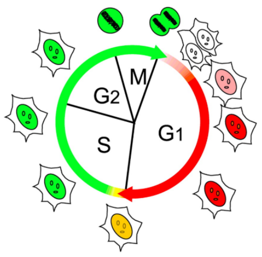


**Figure 3.4:** Schematic representation of the labeling provided by FUCCI for cells in different stages of the cell cycle. Nuclei in phase G1 are stained with red, whereas those in S/G2/M phases are stained with green. Nuclei in G1/S transition are stained with yellow. Image retrieved from [70].

## 3.4 State-of-the-Art

In this section state-of-the-art regarding cell cycle staging from microscopy images is presented.

### 3.4.1 Cell Cycle Staging and Handcrafted Features

Throughout the years several works have been proposed to infer the cell cycle phase of cells based on their DNA content. In order to measure the DNA content, stains that bind stoichiometrically to the DNA are typically used. For instance, Roukos et al. (2015) [73] proposed a protocol to determine DNA content based on fluorescence images of cells stained with DAPI, nevertheless this protocol can be applied to cells stained with other DNA dyes. In this work, the authors calculated the integrated DAPI intensity of each nucleus, and displayed this feature in histograms. Each histogram represents the nucleus's integrated DAPI intensity (in the horizontal axis), and the percentage of cells (in the vertical axis). In figure 3.5 one example of an histogram for a given experiment is presented, the red lines correspond to manual thresholds that have been defined to divide the histogram in regions that correspond to different cell cycle stages: G1, S and G2/M. Thus, cell cycle stage can be inferred for a given cell based on its integrated intensity and on manual thresholds. The validation framework of this work consists in the comparison of the cell cycle distributions obtained with the proposed method with the ones obtained with flow cytometry. Flow cytometry is a technique that analyses the content of each cell as it flows in suspension through a detector. This technique is widely used to study the cell cycle. For instance, by staining the cells with DAPI, total DNA content can be measured by flow cytometry, which allows to identify the cell distribution during the different stages of the cell cycle [74]. Finally, the authors concluded that DAPI staining allows not only to quantify DNA content, but also to assign cell cycle phase to individual cells.

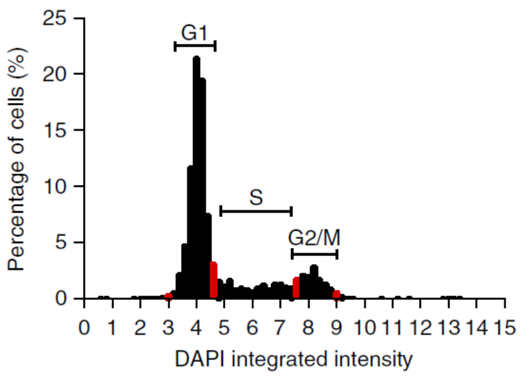


**Figure 3.5:** Example of a DAPI integrated intensity histogram. Image retrieved from [73].

Recently, Ferro et al. (2017) [1] proposed a new framework with the purpose of nuclei segmentation and measurement of DNA content in different stages of the cell cycle, on microscopic images of cells stained with DAPI. They extracted two features, area and total DAPI intensity from each nucleus, and afterwards performed cell cycle classification based on a modified version of k-means clustering, instead of defining manual thresholds as done in [73]. The authors (in [1]) applied this modified k-means clustering to the nuclei of each image. This k-means algorithm requires some prior assumptions regarding the final clusters. For instance, the centroids $\mu_{G1}$ and $\mu_{G2}$ of the cluster G1 and G2, respectively, are correlated by the following formula $\mu_{G2} = 2 \times \mu_{G1}$. This k-means algorithm outputs a label (G1, S or G2) for each nucleus in an image, as represented on the left in figure 3.6. Afterwards, the authors compared these labels with the ground truth labels based on the FUCCI system. These ground truth labels were obtained by manual annotation performed by a biologist. The biologist visually inspected the nuclei, and labeled the red nuclei as G1, the green nuclei as S/G2 and the yellow nuclei as G1/S, as represented on the right in figure 3.6. In this work an overall sensitivity of 94 % was achieved. Their results show that from digital fluorescence microscopy images of cells stained with DAPI it is possible to infer the cell cycle phase. Although this work presents an automatic algorithm for cell cycle staging, it presents some drawbacks. For instance, the nuclei segmentation step of this work was based on a combination of classical approaches (denoising, contrast and intensity adjustment, Otsu's thresholding), therefore an optional operation of manual correction was added to the imaging pipeline in order to obtain the final segmentation masks. Furthermore, the features that are used in this work can vary between different images. For example, the total DAPI intensity depends on the acquisition parameters, which may be different across different images. Additionally, the area of the nuclei can vary between different cell lines. Therefore, when there are new images, the k-means clustering algorithm needs to be applied per image, which may take a lot of time if there are several images. Moreover, the ground truth labels generated in this work result from an intensive work of the biologists and may be subjective.

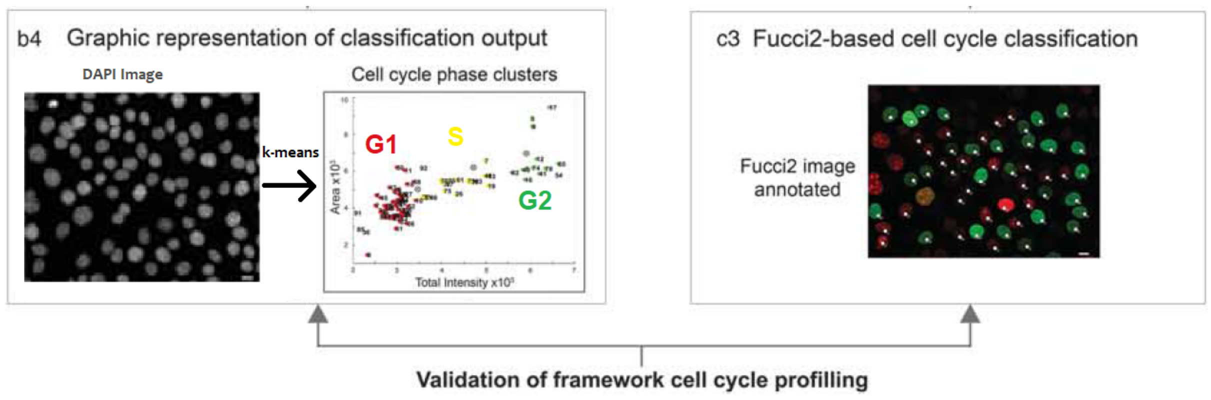

b4 Graphic representation of classification output
DAPI Image
Cell cycle phase clusters
k-means
G1
S
G2
Area x10³
Total Intensity x10⁷
c3 Fucci2-based cell cycle classification
Fucci2 image annotated
Validation of framework cell cycle profilling

**Figure 3.6:** Results of the modified k-means clustering proposed in [1] and the validation framework. Image adapted from [1].

The two works described above focus in the classification of the nuclei in stages of the interphase, based on images of cells stained with DAPI. There have been proposed several works for cell cycle staging based on the cell's DNA content, cell's shape and texture [75–78]. However, they typically focus on the following phases: interphase, prophase, metaphase, anaphase(/telophase). That is, none of these works focuses on the stages of the interphase (G1, S, G2). Furthermore, the nuclei segmentation step of these works [75–78] relies on traditional nuclei segmentation methods which can be sensitive to noise and depend on manual tuning of parameters [3]. Thus, there is lack of computational tools for interphase cell cycle staging.

### 3.4.2 Cell Cycle Staging and Deep Learning

In [79] DeepFlow is proposed, which is a data analysis workflow based on deep learning designed to study the cell cycle of human T cells (figure 3.7). DeepFlow takes as input a brightfield image and a darkfield image of a cell. These images are obtained from imaging flow cytometry (IFC), which is a technique that incorporates features of fluorescence microscopy and flow cytometry. This technique provides one image per cell therefore image segmentation isn't necessary. DeepFlow, an adaptation of the "Inception" architecture, extracts features from the input images and returns a classification label. In this work 7 labels were considered: G1, S, G2, prophase, metaphase, anaphase and telophase. These labels represent the stages of the cell cycle. Moreover, the proposed workflow performs non-linear dimension reduction. This allows the visualization of the learned features in a low dimensional space. In other words, the last layer of DeepFlow is a 336 dimensional vector, therefore dimension reduction is performed to obtain a representation of the data in a lower dimensional space. In this space one can see that in terms of morphology, adjacent classes are more similar than classes that are temporally more separated. In other words, the non-linear reduction allows to represent the data in a feature space and in this space the data is sequentially organized.

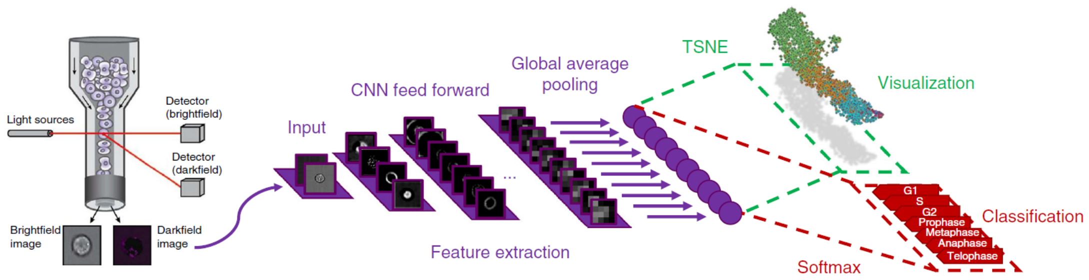


**Figure 3.7:** Schematic representation of the architecture proposed in [79]. Image retrieved from [79].

In this work the authors analyzed a total of 32,266 cells and performed five-fold cross-validation. They obtained an accuracy of $79.40 \pm 0.77\ \%$ when considering the 7 classes. Additionally, in order to compare with a previous study [78], they considered five classes (interphase, prophase, anaphase, metaphase and telophase), in this case the G1, S and G2 phases were considered as a single class, the interphase. By considering five classes they obtained an accuracy of $98.73 \pm 0.16\ \%$ . This is higher than the accuracy of the previous work (92.35 %) [78] and higher than the accuracy obtained when considering the 7 classes. Their results suggest that it is difficult to distinguish cells in G1, S and G2 phases based on the images provided by IFC, because these images are very similar.

To sum up, the proposed deep learning based approach performs classification and allows to reconstruct continuous processes. That is, it provides a visualization of the data points in the feature space consisting of features learned by the network. Furthermore, the predictions made by the DeepFlow are fast enough to be incorporated with the IFC process.

Throughout the years there have been proposed methods based on deep learning to detect cells in mitosis [80–83]. However, these approaches don't focus on interphase cells. In fact, the only work based on deep learning that presents some results regarding classification of cells in stages of the interphase is the one presented by Eulenberg et al. (2017) [79], which was described above.

### 3.4.3 Conclusion

To sum up, two methods [1, 73] for cell cycle staging based on DAPI features were described. Although the work presented by Roukos et al. [73] allows to assign a cell cycle phase to individual cells, it is based on manual thresholds which can be subjective. Furthermore, they only studied one DAPI feature, the fluorescence intensity. Ferro et al. [1] presented a modified version of k-means for cell cycle classification. This clustering method is performed in the feature space (area, total DAPI intensity). Although this approach allows to classify cells automatically, the features considered as input features for the k-means clustering algorithm depend on the experimental conditions. Therefore, the k-means clustering needs to be applied per image to infer the cell cycle, which may be computationally expensive.

The work presented by Eulenberg et al. [79] allows to automatically assign a cell cycle phase to individual cells. However, their results show that it is difficult to distinguish cells in G1, S and G2 phases, since the nuclei images in these phases are very similar. Furthermore, in this work the images are obtained by IFC, which uses cellular suspension cultures. Whereas the approaches presented in [73] and [1] use fluorescence microscopy images of cells stained with DAPI, where the cells are in culture of adherent cells. To study properties of the cells, the culture where they grow should be similar to their real environment. Most human cells are adherent cells, therefore, an adherent cell culture mimics better their real environment when compared to a suspension cell culture. Flow cytometry analysis can be affected when suspension cell culture is created from adherent cells [12, 84]. Therefore, the method proposed in [79] can work properly with cells that grow in suspension, however it can impact negatively the results for adherent cells.

There aren't many works for cell cycle staging based on supervised machine learning techniques, and the work that exists [79] is more directed to cells that grow in suspension. Thus, it is necessary to develop an automatic tool for cell cycle staging that is applicable to adherent cells. Moreover, the results in [79] show that it is difficult to distinguish cells in G1, S and G2 phases. Therefore, it is important to develop tools for interphase cell cycle staging that can be applied to adherent cells.

# 4

# Methodology

## Contents

As stated previously, the main goal of the project is two-fold. First, the goal is to perform cell nuclei segmentation in DAPI and in FUCCI images. One important application of cell nuclei segmentation is the cell cycle staging. Thus, the second goal is to provide a method for cell cycle staging from images of cells stained with DAPI. This chapter starts with the explanation of the overall pipeline for cell nuclei segmentation and cell cycle staging. The proposed approach for cell nuclei segmentation is presented in subsection 4.2.3. The performance of this approach was compared with the performance of other methods, which are described in detail in subsection 4.2.4. Finally, the steps followed to train and test the proposed approach for cell cycle staging are presented in section 4.3.

## 4.1 Overview

Figure 4.1 shows the overall methodology followed to train and test the performance of models for cell nuclei segmentation and cell cycle staging. As stated before, the step of nuclei segmentation is important since it provides valuable information about each nucleus. This step allows to obtain accurate boundaries of each nucleus. In this work, a novel deep learning based approach is presented for cell nuclei instance segmentation. This approach allows not only to segment nuclei in microscopic images of cells stained with DAPI but also to segment nuclei in images of cells stained with FUCCI [1]. For both DAPI and FUCCI images, first a preprocessing step is applied (green blocks in figure 4.1). Afterwards, the preprocessed images and the corresponding ground truth segmentation masks are fed to the deep learning model (steps A and B, for FUCCI and DAPI images, respectively). This will allow the model to learn the parameters $\theta_{FUCCI}$ or $\theta_{DAPI}$, depending if it was trained with FUCCI or DAPI images, respectively. After learning the parameters, when there is a new image the model will be able to provide a segmentation mask for that image (steps C and D, for FUCCI and DAPI images as input images, respectively). Finally, after showing that the model is able to segment nuclei stained with different dyes, the main goal was to study the cell cycle. In this work a novel SVM based approach is presented for cell cycle staging from DAPI images. To train the classifier for cell cycle staging, the segmentation masks obtained for the FUCCI images are used along with the corresponding RGB images. On one hand, the FUCCI image is used to extract the green and the red color components (step E in figure 4.1). Then color normalization is performed and labels are created based on the FUCCI technology (blue blocks in figure 4.1). These are the ground truth labels used for training the cell cycle staging classifier. On the other hand, some features are extracted from the DAPI image (step F in figure 4.1). These features along with the ground truth labels are used to train a machine learning classifier: the SVM (step G in figure 4.1). Note that in order to train the SVM the information from the DAPI image is extracted according to the segmentation mask obtained for the FUCCI image (represented by the dashed line in figure 4.1). When

[1]As shown in the black rectangle on left in figure 4.1, the original images are RGB, the DAPI images correspond to the blue channel of the original image and the FUCCI images are RGB images, where the blue component is zero, and the red and green components are the red and green components of the original image.

there are new images for which one wants to perform cell cycle staging the steps that must be followed are shown within the yellow box. All the other steps are only needed to train the models for nuclei segmentation and cell cycle staging. To sum up, to perform cell cycle staging with the method proposed in this work the cells only need to be stained with DAPI, which is a commonly used fluorescent dye. Cell cycle staging will be performed based on features extracted from images of cells stained with DAPI. First, the proposed deep learning approach will be used to segment cell nuclei. Thereafter, the features from each DAPI nucleus patch will be extracted to feed the SVM. This classifier will classify each nucleus regarding its cell cycle phase, according to the parameters $\theta_{SVM}$ learned during the training step.

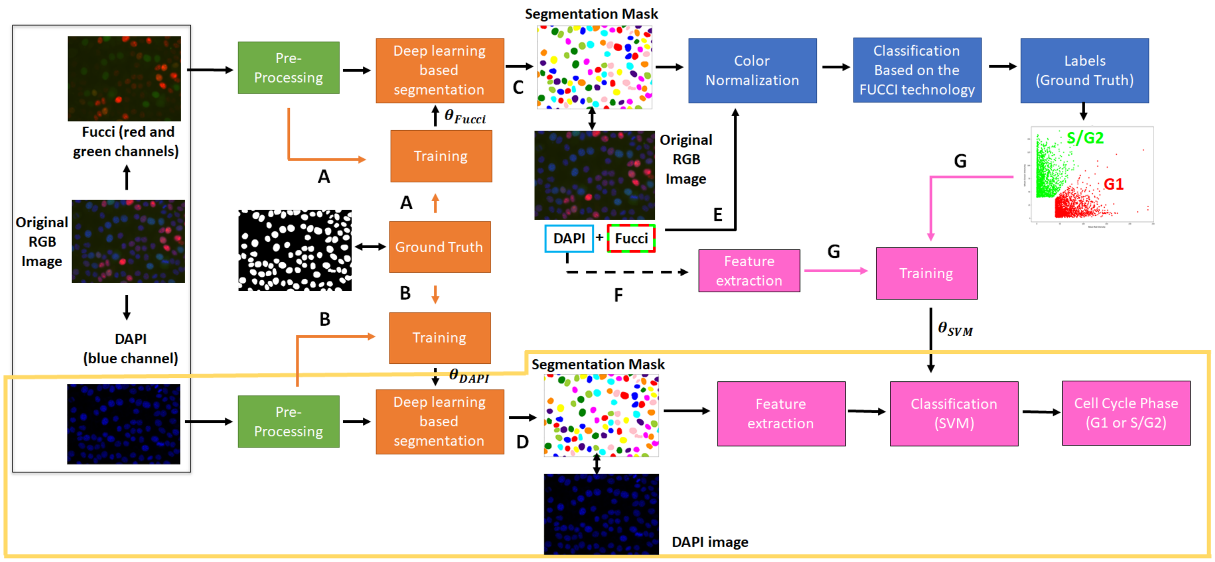


**Figure 4.1:** The original image is an RGB image, where the blue channel represents the DAPI information and the green and red channels show the information regarding FUCCI. The preprocessing blocks are represented in green. The deep learning based segmentation blocks are represented in orange, $\theta_{FUCCI}$ and $\theta_{DAPI}$ denote the parameters learned by the deep learning model when solving the task of nuclei segmentation in FUCCI and DAPI images, respectively. The blocks represented in blue show the steps followed for cell cycle staging based on FUCCI, these steps allow the creation of ground truth labels to train the classifier for cell cycle staging. This classifier is represented in pink, $\theta_{SVM}$ denote the parameters learned by the SVM when solving the task of cell cycle staging from DAPI images.

## 4.2 Cell Nuclei Segmentation

This section starts with an explanation of the creation of ground truth segmentation masks. Afterwards, the image preprocessing steps are described. Thereafter, the proposed approach for cell nuclei segmentation is presented. Finally, the methods with which the proposed method was compared are presented.

### 4.2.1 Ground Truth

In this thesis a DAPI-FUCCI dataset was used. It consists of 130 DAPI stained fluorescence microscopy images, and the corresponding 130 fluorescence microscopy images of cells stained with FUCCI (figure 4.2). The cells stained with DAPI and FUCCI are normal murine mammary gland cells. Images have size $1388 \times 1040$ and they are divided into 13 experiments. This dataset was obtained from the study presented by Ferro et al. [1], which was designed to study cell cycle staging.

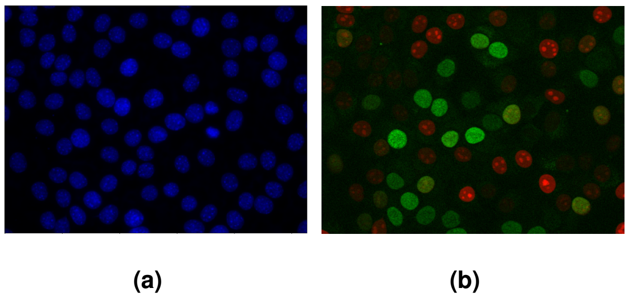


**Figure 4.2:** Example of a DAPI image and corresponding FUCCI image. a) DAPI image. b) FUCCI image.

As mentioned by Ferro et al. [1], in the DAPI-FUCCI dataset only a few nuclei were annotated manually, namely those that were important for the study in question. However, ground truth data is really important in a supervised model, since the model will learn to perform a task based on these data. Therefore, in order to improve the nuclei masks annotated in [1], a manual annotation was performed using GNU Image Manipulation Program (GIMP) [85] (figure 4.3). GIMP provides useful tools such as magic wand, lasso and bucket fill. By using magic wand or lasso, one can obtain the contour of the nucleus, then by applying bucket fill tool, the mask of the nucleus can be obtained, as illustrated in figures 4.3(b) and 4.3(c), respectively.

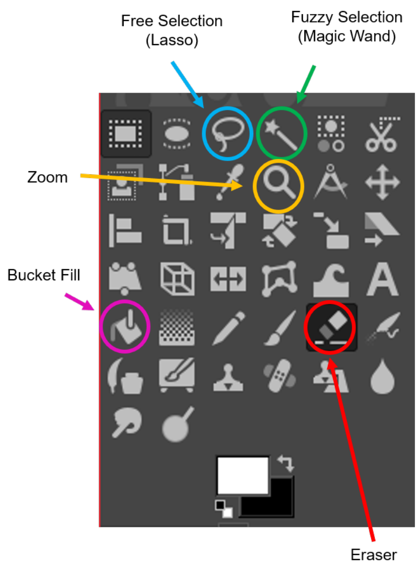
As represented in figure 4.3(b), after applying bucket fill, the edges of the nucleus appear blurred. Moreover, in some cases holes appeared in the nucleus's mask. Therefore, a post-processing step was applied to all masks, using MATrix LABoratory (MATLAB). The code regarding this step is provided in appendix B, listing B.1. In [1], the ground truth masks were created from the DAPI images. Therefore, in this work they were improved based on DAPI images as well.

In figure 4.4, an example is provided regarding improvements that were made to the ground truth masks. Additionally, some masks had touching nuclei (as can be observed at the bottom right corner in figure 4.4(b)). The main goal of this work is to perform instance segmentation, therefore the eraser tool (highlighted in figure 4.3(a)) was used to separate these nuclei.

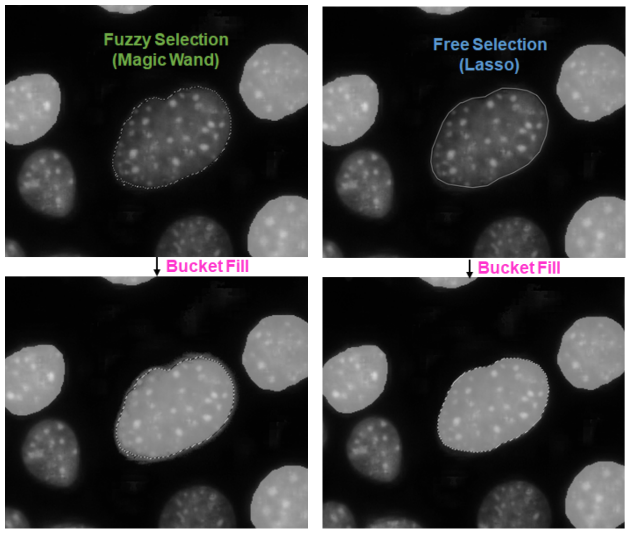
### 4.2.2 Data Pre-Processing

#### Pre-processing for deep learning models

It is known that neural networks work better when the input values are small [22]. Both DAPI images (grayscale images) and FUCCI images (RGB images) have values ranging from 0 to 255. However, a lot of factors influence the image quality, such as the illumination conditions and image acquisition parameters. For instance, one image can have values ranging from 0 to 255 and another image from 0



(a)



(b) (c)

**Figure 4.3:** Representation of some useful GIMP tools that were used to annotate nuclei (image at left), and representation of two ways to generate a mask for a nucleus (middle and right images). Note that the middle and right images are obtained after applying zoom using the zoom tool. **a)** GIMP tools. **b)** Magic wand tool to obtain nucleus contour (represented by the dashed line in the image at top) and bucket fill to create the mask (image at bottom). **c)** Lasso tool to obtain nucleus contour (represented by the continuous line in the image at top) and bucket fill to create the mask (image at bottom).

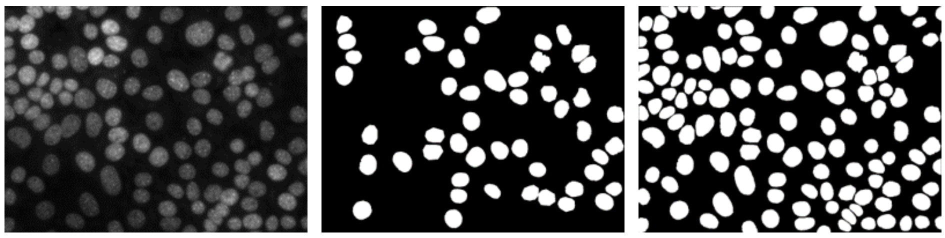

(a) (b) (c)

**Figure 4.4:** Example of a segmentation mask before and after manual annotation performed on GIMP and post processing step performed on MATLAB. **a)** Example of an original DAPI image (it is grayscale because it corresponds to the blue channel of the original RGB image); **b)** Segmentation mask generated in [1], for the image 4.4(a). **c)** Improved segmentation mask, after manually annotating nuclei using GIMP and post processing.

to 100. In order to have all inputs in a comparable range and with small values, a simple normalization step was performed. For each image its values were divided by 255.

### Image Normalization

The images used in this work are acquired under certain experimental conditions that influence their quality. They belong to different experiments, thus, they were acquired under different conditions. In fact, these parameters can be varied even for images acquired during the same experiment. Therefore, the intensities of the images are not comparable. For instance, there may be color variation between images due to the variation of the reagents, due to the staining protocol or due to the acquisition parameters (of the microscope, for example). To solve this problem a normalization step is extremely important.

In the context of this work, the DAPI images acquired in different experiments were similar, that is, there weren't significant differences. However, for the FUCCI images there were clearly some of them in which the background was greenish. In order to make the intensity between all the images comparable, a normalization step was performed for the FUCCI images. In this step, for each image, for each color channel, the mean background intensity was subtracted from the image. That is, for each image there is the ground truth mask, so the average background's intensity for each color channel can be easily calculated, this is denoted as $(\bar{R}, \bar{G}, \bar{B})_{background}$. Thereafter, every pixel in the image is changed by subtracting to its intensity ($(R, G, B)_{pixel}$) the average background's intensity per channel. That is, by computing the following quantity: $(R, G, B)_{pixel} - (\bar{R}, \bar{G}, \bar{B})_{background}$. Figure 4.5 shows an example of an image before and after normalization. Note that, although the ground truth masks were created based on DAPI images, the nuclei positions and shape are the same in the FUCCI images. Therefore, these ground truth segmentation masks are also the ground truth masks for the FUCCI images.

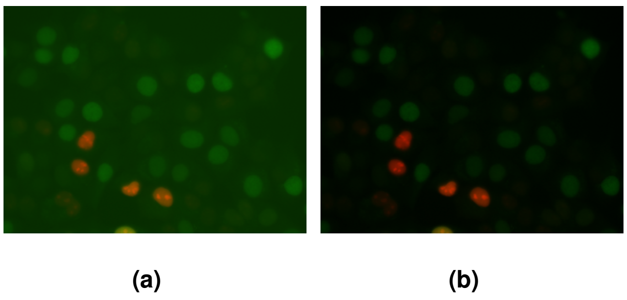


**Figure 4.5:** Illustration of normalization effect. **a)** Original image, where the background is greenish. **b)** Image obtained after the normalization step, where the background is black.

### 4.2.3 Proposed Approach for Cell Nuclei Instance Segmentation

#### Overview

In this work a new approach for cell nuclei instance segmentation is proposed. This approach combines the object detection ability of Fast YOLO [56] with the segmentation capability of U-Net [41]. [2] A schematic representation of the proposed approach is presented in figure 4.6. First, the input image is fed to Fast YOLO (step A), which will divide the image into regions and propose bounding boxes and confidence scores for each region. After non-maximal suppression, Fast YOLO will provide an output

[2] As stated in section 2.5, in [56] the authors proposed YOLO and Fast YOLO. The architecture used in the approach proposed in this work is a smaller version of YOLOv2 [58]. Therefore, it is Fast YOLOv2, however, to simplify it will be designated as Fast YOLO.

with all detected objects and respective classes (step B). In this problem there is only one class: the nucleus. Thereafter, for each nucleus detected by the Fast YOLO, the respective patch will be extracted from the input image and it will be resized to a patch of size $80 \times 80$ (step C). This step is repeated for each object detected by the Fast YOLO. Considering that there are N objects detected, after step C there is a batch of images of size $80 \times 80 \times N$, this is fed to the U-Net (step D). The output of the U-Net is a binary segmentation mask for each image (step D). That is, pixels belonging to the foreground (nucleus) will have value 1 and pixels belonging to background will have value 0. Afterwards, steps E and F are repeated for each nucleus binary mask. Step E resizes the nucleus patch to its original size and finally step F places the patch in the final segmentation mask on the location where it belongs to. Furthermore, each nucleus detected by Fast YOLO will have an unique label. The output segmentation mask is colored to visualize better the quality of the segmentation masks. Note that the colormap used doesn't have enough distinct colors to cover all nuclei, this is used just for visualization purposes.

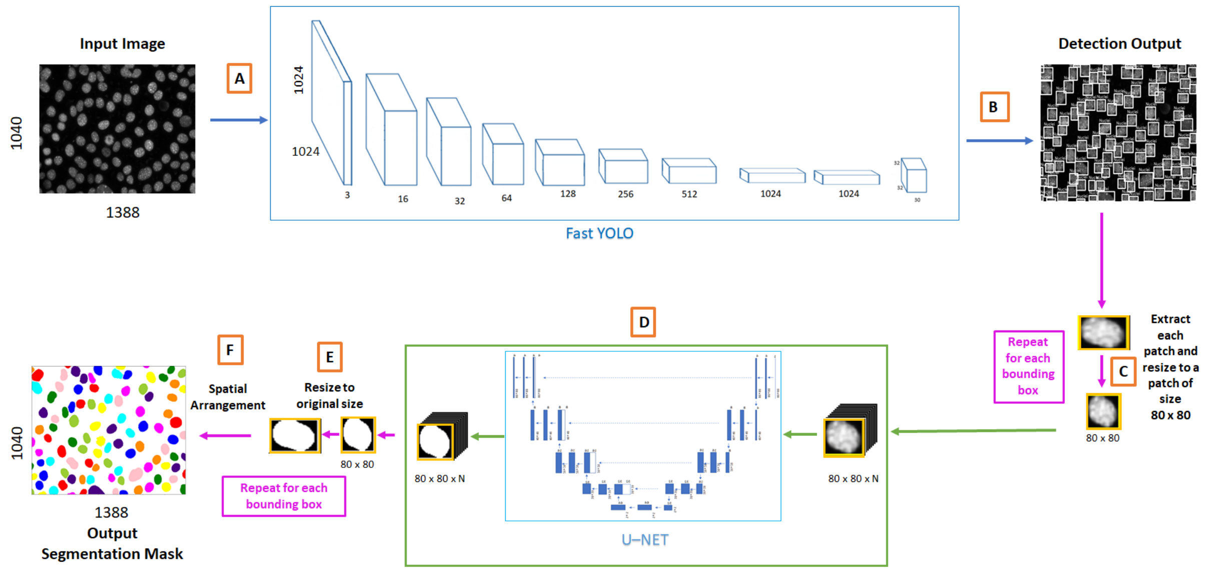
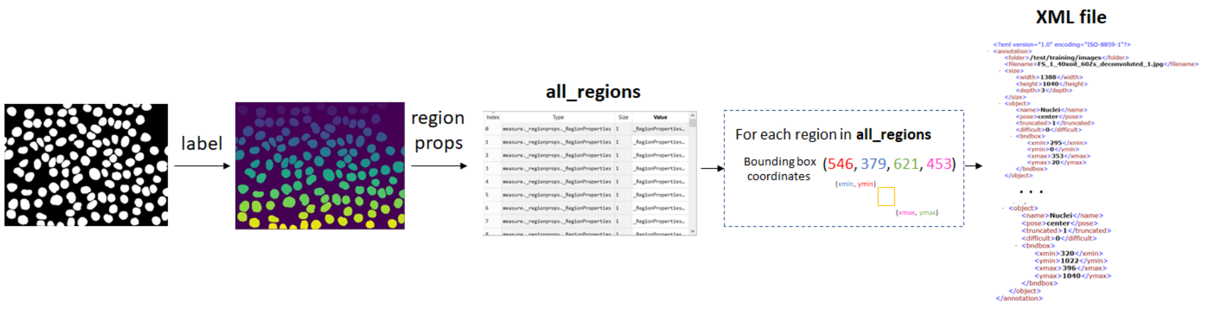


**Figure 4.6:** Overview of the proposed approach for cell nuclei instance segmentation. The input image is fed to the Fast YOLO architecture (step A). Fast YOLO will give as output bounding boxes for all of the detected objects in the input image (step B). Afterwards, each patch inside the bounding box proposed by the previous architecture is resized to $80 \times 80$ (step C). When all the N patches have been resized, a batch of images $(80 \times 80 \times N)$ is obtained. This batch of images is fed to the U-Net (step D). Then each output patch of the U-Net is resized to the original size (step E). Finally, after spatial arrangement, the final segmentation mask is obtained, (step F).

### Implementation

The implementation used for Fast YOLO is based on a publicly available implementation by Thtrieu. This implementation was released under the GNU General Public License v3.0. [86]. To train the Fast YOLO on the dataset of DAPI images, Extensible Markup Language (XML) files, based on Pascal VOC format, were generated. These files were generated from the ground truth masks using label and regionprops tools from skimage [87] and using the lxml.tree module. The process of creation of the XML files is represented in figure 4.7. First, the ground truth binary mask is labeled using the label tool from skimage. Then, regionprops tool is used to obtain the coordinates of the bounding box that encloses each object in the labeled image. Finally, using the lxml module the XML file is created. This file contains the class of each object (nucleus in this problem) and its bounding box coordinates. An example of an XML file generated to train Fast YOLO can be found here.



| Index | Type | Size | Value |
|---|---|---|---|
| 0 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 1 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 2 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 3 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 4 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 5 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 6 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 7 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |

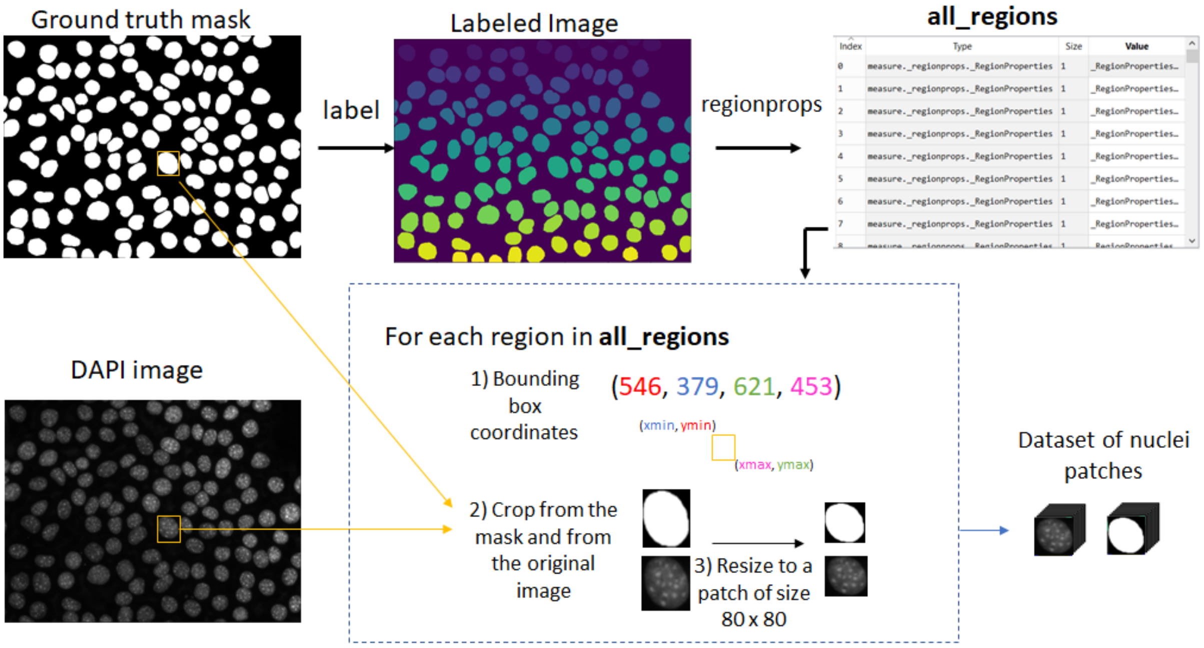
```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <annotation>
    <folder>/test/training/images</folder>
    <filename>FS_1_40xoil_607s_deconvoluted_1.jpg</filename>
  - <size>
      <width>1388</width>
      <height>1040</height>
      <depth>3</depth>
    </size>
  - <object>
      <name>Nuclei</name>
      <pose>center</pose>
      <truncated>1</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>295</xmin>
        <ymin>0</ymin>
        <xmax>353</xmax>
        <ymax>20</ymax>
      </bndbox>
    </object>
      . . .
  - <object>
      <name>Nuclei</name>
      <pose>center</pose>
      <truncated>1</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>320</xmin>
        <ymin>1022</ymin>
        <xmax>396</xmax>
        <ymax>1040</ymax>
      </bndbox>
    </object>
</annotation>
```

**Figure 4.7:** Schematic representation of the procedure performed to create an XML file from a ground truth mask.

After creating the XML files, the number of filters in the last layer of the Fast YOLO was changed according to the formula $5 \times (classes + 5)$ [86]. In this problem the number of classes is 1, therefore the number of filters in the last layer was changed to 30 ($5 \times (1 + 5) = 30$). In YOLOv2 the anchors are obtained by applying k-means clustering to the training data. In order to obtain the anchors to train the Fast YOLO, k-means clustering was performed. A detailed explanation about this algorithm is provided in section B.2, in appendix B.

In order to train the U-Net of the proposed approach, an additional dataset with images of size $80 \times 80$ was created from the original dataset. In order to create this dataset, label and regionprops tools from skimage [87] were applied to ground truth masks. The procedure followed to create this dataset is presented in figure 4.8. When the label tool is applied to a binary mask it returns a labeled image, where each object has an unique label. This labeled image is the input for regionprops which outputs several properties regarding each object, one of these properties corresponds to the four coordinates of the bounding box that encloses an object. For each object two image patches are extracted, one from the ground truth mask and another from the DAPI image. These patches are resized to $80 \times 80$. They are resized to this size since it is, approximately, the mean width and mean height of all nuclei.

The proposed approach was trained to first minimize the loss function of the Fast YOLO and then the loss function of the U-Net. Fast YOLO's loss function is similar to the one represented in equation 2.7. The only difference is in the last term, because in the second version of YOLO the class score is estimated per bounding box instead of being estimated per grid cell, therefore the last term is $\sum_{i=0}^{M^2} \sum_{j=0}^{N} \mathbb{1}_{ij}^{obj} \sum_{c \,\in\, classes} (p_i(c) - \hat{p}_i(c))^2$ instead of $\sum_{i=0}^{M^2} \mathbb{1}_{i}^{obj} \sum_{c \,\in\, classes} (p_i(c) - \hat{p}_i(c))^2$. To train the U-Net the loss function proposed in [47] was minimized, which is defined as:


Ground truth mask
label
Labeled Image
regionprops
all_regions
DAPI image
For each region in all_regions
1) Bounding box coordinates
(546, 379, 621, 453)
(xmin, ymin)
(xmax, ymax)
2) Crop from the mask and from the original image
3) Resize to a patch of size 80 x 80
Dataset of nuclei patches

| Index | Type | Size | Value |
|---|---|---|---|
| 0 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 1 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 2 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 3 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 4 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 5 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 6 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 7 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |
| 8 | measure._regionprops._RegionProperties | 1 | _RegionProperties… |

**Figure 4.8:** Schematic representation of the procedure followed to create a dataset of nuclei patches.

$$L = \frac{1}{2}BCE + \frac{1}{2}(1 - DC) \tag{4.1}$$

where the definition of Binary Cross Entropy (BCE) and Dice Coefficient (DC) is represented in equations 4.2 and 4.3, respectively.

$$BCE = -\frac{1}{N}\sum_{i=1}^{N} y_i log(\hat{y_i}) + (1 - y_i)log(1 - \hat{y_i}) \tag{4.2}$$

where N is the total number of pixels in a given image, $y_i \in \{0, 1\}$ is the true label of pixel i and $\hat{y_i} \in [0, 1]$ denotes the predicted probability for pixel i.

$$DC = \frac{1}{N}\frac{2|X \cap Y|}{|X| + |Y|} \tag{4.3}$$

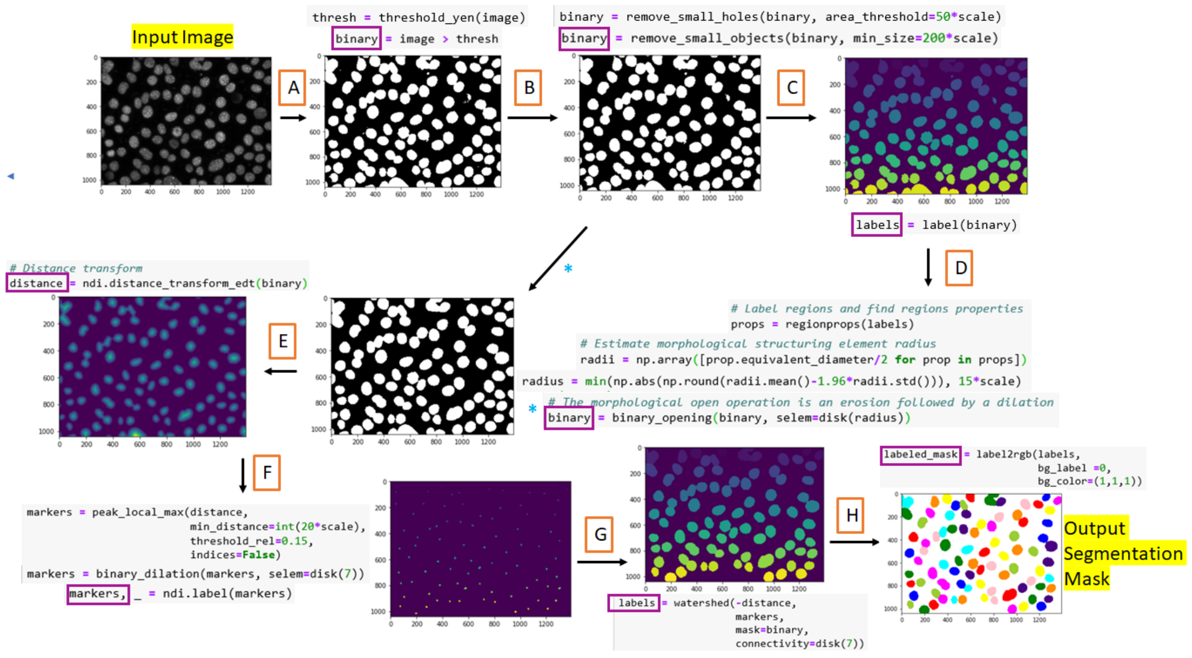
where X is the ground truth binary mask and Y is the predicted probability map.

### 4.2.4 Comparison with other Methods

The performance of the proposed approach was compared with the performance of four other approaches: Yen's thresholding followed by watershed algorithm [2, 88], Original U-Net [41], an approach similar to the winning solution of the kaggle competition in 2018 [47], and the Mask R-CNN [61]. To simplify, these models will be denoted as: Yen + watershed, Original U-Net, Kaggle_2018 and Mask R-CNN, respectively. The following subsections present information for each of these models.

### Yen + Watershed

This is a traditional image processing approach for cell nuclei segmentation [2]. The overall design of this approach is presented in figure 4.9. First, Yen's thresholding method is implemented to obtain a binary segmentation mask. Then watershed algorithm is applied to this binary mask. Inspired by the division of terrestrial surfaces into catchment basins, the watershed transform is a morphological technique for image segmentation [89]. Watershed is usually applied after a thresholding technique [2].



```
thresh = threshold_yen(image)
binary = image > thresh
```

```
binary = remove_small_holes(binary, area_threshold=50*scale)
binary = remove_small_objects(binary, min_size=200*scale)
```

```
labels = label(binary)
```

```
# Label regions and find regions properties
props = regionprops(labels)
# Estimate morphological structuring element radius
radii = np.array([prop.equivalent_diameter/2 for prop in props])
radius = min(np.abs(np.round(radii.mean()-1.96*radii.std())), 15*scale)
# The morphological open operation is an erosion followed by a dilation
binary = binary_opening(binary, selem=disk(radius))
```

```
# Distance transform
distance = ndi.distance_transform_edt(binary)
```

```
markers = peak_local_max(distance,
                         min_distance=int(20*scale),
                         threshold_rel=0.15,
                         indices=False)
markers = binary_dilation(markers, selem=disk(7))
markers, _ = ndi.label(markers)
```

```
labels = watershed(-distance,
                   markers,
                   mask=binary,
                   connectivity=disk(7))
```

```
labeled_mask = label2rgb(labels,
                         bg_label =0,
                         bg_color=(1,1,1))
```

**Figure 4.9:** Overview of the approach Yen + watershed. The variables inside purple boxes are those that are displayed as images.

First, the thresholding method calculates a threshold which is applied to the image. That is, each image pixel will be replaced by a black pixel if its intensity is smaller than the threshold, and by a white pixel otherwise (step A in figure 4.9). Afterwards, some small objects are removed, and some holes are filled (step B in figure 4.9). Then, the binary mask is labeled (step C in figure 4.9) to compute the parameters for the next step, which is a opening operation (step D in figure 4.9). Thereafter, distance_transform_edt tool from scipy module [90] is used to calculate the Euclidean distance of each pixel to its closest background pixel (step E in figure 4.9). This results in a distance map. The inverted distance map is seen as a relief. Therefore, each maximum of the distance map will correspond to a minimum in the relief. The basic idea is to imagine a source of water at every minimum. This water will flood the entire area from these sources and build watershed lines when different sources of water meet [89]. Therefore, the maximum values from this distance map (step F in figure 4.9) are the markers/seeds for the watershed algorithm (step G in figure 4.9). Additionally, the mask argument of the watershed function is equal to the binary mask obtained with the thresholding technique. Thus, the water from different sources only fills up pixels with value one in that binary mask. The watershed technique is used only to separate touching nuclei in the binary segmentation mask obtained with the thresholding technique [89]. Finally, the mask is colored with another colormap for visualization purposes (step H in figure 4.9).

### Original U-Net

U-Net architecture is used to predict two classes: nucleus and background. This architecture is designed for semantic segmentation, therefore, its output is a binary segmentation mask. In this mask pixels with value 1 belong to nuclei and pixels with value 0 belong to the background.

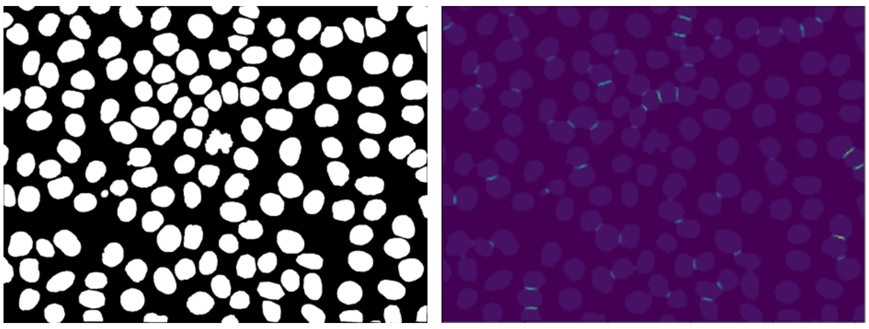
This architecture is trained to minimize a weighted binary cross entropy loss function:

$$WBCE = -\frac{1}{N}\sum_{i=1}^{N} w_i(y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})) \tag{4.4}$$

which it is similar to the loss function represented in equation 4.2, with an additional term $w_i$ which represents the weighting factor associated with pixel i. $w$ is a weight map (see figure 4.10(b)) calculated for each image from the corresponding ground truth mask (figure 4.10(a)), this weight map gives higher weights in the loss function to the pixels belonging to touching borders between nuclei. It is calculated as proposed in [41]:

$$w(x) = w_c(x) + w_0 \cdot exp\left(-\frac{(d_1(x) + d_2(x))^2}{2\sigma^2}\right) \tag{4.5}$$

where $w_c(x)$ is the weight map that takes into account the class frequencies, $d_1(x)$ and $d_2(x)$ denote the distance of the pixel $(x)$ to the border of the nearest and second nearest nuclei, respectively. The parameters $w_0$ and $\sigma$ are set to 10 and 5, respectively.



**(a)** **(b)**

**Figure 4.10: a)** Ground truth mask for a given image. **b)** Weight map for the ground truth mask shown in a).

**Kaggle_2018**

This approach is similar to the original U-Net, but here an extra class is predicted: the touching borders between close nuclei. Hence, the output of the U-Net has 3 classes: the nucleus, the background and the touching borders between close nuclei, as represented in figure 4.11.

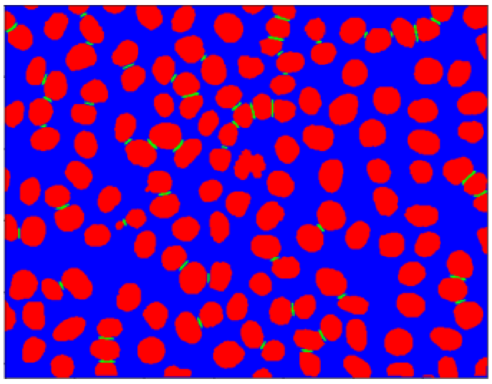


**Figure 4.11:** Example of the output of Kaggle_2018 model. The red, green and blue pixels correspond to the nucleus, touching borders and background, respectively.

This approach is trained to minimize a weighted categorical cross entropy loss, which is defined as:

$$WCCE = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{3} w(c)t_i(c)log(p_i(c)) \tag{4.6}$$

where N is the total number of pixels in the image, c goes from 1 to 3 because this problem has 3 classes, $w(c)$ is the weight of the class c (in this loss function, higher weight is given to the underrepresented green class), $\boldsymbol{t_i}$ is the true output for pixel i, and $\boldsymbol{p_i}$ is the predicted output for pixel i, both are three dimensional (3D) vectors.

In this approach, for each pixel of the input image the output is an array with 3 values. The first, second and third values represent the probabilities of nucleus, touching border and background, respectively. The class assigned to the pixel is the one with the highest probability. To compare with the other models, a binary mask is created from the ternary mask, where the pixels belonging to the nuclei have value 1 and the other pixels have value 0.

In order to train the model, ternary ground truth masks were generated from the binary ground truth masks. This procedure is shown in figure 4.12. First, the ground truth mask is labeled (step A), each object in the labeled mask will be a seed for the watershed algorithm. The mask argument for the watershed algorithm is an image obtained by applying dilation (step B) to the ground truth mask. The output of the watershed is converted to a binary image (step C). Then a bitwise XOR operation is performed between the dilated mask and the binary output of the watershed (step D). This operation will check the differences between these two images. These differences correspond to the watershed lines. Thereafter, a dilation operation is performed (step E). Finally, the ternary mask is obtained, where the red channel represents the nuclei, the green channel the dilated watershed lines (touching borders), and the blue channel the background. The implementation used to create the ternary masks is based on a publicly available implementation by Selimsef which was released under the MIT license [91] and won the Kaggle competition in 2018 [47].

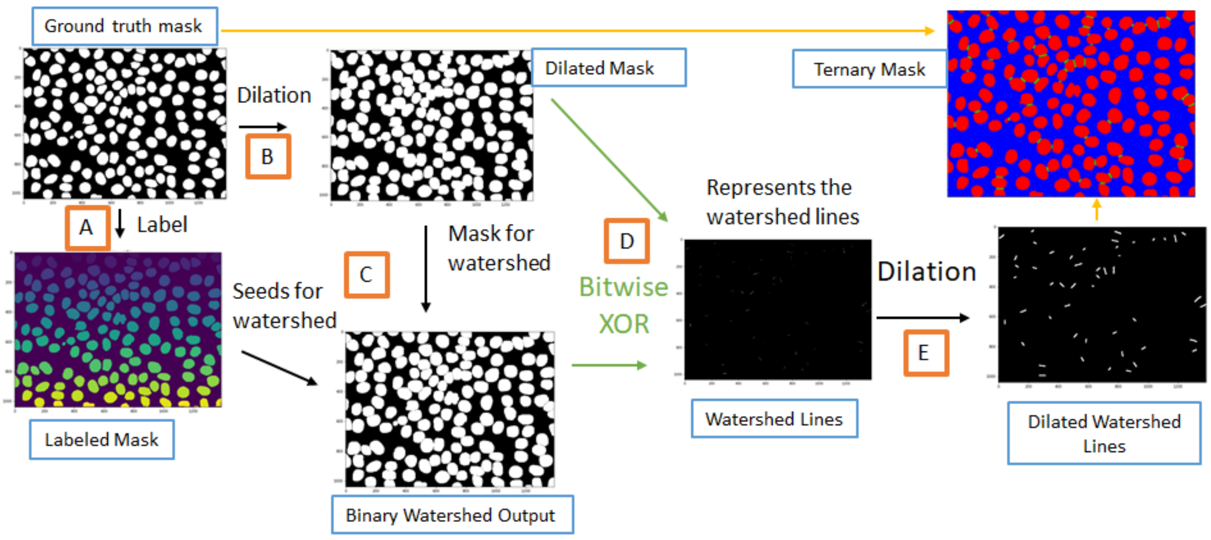


**Figure 4.12:** Procedure followed to generate ternary ground truth masks from binary ground truth masks.

### Mask R-CNN

The implementation used for Mask R-CNN is based on a publicly available implementation by Matterport which was released under the MIT license [92]. As explained in section 2.5.7, Mask R-CNN is an architecture designed for instance segmentation. Thus, different instances of the same object will have different labels. In this problem, all instances belong to the same class: nucleus. The output of this model will be a segmentation mask for each detected nucleus. Therefore, to train this model, a binary mask was created for each nucleus in the ground truth mask (see algorithm B.2, in appendix B). That is, the ground truth binary mask was labeled, thereafter for each nucleus in the labeled image a $1388 \times 1040$ binary mask was created. In this mask, the pixels belonging to the nucleus have value 1 and the other pixels have value 0 (see the example shown in figure 4.13).

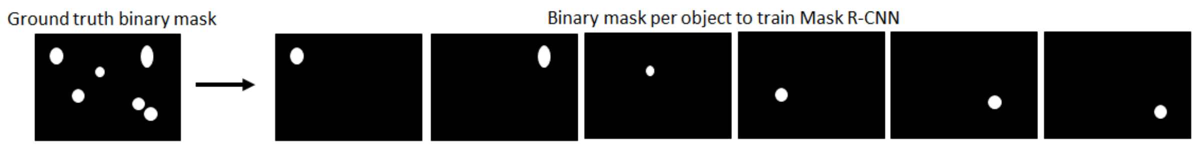


**Figure 4.13:** Illustrative example of the creation of a binary mask for each nucleus in the ground truth mask. Ground truth mask is represented on the left, and the output after applying the code represented in listing B.2 (of appendix B) is represented on the right.

### 4.2.5 Cross Validation

In order to demonstrate the robustness of each approach, a 13-fold cross validation was performed, which in this case is equivalent to a leave-one-experiment-out cross-validation. The DAPI-FUCCI dataset was divided into 13 folders. As stated before, this dataset corresponds to 13 different experiments. Each folder from a given experiment has 10 images. Each model was trained with 120 images from 12 experiments and its performance was measured in 10 images from a different experiment. This avoids the bias that would be introduced by testing a model in images that belong to the same experiment as some images used to train the model.

### 4.2.6 Training Time and Test Time

Training time is the time a model takes to learn a given task, in this case the task of nuclei segmentation. The test time is the time a model takes to give a segmentation prediction for an image. In this work the test time was calculated as the mean test time per image. That is, if $T$ is the time that a model takes to give the predictions for 130 images, then mean test time per image is $\frac{T}{130}$.

## 4.3 Cell Cycle Staging

This section describes the steps followed to build a classifier for interphase cell cycle staging. To train the supervised cell cycle staging classifier input features and ground truth labels are needed. The input features are extracted from the DAPI images, and the ground truth labels are created based on features extracted from the FUCCI images. Thus, first, the features that were extracted from both DAPI and FUCCI images are presented. Thereafter, the steps followed to create the ground truth labels from FUCCI features are presented. Afterwards, the proposed approach for cell cycle staging from DAPI images is described.

### 4.3.1 Feature Extraction

After obtaining the segmentation mask for each FUCCI image, features were extracted for each nucleus. Let $b\_p$ represent the n x m binary mask for each nucleus, where 0 denotes background and 1 denotes foreground/nucleus. $D\_p$ represents the corresponding nucleus in the DAPI image, and $F\_p$ represents the corresponding RGB nucleus in the FUCCI image. As explained before, for each FUCCI image there is the corresponding DAPI image, (see figure 4.2). Therefore, first the segmentation masks for FUCCI images are obtained. Thereafter, for each segmented nucleus in the FUCCI image the corresponding DAPI patch can be obtained. Then, for each nucleus some features were extracted which are summarized in table 4.1. Additionally, based on these features, other quantities were calculated for each nucleus, which are shown in table 4.2. Note that nuclei located at image borders were not considered in further analysis since the information provided by them is incomplete.

**Table 4.1:** Feature Extraction from the DAPI and FUCCI images

| | | |
|---|---|---|
| **Area** ($A$) - Number of pixels belonging to the nucleus | $A = \sum_{i=1}^{n} \sum_{j=1}^{m} b\_p(i,j)$ | (4.7) |
| **Total DAPI Intensity** ($T_D$) - Sum the intensities of the pixels belonging to the nucleus extracted from the DAPI image | $T_D = \sum_{i=1}^{n} \sum_{j=1}^{m} b\_p(i,j) D\_p(i,j)$ | (4.8) |
| **Total Red Intensity** ($T_R$) – Sum the red channel's intensities of the pixels belonging to the nucleus extracted from the FUCCI image | $T_R = \sum_{i=1}^{n} \sum_{j=1}^{m} b\_p(i,j) F\_p(i,j,0)$ ($F\_p(i,j,0)$ denotes the red channel) | (4.9) |
| **Total Green Intensity** ($T_G$) – Sum the green channel's intensities of pixels belonging to the nucleus extracted from the FUCCI image | $T_G = \sum_{i=1}^{n} \sum_{j=1}^{m} b\_p(i,j) F\_p(i,j,1)$ ($F\_p(i,j,1)$ denotes the green channel) | (4.10) |

**Table 4.2:** Quantities calculated from the features shown in table 4.1

| | | |
|---|---|---|
| Mean Green Intensity ($\mu_G$) | $\mu_G = \frac{T_G}{A}$ | (4.11) |
| Mean Red Intensity ($\mu_R$) | $\mu_R = \frac{T_R}{A}$ | (4.12) |
| Normalized Red Intensity ($\hat{R}$) | $\hat{R} = \frac{\mu_R}{\sqrt{{\mu_R}^2 + {\mu_G}^2}}$ | (4.13) |
| Normalized Green Intensity ($\hat{G}$) | $\hat{G} = \frac{\mu_G}{\sqrt{{\mu_R}^2 + {\mu_G}^2}}$ | (4.14) |
| Normalized Area ($\hat{A}$) for each nucleus (normalized per image)[3] | $\hat{A} = \frac{A - \mu_A}{\sigma_A}$ [4] | (4.15) |
| Normalized DAPI Intensity ($\hat{I_D}$) per nucleus (normalized per image) | $\hat{I_D} = \frac{T_D - \mu_{T_D}}{\sigma_{T_D}}$ [5] | (4.16) |

### 4.3.2 Cell Cycle Staging from FUCCI to Create the Ground Truth Labels

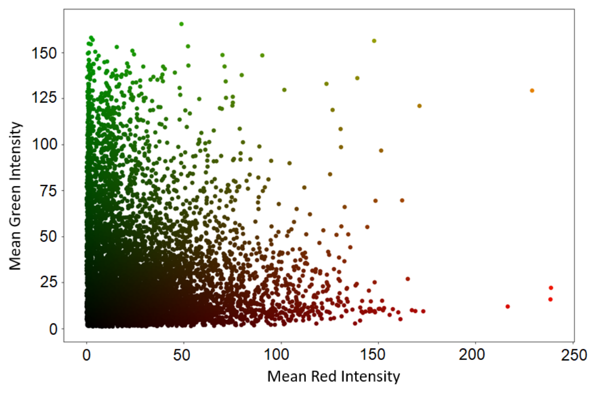
According to the FUCCI system, the red nuclei are in G1 phase, while green nuclei are in S/G2/M phases [70, 71]. This information was used to obtain a label for each nucleus. First, each nucleus was represented in the two dimensional (2D) space (Mean Red Intensity, Mean Green Intensity). That is, each nucleus is represented by 2 coordinates: the mean green intensity (equation 4.11) and the mean red intensity (equation 4.12). Additionally, on one hand, each point was colored with the following RGB coordinates (Mean Red Intensity, Mean Green Intensity, 0) (figure 4.14(a)). On the other hand, each point was colored with the normalized RGB coordinates, that is, (R,G,B) = (Normalized Red Intensity, Normalized Green Intensity, 0) (figure 4.14(b)).

In the previous work [1] the biologists labeled each nucleus manually based on its color, that is, if it was green it was labeled as S/G2 and if it was red it was labeled as G1. The color that they observed for each nucleus is represented in figure 4.14(a). In this figure there are clearly green nuclei, red nuclei, and some colorless nuclei (the ones with very low intensity). In this work, an automatic way to obtain the label for each nucleus is presented. This algorithm takes into account the color of each nucleus in the FUCCI image. In order to better visualize the color of each nucleus, the RGB intensities were normalized according to equations 4.13 and 4.14. Figure 4.14(b) shows each nucleus with the normalized intensity.

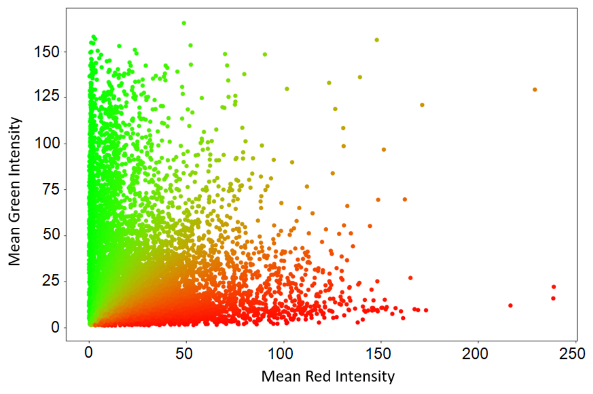
[3]The area and intensity were normalized per image for the reason stated before, that is, there can be experimental variations between images acquired in the same experiment.

[4]$\mu_A$ and $\sigma_A$ denote the mean and the standard deviation of the area, respectively, of all nuclei belonging to a given image.

[5]$\mu_{T_D}$ and $\sigma_{T_D}$ denote the mean and the standard deviation of the total DAPI intensity, respectively, of all nuclei belonging to a given image.



(a)



(b)

**Figure 4.14:** Representation of each nucleus in the 2D space (Mean Red Intensity, Mean Green Intensity). **a)** In this plot each point is colored with the following intensities (R,G,B) = (Mean Red Intensity, Mean Green Intensity, 0). **b)** In this plot each point is colored with the following intensities (R, G, B) = (Normalized Red Intensity, Normalized Green Intensity, 0).

Based on this plot and on the information given by the FUCCI system it is easy to obtain the label for each nucleus. The line $Mean\ Green\ Intensity = Mean\ Red\ Intensity$ can be drawn, and every point located in the semi-plane $Mean\ Green\ Intensity > Mean\ Red\ Intensity$ is labeled as S/G2, whereas every point located in the semi-plane $Mean\ Green\ Intensity < Mean\ Red\ Intensity$ is labeled as G1. Note that in this work nuclei in M phase weren't considered, that's why green nuclei are labeled as S/G2.

The normalization step was performed to better visualize the color of each point. But this step will convert the colorless nuclei into green or red nuclei, and it isn't correct to consider these nuclei. This is due to the fact that both early G1 and G0 cells are colorless, therefore it is difficult to discriminate visually these cells [12]. Thus, the next step consisted in the removal of these colorless nuclei from further analysis. Furthermore, some nuclei belonging to the transition between red and green (nuclei closer to the line Mean Green Intensity = Mean Red Intensity) were removed as well, since they are more difficult to evaluate. Hence, the nuclei that were labeled and considered in this study are represented in figure 4.15(b). The nuclei that were removed (represented in gray in figure 4.15(a)) are the ones that verified at least one of the following conditions:

1. $\mu_R < T_1 \ \ and \ \ \mu_G < T_2$
2. $\mu_R < \theta_1 \mu_G \ \ and \ \ \mu_G < \theta_2 \mu_R$
3. $A < T_3$
4. $A > T_4$

where $T_1 = T_2 = 40$, $\theta_1 = \theta_2 = 1.1$, $T_3 = 3000$ and $T_4 = 9000$, all these variables were chosen by visual inspection. The first equation selects the colorless nuclei, the second equation the ones belonging to the transition between red and green, and finally the two last equations select the outliers. All the nuclei selected by these equations were removed. All the other nuclei were labeled according to the algorithm

4.1.

**Algorithm 4.1:** Automatic algorithm to assign a label to each nucleus based on FUCCI technology.

```
if $\mu_G > \mu_R$ then
    $label = S/G2$
if $\mu_R > \mu_G$ then
    $label = G1$
```

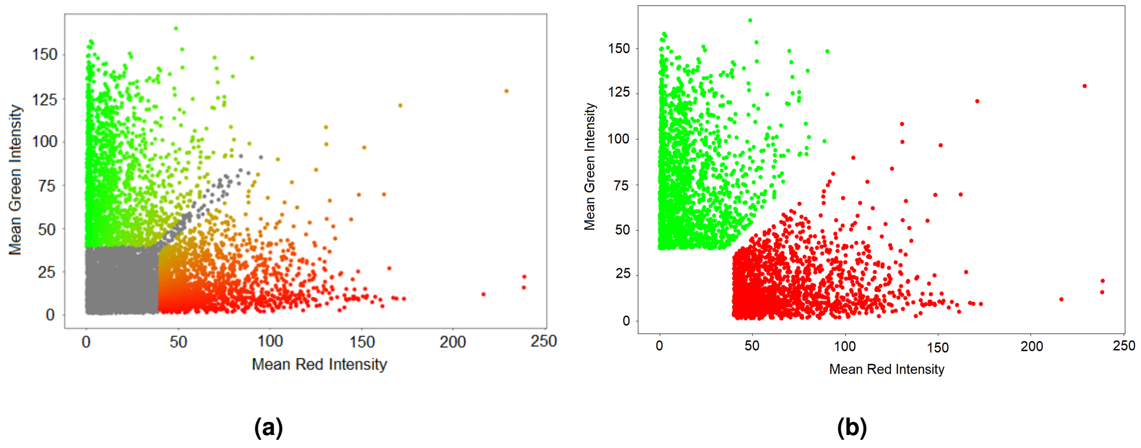
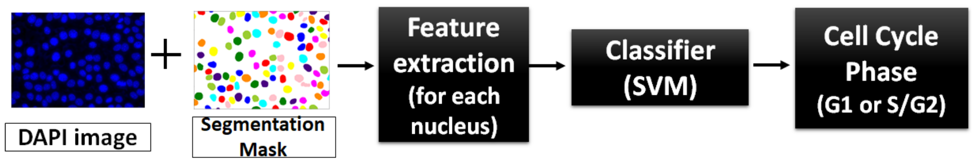


**Figure 4.15:** **a)** Representation of each nucleus in the 2D space (Mean Red Intensity, Mean Green Intensity). The nuclei that were removed are colored with gray. All of the other nuclei are colored with the following intensities (R, G, B) = (Normalized Red Intensity, Normalized Green Intensity, 0). **b)** Representation of each nucleus in the 2D space (Mean Red Intensity, Mean Green Intensity) after removing the colorless nuclei, outliers, and some nuclei in the transition between red and green. Each point is colored with its label, i.e., nuclei labeled as S/G2 are colored with green, whereas nuclei labeled as G1 are colored with red.

### 4.3.3 Proposed Approach for Cell Cycle Staging from DAPI images

The FUCCI technology depends on genetically encoded proteins, which may be toxic for the cells and alter their behaviour [12]. Therefore, in this work a new approach is proposed for cell cycle staging. This approach performs cell cycle staging based on features extracted from images of cells stained with DAPI, a commonly used fluorescent dye which presents low cytotoxicity [12]. Thus, after obtaining the labels for the nuclei, a machine learning algorithm (SVM) was trained and tested with features extracted from DAPI images as input features and with the corresponding ground truth labels generated based on the FUCCI images. For each nucleus, this classifier assigns the cell cycle phase (G1 or S/G2), based on the features extracted from the respective patch in the DAPI image (figure 4.16).



DAPI image
Segmentation Mask
Feature extraction (for each nucleus)
Classifier (SVM)
Cell Cycle Phase (G1 or S/G2)

**Figure 4.16:** Supervised machine learning classifier (SVM) for interphase cell cycle staging.

**SVM: Theory**

SVM is a supervised machine learning algorithm, designed to solve binary classification problems. SVM gives an optimal hyperplane in the feature space, in order to separate the examples belonging to different classes [93]. A detailed explanation regarding SVM is provided below.

- **Linearly separable binary classification**

Considering the training data consisting of M points, $\{\boldsymbol{x}_i, y_i\} \quad i = 1, ..., M$, where $y_i \in \{-1, 1\}$ and $\boldsymbol{x} \in \mathbb{R}^N$, $\boldsymbol{x}_i$ is the input i of dimension N and $y_i$ is the respective label. If this training data is linearly separable, that is, if there is a hyperplane in $\mathbb{R}^N$ that separates the training patterns ($\boldsymbol{x_i} \quad i = 1, ..., M$) into two non-overlapping classes, then the SVM outputs a hyperplane $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ that splits the input space, where $\boldsymbol{w}$ is the normal vector to the hyperplane, and $\frac{b}{\|\boldsymbol{w}\|}$ is the perpendicular distance from the hyperplane to the origin. SVM selects $\boldsymbol{w}$ and $b$ in a way that the training data is described by [94]:

$$\begin{cases} \boldsymbol{x}_i \cdot \boldsymbol{w} + b \geq 1 & \text{for } y_i = 1 \\ \boldsymbol{x}_i \cdot \boldsymbol{w} + b \leq -1 & \text{for } y_i = -1 \end{cases} \tag{4.17}$$

which is equivalent to:

$$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 \geq 0, \forall_i \tag{4.18}$$

When the training data is linearly separable there are several hyperplanes that divide correctly the training data. The hyperplane that is chosen is the one with the largest margin, this allows to generalize better to new unseen data. The margin is given by $\frac{1}{\|\boldsymbol{w}\|}$, maximizing it is the same as minimizing $\|\boldsymbol{w}\|$, which is equivalent to minimizing $\frac{1}{2}\|\boldsymbol{w}\|^2$. Hence, the goal is to find [94]:

$$min \ \ \frac{1}{2}\|\boldsymbol{w}\|^2 \quad s.t. \ \ y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 \geq 0, \forall_i \tag{4.19}$$

When there is a new sample $\boldsymbol{x}_{test}$, the classification function $c = sgn(\boldsymbol{w} \cdot \boldsymbol{x}_{test} + b)$ is used to assign a label to $\boldsymbol{x}_{test}$, it tells the position of $\boldsymbol{x}_{test}$ with respect to the hyperplane. The function $sgn(x)$ is defined as [94]:

$$sgn(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{4.20}$$

- **Binary classification for data that is not fully linearly separable**

When the training data is not linearly separable, a positive slack variable ($\xi_i, i = 1, ..., M$) is introduced in the objective function to allow some missclassified points. In this case the goal is to find:

$$min \ \ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{M}\xi_i \quad s.t. \ \ y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i \geq 0, \forall_i \tag{4.21}$$

where C represents the trade-off between the penalty given by the slack variable and the margin size, it is a hyperparameter [94].

- **Non-linear SVM**

  There are some classification problems that aren't linearly separable in the input space, but they are linearly separable in a higher dimensional feature space obtained using a convenient mapping, $\boldsymbol{x} \rightarrow \phi(\boldsymbol{x})$, where $\phi(\boldsymbol{x})$ is the non linear mapping function from the input space ($\boldsymbol{x} \in \mathbb{R}^N$) to the feature space ($\tilde{\boldsymbol{x}} = \phi(\boldsymbol{x}) \in \mathbb{R}^{N'}$), where $N' > N$ since the feature space has usually higher dimension than the input space [94].

  During the training phase the inner products $\phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j)$ must be computed, which may involve high dimensional vectors. That is, computing the coordinates of the data points in the feature space may be computationally expensive. To avoid very demanding calculations the "kernel trick" is used. In this approach, the kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j)$ is defined, this function computes the inner products without the need of going to the output dimension. The commonly used kernel functions are [94]:

  - Radial basis function (rbf) kernel: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{\|(\boldsymbol{x}_i - \boldsymbol{x}_j)\|^2}{2\sigma^2}}$, $\sigma$ is a parameter.

  - Polynomial kernel: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j + a)^b$, $a$ and $b$ are parameters.

  - Sigmoidal kernel: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = tanh(a\boldsymbol{x}_i \cdot \boldsymbol{x}_j - b)$, $a$ and $b$ are parameters.

  - Linear kernel: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$

  When there is a new sample $\boldsymbol{x}_{test}$, the classification function $c = sgn(\boldsymbol{w} \cdot \phi(\boldsymbol{x}_{test}) + b)$ is used to assign a label to $\boldsymbol{x}_{test}$. The mapping from the input space to the feature space is non-linear, thus the hyperplanes in the feature space correspond to non-linear decision boundaries in the input space [94].

**SVM: Training**

The SVM was trained and its performance was evaluated in two input spaces:

- SVM applied in the non-normalized input space, i.e., in the 2D input space (Area (equation 4.7), Total DAPI intensity (equation 4.8)), this is denoted as SVM_A. This is the input space proposed in [1] for the k-means algorithm for cell cycle staging.

- SVM applied in the normalized input space, i.e., in the 2D input space (Normalized Area (equation 4.15), Normalized DAPI intensity (equation 4.16)), this is denoted as SVM_B. This is the input space proposed in this work.

First, a five-fold cross validation was performed, which is useful when one wants to compare the performance of different classifiers, in this case, the performance of the SVM in two input spaces. To divide the dataset into 5 folds, the tool KFold from sklearn.model_selection [6] was used. Information regarding each folder is presented in table 4.3. For the five fold cross-validation 5 models were trained, and the final results represent an average $\pm$ standard deviation between the results of the five test folds.

**Table 4.3:** Information regarding each of the 5 folds generated to perform five-fold cross validation. The first column denotes the fold number. The second and third column denote the number of nuclei in stage G1 and S/G2, respectively. The fourth column represents the total number of nuclei in that fold. Finally, the last two columns denote the percentage of G1 nuclei and S/G2 nuclei, respectively.

| Fold | G1 | S/G2 | Total | G1 (%) | S/G2 (%) |
|---|---|---|---|---|---|
| 1 | 462.0 | 249.0 | 711 | 64.98 | 35.02 |
| 2 | 470.0 | 241.0 | 711 | 66.10 | 33.90 |
| 3 | 463.0 | 248.0 | 711 | 65.12 | 34.88 |
| 4 | 445.0 | 265.0 | 710 | 62.68 | 37.32 |
| 5 | 451.0 | 259.0 | 710 | 63.52 | 36.48 |

Furthermore, for the SVM_B a 13-fold cross-validation was performed. Since the dataset consists of images belonging to 13 experiments, this is equivalent to a leave-one-experiment-out cross-validation. That is, a model was trained with all nuclei belonging to images of 12 experiments, and its performance was tested on nuclei belonging to images of the other experiment. This was repeated 13 times. There are a total of 130 images, therefore 130 values of average F1 score between class G1 and S/G2 were obtained (the concept of F1 score is explained in appendix C). These values were represented in a box plot and in a violin plot in order to understand the distribution of the data (detailed explanation regarding box plot and violin plot is provided in appendix C). As stated before, the features of the proposed input space were normalized image by image. The 13-fold cross-validation was conducted just for the SVM_B in order to understand how the feature normalization step influences the performance of the proposed approach in each image.

Additionally, for both experiments, the class_weight argument of the SVC function from sklearn_svm is set to 'balanced'. So that the weights are inversely proportional to the class frequencies in the training data [96]:

$$class\_weight\_i = \frac{n\_samples}{n\_classes \times n\_samples\_i} \tag{4.22}$$

where class_weight_i is the class weight for the class i, n_samples denotes the number of samples in the training data, n_classes the number of classes in the classification problem, and n_samples_i the number of samples belonging to class i.

This is a technique used to deal with imbalanced datasets, it will train the SVM where the training

[6]scikit-learn, also known as sklearn is a python's module which provides tools for data mining and data analysis. [95]

samples have different cost weights in the objective function, these weights depend on the class size [97] as defined in equation 4.22. In this problem, there are more nuclei in G1 phase than nuclei in phases S/G2, therefore the class weight will be higher for S/G2 nuclei.

### Test the Performance of the Trained Classifiers on a Different Cell Line

Additional experiments were performed in order to study the potential of the SVM for cell cycle staging of cells from other cell lines different from the one that was used to train it. The SVM_A and SVM_B were trained with all the nuclei 3553 (from images of normal murine mammary gland cells), and their performance was tested on images of gastric cancer cells stained with DAPI.

An example of an image from the gastric cancer cell line is shown in figure 4.17. In these images the nuclei in G2 phase were annotated manually by the biologists based on the expression of the cyclin B1 (marked with an antibody). This cyclin is a marker for cells in G2/M phases (note that the M phase is not considered in this study). The red arrows in figure 4.17 denote the nuclei labeled by the biologist as G2. The dataset of microscopic images of gastric cancer cells stained with DAPI contains 15 images, and a total of 73 nuclei were annotated as being in G2 phase. First these images were segmented, then features (Area, Total DAPI Intensity) were extracted from all nuclei, and normalized area and normalized total intensity were calculated. Afterwards, the features of the 73 nuclei labeled as G2 were fed to SVM_A and SVM_B, and their performance was measured.

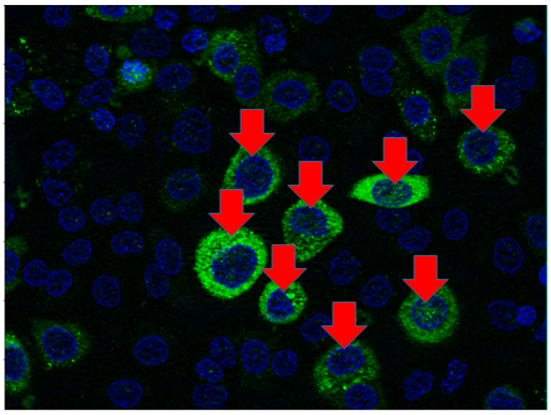


**Figure 4.17:** Example of an image from the gastric cancer cell line. The red arrows indicate the nuclei labeled by the biologists as nuclei in G2 phase.

# 5 Experimental Results

**Contents**

This chapter presents the experimental results and discussion regarding cell nuclei segmentation and cell cycle staging.

## 5.1 Computational Environment

All deep learning implementations are based on open-source deep learning libraries Tensorflow and Keras [98, 99]. Keras is a high-level application programming interface (API) of Tensorflow. These are frameworks that compute backpropagation and update the parameters automatically. The validation loss and other metrics (validation accuracy, validation F1 score, ...) were monitored on Tensorboard, which is a set of visualization tools for Tensorflow programs. Additionally, all experiments were carried out in python 3.6, on a laptop with the following specifications:

- Processor: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz;
- Memory: 8 GB RAM;
- Storage: 128 GB SSD + 1 TB HDD;
- Graphics Processing Unit (GPU): NVIDIA GeForce GTX 1050 with 4 GB of VRAM.

## 5.2 Cell Nuclei Segmentation

In this section results regarding cell nuclei segmentation in DAPI images and in FUCCI images are presented.

### 5.2.1 DAPI

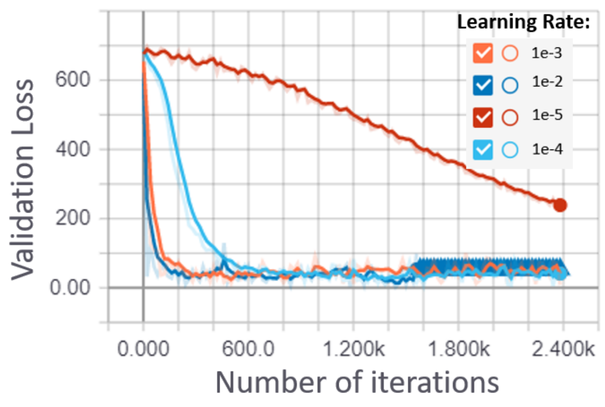
This subsection presents the results obtained by training and testing models with DAPI images.

#### Implementation Details

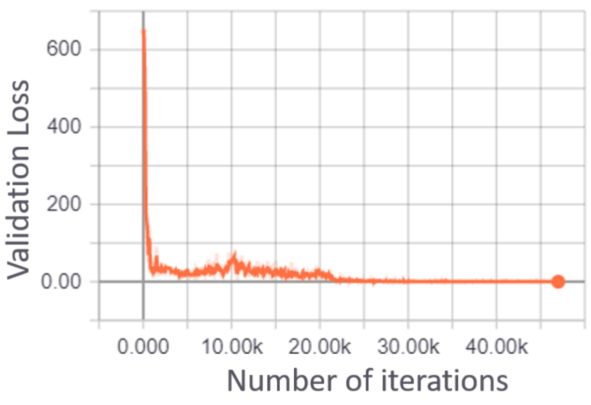
First, details regarding the choice of the learning rate and other parameters of the deep learning models will be presented.

- **Proposed Approach**

First, Fast YOLO was trained for 40 epochs with different learning rates in order to monitor the validation loss, as represented in figure 5.1(a). By observing these plots the learning rate that allows convergence and for which the validation loss presents less oscillations is 1e-4. Hence, Fast YOLO was trained from scratch and using ADAM optimizer, first it was trained for 200 epochs with learning rate 1e-4. Then it was trained for another 600 epochs with learning rate 1e-5. The learning rate 1e-4 was chosen based on the plots shown in figure 5.1(a). After 200 epochs it was decreased to 1e-5. Although this learning rate was too small at the beginning (red plot in figure 5.1(a)), after 200 epochs the training loss is closer to the minimum of the loss function, therefore it is better to decrease the learning rate. The training was stopped after 800 epochs because the validation loss became smooth and small (approximately 1), as represented in figure 5.1(b).



**(a)**



**(b)**

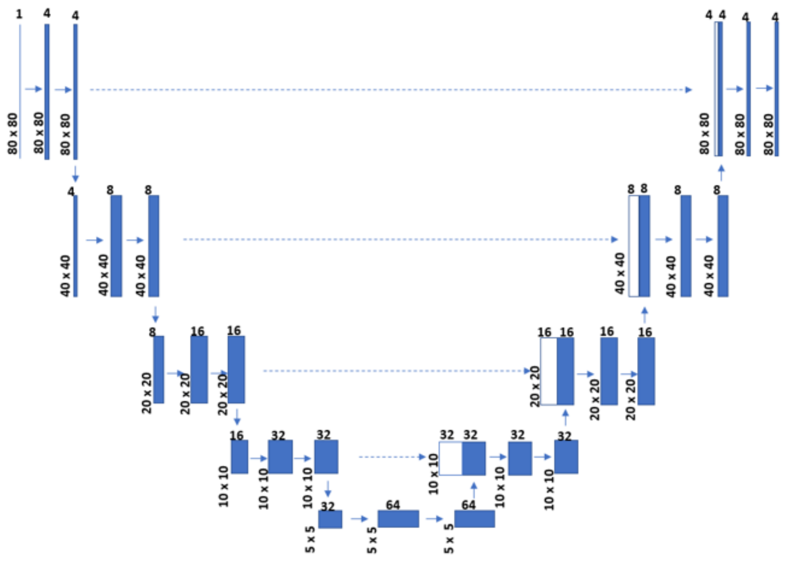
**Figure 5.1:** Validation loss plots for Fast YOLO. **a)** Validation loss as function of the training iterations, for different learning rates. It was trained for 40 epochs which is equivalent to 2,400 iterations. Plots obtained from Tensorboard. **b)** Validation loss during the training of Fast YOLO, for 800 epochs, which corresponds to 48,000 iterations. Plot obtained from Tensorboard.

The U-Net of the proposed approach was implemented using Keras [99] with Tensorflow backend [98]. The architecture implemented is represented in figure 5.2(a). It was trained from scratch, for 100 epochs, with learning rate 1e-4, using ADAM optimizer with the argument amsgrad equal to True[1], without dropout, with batch size equal to 20, with Xavier initialization and with ReLU as activations functions, except in the last layer, which has a sigmoid activation function. Thus, the output image has values in the $[0, 1]$ interval. In order to obtain the binary mask a threshold of 0.5 was applied to the U-Net's output. Pixels above this value belong to the nucleus and pixels below this value belong to the background. Moreover, to choose the learning rate, the model was trained for 100 epochs with different learning rates, as represented in figure 5.2(b). The lowest validation loss was obtained with learning rate 1e-4, which is the learning rate used to train the U-Net. A plot of the validation loss during training of the U-Net is shown in figure 5.2(c).

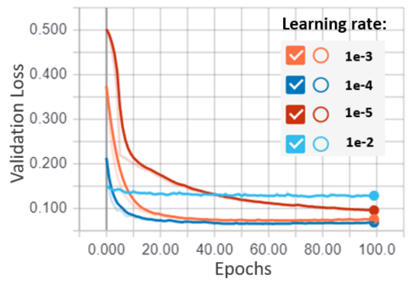
- **Original U-Net**

The Original U-Net was implemented using Keras and Tensorflow. The input images and the respective ground truth masks were resized to images of size $256 \times 256$, due to limited GPU memory. It was trained for 200 epochs, from scratch, using AMSGrad variant of the ADAM optimizer, with learning rate 1e-3, with Xavier initialization, with batch size of 8, with ReLU as activation functions, except in the last layer

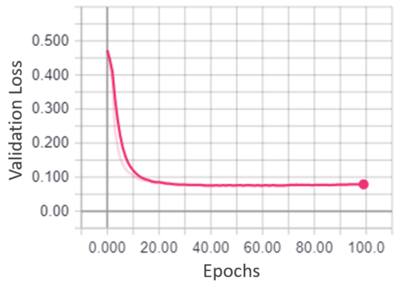
[1]By setting this parameter as true, AMSGrad variant of ADAM optimizer is used, which was presented in [100].



(a)



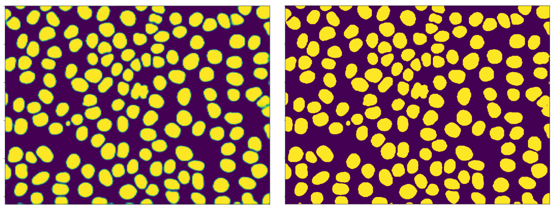
(b)



(c)

**Figure 5.2:** U-Net of the proposed approach. **a)** U-Net architecture used for the segmentation step of the proposed approach. **b)** Validation loss as function of the number of iterations, for different learning rates. Plots obtained from Tensorboard. **c)** Validation loss during the training of the U-Net, for 100 epochs. Plot obtained from Tensorboard.

which has a sigmoid activation function. Thus, the output is an image with values from 0 to 1 (figure 5.3(a)). In order to obtain the final segmentation mask (figure 5.3(b)) a threshold of 0.5 is applied to the U-Net's output. Validation loss plots to choose the best training parameters and the plot of the validation loss during the training of the U-Net are presented in appendix D.



(a) (b)

**Figure 5.3:** **a)** Example of the output of the U-Net. **b)** Binary image obtained after applying thresholding to image a). Images a) and b) are represented with a different colormap to visualize better the difference between U-Net's output and the output after thresholding.

- **Kaggle_2018**

The images and the ground truth ternary masks were resized to $256 \times 256$. The model was trained for 200 epochs, from scratch, with AMSGrad variant of ADAM optimizer, with Xavier initialization, with learning rate 3e-4, batch size of 2, ReLU as activation functions, except in the last layer, which has a softmax activation function. Validation loss plots to choose the learning rate and the plot of the validation loss during the training of Kaggle_2018 are presented in appendix D.

- **Mask R-CNN**

The images and masks were resized to $256 \times 256$. The model was trained for 200 epochs, with learning rate 1e-4. The convolutional backbone was initialized with ResNet50 parameters, this is known as transfer learning. It is an approach to transfer the knowledge acquired on a given task to the task at hand. ResNet50 is a pre-trained model on a huge dataset, for example on the ImageNet [2] [15]. The weights that are learned in that task are reused as the starting point for the Mask R-CNN on the task of nuclei segmentation. In a complex model like the Mask R-CNN this can allow for faster convergence. Validation loss plots to choose the learning rate and the plot of the validation loss during the training of Mask R-CNN are presented in appendix D.

### Comparison Between Different Approaches

Figure 5.4 shows a visual comparison between different models, regarding nuclei segmentation. The images shown in the first row were chosen in order to show the variability that exists between different input images. Some images have a lot of nuclei that are overlapping and/or touching, others are more simple to segment since they contain few nuclei, and there are some images where the pixel intensity varies along the image, that is some nuclei are very bright and others are darker. The second row in figure 5.4 represents the ground truth masks generated for each image represented in the first row, the deep learning based models were trained using these ground truth masks. The third row shows the segmentation masks obtained by applying Yen + watershed. By observing these masks it can be concluded that in some cases (for instance, the one highlighted with a black circle) the segmentation masks are bigger than the ground truth masks. It is, in fact, a disadvantage of the thresholding methods, because it is based on a threshold to define what is foreground and what is background. Moreover, the watershed algorithm doesn't split all the overlapping nuclei, there are some merges present in these segmentation masks, as highlighted with dashed black circles. In fact, when compared to the other approaches, Yen + watershed is the one that presents more merges. Finally, in the last image (fourth column) there is high intensity variation, therefore the classical method struggles in detecting nuclei located in the lower left corner, highlighted with a black square.

Results regarding Original U-Net (fourth row) and Kaggle_2018 (fifth row) show that although these approaches separate better the touching nuclei, in some cases there are gaps between close nuclei

[2]ImageNet "Large Scale Visual Recognition Challenge" is a competition where algorithms for image classification and object detection are evaluated on large datasets that have been carefully annotated [101].

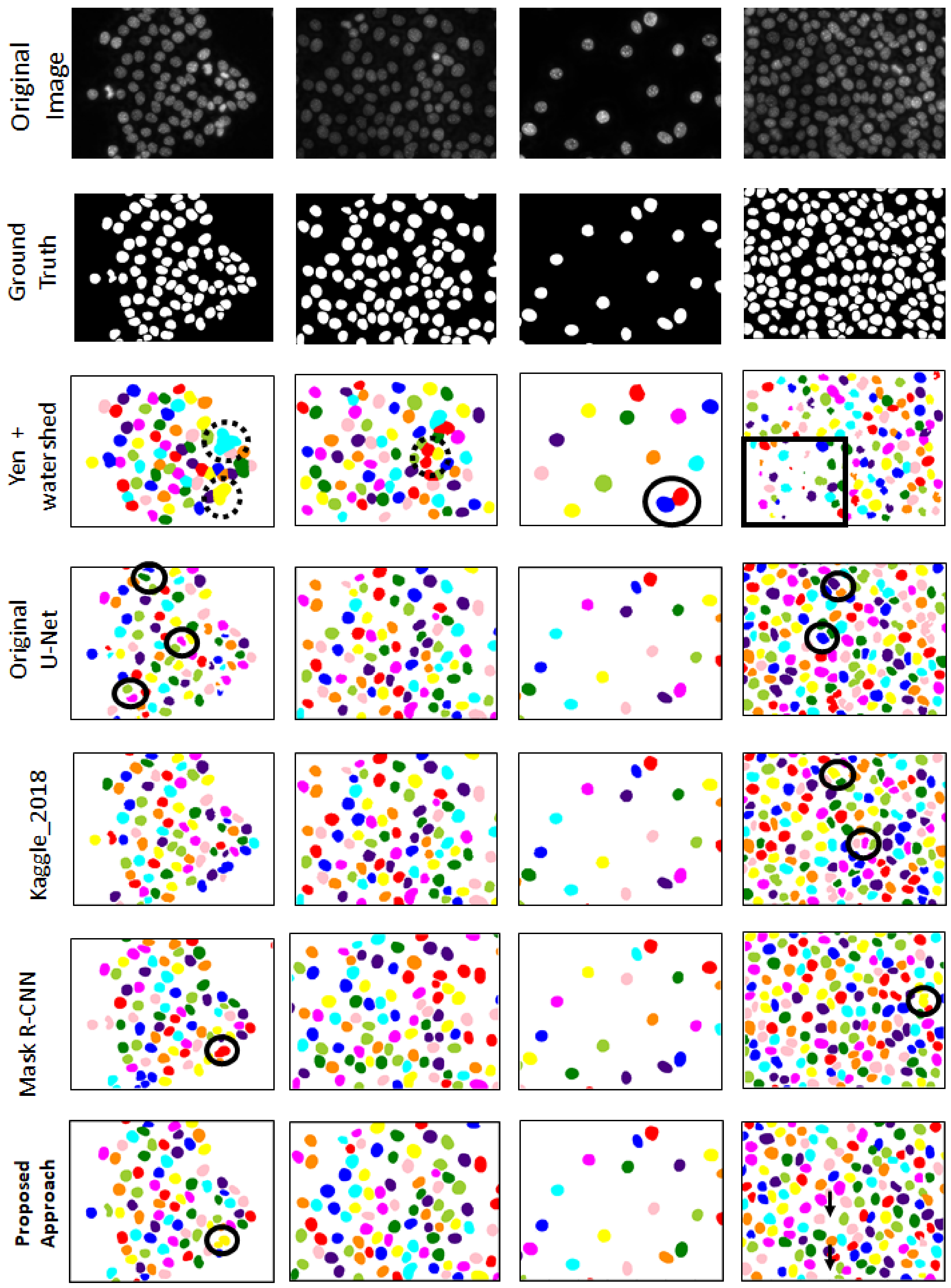


**Figure 5.4:** Nuclei segmentation results obtained by applying different methods to DAPI images. The first row represents examples of the original images, for which the segmentation masks are to be obtained. The second row represents the corresponding ground truth masks. Finally, the third, fourth, fifth, sixth and seventh rows represent the corresponding segmentation results obtained by applying Yen + watershed, Original U-Net, Kaggle_2018, Mask R-CNN and the proposed approach, respectively. Some errors are highlighted by black circles, arrows and square.

(highlighted with black circles). On one hand, this can be explained by the fact that the ground truth data also has some gaps between touching nuclei (as explained in subsection 4.2.1). On the other hand, as stated in subsection 2.5.5, the majority of the works that used U-Net had some post-processing step to refine the nuclei boundaries, because the U-Net is designed for semantic segmentation. Thus, it makes the classification at a pixel level, that's why the nuclei boundaries obtained with the Original U-Net and Kaggle_2018 aren't that accurate. So, a post-processing step could improve the segmentation masks obtained with these two approaches, however this step typically increases the computational burden. Finally, in the results obtained with the Mask R-CNN (sixth row) and with the proposed approach (seventh row) those gaps disappear, since these two approaches are designed for instance segmentation. Although these approaches are the ones that present the best results there are still some merges in the output segmentation masks (highlighted with black circles).

Furthermore, by comparing the segmentation masks obtained for the last image (fourth column), with Mask R-CNN and with the proposed approach, it is observed that the proposed approach fails to identify some of the occluded or very close nuclei (represented with black arrows). This is due to the detection performance of Fast YOLO which is worse than that of the Mask R-CNN. Nevertheless, for all of the other three images the segmentation masks obtained with the proposed approach and the ones obtained with the Mask R-CNN are very similar. This shows that the proposed approach achieves results that are comparable to the state-of-the-art method for instance segmentation.

The evaluation metric (F1 score at different thresholds of IoU) which is used to measure the performance of the models for nuclei segmentation is explained in detail in appendix C (section C.1). In figure 5.5 plots of the F1 score across increasing thresholds of IoU are shown. In fact, it is the average F1 score over the 13 models trained and tested for each approach. By observing these plots it can be concluded that the deep learning models significantly outperform the classical method (Yen + watershed). By comparing the performance of the Original U-Net and the Kaggle_2018 it can be concluded that Kaggle_2018 performs better than Original U-Net. This shows that the additional channel to predict touching borders between nuclei allows to separate better touching and/or very close nuclei. For IoU $threshold < 0.8$ the deep learning based approaches present F1 score values above 0.9, which is very good. Additionally, for IoU $threshold < 0.75$ the proposed approach and the Mask R-CNN present similar performance and better than that of all the other methods.

Moreover, an accentuated decrease of the F1 score is observed, at IoU $\approx 0.80$. This is explained by the presence of some inaccurate boundaries on the ground truth data. As explained above, to separate touching nuclei and to solve the problem as an instance segmentation problem, lines were drawn between touching nuclei and the pixels belonging to these lines were considered background pixels. To better understand this decrease of the F1 score, figure 5.6 represents a patch of an image, the corresponding ground truth patch and the predictions made with Yen + watershed, original U-Net,

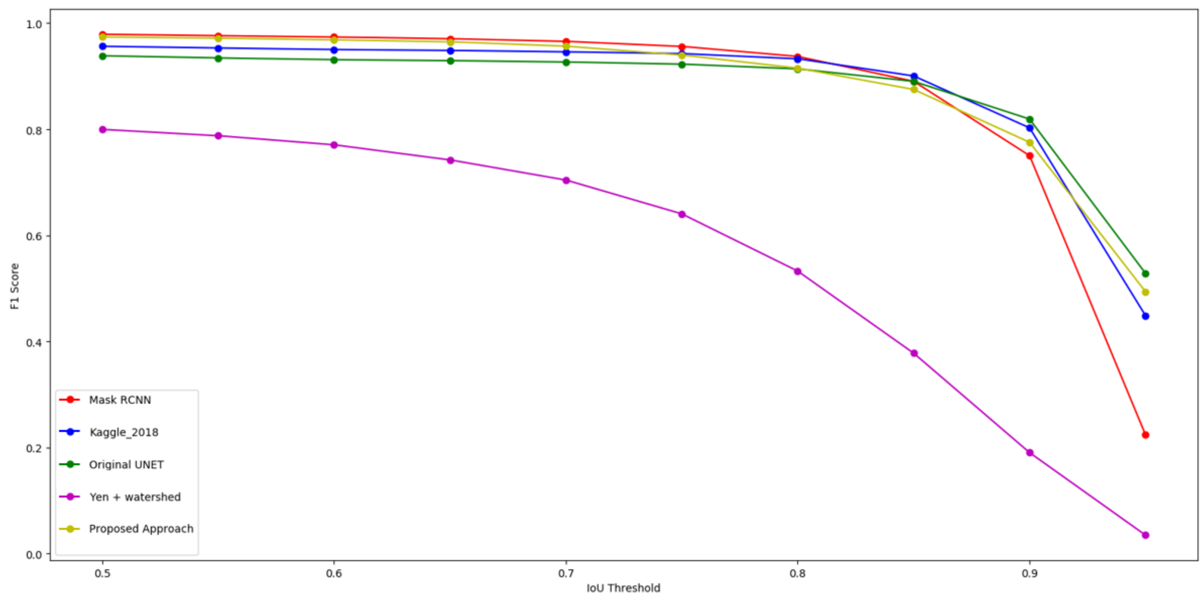


**Figure 5.5:** Average F1 Score vs IoU threshold, comparison between different models: Yen + watershed (purple), original U-Net (green), Kaggle_2018 (blue), Mask R-CNN (red), proposed approach (yellow).

Kaggle_2018, Mask R-CNN and the proposed approach. On one hand, the mask provided by Yen + watershed (figure 5.6(c)) is bigger than the ground truth mask (figure 5.6(b)). Moreover, for the approaches based on the U-Net (figures 5.6(d) and 5.6(e)) the segmentation masks contain gaps bigger than those contained in the ground truth mask (figure 5.6(b)). Both situations impact negatively the F1 score for high IoU thresholds. On the other hand, for the models that perform instance segmentation there aren't gaps between touching nuclei (figures 5.6(f) and 5.6(g)). These models are able to identify accurately each nucleus separately. However, in the ground truth mask there are be gaps between those nuclei (figure 5.6(b)). This can explain the decrease of F1 score for high IoU thresholds.

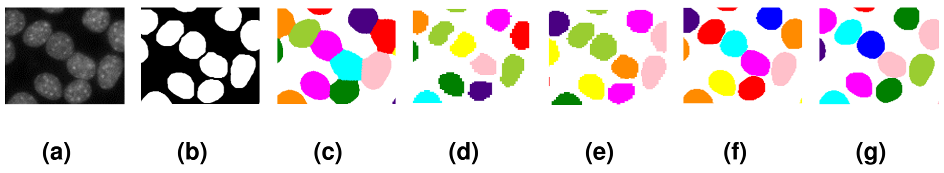


**Figure 5.6:** **a)** Example of a DAPI image patch. **b)** Corresponding ground truth patch. **c),d),e),f),g)** segmentation masks obtained for the image patch shown in a), using Yen + watershed, original U-Net, Kaggle_2018, Mask R-CNN and proposed approach, respectively.

Regarding computational efficiency training and test time were compared for all the methods. Figures 5.7(a) and 5.7(b) show the results regarding training and test time, respectively. By observing figure 5.7(a) it can be concluded that Mask R-CNN is the model that requires significantly more time to learn the task of nuclei segmentation when compared to all the other models. It takes about 1420 minutes to learn this task. On the other hand, although the proposed approach takes more time to train (450 minutes) than Kaggle_2018 (100 minutes) and Original U-Net (14 minutes), it presents better performance when compared to these methods (as shown in figure 5.5).

Regarding test time (figure 5.7(b)), the results show that Mask R-CNN is the model that takes more time to give a segmentation prediction for an image. The proposed approach in comparison with Mask R-CNN is about nine times faster. Furthermore, Yen + watershed requires 1.8 seconds, which is of the same order of magnitude as the test time of the proposed approach (1.6 seconds), however Yen + watershed presents the worst performance, as observed in figure 5.5. Finally, the approaches based on the U-Net present lower test time when compared to the proposed method, however, the proposed approach presents better performance than both U-Net based approaches (figure 5.5).

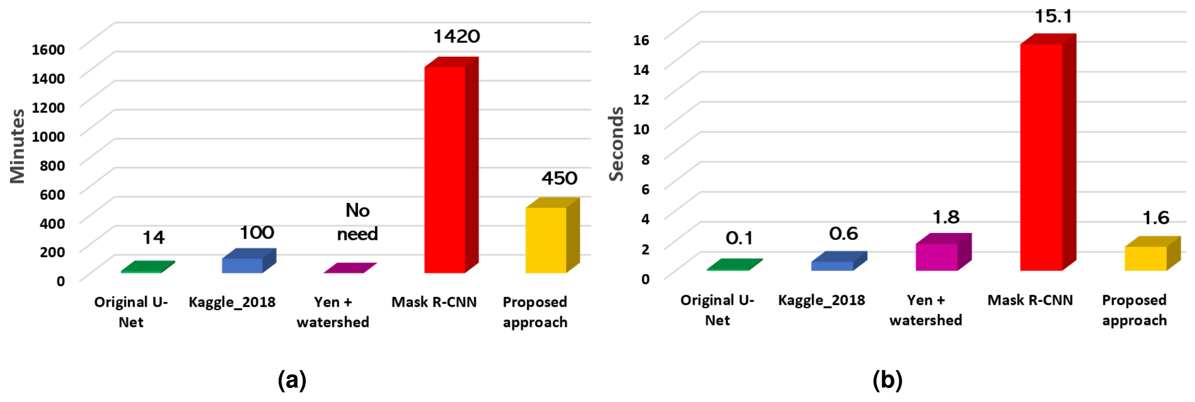


**Figure 5.7:** **a)** Training time (in minutes) associated with each model. **b)** Mean test time per image (in seconds) for each model, for images of size $1388 \times 1040$.

To sum up, by training and testing the models in DAPI images it was shown that, when compared to the state-of-the-art method Mask R-CNN, the proposed approach presents a small lost in performance, however it is much faster and computationally more efficient. After showing this, the main goal was to show that the proposed approach for cell nuclei segmentation can be applied for the segmentation of images of cells stained with other dyes. In this work, the FUCCI images were used, which are more challenging than the DAPI images. Moreover, the segmentation of FUCCI images is really important in this work since the ground truth labels to train the cell cycle staging classifiers will be generated based on the information extracted from the FUCCI images. In the next subsection, results regarding segmentation of FUCCI images are presented.

### 5.2.2 FUCCI

This subsection presents the results obtained by training and testing models with FUCCI images.

#### Implementation Details

The watershed algorithm is designed for grayscale images, however the FUCCI images are RGB. Therefore, to test the performance of Yen + watershed in the segmentation of nuclei in FUCCI images, these images were converted to grayscale. The intensity of the pixel in the grayscale image is given by: $Y = 0.2125R + 0.7154G + 0.0721B$, where $(R, G, B)$ denotes the intensity of the pixel in the RGB image [102]. For the deep learning based approaches the input images are RGB. Hence, the depth of the filters in the first layer was changed from one to three, for all models. Note that the Fast YOLO of the proposed approach already had filters with depth 3 in the first layer. However, the U-Net of the proposed approach had filters with depth 1 in the first layer, so this number was changed to 3.

Table 5.1 summarizes the training parameters of the deep learning models trained and tested for the segmentation of nuclei in FUCCI images. The only parameter that was changed in almost all models was the number of epochs. All of the other parameters remained unchanged. FUCCI images are more challenging than DAPI images, therefore the models may need more time to learn the task of nuclei segmentation in FUCCI images when compared to DAPI images.

**Table 5.1:** Training parameters of the deep learning models used for nuclei segmentation in FUCCI images.

| | Original U-Net | Kaggle_2018 | Mask R-CNN | Proposed Approach | |
|---|---|---|---|---|---|
| | | | | Fast YOLO | U-Net |
| Learning Rate | 1e-3 | 3e-4 | 1e-4 | 1e-4 + 1e-5 | 1e-3 |
| Number of epochs | 200 | 300 | 300 | 200 + 800 | 100 |

### Comparison Between Different Approaches

The proposed approach and the approaches described in subsection 4.2.4 were trained with FUCCI images and their performance was measured. The results regarding these experiments are shown in this subsection.

Figure 5.8 shows a visual comparison between different models, regarding nuclei segmentation. Again, it is clearly visible that deep learning approaches are better than the traditional method (Yen + watershed). The segmentation masks provided by Yen + watershed have a lot of false positives (noise detected as nuclei, highlighted with black circles) and false negatives (nuclei that aren't detected, highlighted with dashed black circles). Regarding the approaches based on the U-Net the results are better in comparison with the results of Yen + watershed, however in the image of the second column there are some merges (highlighted with dashed black circles). Finally, the two approaches designed for instance segmentation (Mask R-CNN and proposed approach) are the ones that provide the best segmentation masks. There aren't gaps between touching nuclei, so the nuclei boundaries are accurate. Moreover, for the images presented in figure 5.8 both instance segmentation models are able to identify each nucleus separately, that is, there aren't merges in the segmentation masks (with one exception highlighted with a dashed circle, however these nuclei belong to the image border so they will not be considered in further analysis). Nevertheless, both models present some missdetections (nuclei that aren't detected), as highlighted with black circles along with arrows, and again the proposed approach presents more missdetections than the Mask R-CNN. Despite this, both provide segmentation masks

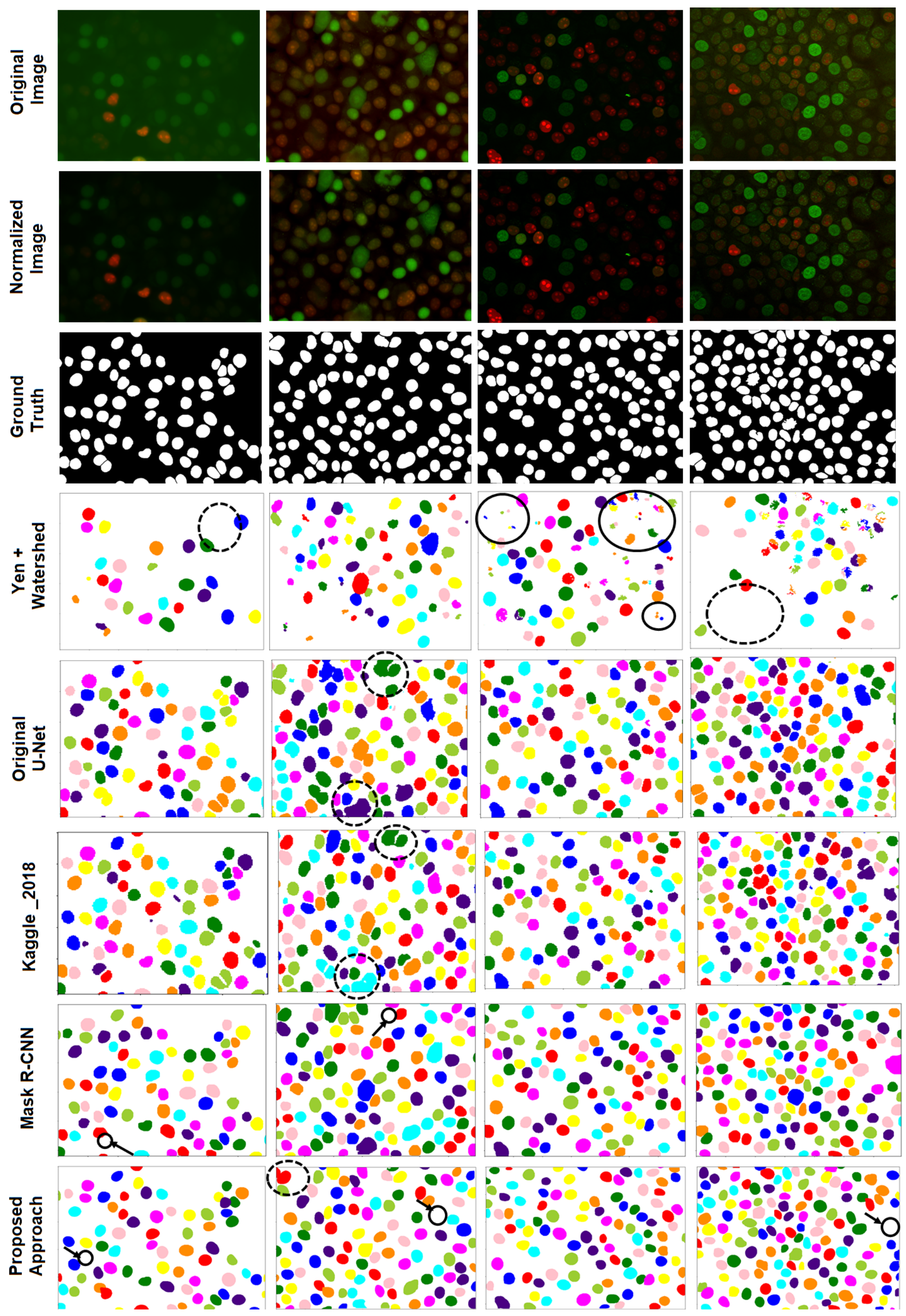


**Figure 5.8:** Nuclei segmentation results obtained by applying different methods to FUCCI images. The first row represents examples of the original images, for which the segmentation masks are to be obtained. The second row represents the corresponding normalized images, which will be the input images for the models. The third row represents the corresponding ground truth masks. Finally, the fourth, fifth, sixth, seventh and eighth rows represent the corresponding segmentation results obtained by applying Yen + watershed, Original U-Net, Kaggle_2018, Mask R-CNN and the proposed approach, respectively. Some errors are highlighted with black circles and arrows.

that are better than those provided by the other methods.

Figure 5.9 shows the F1 Score at different thresholds of the IoU. It can be concluded again that deep learning based approaches perform better than the traditional method. Additionally, the best methods for nuclei segmentation are the Mask R-CNN and the proposed approach, both present a good F1 score (above 0.8) and similar performance for thresholds below 0.75.

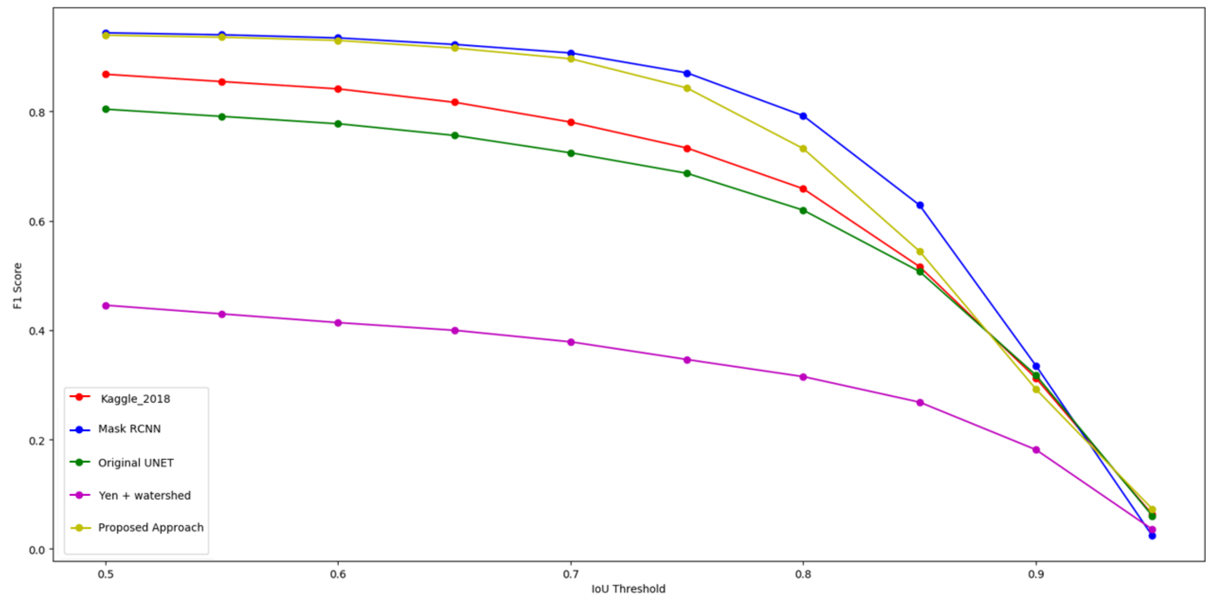


**Figure 5.9:** Average F1 Score vs IoU threshold, comparison between different models: Yen + watershed (purple), original U-Net (green), Kaggle_2018 (red), Mask R-CNN (blue) and proposed approach (yellow).

The results in figure 5.9 are for the models that were trained and tested with the FUCCI images and with the ground truth masks generated from the DAPI images. Although the nuclei positions and shapes are the same in FUCCI and DAPI images, there are some nuclei in FUCCI images that are colorless (those in G0 and early G1 phases [12]), that is, their green and red intensities are very low. Therefore, the segmentation models may not detect these nuclei. This can be one reason that explains why the results in figure 5.9 aren't as good as the ones shown in figure 5.5.

Moreover, the shape of these curves is a little bit different from the shape of the curves shown in figure 5.5. Figure 5.10 shows some patches of the FUCCI images, corresponding DAPI patches, corresponding ground truth patches and the segmentation masks obtained with the proposed approach for the FUCCI patches. By showing these patches the goal is to exemplify one reason that can explain the decrease of F1 score in FUCCI images. As indicated by arrows in figure 5.10, some green stained nuclei in the FUCCI images appear bigger than the nuclei in the DAPI images. Specially, those in mitosis (last two columns in figure 5.10). Moreover, it is observed that the predictions are according to the FUCCI images, but the ground truth masks are according to the DAPI images. For low IoU thresholds these situations are considered as true positives, however for high IoU thresholds they are considered as false positives and the F1 score will decrease. Additionally, note that these nuclei were removed from further analysis as they were considered as being outliers (nuclei with area $> 9000$). Hence, the results of the cell cycle are not influenced by these situations.

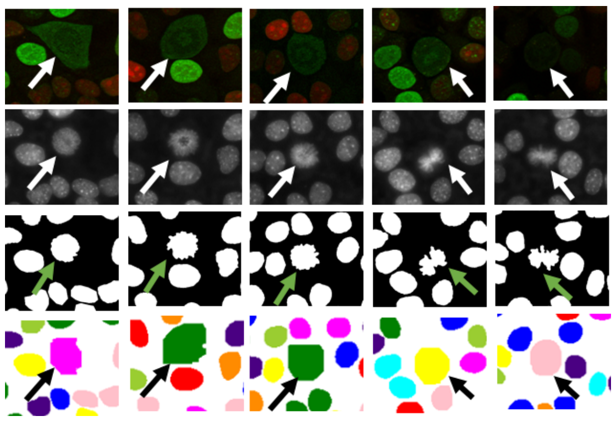


**Figure 5.10:** The first, second and third rows show examples of FUCCI patches, corresponding DAPI patches and corresponding ground truth patches, respectively. The last row shows the predictions made by the proposed approach for the FUCCI patches shown in the first column.

### 5.2.3 Conclusion

To sum up, the two approaches that present the best results in both DAPI and FUCCI images, are the Mask R-CNN and the proposed approach. Although the Mask R-CNN is slightly better than the proposed approach in terms of segmentation performance, it was shown that the proposed approach is computationally much less demanding when compared to Mask R-CNN. Furthermore, it was shown that the proposed approach is able to segment nuclei in images of cells stained with different dyes, in particular, cells stained with DAPI and cells stained with FUCCI.

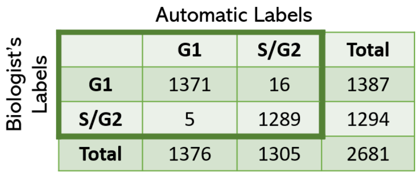
## 5.3 Cell Cycle Staging

In this section results regarding cell cycle staging are presented.

### 5.3.1 Comparison Between Automatic Labels and the Labels Given by the Biologist

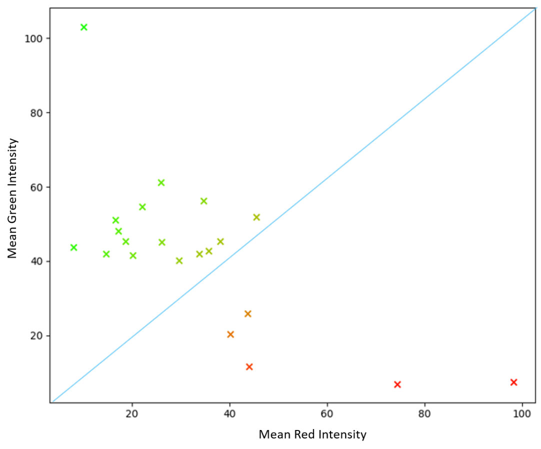
In this work, a total of 3553 nuclei were automatically labeled, where approximately 64 % were labeled as G1 and 36 % as S/G2. The labels generated in this work were compared with the labels given by the biologist (in [1]). In this comparison only the nuclei that were labeled by both were considered, that is, 2681 nuclei. It was observed that only 21 nuclei were labeled differently by both methods. The confusion matrix represented in figure 5.11(a) summarizes the comparison: 5 nuclei labeled as G1 with algorithm 4.1 were labeled by the biologist as S/G2, and 16 nuclei labeled as S/G2 with algorithm

4.1 were labeled by the biologist as G1. Figure 5.11(b) represents each of those 21 nuclei that were labeled differently by both methods. The blue line has the following equation: $Mean\ Green\ Intensity = Mean\ Red\ Intensity$. In figure 5.11(b), the 16 nuclei above the blue line were labeled by the biologist as G1, and the 5 nuclei below this line were labeled by the biologist as S/G2, which shows some subjectiveness associated with the biologist's labels.



| Biologist's Labels | Automatic Labels: G1 | Automatic Labels: S/G2 | Total |
|---|---|---|---|
| G1 | 1371 | 16 | 1387 |
| S/G2 | 5 | 1289 | 1294 |
| Total | 1376 | 1305 | 2681 |

(a)



(b)

**Figure 5.11:** Comparison with the labels given by the biologist. **a)** Confusion matrix between the biologist's labels and the labels generated according to algorithm 4.1. **b)** Representation of each nucleus, labeled differently by both methods, in the 2D space (Mean Red Intensity, Mean Green Intensity). Each point is colored with the following intensities (R, G, B) = (Normalized Red Intensity, Normalized Green Intensity, 0). The blue line has the following equation Mean Green Intensity = Mean Red Intensity.

To sum up, in this work an automatic method for nuclei labeling based on the FUCCI system was proposed in order to create the ground truth labels for the nuclei. This method can reduce the biologist's workload and provide more objective labels. In fact, it was verified that for 2660 nuclei the labels obtained with the automatic method and the ones given by the biologist coincided. This shows that the automatic method can efficiently and accurately replace the intensive work of the biologist.

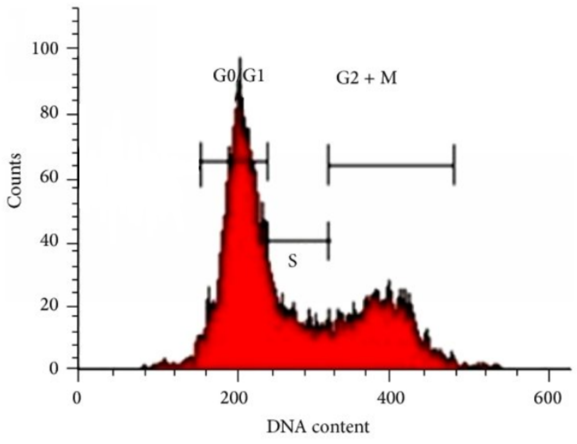
Finally, note that the nuclei that were considered in all the further experiments are the ones that were labeled in this work, that is, 3553 nuclei.

### 5.3.2 Cell Cycle Profiles Using the Labels Based on FUCCI Technology

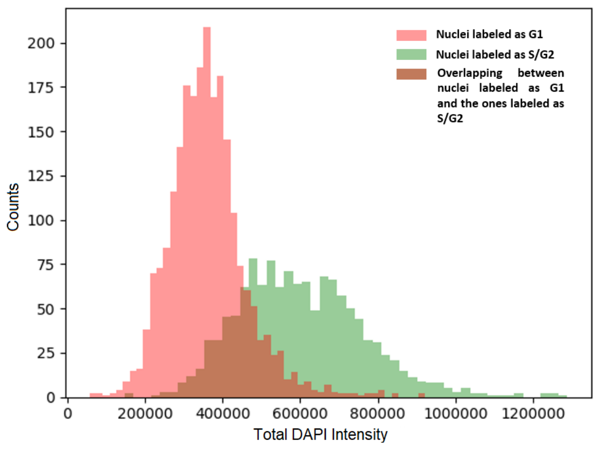
There is a correlation between the DNA content of a population of cells and the cell cycle phase, this is commonly represented as a DNA histogram [12], (figure 5.12(a)). In this histogram a typical distribution of cells within the cell cycle phases is observed. That is, there are more nuclei in phase G1, and the nuclei in phases S/G2 have higher DNA content than those in G1 phase.

As stated before, the method presented in [73] is based on thresholds applied to histograms of the

DAPI intensity. DAPI intensity histogram and DNA content histogram are equivalent, because DAPI reflects the DNA content of the nuclei. Therefore, in order to study if the labeled nuclei follow this distribution, they were represented in a histogram of total DAPI intensity (figure 5.12(b)). It was shown that the histogram obtained from the distribution of the labeled nuclei within the considered phases (figure 5.12(b)) is consistent with the typical distribution of cells in a regular DNA histogram (figure 5.12(a)).



**(a)**



**(b)**

**Figure 5.12: a)** Typical histogram of the DNA content. Image retrieved from [103]. **b)** Histogram of the total DAPI intensity, for the labeled nuclei. The red and green bars indicate counts regarding nuclei that were labeled as G1 and S/G2, respectively. The brown area shows the overlapping between nuclei labeled as G1 and the ones labeled as S/G2.

## 5.3.3 Comparison Between FUCCI Labels and DAPI Features

In [1] the authors have demonstrated the relation between DAPI features and FUCCI labels. They have shown that nuclei in phase G2 are bigger and present higher DAPI intensity when compared to nuclei in phase G1. In order to study the relation between the information provided by the FUCCI and the DAPI, the 3553 labeled nuclei were represented in the 2D space (Area, Total DAPI Intensity), as depicted in figure 5.13(a). By observing this plot it can be concluded that the green nuclei present higher area and DAPI intensity than the red nuclei. Additionally, the 3553 nuclei were also represented in the 2D space (Normalized Area, Normalized DAPI intensity), see figure 5.13(b). The normalized area and normalized DAPI intensity have been defined in equations 4.15 and 4.16, respectively. By comparing the plots in figures 5.13(a) and 5.13(b) it can be concluded that the normalized area and intensity provide a plot where it is easier to visualize two 'separable' clouds. In figure 5.13(b) it is clearly visible that green nuclei are brighter and bigger than red nuclei. In fact, the increase in intensity is greater than the increase in area. The DNA content doubles in nuclei in S/G2 phases when compared to nuclei in G1 phase, this explains the increase in DAPI intensity, because DAPI binds to the DNA. The area also increases, but not as much as the intensity.

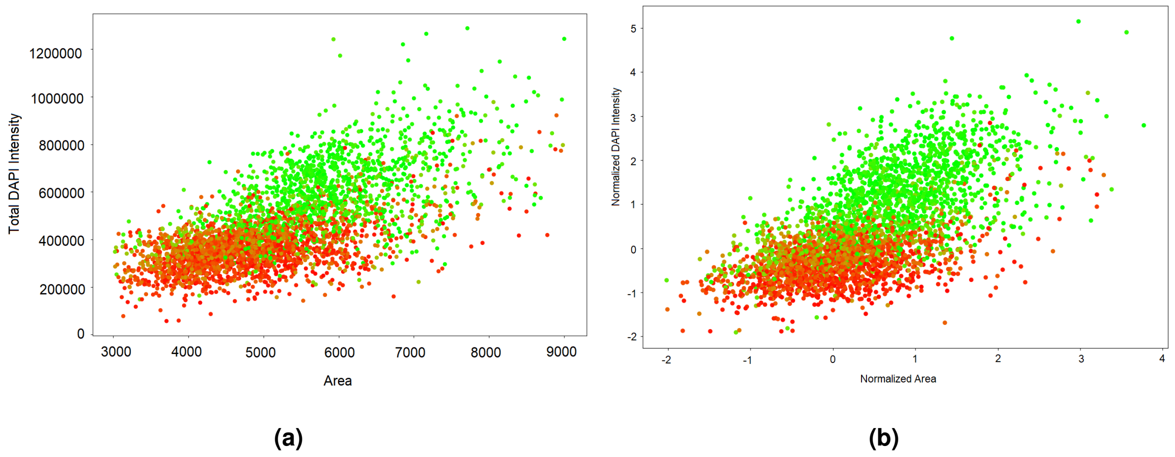


**Figure 5.13:** Relation between FUCCI and DAPI features. In both plots each point is colored with the following intensities (R, G, B) = (Normalized Red Intensity, Normalized Green Intensity, 0). **a)** Representation of each nucleus in the 2D space (Area, Total DAPI intensity). **b)** Representation of each nucleus in the 2D space (Normalized Area, Normalized DAPI intensity).

Furthermore, these plots show that the information regarding the cell cycle obtained from the FUCCI images can be obtained from the DAPI images as well, that is, the nucleus area and DAPI intensity provide useful information regarding the cell cycle. Therefore, the SVM was trained and tested with DAPI features as input and the ground truth labels generated from the FUCCI images. Results regarding the classification using SVM are presented in the next subsection.

### 5.3.4 Nuclei Classification

#### Five-Fold Cross-Validation

The evaluation metrics (precision, recall and F1 score) used to measure the performance of the models for cell cycle staging are explained in detail in appendix C (section C.2).

Tables 5.2 and 5.3 show the classification results obtained with SVM_A and SVM_B, respectively. In each table, the results correspond to the mean and standard deviation over the five models obtained by performing five-fold cross-validation. For both experiments, the steps followed to perform hyperparameter optimization are described in detail in section D.2 of appendix D.

By comparing the two approaches based on the SVM it is clearly observed that the one that presents better results is the SVM_B. This means that the normalization of the area and intensity, per image, helps in the separation between classes G1 and S/G2. In fact, as observed in figure 5.13(b), in the normalized feature space the green and red points are more separable. These results show that what matters for cell cycle staging is the relative area and DAPI intensity of the nuclei.

**Table 5.2:** Classification results for SVM_A.

| | Precision | Recall | F1-Score |
|---|---|---|---|
| G1 | $0.698 \pm 0.019$ | $0.900 \pm 0.020$ | $0.786 \pm 0.012$ |
| S/G2 | $0.622 \pm 0.061$ | $0.294 \pm 0.018$ | $0.399 \pm 0.024$ |
| Average | $0.660 \pm 0.032$ | $0.597 \pm 0.014$ | $0.592 \pm 0.013$ |

**Table 5.3:** Classification results for SVM_B.

| | Precision | Recall | F1-Score |
|---|---|---|---|
| G1 | $0.904 \pm 0.021$ | $0.926 \pm 0.015$ | $0.915 \pm 0.006$ |
| S/G2 | $0.858 \pm 0.025$ | $0.822 \pm 0.040$ | $0.839 \pm 0.019$ |
| Average | $0.881 \pm 0.016$ | $0.874 \pm 0.022$ | $0.877 \pm 0.010$ |

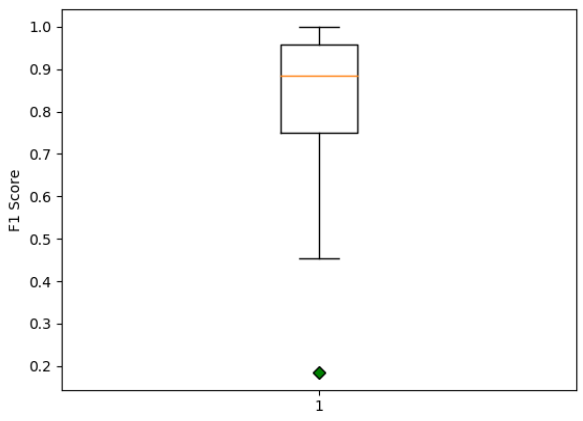
Ferro et al. [1] applied the modified k-means clustering algorithm per image, whereas in this work the features from all nuclei from all images were used to train the SVM_A and SVM_B. In [1] a non-normalized input space (area, total intensity) was used, and since there may be variations in the acquisition parameters between different images, the k-means algorithm was applied per image. In this work, on one hand, the SVM_A was trained in the non-normalized input space (area, total intensity) with features from different images and it was verified that the results aren't that good (F1 scores $0.786 \pm 0.012$ and $0.399 \pm 0.024$ for class G1 and class S/G2, respectively). These results are expectable because, for instance, the intensities between different images may not be comparable. On the other hand, the SVM_B was trained and tested in the normalized input space with features from different images and provided good results (F1 scores $0.915 \pm 0.006$ and $0.839 \pm 0.019$ for class G1 and class S/G2, respectively). These results are better than those obtained with SVM_A, because the F1 score values obtained are higher and the standard deviation across the five folds is lower, that is, the diversity in different folds is lower. This is due to the fact that SVM_B takes into account the relative area and intensity for each nucleus.

To sum up, the SVM_B presented in this work for cell cycle staging is better than the approach presented in [1]. While the method proposed in [1] needs to be applied image by image, which may be computationally expensive, the proposed SVM_B can take as input nuclei features from all images to perform cell cycle staging.

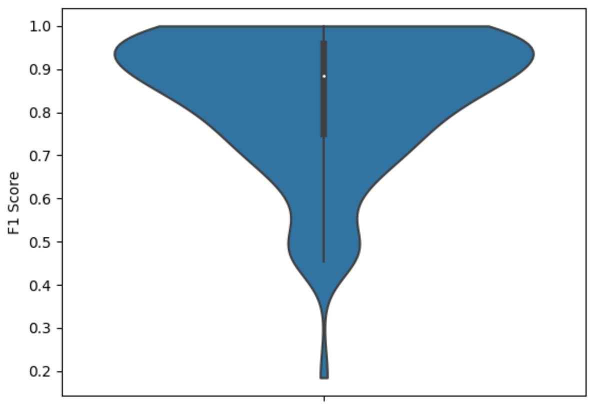
### Leave-One-Experiment-Out Cross-Validation

Results of the leave-one-experiment-out cross-validation performed with the SVM_B are presented here. It was observed that in 21 of 130 images the classifier assigned the correct class to all nuclei belonging to those images (F1 score of 100 %). Furthermore, as observed in figures 5.14(a) and 5.14(b) the distribution of the results isn't symmetric, in fact in 91 of 130 images the F1 score is higher or equal than 80 %, which is good. Moreover, as observed in figure 5.14(b) most of the values are clustered around the maximum, that is, the peak with the highest amplitude is located at a F1 score of approximately 93 %. These results show that for the majority of the images the classifier provides good classification results.

In this work it was assumed that there can be variations in the acquisition parameters between images of the same experiment, therefore the area and intensity normalization was done per image. With this normalization step the goal is to obtain the relative area and intensity between nuclei in stage G1 and nuclei in phases S/G2. However, for example, sometimes an image has all nuclei belonging to stage G1, in this case the normalization step will provide the relative area and intensity of these



(a)



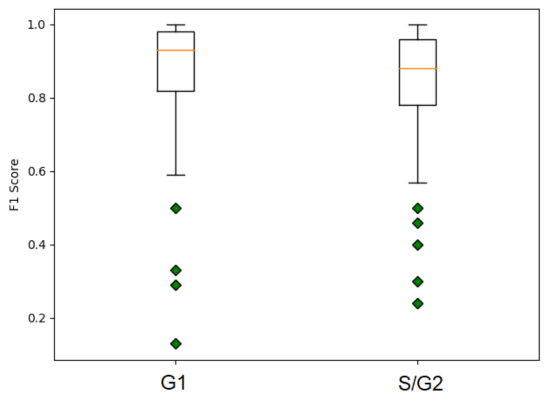
(b)

**Figure 5.14: a)** Box plot of F1 score of 130 images. **b)** Violin plot of F1 score of 130 images.

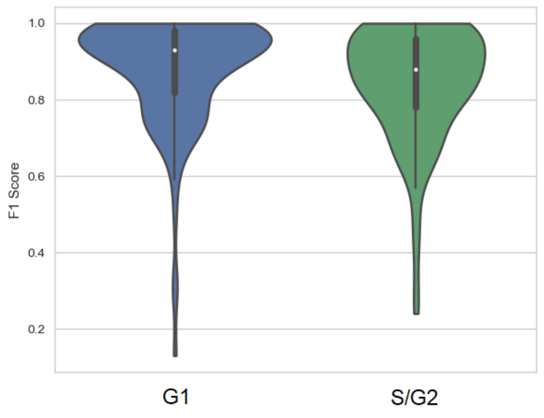
nuclei in stage G1. There is no information regarding nuclei in stages S/G2 in these images. Note that, although less likely, the opposite can also happen, that is, an image that only has nuclei in stages S/G2. The images which only had nuclei belonging to one of the classes were identified, and it turned out that these are the images for which the values of F1 score are lower than 50 % (which are few as observed in figure 5.14(b)). Moreover, in figure 5.14(a) one can observe the presence of an outlier, this corresponds to an image that contains 1 nucleus in G1 and 15 nuclei in S/G2. An alternative to avoid these situations is to do the normalization per experiment instead of doing it per image, in this case it would be guaranteed that there are enough nuclei in G1 and in S/G2. However, this normalization may be negatively influenced due to acquisition differences between the images.

In figure 5.15 the box and violin plots of F1 score for class G1 and for class S/G2 are shown. The results obtained for class G1 are better than those obtained for class S/G2. This is due to the fact that the dataset is imbalanced. The outliers in figure 5.15(a) correspond again to images that have all nuclei or the majority of the nuclei belonging to one of the two possible classes. By observing the violin plots in figure 5.15(b) it can be concluded that for about 75 % of the images the F1 scores for class G1 are higher than 80 %. Additionally, for about 75 % of the images the F1 scores for class S/G2 are higher than 78 %. For both classes the maximum F1 score that is obtained is 100 %.

To sum up, it can be concluded that the method proposed in this work for cell cycle staging achieves good classification results. Moreover, the validation method used in this work is better than the one used by Roukos et al. [73]. Their validation step consisted in the comparison of the cell cycle distributions obtained with the proposed method with the ones obtained with flow cytometry. In this work, a nucleus by nucleus validation was performed, that is, for each nucleus the cell cycle phase predicted with the proposed method was compared with the ground truth label of that nucleus.



(a)



(b)

**Figure 5.15: a)** Box plot of F1 Score for class G1 (on the left) and class S/G2 (on the right). **b)** Violin plot of F1 Score for class G1 (on the left) and for class S/G2 (on the right).

### 5.3.5 Testing on a Different Cell Line

Finally, the goal was to study the performance of the classifiers (SVM_A and SVM_B) on another cell line. Therefore, the classifiers trained on images of normal murine mammary gland cells were used to classify nuclei in images of gastric cancer cells stained with DAPI and their performance was measured. The classification results are shown in table 5.4. SVM_A classified correctly 12 nuclei (16 % recall), and SVM_B classified correctly 68 nuclei (93 % recall). Once again, the importance of the normalized feature space is shown. It can be concluded that SVM_B presents a good generalization capacity due to the feature normalization step. This shows that it is always important to take into account the information regarding the population of cells. As explained before, images acquired in different experiments are acquired under different conditions. Moreover, the typical size of the cells in different cell lines is different. Therefore, one cannot rely on area and total intensity to build a cell cycle staging classifier with high generalization capability. In fact, it is important to take into account the information from the population of cells and calculate the normalized/relative areas and intensities.

**Table 5.4:** Classification results for SVM_A and SVM_B in a different cell line.

| | **TP** | **FN** | **Recall** |
|---|---|---|---|
| **SVM_A** | 12 | 61 | 16 % |
| **SVM_B** | 68 | 5 | 93 % |

To sum up, these results show that SVM_B presents good results in this new cell line, emphasizing once again the importance of area and intensity normalization. Therefore, it is concluded that although SVM_B was trained with features extracted from images of normal murine mammary gland cells stained with DAPI, it can be used for cell cycle staging of cells belonging to other cell lines (in this case gastric cancer cells).

# 6

# Conclusions and Future Work

Contents

In this section conclusions and topics requiring future studies are presented.

## 6.1 Conclusions

This thesis had two main goals: development of a deep learning based approach for cell nuclei instance segmentation and development of an automatic tool for cell cycle staging from microscopic images of cells stained with DAPI. Both objectives have been accomplished.

In this work the important problem of cell nuclei segmentation for high-throughput applications was tackled. A new approach based on deep learning was presented for cell nuclei instance segmentation. This approach combines the object detection capability of Fast YOLO with the segmentation ability of the U-Net. The segmentation quality obtained with the proposed method is comparable to the existing deep learning based state-of-the-art methods, e.g. Mask R-CNN, but a significant reduction of about 9 times on the segmentation time was obtained.

Furthermore, in this work a new approach for interphase cell cycle staging from images of cells stained with DAPI was presented. This technique is based on a supervised machine learning method, the SVM, trained in a normalized input space (normalized area, normalized DAPI intensity). The presented technique depends on a widely used fluorescent dye (DAPI), thus it is a broadly accessible tool for cell cycle staging. Furthermore, the proposed approach can be applied for the cell cycle staging of adherent cells without the need of destroying their natural architecture. This is an advantage when compared to the IFC where the cells need to be in suspension. Moreover, the presented approach presents good results and a good generalization capability. In fact, it was shown that the trained SVM can be applied for cell cycle staging of cells belonging to other cell lines, different from the one that was used for training.

## 6.2 Future Work

The segmentation method proposed in this thesis was already presented (oral presentation) and published in the proceedings of the international conference IbPRIA 2019 [104]. A paper describing the cell cycle staging algorithm is under preparation to be submitted to a high impact international journal.

The methods proposed in this work will be of great importance for the research line focused on the diagnosis of hereditary gastric cancer from fluorescence microscopy images. For instance, the tools presented in this work will be used to study cells with mutations in proteins associated with cancer. First, the proposed approach for nuclei segmentation will be applied to segment nuclei of these cells. Afterwards, the SVM trained in the normalized input space will be used for cell cycle staging of these cells. In this way, a comparison between cells with wild-type proteins and cells with mutations in proteins associated with cancer can be made, which will allow to understand if there is dysregulation of the cell cycle in cancerous cells. Furthermore, in the future, morphological and textural features will be extracted from each segmented nucleus to correlate them with mutations in proteins related with cancer.

# Bibliography

[1] A. Ferro, T. Mestre, P. Carneiro, I. Sahumbaiev, R. Seruca, and J. M. Sanches, “Blue intensity matters for cell cycle profiling in fluorescence DAPI-stained images,” *Laboratory Investigation*, vol. 97, no. 5, p. 615, 2017.

[2] H. Irshad, A. Veillard, L. Roux, and D. Racoceanu, “Methods for nuclei detection, segmentation, and classification in digital histopathology: a review—current status and future potential,” *IEEE reviews in biomedical engineering*, vol. 7, pp. 97–114, 2014.

[3] F. Xing and L. Yang, “Robust nucleus/cell detection and segmentation in digital pathology and microscopy images: a comprehensive review,” *IEEE reviews in biomedical engineering*, vol. 9, pp. 234–263, 2016.

[4] T. Mestre, J. Figueiredo, A. S. Ribeiro, J. Paredes, R. Seruca, and J. M. Sanches, “Quantification of topological features in cell meshes to explore e-cadherin dysfunction,” *Scientific reports*, vol. 6, p. 25101, 2016.

[5] B. S. Deshmukh and V. H. Mankar, “Segmentation of microscopic images: a survey,” in *Electronic Systems, Signal Processing and Computing Technologies (ICESC), 2014 International Conference on*. IEEE, 2014, pp. 362–364.

[6] M. Dabass, R. Vig, and S. Vashisth, “Review of histopathological image segmentation via current deep learning approaches,” in *2018 4th International Conference on Computing Communication and Automation (ICCCA)*. IEEE, 2018, pp. 1–6.

[7] D. Shen, G. Wu, and H.-I. Suk, “Deep learning in medical image analysis,” *Annual review of biomedical engineering*, vol. 19, pp. 221–248, 2017.

[8] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

[9] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[10] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep learning for computer vision: A brief review,” *Computational intelligence and neuroscience*, vol. 2018, 2018.

[11] A. Janowczyk and A. Madabhushi, “Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases,” *Journal of pathology informatics*, vol. 7, 2016.

[12] R. Seruca, J. S. Suri, and J. M. Sanches, *Fluorescence imaging and biological quantification*. CRC Press, 2017.

[13] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[14] W. Rawat and Z. Wang, “Deep convolutional neural networks for image classification: A comprehensive review,” *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[15] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, “A guide to convolutional neural networks for computer vision,” *Synthesis Lectures on Computer Vision*, vol. 8, no. 1, pp. 1–207, 2018.

[16] L. Lundy-Ekman, *Neuroscience-E-Book: Fundamentals for Rehabilitation*. Elsevier Health Sciences, 2013.

[17] F. Fei, J. Johnson, and S. Yeung. (2017) Stanford University CS231 Slides. Accessed 01-02-2019. [Online]. Available: http://cs231n.stanford.edu/slides/2017/

[18] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[19] E. Anderson, “The species problem in Iris,” *Annals of the Missouri Botanical Garden*, vol. 23, no. 3, pp. 457–509, 1936.

[20] J. Blumberg and G. Kreiman, “How cortical neurons help us see: visual recognition in the human brain,” *The Journal of clinical investigation*, vol. 120, no. 9, pp. 3054–3063, 2010.

[21] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *arXiv preprint arXiv:1603.07285*, 2016.

[22] F. Chollet, *Deep learning with python*. Manning Publications Co., 2017.

[23] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation functions: Comparison of trends in practice and research for deep learning,” *arXiv preprint arXiv:1811.03378*, 2018.

[24] D. A. Van Valen, T. Kudo, K. M. Lane, D. N. Macklin, N. T. Quach, M. M. DeFelice, I. Maayan, Y. Tanouchi, E. A. Ashley, and M. W. Covert, “Deep learning automates the quantitative analysis

of individual cells in live-cell imaging experiments," *PLoS computational biology*, vol. 12, no. 11, p. e1005177, 2016.

[25] A. Deshpande. (2016) A Beginner's Guide To Understanding Convolutional Neural Networks. Accessed 09-04-2019. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[26] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[27] Y. LeCun, "The MNIST database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[28] T. Hastie, R. Tibshirani, and J. Friedman, "The elements of statistical learning: data mining, inference, and prediction, springer series in statistics," 2009.

[29] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A review on deep learning techniques applied to semantic segmentation," *arXiv preprint arXiv:1704.06857*, 2017.

[30] F. Xing, Y. Xie, and L. Yang, "An automatic learning-based framework for robust nucleus segmentation," *IEEE transactions on medical imaging*, vol. 35, no. 2, pp. 550–566, 2016.

[31] Y. Liu, P. Zhang, Q. Song, A. Li, P. Zhang, and Z. Gui, "Automatic segmentation of cervical nuclei based on deep learning and a conditional random field," *IEEE Access*, vol. 6, pp. 53 709–53 721, 2018.

[32] Y. Cui, G. Zhang, Z. Liu, Z. Xiong, and J. Hu, "A deep learning algorithm for one-step contour aware nuclei segmentation of histopathological images," *arXiv preprint arXiv:1803.02786*, 2018.

[33] H. Höfener, A. Homeyer, N. Weiss, J. Molin, C. F. Lundström, and H. K. Hahn, "Deep learning nuclei detection: A simple approach can deliver state-of-the-art results," *Computerized Medical Imaging and Graphics*, vol. 70, pp. 43–52, 2018.

[34] M. Khoshdeli and B. Parvin, "Deep leaning models delineates multiple nuclear phenotypes in h&e stained histology sections," *arXiv preprint arXiv:1802.04427*, 2018.

[35] Y.-l. Kuo and C.-c. Ko, "Cell nuclei segmentation from stain images prior to intraductal breast lesion assessment," in *International Conference on Artifical Intelligence and Software Engineering*, 2014, pp. 266–270.

[36] M. Veta, A. Huisman, M. A. Viergever, P. J. van Diest, and J. P. Pluim, "Marker-controlled watershed segmentation of nuclei in H&E stained breast cancer biopsy images," in *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. IEEE, 2011, pp. 618–621.

[37] Y. Song, L. Zhang, S. Chen, D. Ni, B. Li, Y. Zhou, B. Lei, and T. Wang, “A deep learning based framework for accurate segmentation of cervical cytoplasm and nuclei,” in *Engineering in Medicine and Biology Society (EMBC), 2014 36th annual international conference of the IEEE*. IEEE, 2014, pp. 2903–2906.

[38] N. Kumar, R. Verma, S. Sharma, S. Bhargava, A. Vahadane, and A. Sethi, “A dataset and a technique for generalized nuclear segmentation for computational pathology,” *IEEE transactions on medical imaging*, vol. 36, no. 7, pp. 1550–1560, 2017.

[39] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[40] P. Naylor, M. Lae, F. Reyal, and T. Walter, “Nuclei segmentation in histopathology images using deep neural networks,” in *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*. IEEE, 2017, pp. 933–936.

[41] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[42] V. Patel and K. Mistree, “A review on different image interpolation techniques for image enhancement,” *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 12, pp. 129–133, 2013.

[43] H. Noh, S. Hong, and B. Han, “Learning deconvolution network for semantic segmentation,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1520–1528.

[44] J. C. Caicedo, J. Roth, A. Goodman, T. Becker, K. W. Karhohs, M. Broisin, M. Csaba, C. McQuin, S. Singh, F. Theis *et al.*, “Evaluation of deep learning strategies for nucleus segmentation in fluorescence images,” *BioRxiv*, p. 335216, 2019.

[45] V. Iglovikov, S. Seferbekov, A. Buslaev, and A. Shvets, “Ternausnetv2: Fully convolutional network for instance segmentation,” *arXiv preprint arXiv:1806.00844*, 2018.

[46] “Find the nuclei in divergent images to advance medical discovery,” https://www.kaggle.com/c/data-science-bowl-2018/discussion, accessed: 2019-02-20.

[47] “[ods.ai] topcoders, 1st place solution on data science bowl 2018,” https://www.kaggle.com/c/data-science-bowl-2018/discussion/54741#latest-477226, accessed: 2019-02-20.

[48] F. A. Guerrero-Pena, P. D. M. Fernandez, T. I. Ren, M. Yui, E. Rothenberg, and A. Cunha, "Multi-class weighted loss for instance segmentation of cluttered cells," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 2451–2455.

[49] Z. Zeng, W. Xie, Y. Zhang, and Y. Lu, "RIC-Unet: An improved neural network based on unet for nuclei segmentation in histology images," *IEEE Access*, 2019.

[50] Z. Xu, F. Sobhani, C. F. Moro, and Q. Zhang, "US-net for robust and efficient nuclei instance segmentation," *arXiv preprint arXiv:1902.00125*, 2019.

[51] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[52] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[53] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[54] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[55] K. He, G. Gkioxari, P. Dollár, and R. Girshick. (2017) Mask R-CNN (ICCV presentation). Accessed 09-04-2019. [Online]. Available: https://www.slideshare.net/windmdk/mask-rcnn

[56] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[57] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, 2019.

[58] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[59] J. Dai, K. He, and J. Sun, "Instance-aware semantic segmentation via multi-task network cascades," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3150–3158.

[60] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, “Fully convolutional instance-aware semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2359–2367.

[61] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[62] J. W. Johnson, “Adapting Mask-RCNN for automatic nucleus segmentation,” *arXiv preprint arXiv:1805.00500*, 2018.

[63] A. O. Vuola, S. U. Akram, and J. Kannala, “Mask-RCNN and U-net ensembled for nuclei segmentation,” *arXiv preprint arXiv:1901.10170*, 2019.

[64] H. Lodish, J. E. Darnell, A. Berk, C. A. Kaiser, M. Krieger, M. P. Scott, A. Bretscher, H. Ploegh, P. Matsudaira *et al.*, *Molecular cell biology*. Macmillan, 2008.

[65] G. M. Cooper, R. E. Hausman, and R. E. Hausman, *The cell: a molecular approach*. ASM press Washington, DC, 2000, vol. 10.

[66] M. J. Sanderson, I. Smith, I. Parker, and M. D. Bootman, “Fluorescence microscopy,” *Cold Spring Harbor Protocols*, vol. 2014, no. 10, pp. pdb–top071 795, 2014.

[67] M. R. Beccia, T. Biver, A. Pardini, J. Spinelli, F. Secco, M. Venturini, N. Busto Vázquez, M. P. Lopez Cornejo, V. I. Martin Herrera, and R. Prado Gotor, “The fluorophore 4′, 6-diamidino-2-phenylindole (DAPI) induces DNA folding in long double-stranded DNA,” *Chemistry–An Asian Journal*, vol. 7, no. 8, pp. 1803–1810, 2012.

[68] M. Jež, T. Bas, M. Veber, A. Košir, T. Dominko, R. Page, and P. Rožman, “The hazards of DAPI photoconversion: effects of dye, mounting media and fixative, and how to minimize the problem,” *Histochemistry and cell biology*, vol. 139, no. 1, pp. 195–204, 2013.

[69] F. Otto, “DAPI staining of fixed cells for high-resolution flow cytometry of nuclear DNA,” in *Methods in cell biology*. Elsevier, 1990, vol. 33, pp. 105–110.

[70] A. Sakaue-Sawano, H. Kurokawa, T. Morimura, A. Hanyu, H. Hama, H. Osawa, S. Kashiwagi, K. Fukami, T. Miyata, H. Miyoshi *et al.*, “Visualizing spatiotemporal dynamics of multicellular cell-cycle progression,” *Cell*, vol. 132, no. 3, pp. 487–498, 2008.

[71] T. Saitou and T. Imamura, “Quantitative imaging with FUCCI and mathematics to uncover temporal dynamics of cell cycle progression,” *Development, growth & differentiation*, vol. 58, no. 1, pp. 6–15, 2016.

[72] N. Zielke and B. Edgar, “FUCCI sensors: powerful new tools for analysis of cell proliferation,” *Wiley Interdisciplinary Reviews: Developmental Biology*, vol. 4, no. 5, pp. 469–487, 2015.

[73] V. Roukos, G. Pegoraro, T. C. Voss, and T. Misteli, “Cell cycle staging of individual cells by fluorescence microscopy,” *Nature protocols*, vol. 10, no. 2, p. 334, 2015.

[74] R. Nunez, “DNA measurement and cell cycle analysis by flow cytometry,” *Current issues in molecular biology*, vol. 3, pp. 67–70, 2001.

[75] X. Chen, X. Zhou, and S. T. Wong, “Automated segmentation, classification, and tracking of cancer cell nuclei in time-lapse microscopy,” *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 4, pp. 762–766, 2006.

[76] J. Yan, X. Zhou, Q. Yang, N. Liu, Q. Cheng, and S. T. Wong, “An effective system for optical microscopy cell image segmentation, tracking and cell phase identification,” in *2006 International Conference on Image Processing*. IEEE, 2006, pp. 1917–1920.

[77] M. Wang, X. Zhou, F. Li, J. Huckins, R. W. King, and S. T. Wong, “Novel cell segmentation and online SVM for cell cycle phase identification in automated microscopy,” *Bioinformatics*, vol. 24, no. 1, pp. 94–101, 2007.

[78] T. Blasi, H. Hennig, H. D. Summers, F. J. Theis, J. Cerveira, J. O. Patterson, D. Davies, A. Filby, A. E. Carpenter, and P. Rees, “Label-free cell cycle analysis for high-throughput imaging flow cytometry,” *Nature communications*, vol. 7, p. 10256, 2016.

[79] P. Eulenberg, N. Köhler, T. Blasi, A. Filby, A. E. Carpenter, P. Rees, F. J. Theis, and F. A. Wolf, “Reconstructing cell cycle and disease progression using deep learning,” *Nature communications*, vol. 8, no. 1, p. 463, 2017.

[80] D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, “Mitosis detection in breast cancer histology images with deep neural networks,” in *International Conference on Medical Image Computing and Computer-assisted Intervention*. Springer, 2013, pp. 411–418.

[81] M. Saha, C. Chakraborty, and D. Racoceanu, “Efficient deep learning model for mitosis detection using breast histopathology images,” *Computerized Medical Imaging and Graphics*, vol. 64, pp. 29–40, 2018.

[82] C. Li, X. Wang, W. Liu, and L. J. Latecki, “Deepmitosis: Mitosis detection via deep detection, verification and segmentation networks,” *Medical image analysis*, vol. 45, pp. 121–133, 2018.

[83] Y. Mao, L. Han, and Z. Yin, “Cell mitosis event analysis in phase contrast microscopy images using deep learning,” *Medical Image Analysis*, 2019.

[84] W. Godfrey, D. Hill, J. Kilgore, G. Buller, J. Bradford, D. Gray, I. Clements, K. Oakleaf, J. Salisbury, M. Ignatius *et al.*, "Complementarity of flow cytometry and fluorescence microscopy," *Microscopy and Microanalysis*, vol. 11, no. S02, pp. 246–247, 2005.

[85] T. G. Team. (1977 - 2019) GNU Image Manipulation Program: GIMP 2.8.10. Accessed 21-02-2019. [Online]. Available: https://www.gimp.org/

[86] "Darkflow," https://github.com/thtrieu/darkflow, 2018, accessed: 2019-04-01.

[87] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: image processing in Python," *PeerJ*, vol. 2, p. e453, 6 2014. [Online]. Available: https://doi.org/10.7717/peerj.453

[88] J.-C. Yen, F.-J. Chang, and S. Chang, "A new criterion for automatic multilevel thresholding," *IEEE Transactions on Image Processing*, vol. 4, no. 3, pp. 370–378, 1995.

[89] J. B. Roerdink and A. Meijster, "The watershed transform: Definitions, algorithms and parallelization strategies," *Fundamenta informaticae*, vol. 41, no. 1, 2, pp. 187–228, 2000.

[90] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–, accessed: 2019-02-10. [Online]. Available: http://www.scipy.org/

[91] "dsb2018_topcoders," https://github.com/selimsef/dsb2018_topcoders, 2018, accessed: 2019-02-20.

[92] W. Abdulla, "Mask R-CNN for object detection and instance segmentation on keras and tensorflow," https://github.com/matterport/Mask_RCNN, 2017, accessed: 2019-03-15.

[93] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[94] T. Fletcher, "Support vector machines explained," *Tutorial paper*, 2009.

[95] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[96] "Scikit learn: sklearn.svm.svc," https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html, accessed: 2019-06-04.

[97] Y.-M. Huang and S.-X. Du, "Weighted support vector machine for classification with uneven training class sizes," in *2005 International Conference on Machine Learning and Cybernetics*, vol. 7. IEEE, 2005, pp. 4365–4369.

[98] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[99] F. Chollet *et al.*, “Keras,” https://keras.io, 2015, accessed: 2019-02-08.

[100] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of ADAM and beyond,” *arXiv preprint arXiv:1904.09237*, 2019.

[101] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[102] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. (2011) RGB to grayscale. Accessed 10-05-2019. [Online]. Available: https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_rgb_to_gray.html

[103] F. Namvar, H. S. Rahman, R. Mohamad, A. Rasedee, S. K. Yeap, M. S. Chartrand, S. Azizi, and P. M. Tahir, “Apoptosis induction in human leukemia cell lines by gold nanoparticles synthesized using the green biosynthetic approach,” *Journal of Nanomaterials*, vol. 16, no. 1, p. 330, 2015.

[104] H. Narotamo, J. M. Sanches, and M. Silveira, “Segmentation of cell nuclei in fluorescence microscopy images using deep learning,” in *Iberian Conference on Pattern Recognition and Image Analysis*. Springer, 2019, pp. 53–64.

[105] “COCO API,” https://github.com/cocodataset/cocoapi, 2018, accessed: 2019-03-06.

[106] K. Potter, H. Hagen, A. Kerren, and P. Dannenmann, “Methods for presenting statistical information: The box plot,” *Visualization of large and unstructured data sets*, vol. 4, pp. 97–106, 2006.

[107] D. L. Massart, A. J. Smeyers-verbeke *et al.*, “Practical data handling visual presentation of data by means of box plots,” 2005.

[108] D. F. Williamson, R. A. Parker, and J. S. Kendrick, “The box plot: a simple visual method to interpret data,” *Annals of internal medicine*, vol. 110, no. 11, pp. 916–921, 1989.

[109] J. L. Hintze and R. D. Nelson, “Violin plots: a box plot-density trace synergism,” *The American Statistician*, vol. 52, no. 2, pp. 181–184, 1998.

[110] S. M. Ross, *Introduction to probability and statistics for engineers and scientists*. Elsevier, 2004.

[111] R. Amami, D. B. Ayed, and N. Ellouze, “Practical selection of SVM supervised parameters with different feature representations for vowel recognition,” *arXiv preprint arXiv:1507.06020*, 2015.

# A

# Deep Learning State-of-The-Art in Table Format

**Table A.1:** Deep Learning state-of-the-art for cell nuclei segmentation.

| | Main Goal | Dataset | Main Architecture | Pre-Processing | Train & Test Time | Nucleus Splitting | Main Results |
|---|---|---|---|---|---|---|---|
| [37] | Cytoplasm and nuclei segmentation in cervical cancer images | HE stained images | CNN to detect the nucleus; superpixel algorithm to segment nuclei | Trimmed mean filter + RGB space to CIE lab space | Test: 50 s for an image of size $1024 \times 1360$ | ———— | **Precision:** 0.91, **Recall:** 0.87, **F1 Score:** 0.90 |
| [30] | Nuclear segmentation applicable to histopathology images with different stains | Histopathology images: pancreatic neuroendocrine tumor, brain tumor and breast cancer | CNN2 | ———— | Test: 54 s for an image of size 220 $\times$ 230 | Novel algorithm that combines robust selection based sparse shape model and local repulsive deformable model. | **Brain Tumor** Dice similarity coefficient (DSC): 0.85 Hausdorff distance (HD): 5.06 Mean absolute distance (MAD): 3.26 **Pancreatic Tumor** DSC: 0.92 HD: 2.41 MAD: 1.58 **Breast Cancer** DSC: 0.80 HD: 8.60 MAD: 6.24 |
| [38] | Nuclei segmentation at instance level and release a dataset of labeled images | HE stained images | CNN3 | Color normalization | Train: 16.5 hours, Test: 30 s for an image of size 1000 $\times$ 1000 | Third class of pixels which represent nuclear boundary | CNN3 DSC: 0.76 HD: 7.66 F1 Score: 0.83 CNN2 DSC: 0.69 HD: 8.69 F1 Score: 0.75 |

Continued on next page

**Table A.1 – continued from previous page**

| | Main Goal | Dataset | Main Architecture | Pre-Processing | Train & Test Time | Nucleus Splitting | Main Results |
|---|---|---|---|---|---|---|---|
| [40] | Nuclei segmentation in histopathology images and release a dataset of manually annotated images | HE stained histopathology images | FCN | ———— | ———— | Watershed algorithm | **Accuracy:** 0.94 **Recall:** 0.75 **Precision:** 0.82 **F1 Score:** 0.76 |
| [33] | Detection of nuclei in histopathological images | HE stained images | FCN | ———— | ———— | ———— | **F1 Score** varies between 0.817 and 0.823 |
| [41] | Segmentation of neuronal structures and Cell segmentation | Electron microscopy images of neuronal structures and light microscopy images of cells | U-Net | ———— | Test: Less than one second for an image of size 512 × 512 | Weighted loss map to force the network to learn the separation between close cells | IoU: PhC-U373 dataset: 0.92, DIC-HeLa dataset: 0.78 |
| [32] | Nuclei segmentation in histopathological images | HE stained images from different organs (3 datasets: MOD, BCD, BNS) | U-Net | Color normalization | Train: MOD dataset 300 epochs in 7.5 hours, Test: 5 s for an image of size 1000 × 1000 | Simultaneously predicts nuclei and their contours | **Precision** MOD: 0.81 BCD: 0.94 BNS: 0.92 **Recall** MOD: 0.91 BCD: 0.92 BNS: 0.78 **F1 Score** MOD: 0.85 BCD: 0.92 BNS: 0.84 |

Continued on next page

**Table A.1 – continued from previous page**

| | Main Goal | Dataset | Main Architecture | Pre-Processing | Train & Test Time | Nucleus Splitting | Main Results |
|---|---|---|---|---|---|---|---|
| [44] | Investigate the potential of deep learning strategies for nuclei segmentation, segment each nucleus individually | Fluorescence microscopy images | U-Net | Illumination correction, median filters, opening operators | Train: 1 hour, Test: 0.6 - 10.1 seconds | Consider a boundary class that represents the nuclei contours, give 10 times more weight to the boundary class in the loss function | **Jaccard Index:** 0.91, **F1 Score:** 0.85 |
| [34] | Nuclear segmentation | HE stained histology section images | E-Net | Color decomposition, convert RGB into 2 channels one with DNA information and other with protein contents | Test: 60 ms for an image of size 360 × 480 | Combination of two E-Nets. One E-Net predicts nuclei and background pixels, and the other E-Net predicts nuclei boundaries, thereafter the outputs of both E-Nets are fed into a third E-Net. Additionally, a watershed based post-processing step is performed. | **Precision:** 0.94, **Recall:** 0.88, **F1 Score:** 0.91 |
| [45] | Building detection in satellite images | Satellite images | TernausNetV2 | Min-Max normalization per channel | Test: 10 samples/s | In this case there is an additional channel that predicts areas where the buildings touch or are very close, this allows to split instances of the same object. An additional post-processing method based on watershed is applied. | ——— |

Continued on next page

**Table A.1 – continued from previous page**

| | Main Goal | Dataset | Main Architecture | Pre-Processing | Train & Test Time | Nucleus Splitting | Main Results |
|---|---|---|---|---|---|---|---|
| [47] | Cell nuclei instance segmentation | Brightfield and fluorescence microscopy images | Encoder Decoder type architecture based on U-Net | ——— | ——— | Additional channel to predict touching borders between nuclei, and post-processing step based on watershed. | Winner of kaggle data competition 2018 |
| [48] | Instance segmentation of T-Cells | Microscopic images of T-Cells | U-Net | ——— | ——— | 3 classes: background, foreground and touching class. And a weighted loss function where the weight map gives more importance to underrepresented parts. | **Precision:** 0.811, **Recall:** 0.671, **F1 metric:** 0.735. |
| [49] | Nuclei segmentation at instance level | HE stained histology images, TCGA (the cancer genome Atlas) | Ric-Unet (Network based on U-Net) | Color normalization | Test: Less than 1 second for an image of size 1000 $\times$ 1000 | Separate up-sampling branch to predict nuclei contours and a post-processing step applied to both the contour and nuclei predictions maps. | **Aggregated Jaccard Index:** 0.56 **F1 Score:** 0.83 DSC: 0.80 |
| [50] | Nuclei segmentation in histopathology images | Segmentation of nuclei in images contest and MICCAI MoNuSeg (HE stained images) | US-Net (Combination of U-Net and SSD) | ——— | ——— | Combines the object instance detection capability of the SSD with the segmentation ability of the U-Net. Post-processing step to perform instance level segmentation. | **The authors didn't provide numerical results.** |
| [62] | Nuclei instance segmentation in microscopy images | Microscopy images from BBC038V1 | Mask R-CNN | Normalization of the channel means and image upsampling | ——— | Uses an instance segmentation architecture. | **AP**: 59.40 % and IoU: 70.54 % |

Continued on next page

**Table A.1 – continued from previous page**

| | Main Goal | Dataset | Main Architecture | Pre-Processing | Train & Test Time | Nucleus Splitting | Main Results |
|---|---|---|---|---|---|---|---|
| [31] | Cell nuclei instance segmentation | Herlev dataset | Mask R-CNN + LFCCRF | ———— | ———— | Uses an instance segmentation architecture. | **Average Precision** = 96 % and **Average Recall** = 96 % |
| [63] | Compare the performance of the U-Net and Mask R-CNN in the task of nuclei instance segmentation | Fluorescence miscroscopy images and histology images | U-Net, Mask R-CNN and ensemble between U-Net and Mask R-CNN | ———— | ———— | In the U-Net they predict not only the nuclei but also the touching borders between close nuclei, and additional post-processing step based on watershed is applied. | **Precision:** Mask R-CNN - 0.841, U-Net - 0.733, ensemble - 0.767. **Recall:** Mask R-CNN - 0.663, U-Net - 0.643, ensemble - 0.664. |

# B

# Project Code

This appendix presents the code of some parts of this project.

## B.1 MATLAB Code for the Postprocessing of Masks Generated with GIMP

In this section, the MATLAB code for the postprocessing step applied to masks generated using GIMP is provided.

**Listing B.1:** Matlab Script for the Postprocessing step applied to masks generated using GIMP

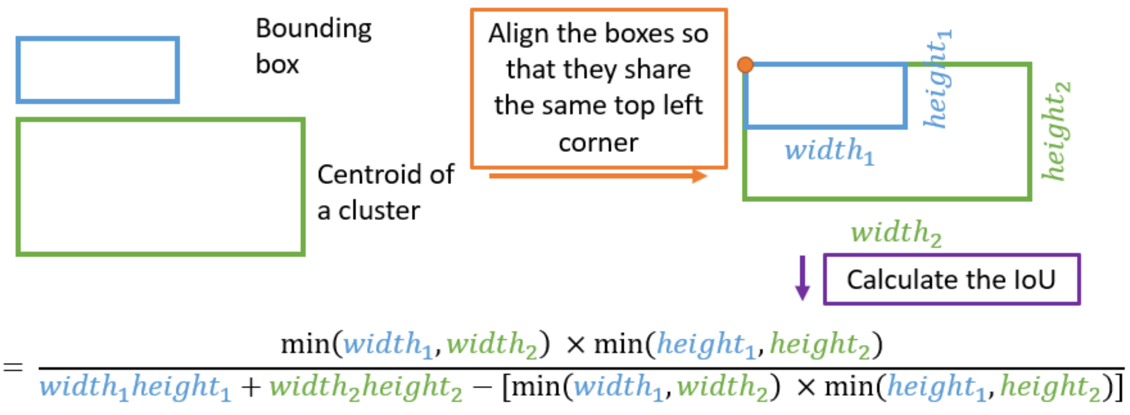
```
D = 'C:\Users\hemaxi\Desktop\Clones\Better_Masks'; %directory where the files are saved
S = dir(fullfile(D,'*.tif*')); % pattern to match filenames.

for k = 1:numel(S)
    F = fullfile(D,S(k).name);
    I = imread(F); %read the image
    aux = I;
    aux(aux>220)=255;
    aux(aux~=255)=0;
    aux = aux/255;
    CC = bwconncomp(aux);
    AA = regionprops(CC, 'Area'); %obtain the area of each object
    L = labelmatrix(CC);
    BW2 = ismember(L, find([AA.Area] >= 10)); %remove very small objects
    BW2 = imfill(BW2,'holes'); %fill the holes
    imwrite(logical(BW2), strcat('C:\Users\hemaxi\Desktop\Clones\Clones_Masks_Final\',
    S(k).name)); %directory where the output files will be saved
end
```

## B.2 K-means Clustering to Obtain Good Priors to Train the Fast YOLO

Algorithm B.1 presents the pseudo-code for the k-means clustering algorithm performed to obtain good priors to train the Fast YOLO. Where M is the number of nuclei in the training dataset, and K = 5 represents the number of anchors. The distance measure of this algorithm requires the calculation of $IoU(\{centroid\_width, centroid\_height\}, \{bbox\_width, bbox\_height\})$. Figure B.1 shows how this value is calculated. Note that, a bounding box is represented by 4 coordinates, but in this problem the goal is to compare the shapes of the bounding boxes. Therefore, only two coordinates per bounding box are necessary, the width and the height.

```
Algorithm B.1: K-means clustering to generate anchors
  Input: K(int): number of clusters, D(list): width and height of the M bounding boxes, i.e.,
    $(\{bbox\_width_1, bbox\_height_1\}, ..., \{bbox\_width_M, bbox\_height_M\})$
  Output: Width and Height of the centroids of K clusters
    $(\{centroid\_width_1, centroid\_height_1\}, ..., \{centroid\_width_K, centroid\_height_K\})$
  Initialization: Initialize the clusters randomly, pick K elements of D
  while No change in the clusters do
      1 ) Assign each object of D (represented by 2 coordinates, the width and height of the
       bounding box: $\{bbox\_width, bbox\_height\}$) to the cluster to which the object is the most
       similar, this similarity is measured by the following distance:
       $distance = 1 - IoU(\{centroid\_width, centroid\_height\}, \{bbox\_width, bbox\_height\})$ ;
      2) Update the cluster centroids, that is, calculate the mean width and mean height of all
       elements in each cluster
  end
```



$$IoU = \frac{\min(width_1, width_2) \times \min(height_1, height_2)}{width_1 height_1 + width_2 height_2 - [\min(width_1, width_2) \times \min(height_1, height_2)]}$$

**Figure B.1:** Illustrative example for calculating the IoU between two boxes, based on the width and height of each bounding box.

## B.3 Code to Generate Binary Mask per Object

In this section the code to generate a binary mask per object is presented (listing B.2). These binary masks per object were created to train the Mask R-CNN.

**Listing B.2:** Code to create a binary mask per object, to train the Mask R-CNN.

```python
#import the necessary packages
import cv2
from skimage.measure import label
import numpy as np
import os

masks_dir = r'D:\Ambiente_de_Trabalho\Tese\Dataset1\Masks' #directory of the
    binary masks

for msk in os.listdir(masks_dir): #iterate through each mask in the masks
    directory
    img_aux = msk.replace('.tif', '')
    os.mkdir(r'D:/Ambiente_de_Trabalho/Tese/Dataset1/MRCNN_Masks/' + img_aux)
        #create a folder with the mask name
    ground_truth = cv2.imread(os.path.join(masks_dir, msk), cv2.
        IMREAD_GRAYSCALE) #read the binary mask
    labeled_mask = label(ground_truth) #label the ground truth binary mask,
        now each object has an unique label
    nb_objects = np.max(labeled_mask) #count the number of different objects
    for i in range(nb_objects): #iterate through all the objects in the
        ground truth mask
        aux_mask = np.zeros(np.shape(ground_truth)) #create an image with
            zeros
        aux_mask[labeled_mask==i+1] = 1 #start in i+1, because i=0 is the
            background, this will create a binary mask, where only the pixels
             that belong to object i+1 have value 1, the other pixels have
            value 0
        cv2.imwrite(r'D:/Ambiente_de_Trabalho/Tese/Dataset1/MRCNN_Masks/' +
            img_aux + '/' + img_aux + str(i) + '.png', aux_mask * 255.0) #
            save the image
```

# C

# Evaluation Metrics

In this appendix a detailed explanation regarding the evaluation metrics is provided. These metrics were used to evaluate the performance of the models for nuclei segmentation (section C.1) and nuclei classification (section C.2). Moreover, in section C.3 an explanation regarding box plots and violin plots is provided.

## C.1 Cell Nuclei Segmentation

To evaluate the performance of different models for cell nuclei segmentation, F1 Score (equation C.1) was calculated at different thresholds of the IoU (equation C.2). In this case, the threshold varied from 0.5 to 0.95 by steps of size 0.05. It is similar to the evaluation criteria used in the COCO dataset [1] [105], where the mAP is calculated for IoU varying from 0.5 to 0.95, by steps of size 0.05. To calculate the F1 Score, first the precision (equation C.3) and recall (equation C.4) must be calculated. Where TP, FP and FN stand for true positives, false positives and false negatives, respectively.

[1] COCO dataset is a dataset for large-scale object detection, segmentation and captioning.

$$F1\,Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{C.1}$$

$$IoU = \frac{Area\,of\,overlap}{Area\,of\,union} \tag{C.2}$$

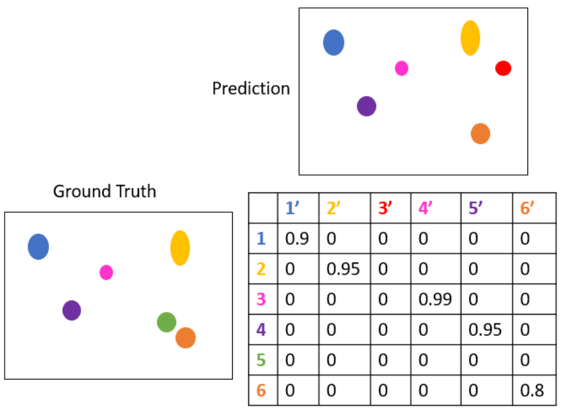
$$Precision = \frac{TP}{TP + FP} \tag{C.3}$$

$$Recall = \frac{TP}{FN + TP} \tag{C.4}$$

For each fluorescence microscopy image there is a ground truth segmentation mask and a segmentation mask predicted by the model. To evaluate the quality of the predicted mask, it must be compared with the ground truth mask. For each image a matrix is built (figure C.1(a)), where the component $(i, j)$ represents the IoU between the object $i$ in the ground truth mask and object $j$ in the predicted mask. After obtaining this matrix, different thresholds are applied to it. For instance, by applying a threshold of 0.7, everything above this value will be true, everything below will be false (see figure C.1(b)). Now the TP, FN and FP can be calculated. In other words, for each row if there is a true it means that an object in the ground truth mask presents high overlap with an object in the predicted mask, it is a TP. If for a given row every entry has a false (F), it means that the object in the ground truth mask doesn't have a corresponding object in the prediction, it is a FN. Finally, if for a given column every entry has a false (F), it means that the object in the prediction doesn't have a corresponding object in the ground truth, it is a FP. For a given threshold (T), by counting the FP, TP and FN, precision (equation C.3), recall (equation C.4) and consequently F1 Score (equation C.1) can be calculated at that threshold. To sum up, a nucleus detected by an automatic technique is considered as TP if for a given threshold (T), its IoU with some ground truth nucleus is higher than T. On the other hand, if its IoU with all nuclei in the ground truth is lower than T, it is considered as FP (extra object). Finally, if for a given ground truth nucleus there isn't a corresponding detection, it will be considered as FN (miss detection).

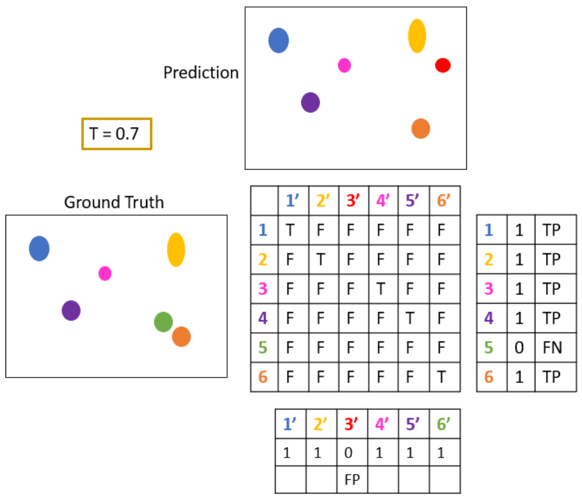
The example illustrated in figure C.1 was a simple example where there aren't merges and splits in the prediction. A merge is when two nuclei in the ground truth are predicted as being a single nucleus in the prediction mask. A split occurs when one nucleus in the ground truth mask is predicted as two nuclei in the prediction mask. In this work, one merge is counted as two FN and one FP. Whereas, one split is counted as one FN and two FP. Since the main goal is to perform nuclei segmentation at instance level, this criterion will penalize more the performance of models whose output presents more merges.

In order to explain how merges and splits were counted, a more complex case is represented in figure C.2. Figure C.2(a) shows the matrix built for the case where in the predicted mask there is one



| | 1' | 2' | 3' | 4' | 5' | 6' |
|---|---|---|---|---|---|---|
| 1 | 0.9 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0.95 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0.99 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0.95 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0.8 |

(a)



| | 1' | 2' | 3' | 4' | 5' | 6' |
|---|---|---|---|---|---|---|
| 1 | T | F | F | F | F | F |
| 2 | F | T | F | F | F | F |
| 3 | F | F | F | T | F | F |
| 4 | F | F | F | F | T | F |
| 5 | F | F | F | F | F | F |
| 6 | F | F | F | F | F | T |

| | | |
|---|---|---|
| 1 | 1 | TP |
| 2 | 1 | TP |
| 3 | 1 | TP |
| 4 | 1 | TP |
| 5 | 0 | FN |
| 6 | 1 | TP |

| 1' | 2' | 3' | 4' | 5' | 6' |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 |
| | | FP | | | |

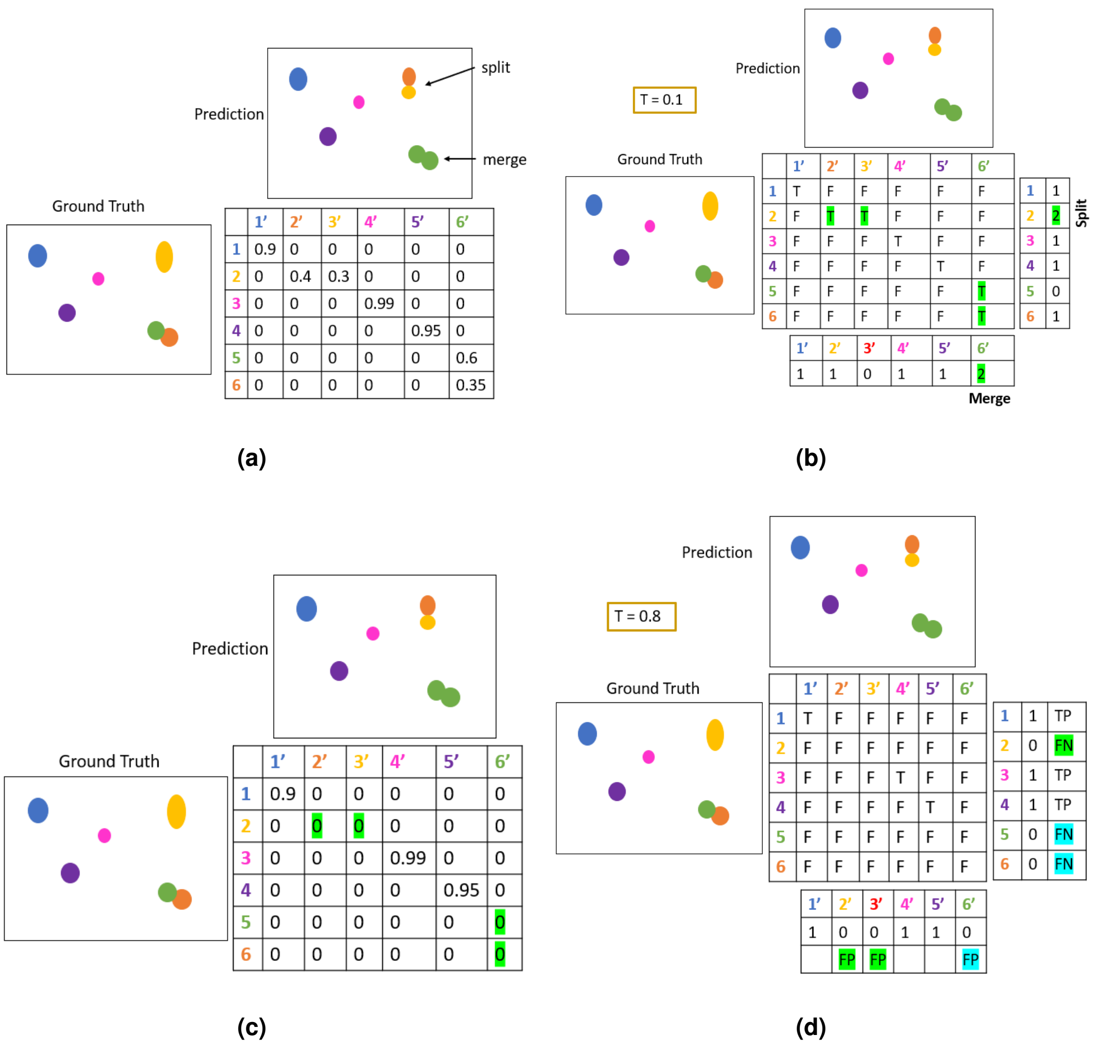
(b)

**Figure C.1:** Simple example to explain the evaluation of the models designed for nuclei segmentation. **a)** Example of a matrix built to evaluate the performance of the models designed for nuclei segmentation. **b)** Illustration of the procedure followed to calculate the TP, FN and FP at a given IoU threshold.

merge and one split. The first step consists in applying a small threshold to this matrix (T = 0.1 as done in [44]), and everything above this value is true (T) and everything below is false (F) (figure C.2(b)). On one hand, when there are two T in a row, it means that one ground truth object has two corresponding objects in the prediction, it represents a split. On the other hand, when there are two T in a column, it means that one object in the prediction has two corresponding objects in the ground truth, it represents a merge. Thereafter, in the matrix represented in figure C.2(a), the values at the positions highlighted in green in the matrix shown in figure C.1(b) are changed to 0 (see figure C.2(c)). Finally, different thresholds are applied to this matrix (from 0.5 to 0.95). For instance, by applying a threshold of 0.8 (see figure C.2(d)), the TP, FN and FP at that threshold can be calculated. Note that the merge is counted as 2 FN and 1 FP (highlighted in blue), whereas a split is counted as 2 FP and 1 FN (highlighted in green).

## C.2 Cell Cycle Staging

To test the performance of each model, classification_report tool from sklearn.metrics was used. This tool gives the precision (equation C.3), recall (equation C.4) and F1 score (equation C.1) for each class in the classification problem. It outputs the matrix represented in table C.1. The second row represents the precision, recall and F1 score when the class G1 is considered positive. The third row shows the precision, recall and F1 score when it is considered that the class S/G2 is the positive class. Finally, the last row shows the average precision, recall and F1 score between the two classes.



(a)

| | 1′ | 2′ | 3′ | 4′ | 5′ | 6′ |
|---|---|---|---|---|---|---|
| 1 | 0.9 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0.4 | 0.3 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0.99 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0.95 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0.6 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0.35 |

(b)

| | 1′ | 2′ | 3′ | 4′ | 5′ | 6′ | | Split |
|---|---|---|---|---|---|---|---|---|
| 1 | T | F | F | F | F | F | 1 | 1 |
| 2 | F | T | T | F | F | F | 2 | 2 |
| 3 | F | F | F | T | F | F | 3 | 1 |
| 4 | F | F | F | F | T | F | 4 | 1 |
| 5 | F | F | F | F | F | T | 5 | 0 |
| 6 | F | F | F | F | F | T | 6 | 1 |

| 1′ | 2′ | 3′ | 4′ | 5′ | 6′ |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 2 |

Merge

(c)

| | 1′ | 2′ | 3′ | 4′ | 5′ | 6′ |
|---|---|---|---|---|---|---|
| 1 | 0.9 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0.99 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0.95 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 |

(d)

| | 1′ | 2′ | 3′ | 4′ | 5′ | 6′ | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | T | F | F | F | F | F | 1 | 1 | TP |
| 2 | F | F | F | F | F | F | 2 | 0 | FN |
| 3 | F | F | F | T | F | F | 3 | 1 | TP |
| 4 | F | F | F | F | T | F | 4 | 1 | TP |
| 5 | F | F | F | F | F | F | 5 | 0 | FN |
| 6 | F | F | F | F | F | F | 6 | 0 | FN |

| 1′ | 2′ | 3′ | 4′ | 5′ | 6′ |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 |
| | FP | FP | | | FP |

**Figure C.2:** Complex example to explain the evaluation of the models designed for nuclei segmentation. **a)** Example of a matrix built to evaluate the performance of the models designed for nuclei segmentation. **b)** Threshold of 0.1 applied to the matrix represented in a). **c)** The values at the positions highlighted in green in the matrix represented in b) are changed to 0. **d)** Threshold of 0.8 applied to the matrix represented in c) to calculate the TP, FN and FP.

**Table C.1:** Classification results obtained by using the classification_report tool from sklearn.metrics.

| | Precision | Recall | F1-Score |
|---|---|---|---|
| G1 | G1_Precision | G1_Recall | G1_F1_Score |
| S/G2 | S/G2_Precision | S/G2_Recall | S/G2_F1_Score |
| Average | Average_Precision | Average_Recall | Average_F1_Score |

## C.3 Box Plot and Violin Plot

A box plot provides information about the variability and dispersion of the data. It is built based on summary statistics of the data. The box plot is also known as box and whisker plot, because each box plot has a box and vertical lines that extend from the box (whiskers), as represented in red in figure C.3 [106, 107]. In order to obtain a box plot, first the data points are ordered from least to greatest, then the following five values are obtained [107, 108]:

- Minimum - lowest value in the dataset;
- Median or Second Quartile (Q2) - it is the middle value of the dataset (if there are two middle values it is the mean between these values);
- First Quartile (Q1) - it is the median of the values positioned before Q2 (approximately 25 % of the data points fall below this value);
- Third Quartile (Q3) - it is the median of the values positioned after Q2 (approximately 75 % of the data points fall below this value);
- Maximum - highest value in the dataset.

An example of a box plot obtained based on these five numbers is shown on left in figure C.3. This is a simple box plot. A more complex box plot is obtained when outliers are present in the dataset (box plot on right in figure C.3). An outlier is a value that is much smaller or much larger than the other values in the dataset. The steps that must be followed to find the outliers are [107]:

1. Find the interquantile range (IQR), $IQR = Q3 - Q1$;
2. Calculate $Q1 - 1.5(IQR)$
3. Calculate $Q3 + 1.5(IQR)$
4. Any value lower than the value in step 2 or higher than the value in step 3 is an outlier.

In the presence of outliers, the ends of the whiskers aren't the minimum and the maximum value of the dataset. These ends represent the minimum value of the dataset higher than $Q1 - 1.5(IQR)$ and the maximum value of the dataset lower than $Q3 + 1.5(IQR)$ [107].

A violin plot is similar to the box plot, however, it also shows the distribution (probability density) of the data. Hence, a violin plot is more informative than a box plot [109]. In order to understand better a violin plot, the concept of probability density function will be introduced.

A random variable is some quantity of interest that is obtained by the result of an experiment. The outcome of the experiment determines the value of the random variable, therefore probabilities can be assigned to the possible values that the random variable can take. A random variable can be discrete or continuous [110]. In this work the variable for which the violin plot is built is the F1 score. The F1 score can take infinite number of possible values, that is, it is a continuous random variable.

A continuous random variable (X) is defined over an interval of values. For instance, the probability of X falling in the interval B = [a,b] is given by [110]:

$$P\{a \leq X \leq b\} = \int_a^b f(x)dx \tag{C.5}$$

where $x \in (-\infty, \infty)$ and $f(x)$ is known as the probability density function of the random variable X, and has the following properties:

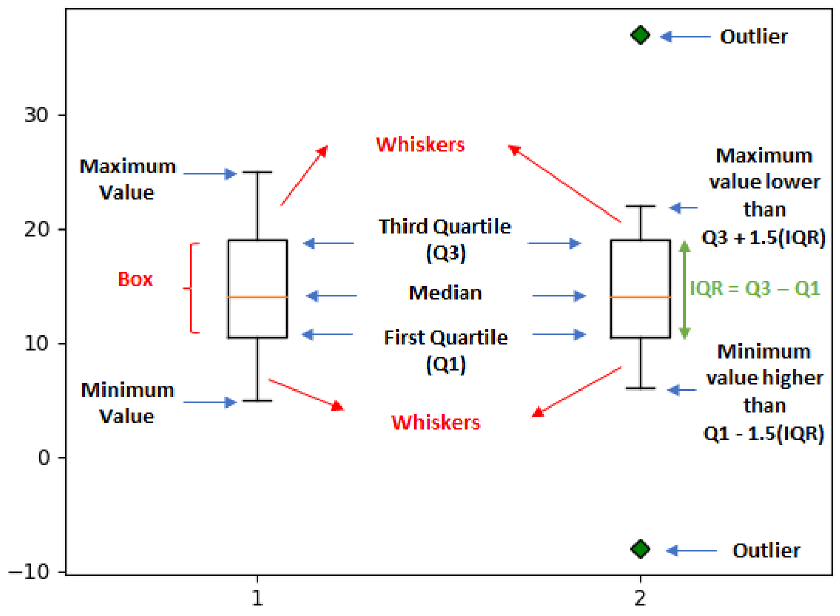


**Figure C.3:** Meaning of the components of a boxplot without the presence of outliers (on the left) and with the presence of outliers (on the right). The plot on left represents data1 = {5, 6, 12, 13, 15, 18, 22, 25}, and the one on the right represents data2 = {-8, 6, 12, 13, 15, 18, 22, 37}. Both plots were obtained with the matplotlib function boxplot.

1. $f(x) \geq 0, \forall x \in \Re$

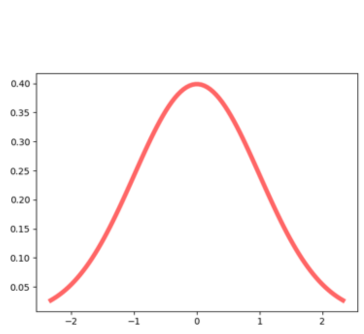
2. $P\{X \in (-\infty, \infty)\} = \int_{-\infty}^{\infty} f(x)dx = 1$

The probability of a continuous random variable assuming a particular value is zero:

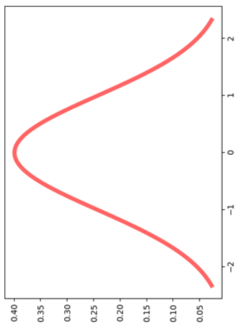
$$P\{X = a\} = \int_{a}^{a} f(x)dx = 0 \tag{C.6}$$

Figure C.4(a) shows an example of a probability density function of a random variable. The violin plot is a rotated version of this plot (figure C.4(b)), where the probability density function is mirrored across the other side (C.4(c)) [109]. Moreover, the violin plot also shows some statistics of the data. Thus, a violin plot combines the statistics and the density shape in a single plot [109], (see figure C.4(d).)

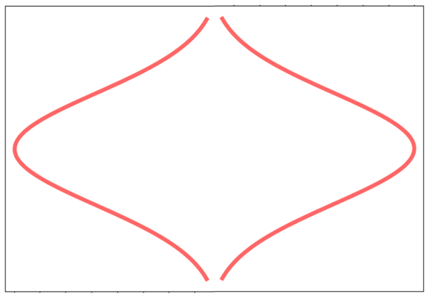
Finally, figure C.5 illustrates the common components of a box plot and a violin plot. Both show the upper and lower adjacent values, the first, second and third quartile and the outliers. When compared to the box plot the violin plot has an additional information regarding the distribution of the data. This information can be useful, for example to compare two populations with similar statistics but different distributions [109].



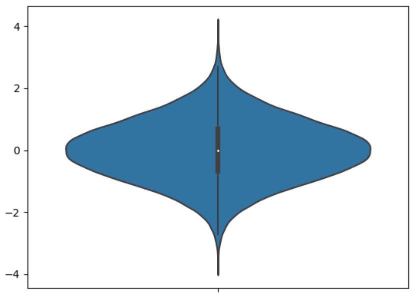
**(a)**



**(b)**



**(c)**



**(d)**

**Figure C.4: a)** Probability density function of a random variable, in this case, $f(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$. **b)** Rotated version of the plot shown in a). **c)** Mirrored version of the plot shown in b). **d)** Example of a violin plot for data with the distribution shown in a).

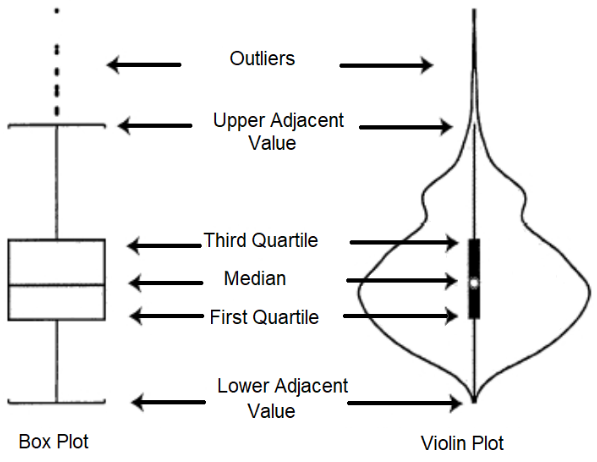


**Figure C.5:** Common components of a box and a violin plot. Image adapted from [109].

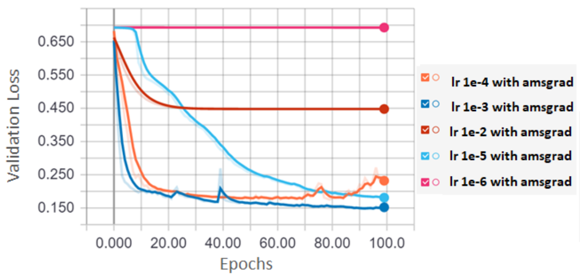
# D

# Hyperparameter Optimization

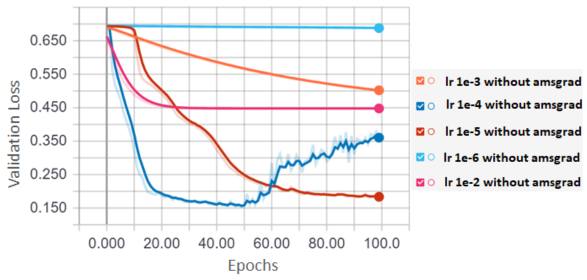
## D.1 Cell Nuclei Segmentation

**Original U-Net**

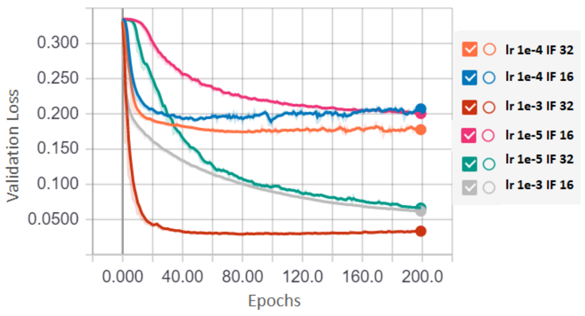
First of all the effect of amsgrad argument of ADAM was studied (figures D.1(a) and D.1(b)). It is observed that amsgrad argument doesn't have a significant effect on the validation loss plots for learning rates 1e-2, 1e-6 and 1e-5. By visualizing the plots of the validation loss for learning rates 1e-3 and 1e-4 with amsgrad (figure D.1(a)) and without amsgrad (figure D.1(b)) it can be concluded that the plots with amsgrad are better than the plots without amsgrad. For instance, for learning rate 1e-3 and without amsgrad the validation loss is much higher when compared to the loss obtained with learning rate 1e-3 and with amsgrad. From these experiments it is concluded that the parameter amsgrad provides better results and from figure D.1(a) it is concluded that the smallest validation loss is obtained with learning rates 1e-3, 1e-4 and 1e-5. Therefore, the effect of different number of filters in the first layer (IF) of the U-Net was studied for these learning rates (figure D.1(c)). By observing these plots learning rate 1e-3 and number of filters 32 have been chosen to train the U-Net. One example of the validation loss during the training of the U-Net with the chosen parameters is shown in figure D.1(d). Note that, IF represents the complexity of the U-Net. The number of filters in the following layers are calculated based on this number.



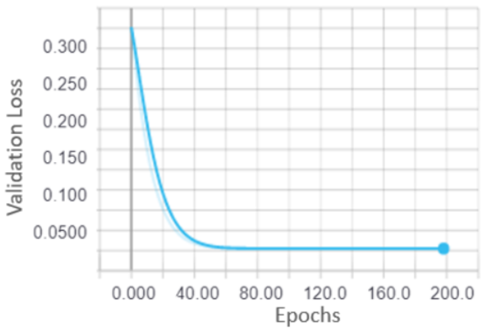
**(a)**



**(b)**



**(c)**



**(d)**

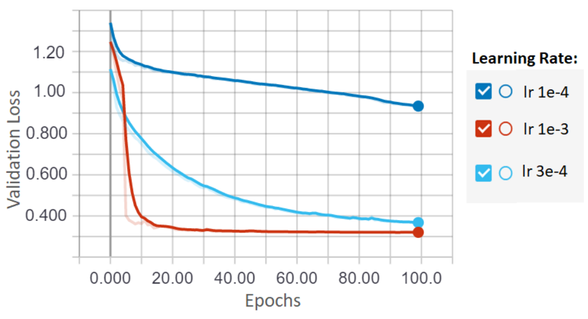
**Figure D.1:** Validation loss plots for the original U-Net. Plots obtained from Tensorboard. **a)** Validation loss as function of the training epochs, for different learning rates and with the argument amsgrad of ADAM equal to true. **b)** Validation loss as function of the training epochs, for different learning rates and with the argument amsgrad of ADAM equal to false. **c)** Validation loss as function of the training epochs, for different learning rates (represented as lr) and with different number of filters in the first layer (represented as IF). **d)** Validation loss during the training of the Original U-Net.

### Kaggle_2018

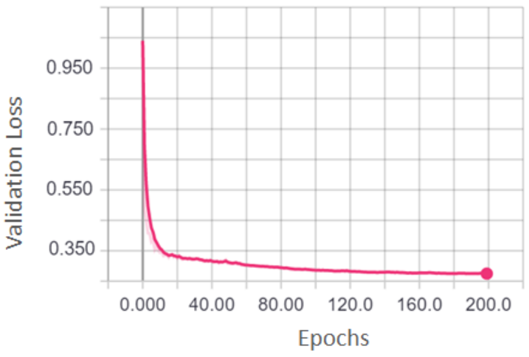
In order to choose the best learning rate, Kaggle_2018 was trained with the following learning rates: 1e-3, 1e-4 and 3e-4 (figure D.2(a)). Although the validation loss for learning rate 1e-3 decreases rapidly, it stabilizes at 0.3. The validation loss for learning rate 1e-4 decreases two slowly. Finally, the loss for 3e-4 decreases more slowly when compared with the loss for learning rate 1e-3. However, it seems like it will continue to decrease and will stabilize at a value lower than 0.3. Therefore, learning rate 3e-4 was chosen to train Kaggle_2018. An example of the validation loss during the training of this model is shown in figure D.2(b).

### Mask R-CNN

In order to choose the best learning rate, Mask R-CNN was trained for 30 epochs with the following learning rates: 1e-6, 1e-5, 1e-4 and 1e-3 (figure D.3(a)). It is observed that with learning rate 1e-3 the



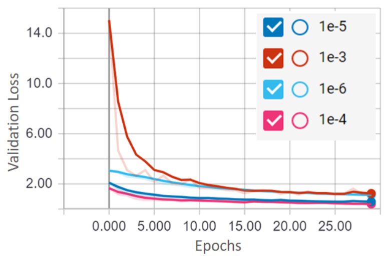
**(a)**



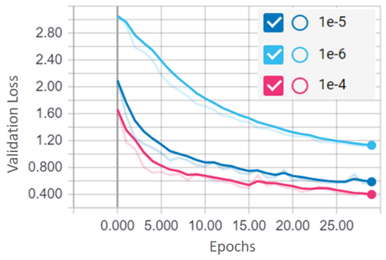
**(b)**

**Figure D.2:** Validation loss plots for Kaggle_2018. Plots obtained from Tensorboard. **a)** Validation loss as function of the training epochs, for different learning rates. **b)** Validation loss during the training of the Kaggle_2018.

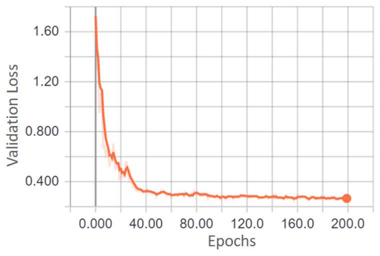
loss is high at the beginning, and then decreases rapidly and stabilizes in a value that is higher than the values of the validation loss obtained with the other learning rates. By removing the validation loss plot for learning rate 1e-3 (figure D.3(b)) it can be observed the best learning rate is 1e-4, because the validation loss obtained with this learning rate is the lowest. Finally, figure D.3(c) shows the validation loss during the training of the Mask R-CNN, the training was stopped when the validation loss was smooth and low.



**(a)**



**(b)**



**(c)**

**Figure D.3:** Validation loss plots for Mask R-CNN. Plots obtained from Tensorboard. **a)**, **b)** Validation loss as function of the training epochs, for different learning rates. **c)** Validation loss during the training of Mask R-CNN.

## D.2 Cell Cycle Staging

In order to perform hyperparameter optimization, for the SVM, GridSearchCV tool from sklearn.model_selection was used. This search was performed in the following parameter space:

- Kernel: 'rbf'; Gamma: [1e-2, 1e-3, 1e-4, 1e-5]; C: [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000];

- Kernel: 'poly'; C: [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]; degree: [1, 2, 3, 4, 5];
- Kernel: 'sigmoid'; Gamma: [1e-2, 1e-3, 1e-4, 1e-5]; C: [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000];
- Kernel: 'linear'; C: [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000];

where for rbf kernel $\gamma = \frac{1}{2\sigma^2}$ [111], for sigmoid kernel $\gamma = a$ [111], and for poly kernel degree = b.

For each of the combinations between these parameters a model is built using the training set and its performance is evaluated on the validation set. The best model is the one that presents the highest performance (average F1 Score between G1 and S/G2) on the validation set. Finally, the performance of this model is evaluated on the test set.

The parameters of the best model for each fold of the 5-fold cross validation, for SVM_A and SVM_B, are summarized in table D.1. For all the models the best kernel was the rbf kernel.

**Table D.1:** Parameters of the best model found for each fold of the 5-fold cross validation. The second column represents the parameters for SVM_A. The third column represents the parameters for SVM_B.

| **Fold** | **SVM_A** | **SVM_B** |
|---|---|---|
| **1** | C: 10<br>Gamma: 1e-5 | C: 50<br>Gamma: 0.0001 |
| **2** | C: 25<br>Gamma: 1e-5 | C: 10<br>Gamma: 0.001 |
| **3** | C: 10<br>Gamma: 1e-5 | C: 50<br>Gamma: 0.0001 |
| **4** | C: 10<br>Gamma: 1e-5 | C: 100<br>Gamma: 0.001 |
| **5** | C: 10<br>Gamma: 1e-5 | C: 10<br>Gamma: 0.001 |