



NOVA

IMS

Information
Management
School

MAA

Mestrado em Métodos Analíticos Avançados

Master Program in Advanced Analytics

**Convolutional Neural Networks for Cell
Detection and Counting**

A Case Study of Human Cell Quantification in
Zebrafish Xenografts using Deep Learning Object
Detection Techniques

Carina Isabel Andrade Albuquerque

Dissertation presented as partial requirement for obtaining
the Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

**NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa**

Convolutional Neural Networks for Cell Detection and Counting
A Case Study of Human Cell Quantification in Zebrafish Xenografts using Deep
Learning Object Detection Techniques

by

Carina Isabel Andrade Albuquerque

Dissertation presented as partial requirement for obtaining the Master's degree in
Advanced Analytics

Advisor: Leonardo Vanneschi
Co Advisor: Roberto Henriques

November 2018

ACKNOWLEDGEMENTS

This thesis would have been impossible without the support and mentoring of my advisors, Leonardo Vanneschi and Roberto Henriques. More than words of wisdom, they also show me the importance of endurance and persistence, and more than mentors, they also have been friends with the right words at the right times. Thank you for your ongoing confidence. A special thank you to Professor Roberto Henriques for all the given opportunities during this academic path. I would also like to express my gratitude to Professor Vitor Santos and Professor Mauro Castelli for the altruist advice, words of enlightenment and support given, and to the many professors who have advised me along the way.

I am grateful to Fundação Champalimaud for the chance of working with them and for giving me the needed "ingredients" to work on in this thesis. A special thank you to Nickolas Papanikolaou, Vanda Póvoa, Rita Fior and José-Maria Moreira for all the patience and transmitted knowledge.

This research has been supported by Microsoft, who I would like to thank for giving me the needed tools to work in this thesis. Without this support, the present project would not be more than one idea. Thank you to Professor Henrique Carreiro for letting me know about this sponsorship.

To my parents, my brother and my sister in law, because they were the ones who suffer most this path with me. A special thanks to my brother for reminding me that for every problem there is always a turn way around, and to make me see further. And to my parents, for years of patience and full support in my decisions, all the love, and guidance.

To Vera Trindade and António Trindade, by all the extra night coffees, friendship, patience, company, support, words of wisdom, pressure, confidence, and so many other things. To Fábria Ribeiro and Maira Perdigão, for being my inspiration source for so many times, and to André Rodrigues for all the support. To Sofia Pinto, the most bright person that I have been proud to meet, and the one that crossed this path with me. Thank you for all the hugs! I would also like to thank for the precious advice from Renier Botha (last minute advice but of great value), Illya Bakurov, Paulo Lapa, Leonardo Custode, Ciro Tecce, Bruno Damásio and Jan Benedikt.

I could not be more grateful to the people mentioned, and so many other friends who I am lucky to have and people that I know, who crossed my life and who walked with me

and lead me to the path where I am today. Thank you for all the support, for the belief in me, and all the words of enlightenment.

This thesis is a product of several convolutions combining data and information obtained in 32 years of experiences, mistakes, inspiration, fortunate events and marvelous and bright people. It was an incredible, sometimes tough, but totally worthwhile, journey.

Finally, I am grateful for life and all the opportunities I had.

"If I have seen further it is by standing on the shoulders of giants."

Sir Isaac Newton

Deep learning had, in recent years, become the state of the art to deal with computer vision problems. On the computer vision research field, object detection is a technique that allows to localize and classify one or more objects in an input image. This approach can be applied to several tasks and problems, as cell counting in medical imaging, as proposed in this thesis.

Cell counting is a frequently needed task in several medical types of research, but often still made manually due to several constraints. The automation of this process can be challenging to achieve, especially when dealing with cell clumping and overlapping, and cells that can assume several shapes and sizes. However, doing this needed process manually turns out to be a bottleneck concerning speed during the research. As so, an automatic tool that allows researchers to quantify cells with different features in an accurate way is a long desired application in the medical community. In this thesis, a fine-tuned architecture based on Faster R-CNN object detection algorithm and Inception Resnet V2 feature extractor is proposed to deal with cell quantification in zebrafish xenografts, an innovative approach for the study of cancer, metastasis, and drug discovery, currently being applied at Fundação Champalimaud, worldwide reference in oncologic treatment innovation.

In this way, it is shown the practical application of the proposed solution to address a problem that remains in the context of medical research in Fundação Champalimaud, where a team of researchers explores the approach of cell counting and histological analysis in zebrafish larvae xenotransplants to evaluate the response of therapies in cancer. As so, this thesis aims to be a contribution to the application of object detection techniques to cell counting tasks, and address several problems usually associated with this process, as the presence of overlapped cells, the high number of objects to be detected and the heterogeneity of cells concerning size and shape.

Keywords: Deep Learning, Faster R-CNN, Cell Counting, Machine Learning, Neural Networks, Computer Vision, Object Detection, Convolutional Neural Networks, Inception ResNet V2

Nos últimos anos, Deep Learning revelou-se como a tecnologia de vanguarda para lidar com problemas de visão computacional. Incluído no campo de pesquisa de visão computacional, a deteção de objetos é uma técnica que permite localizar e classificar um ou mais objetos numa imagem de input. Esta abordagem pode ser aplicada nos mais diversos problemas, tal como contagem de células em imagens médicas, como proposto nesta tese.

A contagem de células é uma tarefa frequentemente necessária em diversas áreas de pesquisa médica, mas maioritariamente ainda realizada manualmente devido a diversas limitações. A automatização deste processo pode ser desafiante de atingir, especialmente quando lidamos com aglomeração e sobreposição de células, e situações em que as células podem assumir diversas formas e tamanhos. No entanto, fazer este processo de modo manual revela-se como uma fase de constrangimento temporal na investigação. Como tal, uma ferramenta automática que permite a investigadores a quantificação de células de diversas características de modo acurado tornou-se uma aplicação fortemente desejada na comunidade médica. Nesta tese, uma arquitetura ajustada baseada no algoritmo de deteção de objetos Faster R-CNN e no Inception Resnet v2 é proposta de modo a lidar com a quantificação de células em xenoenxertos de zebrafish, uma abordagem inovadora para o estudo de cancro, metástases e descoberta de drogas, e atualmente a ser desenvolvida na Fundação Champalimaud, uma referência mundial em termos de inovação no tratamento oncológico.

Desta maneira, é apresentada a aplicação prática da solução proposta para endereçar um problema que se mantém no contexto de pesquisa médica na Fundação Champalimaud, onde uma equipa de investigação explora a contagem de células e a análise histológica em xenotransplantes realizados em zebrafish larvae para avaliar a resposta de terapias em células cancerígenas.

Palavras-Chave: Deep Learning, Faster R-CNN, Quantificação de células, Machine Learning, Redes Neurais, Visão computacional, Deteção de objetos, Redes Neurais Convolucionais, Inception ResNet V2

Contents

subsection	XVI
1 Introduction	1
1.1 Thesis objective and research questions	1
1.2 The task of counting cells in medical imaging	2
1.2.1 The pertinence of counting cells in zebrafish xenografts	4
1.3 Report structure	5
2 Theoretical Background	7
2.1 Machine Learning and Neural Networks	7
2.1.1 Neural Networks and backpropagation	8
2.1.2 Generalization of a model, underfitting and overfitting problems . .	9
2.1.2.1 Avoiding overfitting	9
2.2 Deep learning	11
2.2.1 Deep learning for computer vision and the object detection problem	11
2.3 CNNs	13
2.3.1 The biological inspiration	13
2.3.2 The architecture of a CNN	14
2.3.3 Main layers in a CNN	14
2.3.4 Activation functions	18
2.3.5 Batch normalization	20
2.4 The meta-architecture	21
2.4.1 SSD [30]	21

2.4.2	Faster R-CNN [11] [10] [44]	22
2.4.3	RFCN [3]	25
2.5	The feature extractor	25
2.5.1	Classical Networks	25
2.5.2	Resnet	26
2.5.3	Inception	26
2.5.4	Inception Resnet [52]	27
2.5.5	NASnet [58]	27
3	The Faster R-CNN stages, parameters and hyperparameters: a theoretical introduction	28
3.1	Introduction to Faster R-CNN	28
3.1.1	Quick overview	28
3.2	Faster R-CNN step by step	31
3.2.1	The pre-processing	31
3.2.1.1	Image resizing	31
3.2.1.2	Resize method	32
3.2.1.3	Image normalization	33
3.2.1.4	Converting to grayscale	33
3.2.2	The training process	34
3.2.2.1	The batch size	34
3.2.2.2	The optimizer	35
3.2.2.3	Learning Rate	35
3.2.2.4	Data augmentation	36
3.2.2.5	Gradient clipping	37
3.2.2.6	Transfer learning	38
3.2.3	The feature extractor	38
3.2.3.1	First stage feature stride	38
3.2.4	The first stage of Faster R-CNN	38
3.2.4.1	The anchors	39
3.2.4.2	Atrous rate	39
3.2.4.3	Network hyperparameters	39
3.2.4.4	Minibatch size	40

3.2.4.5	Positive Balance fraction	41
3.2.4.6	NMS - IoU threshold	41
3.2.4.7	Maximum number of proposals	41
3.2.4.8	Loss weights	41
3.2.4.9	ROIs parameters	41
3.2.5	The second stage of Faster R-CNN	42
3.2.5.1	Network hyperparameters	42
3.2.5.2	Dropout	42
3.2.5.3	The IoU Threshold	42
3.2.5.4	The number of detections	42
3.2.5.5	The Loss Weights	43
3.2.5.6	The Classification loss	43
3.2.5.7	The score converter	43
3.3	Measuring the performance of a model	44
3.3.1	mAP	44
3.3.2	Regression performance measures	45
3.3.3	The losses in Faster R-CNN	46
3.3.3.1	First Stage Losses	46
3.3.3.2	The second stage losses	47
4	Data sample and data exploration	50
4.1	Data sample	50
4.2	Data exploration	53
5	Prepare the data to work in TensorFlow	58
5.1	Default parameters	59
6	Experiments	61
6.1	Hardware used	61
6.2	Dealing with a small dataset	61
6.3	Choosing the object detection model	62
6.4	Choosing the feature extractor	64
6.5	Using a pre-trained model	66
6.6	The preprocessing	68

6.6.1	Image resizing	68
6.6.2	Resize method	69
6.6.3	Using normalization	70
6.6.4	Using grayscale	71
6.7	The training process	71
6.7.1	Batch size	71
6.7.2	The optimizer	72
6.7.3	The learning rate	74
6.7.4	Data Augmentation	75
6.7.5	The gradient clipping	76
6.7.6	The number of steps	76
6.7.7	The maximum number of boxes	77
6.8	Configurations on feature extractor	78
6.8.1	The stride	78
6.8.2	Batch norm trainable	78
6.9	First stage of Faster R-CNN	79
6.9.1	Anchor generator	79
6.9.2	Atrous rate	81
6.9.3	Network hyperparameters	82
6.9.4	Box predictor	84
6.9.5	Minibatch size	84
6.9.6	NMS	85
6.9.7	Maximum proposals	86
6.9.8	Loss weights	87
6.9.9	ROI pooling	87
6.10	Second stage of Faster R-CNN	89
6.10.1	Box predictor network hyperparameters	89
6.10.2	Dropout	91
6.10.3	Post processing - Batch Non-Max supression	91
6.10.4	Loss weights	93
6.10.5	Classification loss	93
6.11	Evaluating the regression metrics of the models	93

6.12	Choosing the best model	98
6.13	Use the final models in a new dataset partition	99
7	Final model	101
8	Test evaluation	107
9	Final application	112
10	Conclusions, limitations and future work	114
10.1	Conclusions	114
10.2	Limitations	116
10.3	Future work	116

List of Figures

2.1	Schema of a hierarchical sequence of categorical representations processing a face input stimulus	14
2.2	The main components of a CNN	14
2.3	Different 3x3 kernels applied to the raw image.	16
2.4	Illustration of how max pooling and average pooling significantly reduces parameters as we move up the network	17
2.5	Activation functions widely used in ANN	20
2.6	SSD architecture where we verify that this model adds several feature layers to the end of a base network, which predicts the offsets to default boxes of different scales and aspect ratios and their associated confidences [30].	22
2.7	Architecture of R-CNN [11]	23
2.8	Architecture of Fast R-CNN [10]	23
2.9	The region proposal network system [44]	24
2.10	Architecture of Faster R-CNN [44]	24
2.11	Architecture of LeNet-5 [28]	25
2.12	Architecture of AlexNet [27]	26
2.13	Architecture of ResNet 50 and the concept of residual block [17]	26
2.14	Architecture of GoogleNet/Inception [53]	27
3.1	Overview of Faster R-CNN process	30
3.2	Effect of image resolution in the overall mAP using different meta-architectures [20]	31

3.3	Accuracy stratified by object size, meta-architecture and feature extractor, with an image resolution of 300 pixels [20]	32
3.4	Illustration of the interpolation applied using different methods where the input is a 2x2 matrix and the output a 3x3 matrix	33
3.5	The IoU concept	44
4.1	A raw image example	50
4.2	The process of labeling images using labelImg	51
4.3	Distribution of the quantity of cells per image	54
4.4	Density of cells according to their width and height	56
4.5	Frequency of cells according to their ratio	57
4.6	Possible aspect ratios taking into account the ratio distribution	57
5.1	Basic flow diagram	59
6.1	mAP using different meta-architectures	63
6.2	mAP using different feature extractors	64
6.3	Architecture of the Inception Resnet v2 applied on the present project . . .	65
6.4	Freezing the layers in transfer learning	67
6.5	mAP depending on the dimension of the input image	68
6.6	mAP using different resize methods	70
6.7	mAP using different optimizers	73
6.8	mAP using different initial learning rates (100000 decay steps)	74
6.9	mAP of the five final models	99
6.10	mAP of the model "Best MEDAE" using the two partitions	100
6.11	mAP of the model "Best RMSE & MAE" using the two partitions	100
7.1	Losses for the model "Best RMSE & MAE" with the original partition	101
7.2	Losses for the model "Best RMSE & MAE" with the final partition	102
7.3	Losses for the model "Best MEDAE" with the original partition	103
7.4	Losses for the model "Best MEDAE" with the final partition	103
8.1	Ground truth labeling and inferred detections for images with Low, Medium and High complexity	110

9.1	Initial screen of the application "CellCounter"	112
9.2	Making inferences in the application "CellCounter"	113

List of Tables

3.1	Guidelines for transfer learning	38
4.1	Data partition	52
4.2	Descriptive statistics related to the number of cells in input images	54
4.3	Descriptive statistics related to the width and height of cells present on input images	55
6.1	mAP and speed using different meta architectures	63
6.2	mAP and speed using different feature extractors	64
6.3	mAP and speed by using transfer learning	66
6.4	Total losses, RPN losses in validation, mAP and speed of training for 3000 steps using different input images dimensions	69
6.5	Training loss and speed using image normalization	70
6.6	Training loss and speed using grayscale images	71
6.7	mAP and speed using different batch sizes	72
6.8	mAP and speed using different optimizers	73
6.9	mAP according to the decay steps (initial LR of 0.0002)	74
6.10	mAP using different data augmentation techniques	75
6.11	mAP according to the gradient clipping value	76
6.12	mAP, speed and RPN losses depending on the stride of the feature extractor	78
6.13	mAP and speed depending on the use of the batch normalization	78
6.14	mAP and speed depending on the aspect ratio of the anchors	79
6.15	Width and height of the cells rescaled to the new image resolution	80

6.16 mAP and speed according to different scales used in anchor generator . . .	80
6.17 mAP and speed according to different scales and aspect ratios used in anchor generator	81
6.18 mAP and speed according to the different strides applied on anchor gen- erator	81
6.19 mAP and speed depending on the use of atrous rate	82
6.20 mAP and speed using different regularizers during the first stage of the object detector	82
6.21 mAP and speed according to the activation used in the first stage	83
6.22 mAP and speed depending on the use of batch normalization in the first stage	83
6.23 mAP and speed by using depthwise regularization on the first stage	83
6.24 mAP and speed according to the kernel size of box predictor	84
6.25 mAP and speed according to the depth of box predictor	84
6.26 Total loss, RPN Validation Loss, mAP and speed depending on the size of the minibatch	85
6.27 Total loss, RPN Validation Loss, mAP and speed depending on the positive balance fraction	85
6.28 Total loss, Box Validation Loss, mAP and speed according to the IoU thresh- old used	86
6.29 Total loss, Box Validation Loss, mAP and speed based on the number of proposals	86
6.30 mAP and speed changes using different combinations of localization and objectness loss weights	87
6.31 mAP and speed according to the initial crop size	88
6.32 mAP and speed according to the maxpool kernel size	88
6.33 mAP and speed using different regularizers during the second stage of the object detector	89
6.34 mAP and speed according to the activation used in the second stage	89
6.35 Total Loss, Box validation loss, mAP and speed depending on the use of batch normalization in the second stage	90

6.36	Total Loss, Box validation loss, mAP and speed by using depthwise regularization on the second stage	90
6.37	Total Loss, Box validation loss, mAP and speed by using dropout	91
6.38	Total Loss, Box validation loss, mAP and speed according to the IoU threshold	92
6.39	Total Loss, Box validation loss, mAP and speed according to to maximum detections	92
6.40	Total Loss, Box validation loss, mAP and speed changes using different combinations of localization and classification loss weights	93
6.41	Total Loss, Box validation loss and mAP using different classification losses	93
6.42	Regression metrics and mAP for different configurations in training and data augmentation parameters, where the symbol * represents the default model for Faster R-CNN	94
6.43	Regression metrics and mAP for different configurations in some of the first stage parameters, where the symbol * represents the default model for Faster R-CNN	95
6.44	Regression metrics and mAP for different configurations for the remaining first stage parameters, where the symbol * represents the default model for Faster R-CNN	96
6.45	Regression metrics and mAP for different configurations for the second stage parameters, where the symbol * represents the default model for Faster R-CNN	97
6.46	Main results of five final models	98
6.47	Final data partition	99
7.1	Inference and Regression metrics for model "Best RMSE & MAE" - Low complexity"	104
7.2	Inference and Regression metrics for model "MEDAE - Low complexity" . .	104
7.3	Inference and Regression metrics for model "Best RMSE & MAE" - Medium complexity	105
7.4	Inference and Regression metrics for model "Best MEDAE" - Medium complexity	105

7.5	Inference and Regression metrics for model "Best RMSE & MAE" - High complexity	106
7.6	Inference and Regression metrics for model "MEDAE" - High complexity .	106
8.1	Test Dataset for levels "Low", "Medium" and "High"	107
8.2	Results on test dataset - predicted vs. ground truth for low level using model "Best MEDAE" for a threshold of 0.9	107
8.3	Results on test dataset - predicted vs. ground truth for medium level using model "Best RMSE & MAE for a threshold of 0.4	108
8.4	Results on test dataset - predicted vs. ground truth for high level using model "Best RMSE & MAE for a threshold of 0.2	108
8.5	Regression metrics results on test evaluation	109
8.6	Descriptive statistics associated to the error encountered on the different levels of complexity	109

ACRONYMS / ABBREVIATIONS

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
GA	Genetic Algorithms
GAN	Generative adversarial network
FC	Fully convolutional
IoU	Intersection over Union
mAP	Mean average precision
NN	Neural network
ReLU	Rectified linear unit
RGB	Red-Green-Blue
RFCN	Region-based fully convolutional network
MAE	Mean absolute error
MedAE	Median absolute error

MSE	Mean squared error
R-CNN	Region-based convolutional neural network
RNN	Recurrent neural network
RPN	Region proposal network
ROIs	Regions of interest
SSD	Single shot detector

e.g.	Exempli gratia ("for the sake of an example")
i.e.	Id est("it is")
w.r.t.	With respect to

Over the past decade, deep learning has accomplished notable success in several research areas. Risen from artificial neural networks study field, it has been proven that deep learning can obtain remarkable performance regarding other machine learning algorithms in a wide range of applications, like image processing, natural language processing, voice recognition, among others. In recent years, deep learning has also had a significant role in medical research, showing promising results in several fields, from medical imaging to drug discovery. This project aims to contribute further to the application of deep learning architectures to medical imaging research, by exploring the possibility of creating an automatic counter of cells in microscopic images, a frequently needed task in several medical fields as microbiology, molecular biology, histopathology, and cancer research.

1.1 Thesis objective and research questions

The present project aims to develop a deep learning system that allows to detect accurately and quantify the number of human cells in microscopic images of zebrafish xenografts. Taking into account the available data and the characteristics of it, the application of an object detector was selected as a viable solution in order to achieve the main goal.

During this research, it was also tested the significance of several parameters in the performance of the model to the problem in hand. Due to the particularities of the dataset and the need for final high accuracy in the counting, several modifications to parameters and hyperparameters were considered during the process.

While most object detection studies are applied in images where the objects respect some basic conditions, like the non overlapping problem and the size of the objects itself, this project appears as a contribution where it is shown that those types of difficulties can be surpassed, or at least, minimized, using several strategies and approaches and by fine-tuning some parameters to values adjusted to the problem in hand.

In this way, this thesis proposes to answer the following questions:

1. How well can CNNs perform on medical image recognition?
2. How to deal with situations of overlapping objects and small objects by using an object detection algorithm?
3. What is the impact of changing the different parameters and hyperparameters in an object detector architecture?
4. Is it possible to use object detection for counting objects in images where the number of objects is quite significant (more than one hundred in average)?

1.2 The task of counting cells in medical imaging

The process of counting cells in microscopic images is a frequently needed task and of significant interest in the medical research field, but usually still made manually by scientists, which leads to a laborious task, time-consuming and prone to human error. Moreover, manual cell counting is a subjective task due to a considerable inter-observer and intra-observer variability [4]. Consequently, the automation of this process is a long desired tool among the medical community and has been approached using different techniques over the last years.

Cell counting can be applied to a vast number of applications in life sciences [54], to name a few:

1. In biology and medicine, counting red and white cells in blood allow to make inferences regarding a patient's health;
2. In clinical pathology, the quantification of cells can be used to investigate the hypotheses of pathological processes, and to evaluate the response of a patient's immune system to an infection;

3. In molecular biology, assess cell concentration should be done as an insight to adjust the chemicals' amount to apply in an experiment;
4. In cancer research and diagnosis, cell counting allow to appraise the proliferation rate of a tumor and evaluate the therapy response.

The most ancient approaches for cell counting were based on classical image processing techniques, by using methods like intensity thresholding, feature detection, morphological filtering, region accumulation, and deformable model fitting [31].

With the rise of deep learning and CNNs, several authors had proposed different techniques that could surpass the most traditional approaches. In [56], the authors propose a CNN-based detection method that uses encoding of the output pixel space using compressed sensing. In [18], the authors apply deep learning architectures in order to segment and count cells on microscopic images, but as mentioned, their system fails in situations of high cell overlap and irregular cell shapes. On [35], the authors generate density maps which they sum over to calculate a final count. Other common and simple approach is the area based counting [41], that calculates the number of objects on the total area of objects of interest by using the average size of an individual cell, where the area of all cells is divided by the average size of a cell in order to obtain the number of cells in the image. The problem associated with this last approach is that if dealing with overlapping objects, the total pixel coverage will sometimes be significantly less than the actual coverage, leading to false results. In [54], the authors try to automate the process by applying fully convolutional regression networks in order to regress a cell spatial density across the image, achieving good results regarding the accuracy of the model. However, the proposed approach was tested on images that did not suffer a high density of overlapping.

When dealing with overlapping cells, some recent studies try to apply different procedures to be able to count each cell individually. On [33], the individualization of the cells is achieved based on the contour concavity of the same. Again, this could solve situations of overlapping cells, but in situations that are considered simple to deal with. As so, the task of counting cells can be dealt with in a vast majority of the cases using the approaches mentioned before. However, in situations where there is a high overlap of cells, poor quality imaging, non-uniform cells with different shapes or textures, among other aspects the task became more difficult if not impossible.

Another condition to take into account is the type of labeling data that is available. Depending on the labels, different approaches can and should be used, and in the most approaches presented in the mentioned articles, the ground truth was labeled in a pixelwise level, allowing to apply segmentation, one of the most common approaches in similar situations.

Similar environments can be considered when searching for the best approach to detect and count cells. Taking into account the nature of the problem, cases like fruit counting [41] and crowd counting [57] [55] [45] can also have meaningful insights, due to the similarity of the problem.

This project aims to design a deep learning procedure where the major goal is to count accurately human cells in microscopic images. The proposed strategy is the creation of an object detection model that is robust enough in order to solve this problem. Besides the task of individualizing and detect cells, we are dealing with images where not only human cells are present. In this way, it is necessary to complement the procedure with a classification task that can classify accurately what are the objects of interest.

To our knowledge, there is no state-of-the-art object detector architecture for counting overlapped cells in an image.

1.2.1 The pertinence of counting cells in zebrafish xenografts

The present project was developed as a collaboration with Champalimaud Centre for the Unknown, whose mission is to "develop programmes of advanced biomedical research and provide clinical care of excellence, with a focus on translating pioneering scientific discoveries into solutions which can improve the quality of life of individuals around the world.¹" As center of several researches regarding to cancer treatment, in 2017 a group of researchers of the foundation submitted a paper [6] where they promote zebrafish larvae xenotransplants as a promising approach in evaluating the response of different treatments in cancer cells, and as an innovative technique that offers speed and the need of smaller samples of transplants comparing to mice tumor transplantation. In order to evaluate the therapy response, each of those transplants is sliced, and around three slices for each sample is analyzed. For each slice, the number of human cells is counted in order to achieve the perception if the treatment is having any effect on the cancer cells.

¹<http://first.fchampalimaud.org/en/the-foundation/mission/>

The process of counting cells is, however, a bottleneck concerning speed to the approach developed. Due to the anatomy of the cells, simple pre-processing techniques do not allow to have an accurate counting of the cells and demands a manual counting which turns out to be a laborious and tedious task.

1.3 Report structure

The present document is structured in ten chapters, including the Introduction:

1. Chapter 2 introduces the main concepts associated with machine learning, deep learning, convolutional neural networks, and focus on the object detection systems used;
2. In Chapter 3, a further exploration of the Faster R-CNN model is made, where the different stages of the algorithm are detailed, and the parameters and hyperparameters corresponding to those phases are defined;
3. In Chapter 4, a brief exploration of the raw data and the main statistics associated is performed with the purpose of not only to understand potential risks and problems that we can deal during the project but also extract meaningful insights that can be useful during experiments;
4. In Chapter 5 tensorflow object detection API is introduced, and the documents/files needed are detailed in order to train and make inferences, at a high level;
5. Chapter 6 describes different experiments, from the selection of the architecture and the feature extractor till the fine-tuning process, with technical details and main conclusions regarding the results;
6. Chapter 7 gives a quick overview of the losses associated to the final model, and the best score threshold to use in the final solution for the different sets of image complexity is evaluated and assessed, i.e., the level of confidence where we are going to consider an object as an object of interest;
7. In Chapter 8, the evaluation of the performance of the final model is made on test dataset;

8. Chapter 9 presents the user-friendly tool built with the aim of making inferences regarding the number of cells in microscopic images in a simple and accessible way;
9. Chapter 10 concludes this thesis naming the goals achieved, limitations and future work.

2.1 Machine Learning and Neural Networks

The field of machine learning is focused on developing computer programs that automatically improve with experience [32]. The concept was defined in 1959 by Arthur Samuel as "the field of study that gives computers the ability to learn without being explicitly programmed" [47]. In the last two decades, the machine learning activities evolved significantly due mainly to the increase of computational power and GPU acceleration, and the rise of large repositories of data, being almost ubiquitous nowadays.

In the past decade, machine learning was responsible for the emergence of automatic cars, speech recognition tools, web optimization and other numerous applications across several fields of study. Regarding machine learning systems, we can classify them taking into account the amount and type of supervision they get during training into four different concepts:

1. **Supervised learning** - The training data includes the ground truth, i.e., the desired solutions, also called labels. Those labels were usually defined by humans, and the algorithm is able to predict the label for the unseen data. Common tasks in this field are classification and regression. In the case of object detection, training images that were previously annotated mark the locations and the classes of meaningful objects.
2. **Unsupervised learning** - In unsupervised learning, the training data is unlabeled, and the system tries to learn without guidance. The most relevant algorithms applied in this field are clustering, the visualization and dimensionality reduction,

and the association rule learning. The major goal of unsupervised learning is to discover patterns in unlabeled data.

3. **Semi-supervised learning** - In semi-supervised learning, the training data is composed partially (usually in small proportions) with labeled data. It is a combination of both labeled and unlabeled data to train the model.
4. **Reinforcement learning** - In reinforcement learning, the learning system, denominated as agent, select and perform actions in an environment, getting rewards in return, or penalties as negative rewards, which allows him to learn by itself, and able to identify the best strategy possible in order to get the most rewards. In this field, we can include genetic algorithms, swarm intelligence and many other disciplines.

2.1.1 Neural Networks and backpropagation

The most basic structure of the artificial neural network (ANN) is the linear perceptron. Resembling the brain neurons, an ANN is composed of nodes linked to each other whose function is to receive, process and transmit information.

A linear perceptron receives several inputs and performs a weighted summation to produce an output. The weights are determined during the training process and based on the training data. This training can be done through gradient-based methods, and the output of the perceptron can be passed through an activation function [49].

In order to deal with non-linear separable patterns, one of the limitations of the single perceptron, the multi-layer perceptron (MLP) was developed by adding more layers to the network. This class of ANN uses a supervised learning algorithm denominated as backpropagation to learn. The learning is achieved during the training, which is composed of two main phases:

1. **The forward pass** - The weight of each connection and the biases are already defined, and the input signal is propagated through the network until it ends. In the output, the error is calculated by comparing the output value of the network with the ground truth.
2. **The backward pass** - The error is propagated backward through each network layer, and adjustments are made to the weights and the biases in order to minimize

the error, by using gradient descent, an iterative optimization algorithm, which role is to minimize the cost function.

2.1.2 Generalization of a model, underfitting and overfitting problems

The power of a learning algorithm is highly associated with its capability to generalize, i.e., to handle unseen data. There is no use of creating a model that is only able to deal with the instances that were used for its training. Together with the generalization concept, we can introduce two situations that are common while training a model: underfitting and overfitting. The underfitting can occur when the model trained is too simple to learn the underlying structure of the data, and fails to capture significant variables of the reality of the situation that we are trying to model. On the other hand, overfitting happens in the other way around, usually in the presence of complex models, giving more relevance to unimportant details and noise. Both situations lead to cases of lousy generalization, and they should always be avoided.

In case of underfitting, several strategies can be applied in order to fix it, like selecting a more powerful model, with more parameters, feeding better features to the learning algorithms (a.k.a. feature engineering) [8], or reducing the constraints on the model, by reducing, as for example, the regularization hyperparameter. In the case of overfitting, some strategies can be applied, such as simplification of the model, reduction of the number of parameters (the use of more parameters tend to lead to a model prone to overfitting), gathering more training data (using, for example, data augmentation techniques), among others.

The performance of an algorithm is measured by the quality and quantity of the errors. In order to measure this, one or more loss function are used, that could differ taking into account the type of the problem that we are dealing with. The primary goal during the training phase is to decrease as maximum as possible this loss. [8]

2.1.2.1 Avoiding overfitting

When dealing with neural networks, we typically have thousands, if not millions, of parameters. The advantage of this is that the networks gain a huge amount of flexibility and can fit several problems and complex datasets. On the other way around, the great amount of freedom comes with the disadvantage of easily overfitting the training set.

In order to prevent or to delay the overfitting when training, several approaches were

proposed: early stopping, l_1 and l_2 regularization, dropout, max-norm regularization, data augmentation, transfer-learning, reducing the number of parameters by reducing the complexity of networks and others. Usually, different methods are combined in order to prevent overfitting.

Early stopping As the name suggests, the early stopping is the simple technique of stop training our model as soon as we achieve a status of overfitting, i.e., as soon as the performance on the validation set starts dropping.

l_1 and l_2 regularization The l_1 and l_2 regularization will allow to constrain a neural networks' connection weights.

The L1 regularizer, as presented in equation (2.1), adds a penalty term to the cost function using the sum of absolute values. The L2 regularizer, shown in equation (2.2), adds a squared magnitude of coefficient as a penalty term to the loss function.

$$Cost = Lossfunction + \lambda \sum_{j=0}^M |W_j| \quad (2.1)$$

$$Cost = Lossfunction + \lambda \sum_{j=0}^M W_j^2 \quad (2.2)$$

The main difference between L1 and L2 regularization is that the first one shrinks the less important feature's coefficient to zero, allowing to remove those features, working as a feature selector. On the other hand, the L2 regularizer allows achieving a stable solution, even not so robust as the L1 regularizer [49].

Dropout The dropout technique was proposed by G.E. Hinton in 2012 [19] and detailed by Nitish Srivastava et al. [51].

In the dropout technique, at every training step, every neuron (including the input ones) have a probability associated of being "dropped out" in that step, i.e., to be ignored, but can be active on the next one. The dropout technique is defined by the hyperparameter p (the dropout rate). The dropout is only applied to the training phase.

Max-Norm Regularization The max-norm regularization enforces an absolute threshold on the magnitude of the weight vector for each neuron, using projected gradient descent to enforce the constraint. In this way, if the weight exceeds a specific value, the

max-norm regularizer will scale the whole weight matrix by a factor in order to reduce the norm to the value of the threshold defined. Contrarily to L1 and L2 regularizers, in max-norm the regularization is made directly and not by penalization. [48].

Data augmentation The data augmentation is the process where new instances are created from the existing ones (usually on the fly in order to avoid waste of storage). Several techniques can be applied, like rotation, zoom, change in channel colors, change on the hue and saturation. Those approaches will introduce noise to the training process and force the model to be more tolerant to the possible changes in the images like position, orientation, and size of the picture, allowing the model to become more robust. On the other hand, this could be a useful process when dealing with smaller datasets, since it artificially boosts the size of the training set.

2.2 Deep learning

Deep learning can be considered as a subset of machine learning algorithms that uses artificial neural networks. It has a diverse and wide range of applications, and seek to simulate human brain processing in tasks as visual recognition, natural language processing, and speech recognition. This concept emerges from the necessity of adding more layers to neural networks as a way to deal with more complex problems. However, adding more layers came with the problem of vanishing gradients, where the gradient tends to get smaller on the backward process, leading to poor solutions. The inclusion of innovative activation functions appears as a way to address this problem, allowing to add more layers to a network, and leading to the concept that nowadays is called Deep Learning.

2.2.1 Deep learning for computer vision and the object detection problem

By applying deep learning to the field of computer vision, the aim is to simulate the properties of the human vision on a computer. Different problems arise from this field, that can be solved using different techniques of deep learning.

Image classification Image classification takes as input an image and predicts the object of that image. The goal is to label the whole image with an object or concept.

Object detection Object detection goes further, and introduces the concept of location of that object in the image. In that way, object detection at a high level can be seen as a mix between two tasks: predict bounding boxes constituted by 4 variables (x and y coordinates, width and height) that will allow to uniquely identify a rectangle that circumscribes the object and the classification of the object inside the selected bounding boxes. The object detection approach is modeled as a classification problem where several windows of fixed sizes in the input image are defined at different locations, and those patches are fed into an image classifier.

In order to create an object detection model, the first step is to have labeled data. In the present context, we refer to labeled data annotations of the coordinates of bounding boxes that correspond to a specific object we want to detect and the labels associated.

Semantic segmentation The segmentation problem can be seen as an extension to the object detection problem since it is considered a pixel-wise classification task. The most known architectures to solve this problem at this moment are SegNet, DeepLab, RefiNet, PSPNet, UNET, among others [49].

Similarity learning When applying deep learning on the task of similarity, the aim is to learn how two images are similar. A few architectures have been proposed for this purpose, namely FaceNet, DeepNet, DeepRank, among others [49].

Image captioning In image captioning, the goal is to describe an image with text. In this case, techniques of natural language processing and computer vision are combined [49].

Generative models The purpose of generative models is to generate images as the name implies. Besides creating new images using style transfer, other purposes can exist when using this kind of technique: creating new images to serve as a sample for training, create super-resolution images and others [49].

Video analysis Video analysis can have several applications like sports tracking, intrusion detection, and surveillance cameras [49].

2.3 CNNs

Convolutional Neural Networks (CNNs), a subcategory of ANNs, have been proven highly effective in several areas such as image recognition and classification. On computer vision field, an image is taken as input, and that image will be passed through a series of convolution operations that ends up in a subsequent classification. These individual layers will learn particular structures to extract, and those features will become more complex as deeper as they go through the network, starting from simple edges to complex textures and arrangements.

2.3.1 The biological inspiration

CNNs have as basis inspiration the visual cortex in animals, and its history begin with the neurophysiologists David Hubel and Torsten Wiesel that were able to determine the most basic facts about mammalian vision system works, and to identify the nucleus in the visual cortex whose job was to detect features like edges from the visual information, and denominated as primary visual cortex(V1) [14]. Higher level nucleus tends to detect more complex features, as seen in figure 2.1.

The cells in the visual cortex are sensitive to small sub-regions of the input, and those smaller sub-regions are tiled together to cover the entire visual field. In convolutional networks, the idea is similar: an architecture is designed that resembles the organization of information in the human visual system simplistically. We try to extract features from an image and then classify those features.

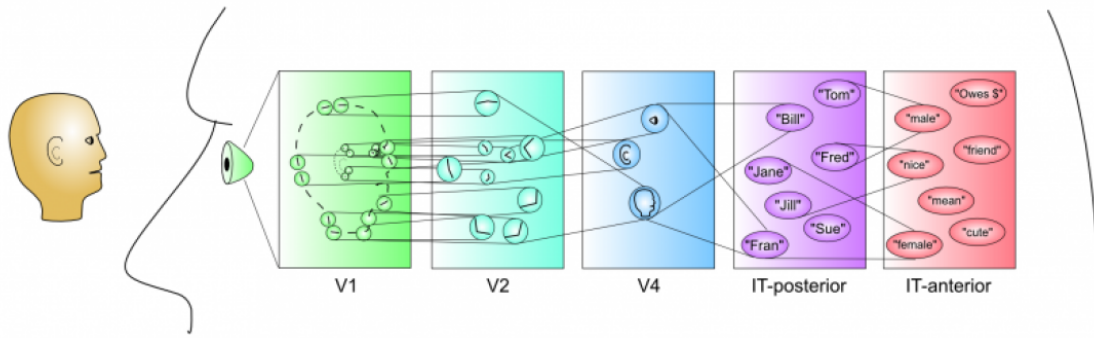


Figure 2.1: Schematic of a hierarchical sequence of categorical representations processing a face input stimulus. In the first level, detection of elementary structures is made, like edges and lines. The higher the level, the more complex will be the visual features, which are arranged by combining processes, until the last level where "semantic" categories are associated with detections [36].

2.3.2 The architecture of a CNN

In a CNN we deal with an input and an output layer, as several hidden layers, as shown in figure 2.2. The input layer is going to be input images that we are going to analyze. The hidden layers are usually composed of convolutional layers, pool layers, and fully connected layers.

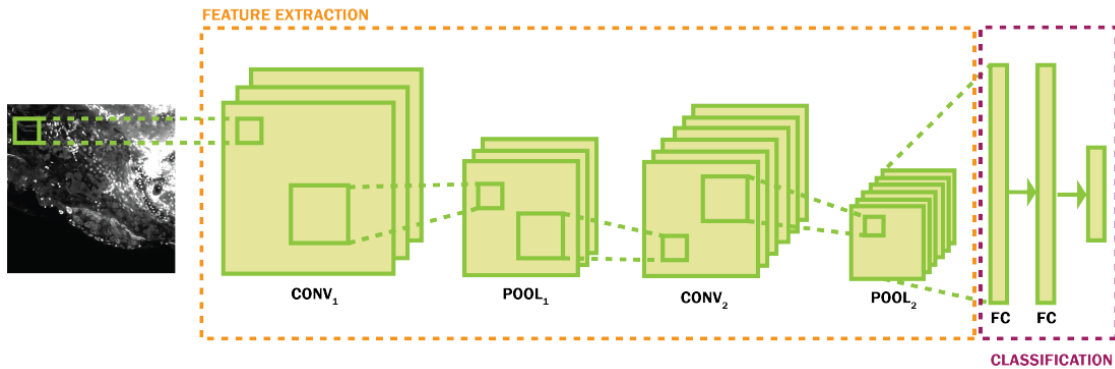


Figure 2.2: The main components of a CNN

In the following section, each type of layer is defined in further detail.

2.3.3 Main layers in a CNN

Input layer The input layer is an image with the dimensions [width x height x depth]. The depth corresponds to the color channels of the images: in the presence of a color image, it is constituted by three channels (RGB), while on a grayscale image the depth

will be one. We can consider an image as an array of pixels where the elements are values that range between 0 and 255. During training, each input image will be passed several times through the network.

Convolutional layer Convolutions are one of the essential operations in image processing. In an abstract level, convolution is a mathematical operation that allows merging two sets of information. We can apply convolutions in inputs of 1D (e.g., speech processing), 2D (e.g., image processing) and 3D (e.g., video processing). When applied to images, the primary purpose of convolution operations is to extract features, and in that way, we can consider the convolutional layer as the core buildings of CNNs.

A convolutional layer is a layer that applies different convolutions in the data that receives. Each convolution operation is composed by a kernel/filter, a matrix smaller than the original input in height and width and extended over the entire depth of the input volume, that will slide through the input horizontally and vertically. The stride will define this movement behavior, and the matrix obtained after this process is denominated as a feature map. The more different filters used, the more features are extracted, which can have a high impact on the performance of the model but will also add constraints to the practicability of the training regarding speed and complexity. As so, the size of the feature map is defined by:

1. **Depth / Channels(K)** - The number of filters used. One convolutional layer can have multiple input channels and multiple output channels (a 2D matrix).
2. **Stride (S)** - The number of pixels that a kernel slides over the input matrix. The convolutional layer can have different values of stride for horizontal and vertical sliding, but commonly they assume the same value.
3. **Zero padding (P)** - The process of padding the input matrix with 0s around the border, a strategy often used to maintain the height and the width for the output matrix (and consequently build deeper networks), and that improves performance by keeping information of the image borders [14].
4. **Filter size (F)** - The size of the filter refers to the dimensions of the filter/kernel in the convolutional network.

The dimension of the output convolution concerning the width (W_{out}), height (H_{out}) and depth (D_{out}) can then be calculated as [1]:

$$W_{out} = \frac{W_{in} - F_w + 2P}{S_w} + 1 \quad H_{out} = \frac{H_{in} - F_h + 2P}{S_h} + 1 \quad D_{out} = K$$

where w and h stand for width and height, respectively. In figure 2.3 we can verify the different effects obtained by using a 3x3 kernel with different values to a input image.

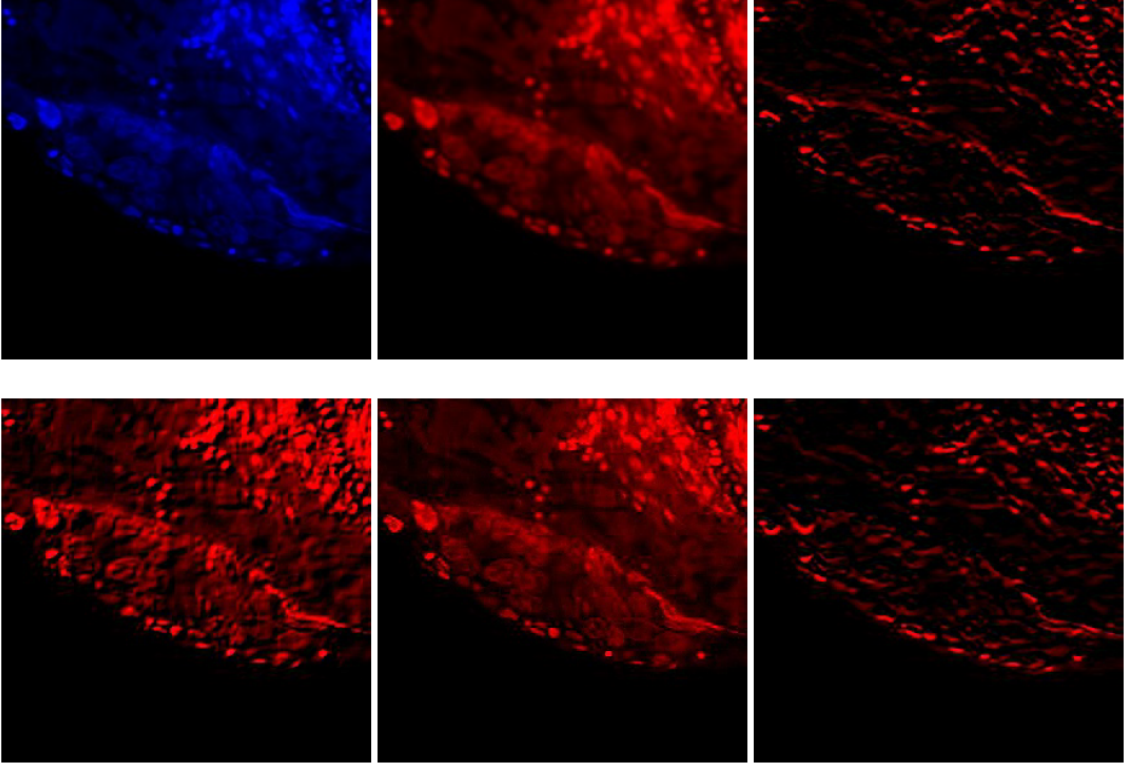


Figure 2.3: Different 3x3 kernels applied to the raw image. From left to right: Top - Raw Image, $\begin{bmatrix} -0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$, $\begin{bmatrix} -1, 2, 1 \\ 0, 0, 0 \\ 1, 2, 1 \end{bmatrix}$; Bottom: $\begin{bmatrix} -2, -1, 0 \\ -1, 1, 1 \\ 0, 1, 2 \end{bmatrix}$, $\begin{bmatrix} 0, -1, 0 \\ -1, 5, -1 \\ 0, -1, 0 \end{bmatrix}$, $\begin{bmatrix} 1, 2, 1 \\ 0, 0, 0 \\ -1, -2, 1 \end{bmatrix}$

Pooling layer The pooling layers are responsible for reducing the dimensionality of the convolutional layers output and try to make the representation robust to small input translations and variances. In convolutional neural networks, the pooling layers are usually inserted between successive convolutional layers in order to reduce the dimension of the feature maps and consequently reduce the number of parameters and computation in the network in a way that also allows controlling overfitting. [39] The most common approach used in pooling is max pooling, i.e., for each tile, it is computed the maxi-

mum value in the tile, and that maximum value is propagated to the corresponding cell of the condensed feature map. [1] The average pooling, as the name implies, calculates the average value taking into account the kernel size and the stride, as seen in figure 2.4. A pooling layer can be defined through two parameters: its spatial extent E and

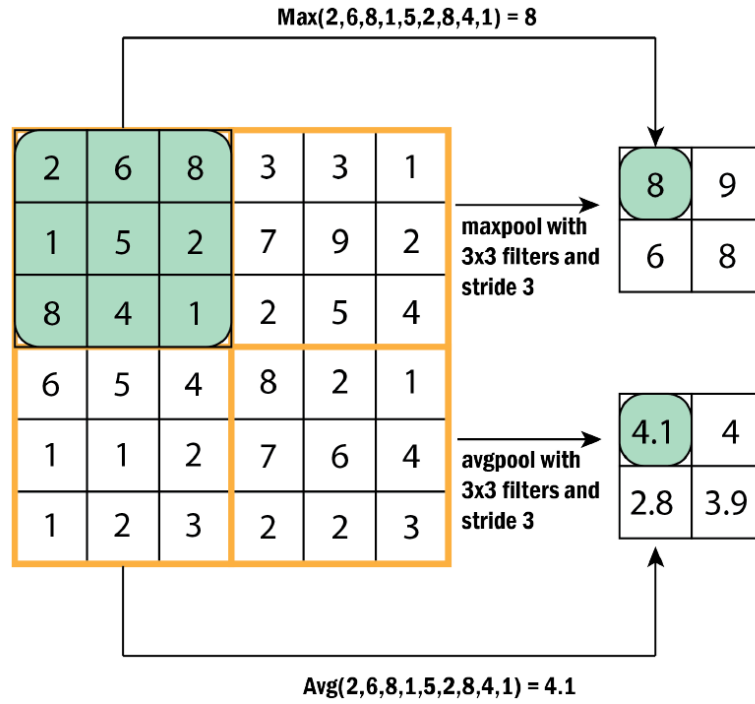


Figure 2.4: Illustration of how max pooling and average pooling significantly reduces parameters as we move up the network

the stride S . We can deal with two different solutions: a nonoverlapping pooling layer where E is equal to s , and a overlapping pooling layer where E is higher than S . The final dimensions for the feature map [1] will be :

$$W_{out} = \left\lfloor \frac{W - E}{S} \right\rfloor + 1 \quad H_{out} = \left\lfloor \frac{H - E}{S} \right\rfloor + 1 \quad D_{out} = D_{in}$$

here W stands for width, H for height and D for depth.

Linear or fully connected layer The fully connected layer is a traditional multi-layer perceptron, where all neurons in the previous layer are connected to every neuron on the next layer. It is placed on the end of the network with the purpose to use the features obtained in the previous layers in order to classify an object based on the training dataset. Apart from classification, the FC layers also allow learning non-linear combinations of

those features. The dimension of the output volume is $[1 \times N]$, where N is the number of output classes that we are evaluating. Each value in N will represent the probability of an object belonging to a specific class. This probability is achieved because the FC layer look at which high level features most strongly correlate to a specific class, and since it has access to the weights of the previous layer, it will compute their weights and the weights of the previous layer in order to obtain the probabilities for each class.

Fully connected layer in object detection problems The dimension of the fully connected layer can differ significantly accordingly to the type of problem we are dealing with. In object detection problems, the output size is going to be the number of classes existent in our dataset (in the project in hand since we are dealing with only one class, the output is going to be just one), and four values for the coordinates of the bounding box. In this way, the final output for each bounding box and classification is going to be in the form of $[p_i, x_i, y_i, w_i, h_i]$, where p_i corresponds to the probability value that the detected object i to belong to the class, and the other parameters to the coordinates of the bounding box.

2.3.4 Activation functions

The activation functions allow passing the output of one layer's node forward to the next layer. In hidden neurons, the activation functions introduce nonlinearity into the networks modeling capabilities. [39]

The non-linearity is an essential aspect to assure that the neural network turns out to be powerful. There are several activation functions, as seen in figure 2.5, that we can apply with this purpose, and while sigmoid and tanh functions were widely used, the ReLU activation in the last years became one of the most used since it has been proved its higher efficiency.

Sigmoid function Sigmoid can convert any value to probabilities and is used on binary classifications. When using the sigmoid function, in the presence of really small logits, the output of a logistic neuron is very close to zero, and if too large, the output will be close to one. In this case, the output values will range from $[0,1]$. The sigmoid function has the following mathematical form:

$$y = \frac{1}{1 + e^{-x}}$$

Tanh function The tanh is a hyperbolic trigonometric function with the following mathematical form:

$$y = \frac{2}{1 + e^{-2x}} - 1$$

The normalized range of tanh function is -1 to 1. The major advantage of this function is that we can deal easily with negative numbers.

Rectified Linear Unit function The RELU activation function has the following mathematical form:

$$y = \max(0, x) \quad (2.3)$$

This function activates a node only if the input is above a certain value. When the input is below zero, the output is zero, but if the input rises above a defined threshold, it has a linear relationship with the dependent variable.

The ReLU has become the activation function of choice in computer vision tasks [1].

Rectified linear Unit Function 6 The RELU6 function is similar to RELU, but defines the maximum threshold as the value 6, and has the following mathematical form:

$$y = \min(\max(x, 0), 6) \quad (2.4)$$

The value six is defined as the best threshold according to [26], where it is confirmed that this value encourages the model to learn sparse features earlier than an infinite amount.

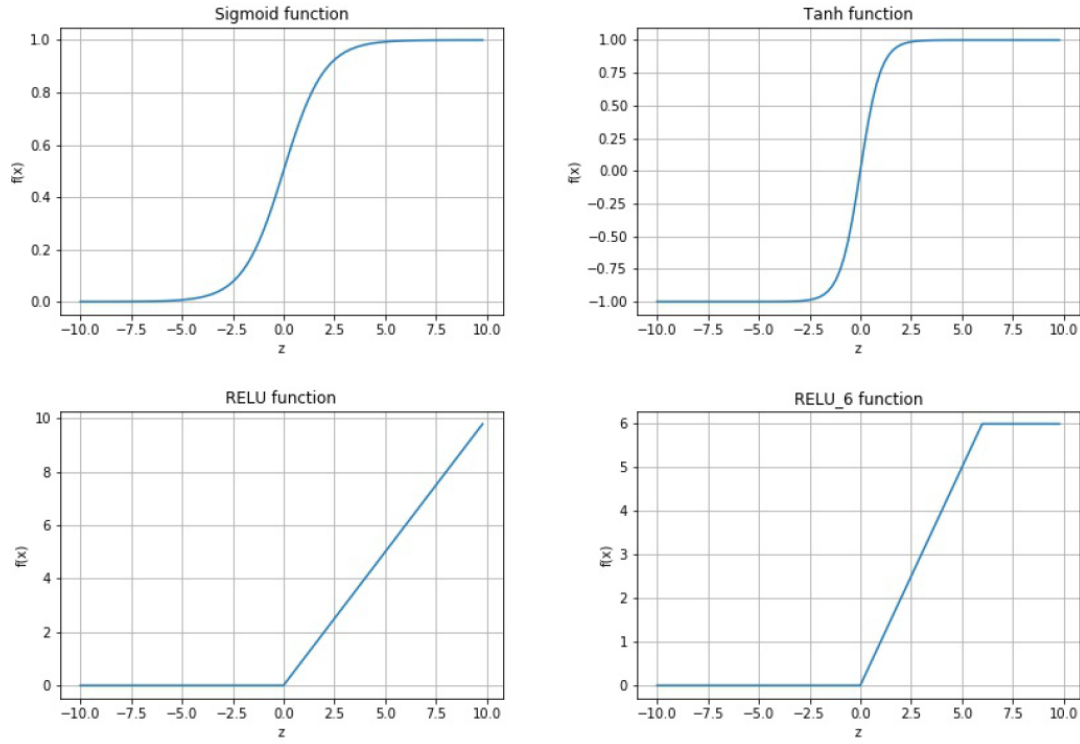


Figure 2.5: Activation functions widely used in ANN

2.3.5 Batch normalization

The batch normalization allows normalizing the activations of the previous layer at each batch, by keeping the mean activation close to 0.0 and the activation standard deviation close to 1.0.

It was also shown that batch normalization speeds up training since due to the normalization effect we can apply higher learning rates, and reduce the sensitivity of training towards weight initialization, acting like a regularizer. [39] The batch normalization is an approach introduced by Ioffe [21] in order to try to reduce the problem of internal covariate shift, defined "as the change in the distribution of network activations due to the change in network parameters during training", on a mini-batch level. By acting similar to normalization in the input layer, batch normalization normalizes the input of a previous activation layer by scaling it using the mean and the variance computed on that mini-batch, which will allow to speed up training. Batch normalization also allows to use higher learning rates, since problems like vanishing and diminishing gradients are addressed, and reduce the concerns associated with initialization parameters. As a plus, acts as a slight regularizer in overfitting situations by adding some noise, while achiev-

ing better accuracy than a model without batch normalization. The regularizer effect will be more significant if the size of the mini-batch is smaller since the noise introduced on the mean and the variance will be higher [34].

2.4 The meta-architecture

The object detection problem is being dealt using several strategies in recent years. As cited in [20], to compare the efficiency of the different systems based on articles and papers is a hard task due to the different base features extractors used, different image resolutions and different software and hardware platforms. However, it is possible to distinguish those systems taking into account the trade-off between speed and accuracy. In this project, the focus goes to three recent meta-architectures where the values in speed and the accuracy are distinguishable: SSD, Faster R-CNN, and RFCN.

While choosing the meta-architecture for a specific problem, the important question is not which detector is the best one, but which detector and the different configurations that we can define on it are the best in order to have the most optimal balance between the speed and the accuracy that our problem requires.

The main difference between the meta-architectures exposed relies on the origin of bounding boxes that will allow detecting each object.

2.4.1 SSD [30]

The single shot detector meta-architecture, presented in figure 2.6, was proposed by Liu et al. [30] and refers to an architecture that uses a single feed-forward convolutional network to predict classes and anchor offsets without the need of a second stage for classification, as it is found in Faster R-CNN and RFCN. This characteristic raises the speed term while decreasing the accuracy of the model, which in cases like video object detection can be preferable. The accuracy can always be increased, in each of the meta-architectures exposed, by using a more powerful feature extractor.

In 2016, a different meta-architecture denominated as YOLO was proposed by Redmon et al. [42], and is recognized by achieving a speed term even higher than SSD, but not without affecting the accuracy value [30]. Due to the need for high accuracy in the problem at hand, YOLO tests were not considered for this project.

SSD Architecture As mentioned previously, the SSD algorithm uses a single feed-forward convolutional network to detect and classify objects. In order to deal with objects of different sizes, feature maps of different sizes are passed to the final convolutional layer, as can be seen in figure 2.6.

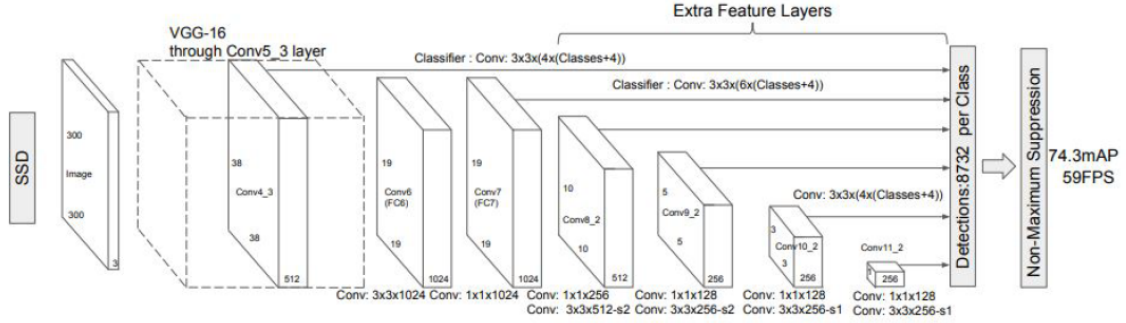


Figure 2.6: SSD architecture where we verify that this model adds several feature layers to the end of a base network, which predicts the offsets to default boxes of different scales and aspect ratios and their associated confidences [30].

Anchor generation in SSD Regarding the generation of anchors, SSD defines a scale value for each feature map layer. In this way, it is possible to define the number of layers to create, but also the scale of the objects to be detected in each layer. In order to apply that, it should be defined the minimum scale and the maximum scale, and SSD will apply for each of the layers a different scale between that range, increasing linearly to the rightmost layer. By combining the scale value with the aspect ratios, it is possible to compute the width and the height of the default boxes. When dealing with small objects, the scales value should be small enough to detect them.

2.4.2 Faster R-CNN [11] [10] [44]

Faster R-CNN was proposed by Ren et al. [44], which was developed based on the architecture of the Fast R-CNN method proposed in 2015 by Girshick [10], which in turn was based on the R-CNN (Regions of convolutional neural networks) method proposed in 2014 [11].

Faster R-CNN is composed of two networks. The first one, denominated as region proposal network (RPN), has the main purpose of generating a set of region proposals where objects could be present. The second network will use the output of the first one in order to try to detect objects in those regions.

In R-CNN, whose workflow is shown in figure 2.7, a selective search method was used to propose regions of potential interest by grouping the color and texture of windows, producing in the end around 2000 region proposals. The main drawbacks were related to the vast quantity of the region proposals which would imply several inferences, the fact that there was no training end-to-end and the requirement of three classifiers, which increases the number of parameters significantly [49].

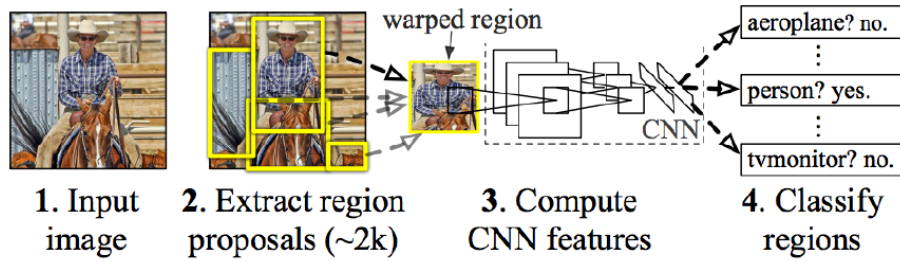


Figure 2.7: Architecture of R-CNN [11]

In Fast R-CNN, presented in figure 2.8, some of the drawbacks of the last model were surpassed. The CNN inference is applied only once, and its output is used to propose bounding boxes. In this way, a new concept was introduced, the Region of Interest Pooling, which takes features from the CNN and pools them in order to select regions of interest. This allows having an end-to-end training, avoiding the multiple classifiers needed in the previous version. The main drawback is the use of a selective search to find regions, which is time consuming [49].

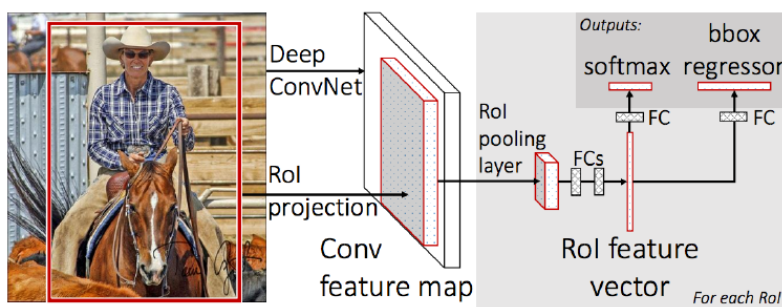


Figure 2.8: Architecture of Fast R-CNN [10]

The Faster R-CNN, presented in figure 2.10, will eliminate the need of using selective search by using CNN features of the architecture for proposals. Those features are passed through the region proposal network, shown in figure 2.9. A sliding window is passed

through different anchor boxes, using different aspect ratios, and outputs a score of the feature [49]. This approach will be detailed in section 3.

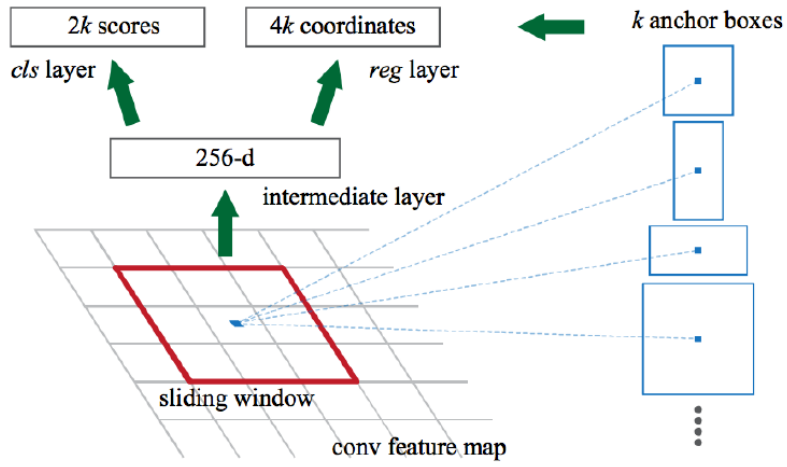


Figure 2.9: The region proposal network system [44]

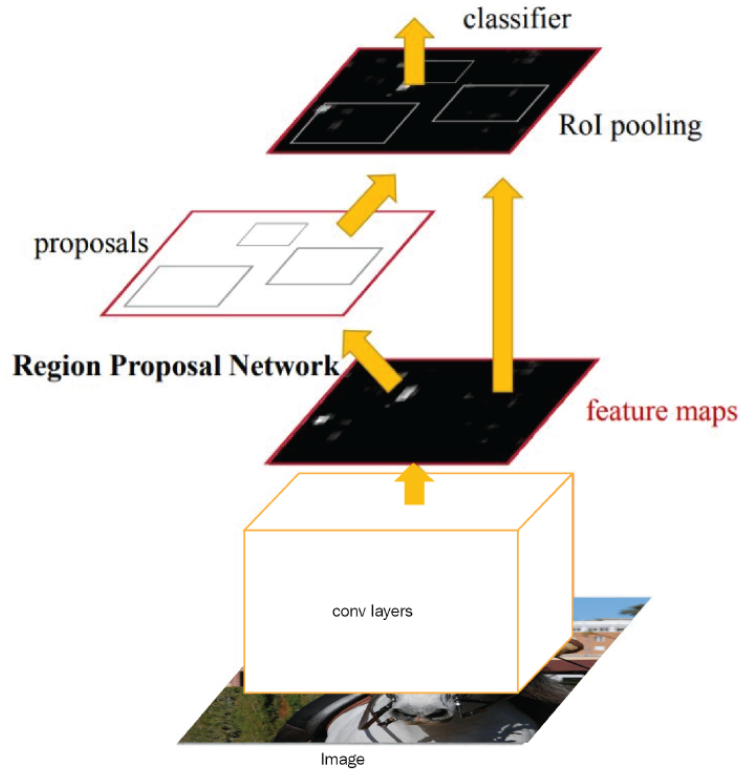


Figure 2.10: Architecture of Faster R-CNN [44]

2.4.3 RFCN [3]

RFCN was presented in 2016 as a new approach that claims to be faster than Faster R-CNN since the region-based detector is fully convolutional with almost all computation shared on the entire image, by proposing a position sensitive score maps to address the problem between translation-invariance in image classification and translation-variance in object detection [3].

2.5 The feature extractor

2.5.1 Classical Networks

LeNet-5 LeNet [28] was proposed in 1998 by Yann LeCun, the actual Director of AI research at Facebook, as a system to recognize handwritten digits. This network took as input images of 32×32 pixels, passing through a convolutional layer (C1), and then to a subsampling layer (S2), which nowadays is replaced by a pooling layer.

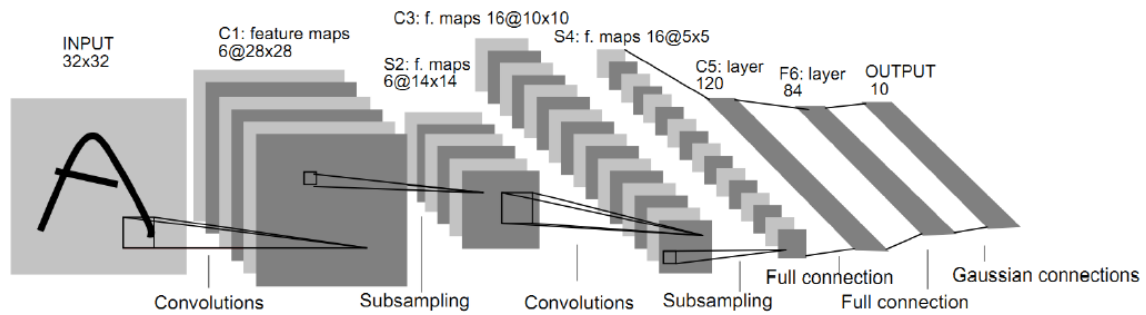


Figure 2.11: Architecture of LeNet-5 [28]

AlexNet AlexNet was proposed in 2012 [27], and introduced the usage of different techniques that proved as useful when dealing with computer vision problems, as the utilization of dropout to reduce overfitting and the implementation of data augmentation techniques. Now obsolete due to the emergence of new architectures more powerful and faster, the AlexNet was a pioneer in this field. With only eight layers, AlexNet has 60 million parameters in total. [49]

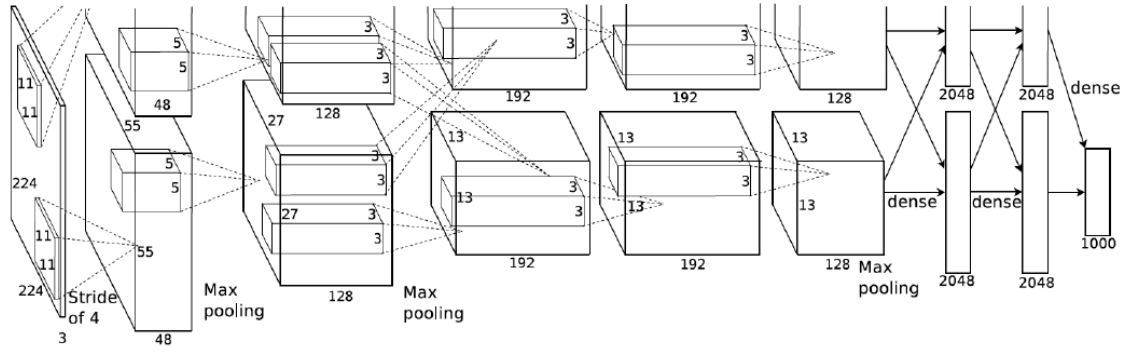


Figure 2.12: Architecture of AlexNet [27]

2.5.2 Resnet

The Residual networks, also known as ResNets, were introduced by Microsoft in 2015, with the architecture ResNet50 [17]. This network introduced the subject of skipping connections in CNNs, a concept denominated as residual blocks, where some activations from one layer are fed into a deeper layer, which allows increasing the number of layers in the network efficiently. [48]

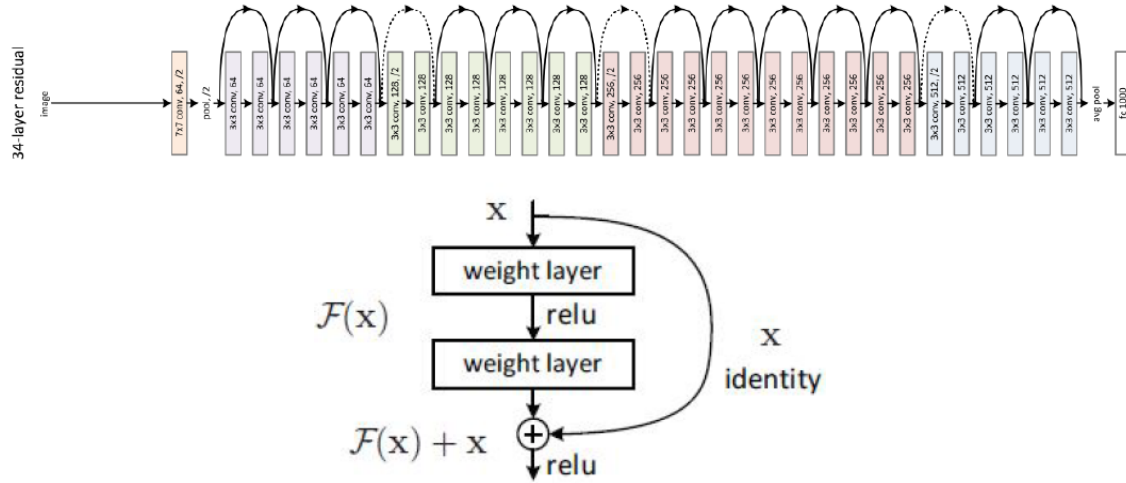


Figure 2.13: Architecture of ResNet 50 and the concept of residual block [17]

2.5.3 Inception

GoogleNet/ Inception [53] was presented in 2015 and introduced the concept of inception module, that reduced the number of parameters, even with 22 layers of depth. The network was constituted by nine inception modules, with a total of 100 layers on them. Inception modules create micro-architectures inside the macro-architecture of the net-

work, where operations happen in parallel, and filters are applied to the output to allow dimensionality reduction [49]. The number of parameters compared to AlexNet was reduced over than 12 times less, and this leads to a decrease in memory usage and an increase in the speed of the training. [48].

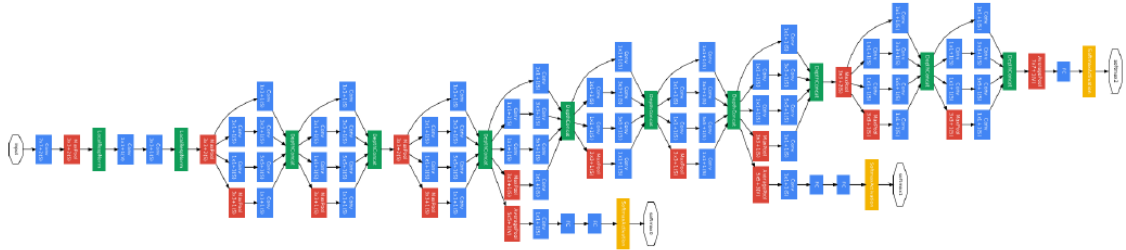


Figure 2.14: Architecture of GoogleNet/Inception [53]

2.5.4 Inception Resnet [52]

The Inception Resnet takes as basis the Inception architecture and introduces a hybrid version with Resnet by applying residual blocks on the architecture. It was found by the authors that Inception-Resnet models were able to achieve higher accuracies at a lower epoch.

2.5.5 NASnet [58]

Nasnet was proposed in 2017, and it was developed by making use of search methods to find good convolutional architectures on a dataset of interest. Inspired by the Neural Architecture Search (NAS) framework, the authors developed a architecture by "searching" a good system on a proxy smaller dataset (CIFAR-10), and then the learned architecture was transferred to a larger dataset (IMAGENET), making use of the concept of scalable architectures, and introducing the concept of flexibility of architectures that can be scaled in terms of computational cost and parameters to address a variety of problems easily. The authors claim that the accuracy of the resulting model exceeds all human-designed models until the article date.

The Faster R-CNN stages, parameters and hyperparameters: a theoretical introduction

3.1 Introduction to Faster R-CNN

3.1.1 Quick overview

Like any other object detection algorithm, the purpose of faster R-CNN is to obtain, from an input image, three types of information:

1. A list of bounding boxes, that will correspond to the detected objects in an image;
2. A label, i.e., a classification of the content, assigned to each bounding box;
3. A probability for each label in a bounding box.

The input images are converted to tensors whose shape will be [height x width x depth] and will be passed through a CNN until an intermediate layer composed by a convolutional feature map. Afterward, a Region proposal Network (RPN) is applied, where the features achieved on the previous step will be used in order to find up a pre-defined number of regions that could contain objects, which will be called Regions of Interest (ROIs). This is achieved by recurring to anchors, which are fixed size reference bounding boxes placed uniformly through the input image. For each anchor, two losses are calculated:

1. The **RPN objectness loss**, where it is measured if a box contains, or not, an object.
2. The **RPN localization loss**, where regression is used in order to evaluate the accuracy between the coordinates (x0, y0, width, height) of a bounding box and the ground truth.

After obtaining a list of possible objects and their locations from the original image, and by using the features extracted by the CNN and the final bounding boxes defined in the RPN phase, all accepted proposals get reshaped to a fixed size, in a process denominated as Region of interest Pooling, in order to extract those features which will correspond to the relevant objects into a new tensor.

Finally, a R-CNN module is applied, where two new losses are computed for each bounding box:

1. The **box classifier localization loss**, where a new regression is calculated in order to improve the fit of the bounding box to the ground truth.
2. The **box classifier classification loss**, where classification of the content is made.

In this way, we can say that the object detection algorithms, and more specifically, the faster R-CNN architecture, is composed of two main stages:

1. Region proposal – While processing an input image, it should identify all the possible locations where objects can be found. The output of this stage will be a list of bounding boxes of likely positions of objects, which we denominate as region proposals or regions of interest.
2. Final classification – For every region proposal defined on the previous step, we apply a deep convolutional network that will allow to classify it as a background or as a specific class of object.

In the first phase, several region proposals should be generated, more than the number of the objects itself on the image. Otherwise, if an object is not detected during this step, it will not suffer the second phase process, and it will not be classified. The high recall needed in this step is achieved by generating a large number of proposals, and most of them will eventually be classified as background in the second phase of the algorithm. This demanding process can be computationally expensive, concerning processing and speed. The described workflow can be seen in figure 3.1.

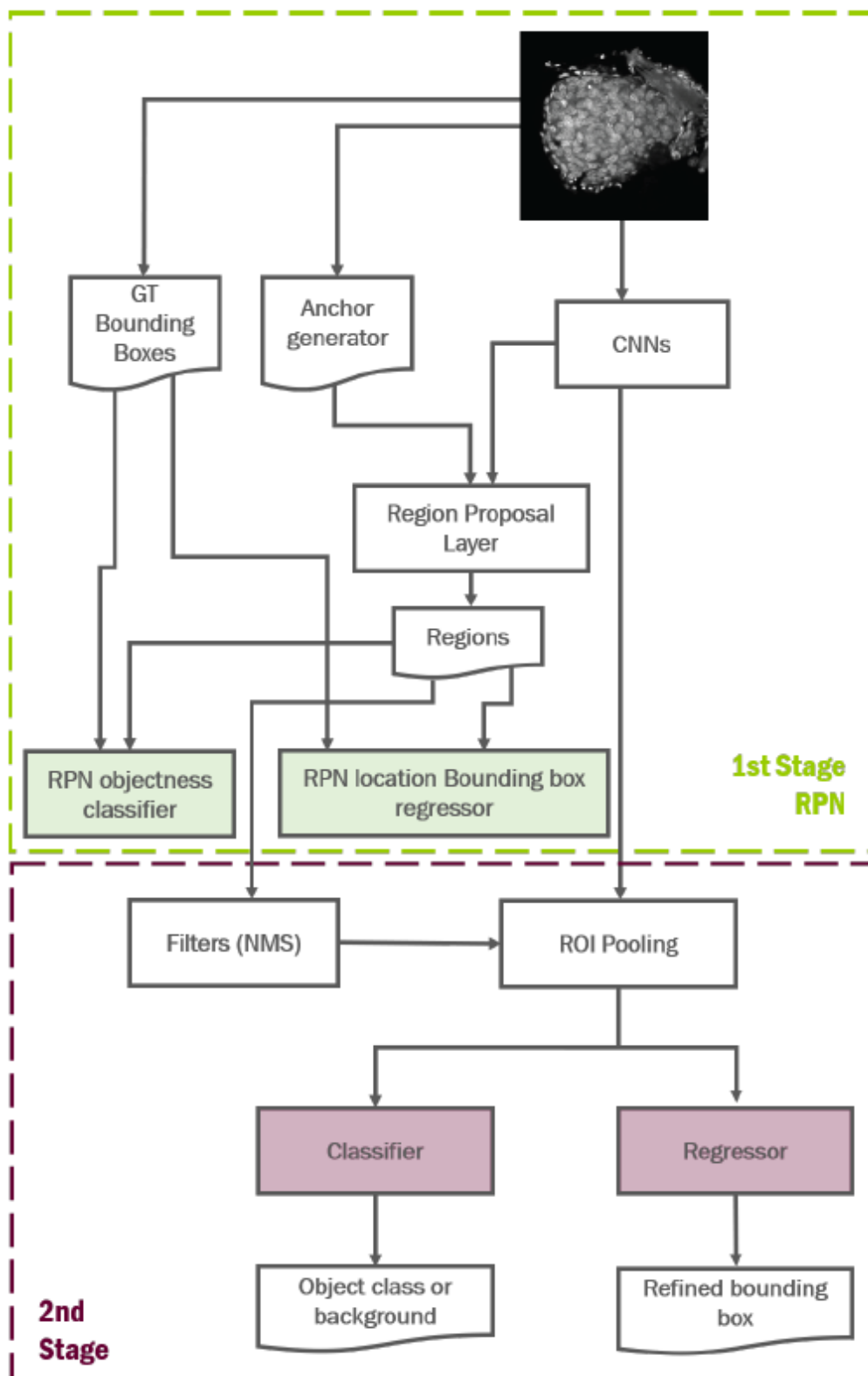


Figure 3.1: Overview of Faster R-CNN process

3.2 Faster R-CNN step by step

3.2.1 The pre-processing

3.2.1.1 Image resizing

When dealing with images of small resolution, the detection accuracy can be significantly reduced. In [20], the authors verify that decreasing resolution by a factor of two in both dimensions reduce accuracy by 15,88% on average, but also lowers inference time by a factor of 27.4% on average. One of the reasons pointed by the authors for this effect is that high-resolution images allow the detection of small objects, one of the main limitations of the existent object detection meta-architectures. The figure 3.2 compares the performance of detectors on large objects (input images of 600x600 pixels) against small objects (input images of 300x300 pixels), confirming that high-resolution images improve the mAP (mean Average Precision) results significantly. This difference is more notorious in meta-architectures like Faster R-CNN, while in SSD the difference is not significant. Figure 3.3 shows the performance for different models on different sizes of objects. It is possible to verify that all methods have a better performance on large objects, with SSD models recognized by their poor performance on smaller objects.



Figure 3.2: Effect of image resolution in the overall mAP using different meta-architectures [20]

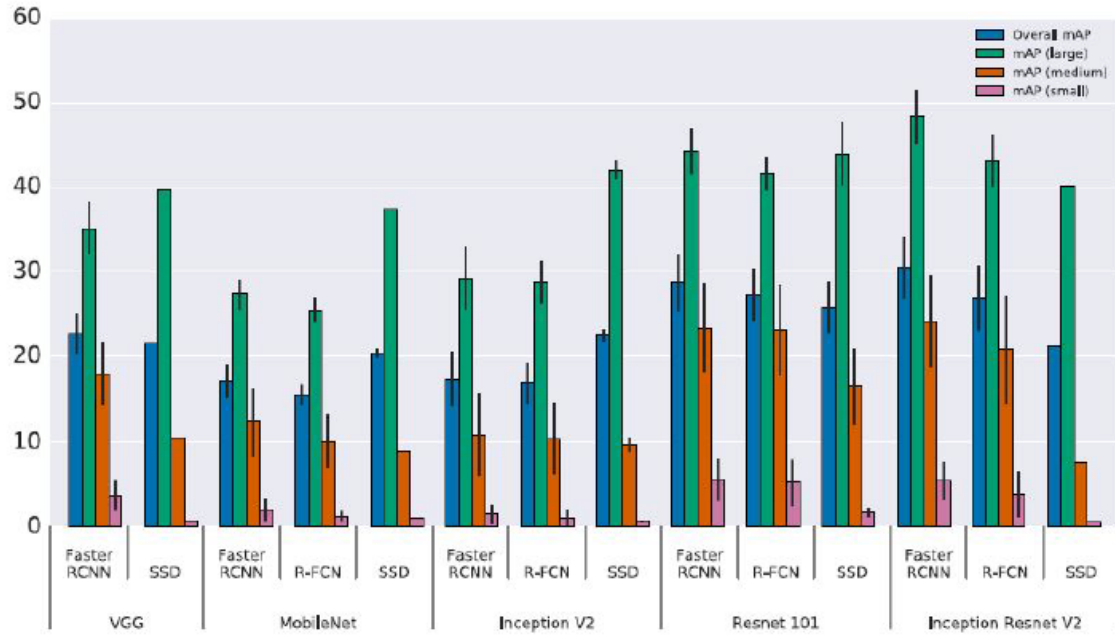


Figure 3.3: Accuracy stratified by object size, meta-architecture and feature extractor, with an image resolution of 300 pixels [20]

3.2.1.2 Resize method

According to [37] and [9], image interpolation is a technique used in order to resize images, where every pixel in the new image would be mapped back to a location in the old image in order to calculate the value of a new pixel. The **nearest neighbor** approach selects the value of the nearest pixel by rounding the coordinates of the desired interpolation point and can create pixelation or stepped edges. The **bilinear** interpolation takes a weighted average of the four neighborhood pixels to calculate the final value, producing smoother images than the original ones. The **bicubic** approach uses the 4x4 neighborhood of known pixels for a total of 16 pixels to calculate the interpolated value, giving more weight to closer pixels, producing sharper images. Finally, the **area** method is similar to the bilinear method, but the coefficient values can vary depending on if the division of the output scale versus the original scale is an integer or not. The different methods are exemplified on figure 3.4.

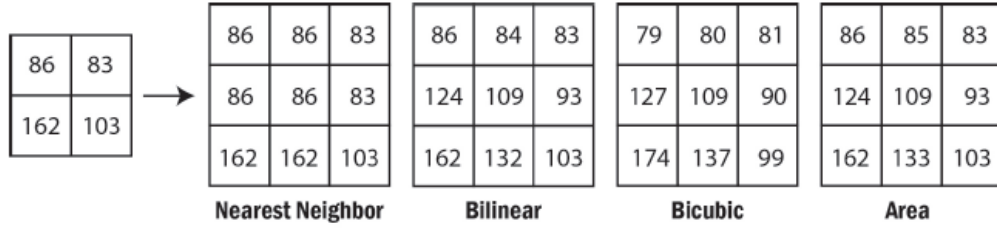


Figure 3.4: Illustration of the interpolation applied using different methods where the input is a 2x2 matrix and the output a 3x3 matrix

3.2.1.3 Image normalization

Image normalization is a technique used in processing images that changes the range of pixel intensity values. This task could in some cases lead to a better convergence on the model. In order to apply normalization on the image, the following formula [13] is applied:

$$I_N = (I - Min) \frac{newMax - newMin}{Max - Min} + newMin$$

For every channel in the image, the pixel values are changed from the range [Min, Max] to [newMin, newMax].

The normalization of an image before the training could be important due to two main reasons:

1. If data is not normalized, pixels with larger values will dominate the ones with lower values, and the contribution of both types of features will not be balanced.
2. Many learning algorithms behave better with normalized data.

However, one important aspect to take into account, if using transfer learning, is if the pre-trained model also received as input normalized images. It is a good practice to maintain the "characteristics" of the input images from the pre-trained models.

3.2.1.4 Converting to grayscale

Many computer vision algorithms can benefit from using grayscale images instead of color ones. The main reason comes from the fact that using grayscale, the depth of the input image will be reduced from 3 to 1, which will be reflected on the training speed. Besides that, in some cases, the color information is not a plus for the detection of objects in images, since features like edges usually have more impact on the model. As in the

case of normalization, when dealing with transfer learning, maintaining the features of the input images from the pre-trained models could be more critical and give better results.

3.2.2 The training process

3.2.2.1 The batch size

The batch size defines the number of samples that will be passed through the network at once. One epoch will correspond to one propagation of all training samples on the network, while the number of steps defines the number of passes one batch size pass through the network. The batch size parameter is mainly selected based on the RAM available on the machine. By using a batch size smaller than the number of samples, the memory needs are not going to be so high, and the network trains faster since the weights of the network parameters are updated at each propagation.

However, the batch size can have several repercussions on the training. Large batch processes tend to converge to sharp minimizers of the training function, that leads to a decrease in the capacity of generalization of the model [24].

On the other hand, the batch size is highly correlated with the speed of the training: the larger the batch, the faster the training process. Another aspect to take into account by defining the batch size is the influence that is going to have on the learning rate parameter. The learning rate is applied once for every batch, and in [50], it is proved that equivalent accuracy results can be achieved after the same number of training epochs, but with fewer parameter updates, which consequently shorts up the training time, by increasing the learning rate and scaling the batch size.

Processes that use the entire training set are denominated as deterministic gradient methods, since they process all the training examples simultaneously, while the ones that use a single sample at once are denominated as stochastic. Most algorithms use a batch size that falls in between, where the primary considerations are based on the memory available and the speed of the training [14].

To assess the number of epochs of the training process, we can apply the following formula:

$$\text{number of steps to complete 1 epoch} = \frac{\text{training samples}}{\text{batch size}}$$

3.2.2.2 The optimizer

Momentum The Momentum optimizer was proposed by Boris Polyac in 1964 [40], and was designed in order to accelerate learning. As exposed in [14], this algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction. The momentum optimizer is defined by a momentum term γ , usually set as 0.9, that increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. Consequently, this leads to situations of faster convergence and diminished oscillations [46].

RMSProp The RMSProp optimizer is an unpublished, adaptive learning algorithm, created by Tijmen Tieleman and Geoffrey Hinton in 2012. It comes as a modification from the AdaGrad algorithm, by improving the performance of the model in the non-convex settings. It is an advanced form of gradient descent that divides the learning rate by an exponentially decaying average of squared gradients [48] [46].

Adam The Adam optimizer stands for Adaptive Moment estimation, and mix the ideas behind the momentum and RMSProp optimizer. Proposed by Kingman in 2014 [25], it keeps track of an exponentially decaying average of past gradients similar to Momentum, and just like RMSProp, it keeps track of an exponentially decaying average of past squared gradients [8] [46]. Authors show that Adam can have superior results than other adaptive learning-method algorithms.

3.2.2.3 Learning Rate

The learning rate hyperparameter in neural networks is one of the most important hyperparameters to set when tuning a neural network since it has a strong impact on both stability and efficiency of training times. If the learning rate is too large, our network training can become unstable and never converges, while if too small, training can take orders of magnitude longer than needed and never progresses. Usually, we want to have a large learning rate at the beginning of the training, that becomes smaller as we approach convergence of the training process.

The process of finding a good initial learning rate can be difficult. One of the approaches to find a feasible learning rate to the problem at hand is to train the model for a few

epochs using different learning rates and to compare the learning curves. The ideal learning rate will learn quickly and converge to a good solution.

The decrease in the learning rate can be applied using different strategies, such as:

1. **Constant learning rate** - In the constant learning rate, the entire training process will suffer from the same value concerning the learning rate, i.e., there will be no decrease. In this case, we need to set a constant to this parameter.
2. **Exponential decay learning rate** - In the exponential decay learning rate, it is applied an exponential decay function to a provided initial learning rate.
3. **Manual step learning rate** - In this case, we can define a specific value of learning rate to a pre-defined step. We can also apply warmup, i.e., if we want to apply a linearly interpolate learning rate for steps in a specific range.
4. **Cosine decay learning rate** - In the cosine decay learning rate, a cosine decay function is applied to an initial learning rate.

3.2.2.4 Data augmentation

The image augmentation is a set of techniques that can be applied in order to mitigate overfitting, and useful in scenarios where the dataset is small. It allows to increase the size of the training dataset, and since it introduces noise during training, it can increase the robustness of the model to various inputs. Common techniques applied are flipping the images, random cropping, rotation, changes on the channels, and others. In the present project, several techniques were applied, namely:

1. **Random horizontal flip** - Randomly flips the image and detections horizontally with a probability of 50%.
2. **Random vertical flip** - Randomly flips the image and the ground truth annotations with a probability of 50% vertically.
3. **Pixel value scale** - Scales randomly the values of all pixels in the image by a constant value between [minval, maxval].
4. **Random Image scale** - Randomly enlarges or shrinks images, keeping the aspect ratio.

5. **RGB to gray** - Randomly convert entire image to grayscale, with a probability of 10%.
6. **Adjust brightness** - Randomly changes image brightness by up to a maximum threshold. Image outputs will be saturated between 0 and 1.
7. **Adjust contrast** - Randomly scales contrast by a value between [min_delta, max_delta].
8. **Adjust hue** - Randomly alters hue by a value of up to a maximum threshold.
9. **Adjust saturation** - Randomly alters saturation by a value between [min_delta, max_delta]
10. **Distort color** - Performs a random color distortion.
11. **Jitter boxes** - Randomly jitters corners of boxes in the image determined by a ratio, i.e., if a box is [100, 200] and the ratio is 0.02, the corners can move by [1, 4].
12. **Crop image** - Randomly crops the image and bounding boxes.
13. **Pad image** - Randomly adds padding to the image.
14. **Crop pad image** - Randomly crops an image followed by a random pad.
15. **Crop to aspect ratio** - Randomly crops an image to a given aspect ratio.
16. **Black patches** - Randomly adds black square patches to an image.
17. **Rotation 90** - Randomly rotates the image and detections by 90 degrees counter-clockwise 50% of the time.

3.2.2.5 Gradient clipping

The gradient clipping can be a valuable procedure when dealing with exploding gradients, i.e., when the gradients can explode during backpropagation, resulting in number overflows. When defined as a value greater than 0.0, the process will clip gradients by the defined threshold which prevents the gradients from getting to large. The proposed solution applies a normalization of the gradients of a parameter vector when its L2 norm exceeds a certain threshold [38].

3.2.2.6 Transfer learning

Transfer learning is the task of learning from a pre-trained model that was trained on a larger dataset [49]. Usually, training with a pre-trained model can give faster convergence. If this original dataset is large enough and general enough, then the spatial hierarchy of features learned by the pre-trained network can effectively act as a generic model of the visual world, and hence its features can prove useful for many different computer vision problems, even though these new problems may involve entirely different classes than those of the original task. Such flexibility of learned features across different problems is a crucial advantage of deep learning compared to many older, shallow learning approaches, and it makes deep learning very effective for small data problems.

Depending on the size and the similarity of the dataset, different approaches should be made when using transfer learning [49], as shown in table 3.1.

Table 3.1: Guidelines for transfer learning

Data size	Similar Dataset	Different Dataset
Smaller Data	Fine-tune the output layers	Fine-tune the deeper layer
Bigger data	Fine-tune the whole model	Train from scratch

At the date, it is possible to gather pre-trained models from a small set of datasets as *ImageNet*, *Pascal VOC*, *KITTI*, *COCO*, *Pets*, among others [7].

3.2.3 The feature extractor

Regarding the behavior of feature extractor overall, two parameters should be defined:

3.2.3.1 First stage feature stride

This parameter corresponds to the output stride of the extracted RPN feature map.

Batch norm trainable This parameter will define if batch normalization should be applied during training. It is highly advisable to apply batch normalization if the batch size is relatively large.

3.2.4 The first stage of Faster R-CNN

After preprocessing the input images, the first step is to find regions of interest in each image for classification. The main goal in this phase is to find optimal rectangular

bounding boxes on the image. Those bounding boxes can have different sizes and aspect ratios, depending on the object shape. Each bounding box will be characterized by four values: Δ_{x_center} , Δ_{y_center} , Δ_{width} and Δ_{height} .

The anchors are fixed bounding boxes placed through the original image with pre-defined aspect ratios and scales or sizes that will work as a reference for predict object locations. With that purpose in mind, in the first stage of the model, several parameters should be defined, not only regarding the convolutional network but also to the several specificities to find regions of interest in the image.

3.2.4.1 The anchors

The anchors can be defined according to 8 different parameters: height and width of the anchor basis, defined as 256 pixels in the original implementation of faster R-CNN [44], height and width of the anchor stride, i.e., the space between each anchor, height and width offsets, list of scales and list of aspect ratios for the anchors.

3.2.4.2 Atrous rate

The atrous rate has been cited as a mechanism that will allow increasing the performance of the models when detecting small objects [15]. This atrous rate is applied on the tensor associated to the features to crop in the first stage to obtain box predictions.

3.2.4.3 Network hyperparameters

The network in the first stage should be adjusted in different parameters, namely those which specify the structure of the network itself and defines how the network will be trained.

1. **The Regularizer** - The regularizer will allow to prevent overfitting situations, as explained in section 2.1.2.1. By default, no regularizer is applied, but apply L_1 (see equation 2.1) or L_2 (see equation 2.2) regularizers is possible. The regularizers will be defined by the weight, which determines how much we penalize higher parameter values.
2. **Activation Function** - As explained in section 2.3.4, the activation functions will define how the output of the convolutions are calculated. By default, Faster R-CNN applies RELU (see equation 2.3) activation functions, but it is also explored

the possibility of applying RELU_6 activation function (see equation 2.4).

3. **The Initializer** - The definition of an initializer in deep learning training algorithms can be an essential step to determine whether the algorithm converges at all [14]. Different approaches can be used, namely:

(a) **Random Normal initilizer** - The random normal initializer is an initializer that generates a tensor with a normal distribution. It is possible to define the value of the mean and the standard deviation, which by default assume the values of 0 and 1, respectively.

(b) **Truncated Normal initializer** - The truncated normal initializer is a initializer that generates a truncated normal distribution, i.e., the values generated are similar to the ones obtained from a random normal initializer, however the values with more than two standard deviations from the mean are excluded and re-drawn. It is the recommended approach for neural network weights and filters.

(c) **Variance Scaling Initializer** - The variance scaling initializer has as advantage to keep the scale of the input variance constant, in order to not explode or vanish to zero by reaching the final layer. In this context, we can cite three different initializer proposed:

1 - MSRA Initialization [16]

2 - Convolutional Architecture for Fast Feature Embedding [22]

3 - Xavier [12]

4. **The Batch Normalization** - Perform a batch normalization using a standard set of parameters. This concept is explained in subsection 2.3.5.

5. **Depthwise regularization** This parameter defines whether or not apply L_2 regularization (see equation 2.2) on the depthwise convolution weights.

3.2.4.4 Minibatch size

It corresponds to the batch size to use for computing the first stage objectness and location losses.

3.2.4.5 Positive Balance fraction

Associated to cases of unbalanced data, this parameter will define the fraction of positive examples per image for the RPN.

3.2.4.6 NMS - IoU threshold

After training, several box predictions are made to a unique object. In order to remove the unnecessary bounding boxes, a Non-Max suppression (NMS) algorithm is applied which will filter out those overlapping boxes taking into account a specific threshold. Using the `score_threshold` parameter, the boxes where the score is lower than the threshold defined are removed immediately. The parameter `iou_threshold` will allow dealing with the situation where more than one bounding box is over only one object, i.e., the IoU threshold is a scalar threshold for IoU, where boxes that have high IOU overlap with previously selected boxes are removed. The algorithm applied in this case is:

1. Select the box with the highest score;
2. Compute its overlap with all other boxes and remove the boxes that overlap more than `iou_threshold`.
3. Iterate until there are no more boxes with a lower score than the currently selected box.

3.2.4.7 Maximum number of proposals

This parameter corresponds to the maximum number of RPN proposals retained after first stage postprocessing.

3.2.4.8 Loss weights

When dealing with multiloss functions, it can be advisable to adjust the magnitude of each loss and search for the best combinations which will lead to better results.

3.2.4.9 ROIs parameters

After defining a good set of proposals, the filtered regions of interest will pass through the process of ROI pooling, a needed step to resize the feature maps that will allow the forward processing of those in the second stage of the algorithm. In this phase, we should define:

1. **The crop size** - The output size (width and height are set to be the same) of the initial bilinear interpolation based cropping during ROI pooling.
2. **The kernel size** - The kernel size of the max pooling operation on the cropped feature map during ROI pooling.
3. **The stride** - The stride of the max pooling operation on the cropped feature map during ROI pooling.

3.2.5 The second stage of Faster R-CNN

In the second stage of Faster R-CNN, and after obtaining a set of proposals, the class of the detection will be defined and the bounding boxes are going to be refined.

3.2.5.1 Network hyperparameters

Taking as input the proposals obtained during RPN, filtered in NMS and reconfigured in ROI Pooling, the network need to be adjusted for the new context. In this way, and similarly to what happened in the first stage of the algorithm, the hyperparameters of the network can be fine-tuned with the parameters exposed in 3.2.4.3.

3.2.5.2 Dropout

Dropout comes as a regularization technique for neural networks that allows preventing overfitting by introducing noise to the training. At each training iteration, a fraction of the neurons is randomly set to zero, as explained in 2.1.2.1. By default, no dropout is applied in Faster R-CNN.

3.2.5.3 The IoU Threshold

Likewise in the first stage, an IoU threshold should be defined on this phase in order to refine the bounding boxes. This process is better defined in 3.2.4.6.

3.2.5.4 The number of detections

The parameters `max_detection_per_class` and `max_total_detections` define the maximum number of detections to maintain after the NMS is applied. Since we are dealing with a problem where we only have one class to predict, those parameters will have the same value. The default value in Faster R-CNN is 100. This value should be adjusted to

the problem in hand, i.e., to at least the maximum number of cells present in the image with the highest number of cells in the dataset.

3.2.5.5 The Loss Weights

Similarly to the first stage, the loss weights of the multiloss function applied in this stage can be adjusted using different combinations of magnitude for each of them.

3.2.5.6 The Classification loss

Related to the loss for the second stage box classifier, the different options analyzed, namely weighted sigmoid, focal sigmoid and bootstrapped sigmoid, are described in subsection 3.3.3.2.

3.2.5.7 The score converter

The score converter will define how the classification is made and it can be of three types:

1. Identity – the input scores equals the output scores;
2. Sigmoid – Applies a sigmoid on input scores, as shown in equation 3.1.

$$F(X_i) = \frac{1}{1 + \text{Exp}(-X_j)} \quad (3.1)$$

3. Softmax – Applies a softmax on input scores, as shown in equation 3.2.

$$F(X_i) = \frac{\text{Exp}(X_i)}{\sum_{j=0}^k \text{Exp}(X_j)} \quad (3.2)$$

While the sigmoid function is used for binary classification, the softmax function calculate the probability distribution of the event over n different events, i.e., it will calculate the probabilities of each target class over all possible target classes. In this project, we are trying to classify an object as a human cell or not, and in this way, we apply the sigmoid function in order to calculate the probability of that object in being from human class.

3.3 Measuring the performance of a model

3.3.1 mAP

The standard metric used to evaluate the performance and accuracy of an object detection algorithm is denominated as mean average precision (mAP). This metric is the product of precision and recall of the detected bounding boxes, and its values range from 0 to 1, where the higher the number, the better. The mAP is a good measure of the sensitivity of the network [49], being based on the Intersection over Union (IoU) concept. The IoU is the ratio of the overlapped area between the ground truth and predicted area to the total area, or area of union, as shown in figure 3.5, and is given by the following formula:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (3.3)$$

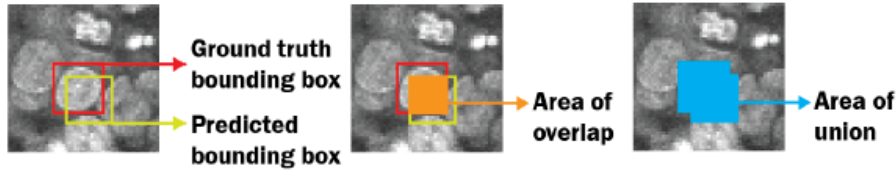


Figure 3.5: The IoU concept

In this way, detections are assigned to ground truth objects and judged to be true or false positives by measuring bounding box overlap. To be considered a correct detection, the overlapping area between the predicted bounding box and ground truth bounding box must exceed 50% . Furthermore, we can define the following concepts [5]:

True positive (TP) = number of detections with $\text{IoU} > 0.5$

False positive (FP) = number of detections with $\text{IoU} \leq 0.5$ or detected more than once

False negative (FN) = number of objects not detected, or detected with an $\text{IoU} \leq 0.5$

Computing the mean average precision of a class afterward is a five-stage process where:

1. Calculate the precision (the proportion of true positives). The precision is defined by:

$$\text{Precision} = \frac{\text{True positive}}{\text{True Positive} + \text{False Positive}} \quad (3.4)$$

2. Calculate the recall (the proportion of true positives out of all possible events):

$$\text{Recall} = \frac{\text{True positive}}{\text{True Positive} + \text{False Negative}} \quad (3.5)$$

3. Compute the precision recall curve for the different thresholds
4. Average together the maximum precision value across all recall levels. By applying a 11 point interpolation, i.e., by averaging the precisions at a set of 11 recall levels, then the average precision score for each class is given by:

$$\text{AP} = \frac{1}{11} \sum_{\text{Recall}_i} \text{Precision}(\text{Recall}_i) \quad (3.6)$$

5. Finally, the mAP is calculated by taking the mean AP over all classes and /or over all IoU Thresholds. In the present project, it is only considered one IoU threshold (mAP@0.5).

3.3.2 Regression performance measures

On top of the mAP metric, the one used to fine tune the model, and taking into account the problem in hand, other metrics were considered in order to evaluate the performance of the model. While mAP is the standard metric used in object detections systems, it has limitations especially in cases where we are dealing with a large number of objects. This happens because if a specific object is not identified on a region proposal on the first stage of the algorithm, it will not be taken into account for the accuracy of the model. In this way, for each model, it was tested the performance of the model in the validation dataset and calculated several metrics that allow measuring regression performance between the predicted value, i.e., the number of objects predicted, versus the ground truth value. The following measures were taken into account [23]:

Root mean squared error (RMSE) The root mean squared error, shown in equation 3.7, measures the average error performed by the model in predicting the outcome for an observation. Mathematically, the RMSE is the square root of the mean squared error (MSE), which is the average squared difference between the observed actual outcome values (y) and the values predicted ($\hat{y}$) by the model.

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{\sum_{i=1}^n (\hat{y}_i - y_i)^2}{n}} \quad (3.7)$$

Mean absolute error (MAE) The mean absolute error, presented in equation 3.8, is the sum of the absolute differences between the predictions and the real values, also denominated as l_1 norm loss. This measure gives an idea of the magnitude of the errors, but not about the direction of those errors, i.e., if there is over or under prediction. A value of 0 indicates a perfect prediction with no errors, and as so, the lowest the value, the better.

$$\text{MAE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (3.8)$$

Median absolute error The median absolute error, in equation 3.9, is the median of all of the absolute values between the predictions and the real values. This measure is more robust than the MAE in the presence of outliers.

$$\text{MEDAE}(y, \hat{y}) = \text{median}(|y_1 - \hat{y}_1|, \dots, |y_n - \hat{y}_n|) \quad (3.9)$$

3.3.3 The losses in Faster R-CNN

The loss in Faster R-CNN is evaluated in the two stages of the algorithm, and each stage measures the error using a multi-loss function, which sums up to four losses in total.

3.3.3.1 First Stage Losses

In the first stage of the algorithm, the RPN, it is measured the performance of the model taking into account the objectness loss and the localization loss, which lead to the multi-loss function defined in equation 3.10.

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \quad (3.10)$$

As so, the RPN takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score [44].

In the equation exposed in 3.10, p stand for the predicted objectness class, p^* for the ground truth objectness class, t for the ground truth bounding box and t^* for the predicted bounding box.

The RPN objectness loss The objectness loss classifies a patch as having, or not, an object. In this phase the goal is not to find the class of the object identified, but if the

patch does have an object, and it is not background. This objectness score will be used to filter the bad predictions for the second stage. With this purpose in mind, Faster R-CNN uses a classifier with 2 possible classes: one for the presence of an object category and one for the background.

The RPN localization loss For the bounding box regression, the authors of Faster R-CNN adopt the parameterizations of the 4 following coordinates, as shown in 3.11.

$$\begin{aligned}
t_x &= \frac{(x - x_a)}{w_a} & t_y &= \frac{(y - y_a)}{h_a} \\
t_w &= \frac{w}{w_a} & t_h &= \log \frac{h}{h_a} \\
t_x^* &= \frac{(x^* - x_a)}{w_a} & t_y^* &= \frac{(y^* - y_a)}{h_a} \\
t_w^* &= \log \frac{w^*}{w_a} & t_h^* &= \log \frac{h^*}{h_a}
\end{aligned} \tag{3.11}$$

where x , y , w and h denote the box's center coordinates and its width and height. The variables x , x_a and x^* are for the predicted box, anchor box and ground truth box respectively (likewise for y , w , h).

3.3.3.2 The second stage losses

For the second stage of the algorithm, two new losses are computed, namely the box predictor localization loss and the box predictor classification loss. Each RoI is labeled with a ground truth class u and a ground truth bounding box regression target v . The multi-loss is defined as seen in equation 3.12:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda [u \geq 1] L_{loc}(t^u, v) \tag{3.12}$$

where p stands for the discrete probability distribution over the categories, and t^u the bounding box regression achieved from the first stage using the RPN localization loss. The indicator function $[u \geq 1]$ evaluates to 1 when $u \geq 1$ and 0 otherwise (being the background labeled as 0). The hyperparameter λ controls the balance between the two losses [10].

Box predictor localization loss The paper [44] applies a smooth-L1 loss on the position (x, y) of top-left of the bounding box, and the logarithm of the heights and widths ,

similar to the one applied on the first stage, as shown in equation 3.13:

$$L_{Loc}(t^u, v) = \sum_{i \in x, y, w, h} \text{smooth}_{L_1}(t_i^u - v_i) \quad (3.13)$$

where

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

where x is the difference between predictions and target.

Choosing a smooth L1 loss comes from the fact that this type of loss is less sensitive to outliers, and in that way, more robust, since when the regression targets are unbounded, training with L2 loss can demand careful tuning of learning rates in order to avoid exploding gradients [10].

This loss aims to refine the localization of the bounding boxes before classification.

Box predictor classification loss Regarding classification loss, by default, it was used weighted sigmoid. However, focal sigmoid classification loss and bootstrapped sigmoid classification loss were also tested.

As a binary classification, all of the former classifications losses are based on the cross-entropy loss for binary classification [29], presented in equation 3.14:

$$CE(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1 - p) & \text{otherwise} \end{cases} \quad (3.14)$$

In equation 3.14, $y \in \{\pm 1\}$ specifies the ground truth class and $p \in [0, 1]$ is the model's estimated probability for the class with label $y = 1$. For notational convenience, p_t is defined as:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise} \end{cases} \quad (3.15)$$

and

$$CE(p, y) = CE(p_t) = -\log(p_t) \quad (3.16)$$

The **weighted sigmoid loss**, presented in 3.17, addresses class imbalance by introducing a weighting factor $\alpha \in [0, 1]$ for class 1 and $1 - \alpha$ for class -1. In this way, we can define the weighted sigmoid classification loss as:

$$CE(p_t) = -\alpha_t \log(p_t) \quad (3.17)$$

Focal loss adds a factor to the standard cross entropy criterion, reducing the relative loss for well-classified examples, where $(p_t) > 0.5$. In this way, and cited in [29], the focal loss can be defined as:

$$FL(p_t) = (1 - p_t)^\gamma \log(p_t) \quad (3.18)$$

where $(1 - p_t)^\gamma$ is the modulating factor for the cross entropy loss. Improved accuracy was detected by the authors when adding an α balanced variant of the focal loss. In this way, the sigmoid focal loss applied by default in this project is denoted as:

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t) \quad (3.19)$$

where γ is 2.0 and *alpha* equals 0.25.

Finally, the **bootstrapped sigmoid classification loss** [43] uses a convex combination of training labels and the model's predictions as training targets in the classification loss. This will allow the model to improve over time since the previous predictions by the model can be used to mitigate the damage of incorrect labels.

4.1 Data sample

In order to build an effective neural network, careful considerations should be applied in several aspects, passing through the input data format till the network architecture. Regarding input data, the most relevant parameters to take into account are the number of images, the dimensions (width and height) and the number of channels. All of those parameters can influence the performance of the model, not only in terms of accuracy but also in terms of speed.

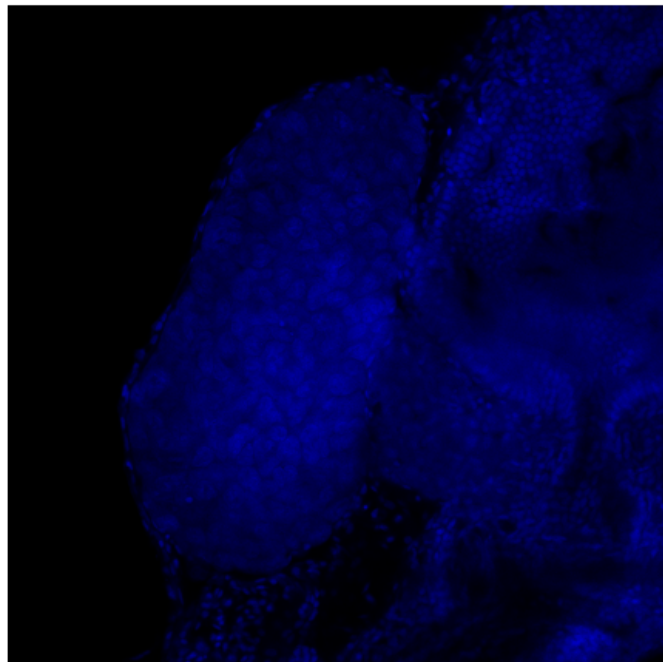


Figure 4.1: A raw image example

The raw data The raw data provided by Fundação Champalimaud included 97 images correctly labeled. A raw image is figured in 4.1 as a matter of example. The labeling of those images was poorly performed for a case of possible segmentation since each cell was annotated by a dot not corresponding exactly to the center of the cell. In this way, another labeling was performed taking into account those annotations, by using the software labelImg, as explained in Chapter 5. This software allows creating bounding boxes around objects in order to extract the coordinates of those boxes in the image, as can be seen in figure 4.2.

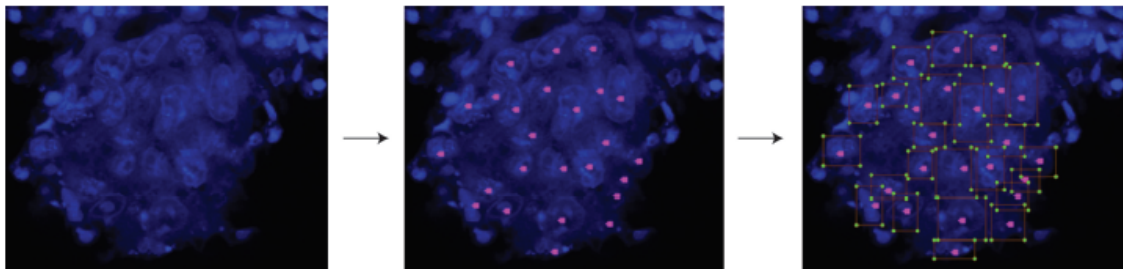


Figure 4.2: The process of labeling images using labelImg

Using the labeling as ground truth, and in order to build an object detector, several files were needed to be created, namely:

1. A XML file with the coordinates annotation for the different images, and the class associated with it. This file should take into account the object detector framework that is going to be applied. As an example, the ones presented in this project use the annotation PASCAL VOC, where the bounding boxes are defined by [xmin, xmax, ymin, ymax]. Other algorithms can use different types of annotations, as the center of the box and the width and height. After data partition in training, validation and test, the information in XML files should be converted to a tfrecord.
2. A ptxt file where each class present in the images is going to be defined by an ID. Different frameworks and APIs could use different formats for this file. As an example, the tensorflow API, the one that is used in this project, should not consider the background as a class while in Keras library, the background should be taken into account.

The number of images and data partition The number of images, in this case, was limited by the number of images given by Fundação Champalimaud that were labeled. In that way, 96 images were considered valid for the purpose of this project. In order to divide the total images available in the three datasets, it was considered not only the number of images itself but also the number of cells/objects available for the several images, and the average number of cells in each image for the different datasets, since this value could range between 18 and 661 cells. The total dataset was split in the three datasets taking into account the following proportions: around 80% for training, 15% to validation and 5% to test as shown in table 4.1.

Table 4.1: Data partition

Dataset	#cells	%cells	#images	%images	Avg. #cells per image
Train	13790	80,5	78	81,3	176.8
Validation	2292	13,4	13	13,5	176.3
Test	1046	6,1	5	5,2	209.2
<i>Total</i>	<i>17128</i>	<i>100</i>	<i>96</i>	<i>100</i>	<i>176.6</i>

In this way, we try to apply a “stratified” partition of the data, where the proportion of the number of cells available for each dataset follow the same proportion as the number of the images, and the average number of cells could also be similar in order to have samples with different types of difficulty (in this case difficulty is associated to the number of cells to be detected in each image) in the different samples.

Image dimension and scaling Regarding to image scaling, it was defined that the best approach was to work with images where the dimension was equal for all cases. In the images given by Champalimaud, 89 of them have the dimension 512x512 with three channels and the remaining 8 the dimension 1280x1280, also with three channels. Since only less than 10% of the given images were of a different dimension from 512x512, the images that had different values were scaled to the default 512x512, using the technique INTER_AREA in the open-cv library, the one that is considered to be the best to apply in cases where the purpose is to decrease images [13].

Uniform aspect ratio While not mandatory for the object detection process, it was assured that all the images to be processed had the same dimensions and aspect ratio. Most

of the neural network models assume a square shape input image, and in our case, all the images originally followed this type of behavior, which allowed us to pass forward this step and avoid processing functions like crop in order to respect this characteristic.

Number of channels The images provided by Fundação Champalimaud were RGB images with 3 channels.

4.2 Data exploration

One of the first steps in any machine learning project is to get to know and be aware of the data available. Since the number of cells can vary highly on the images, in a range of [18,661], the images were divided into three sections regarding the complexity/ density of cells:

1. **Low** - Images where the number of cells is lower than 50;
2. **Medium** - Images where the number of cells is in the range [50,250];
3. **High** - Images where the number of cells is higher than 250;

In order to perceive the distribution of the number of cells in each image, it was plotted a histogram.

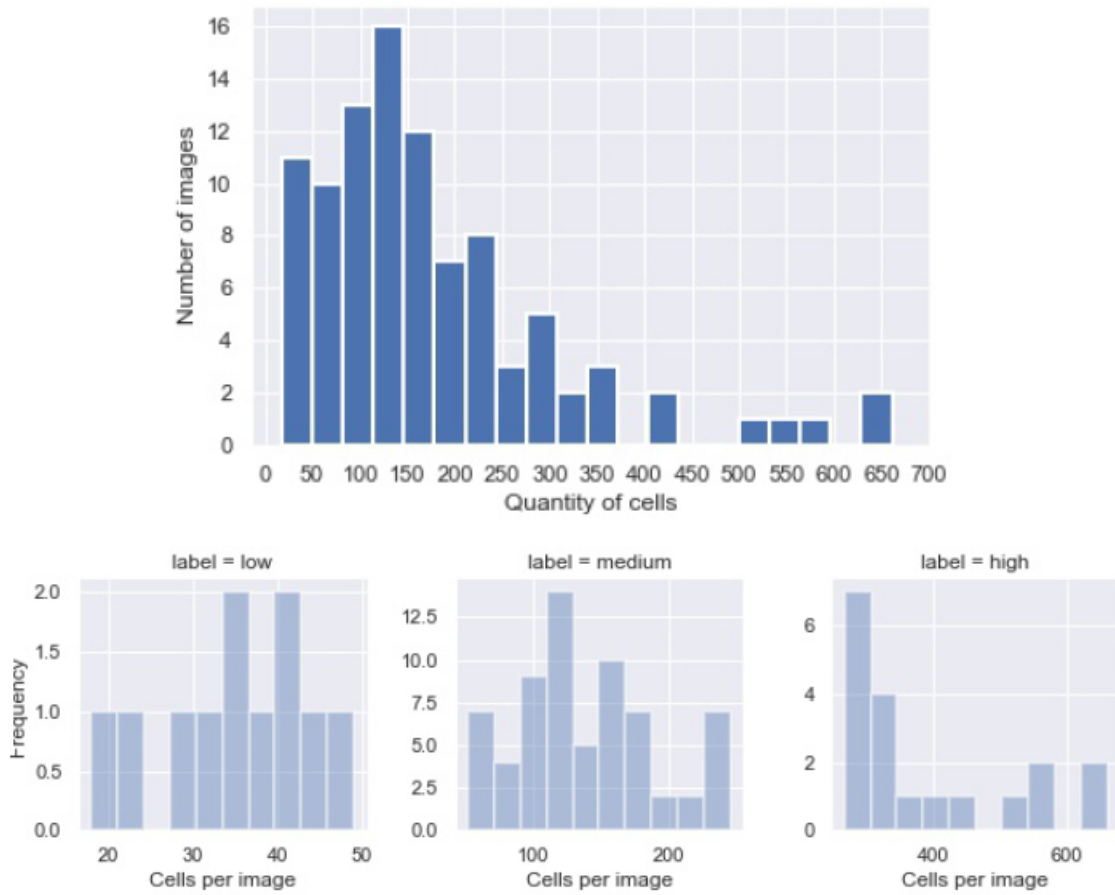


Figure 4.3: Distribution of the quantity of cells per image

According to the results plotted in figure 4.3, we can verify that the number of cells in the majority of the images range between [50,300]. More descriptive statistics regarding cell quantification on raw images is given in table 4.2.

Table 4.2: Descriptive statistics related to the number of cells in input images

Images	Count	Mean	Std	Min	25%	50%	75%	max	skew	kurt
All	96	176.6	132.4	18	95	137	230	661	1.7	3.3
Low	11	35.2	9.1	18	31	35	42	49	-0.5	-0.2
Medium	66	138.9	52.5	52	104	129	173	245	0.4	-0.6
High	19	391.2	131	267	295	343	471	661	1	-0.3

High skewness situations are assumed when the value of skewness is less than -1 or greater than 1 [2]. Similar to skewness, we can consider a high value of kurtosis when we are dealing with values lower than -2 or higher than 2. As it is possible to verify in table 4.2, the whole dataset is positively skewed with a skewness of 1.7, and we also

deal with a leptokurtic (kurtosis > 3) distribution, which can lead to situations where the predictive model will underestimate some of the predictions. With that in mind, the dataset was "partitioned" in three different (low, medium and high) levels in order to reduce the skewness and the kurtosis for each sample.

Taking into account the small number of samples available, the fine-tuning of the model considered all the dataset, but the regression measures calculated take into account this partition.

Besides the number of cells present in each image, it was important, due to the necessity of establishing aspect ratios and scales during anchor generation, to evaluate the distribution of width, height and aspect ratios of the 17128 cells present in the images. The following statistical measures in table 4.3 and presented in figure 4.4 refer to images with dimension 512x512 pixels. If resizing the image will be one of the steps during fine-tuning the model, the scaling of these values should be considered.

Table 4.3: Descriptive statistics related to the width and height of cells present on input images

Dimension	Mean	Std	Min	25%	50%	75%	max	skew	kurt
Width	15.9	4.5	4	13	15	19	44	0.8	1.1
Height	15.2	4.3	3	12	15	18	45	0.9	2.8
Ratio	1	0.3	0.3	0.8	0.9	1.1	3.3	18.9	0.1

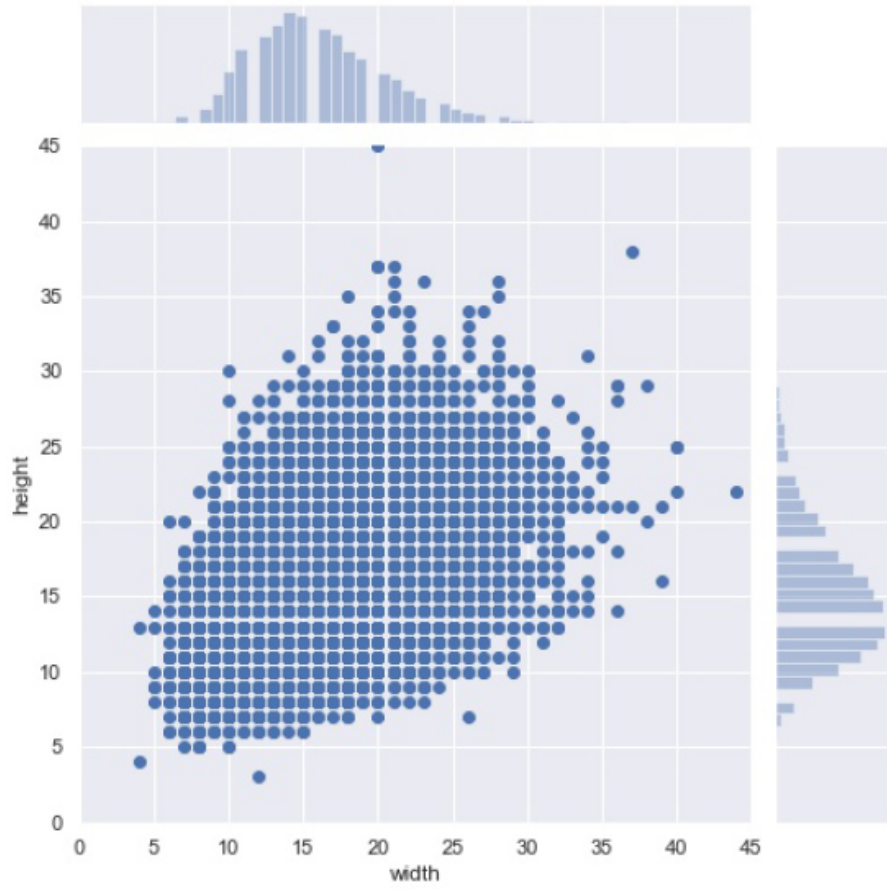


Figure 4.4: Density of cells according to their width and height

We can verify in figure 4.5 that the aspect ratio of the cells range mainly between $[0.5, 2.0]$. One possible approach to select the aspect ratios in anchor generator will be to start with three different ratios (e.g. $[0.5, 1.0, 2.0]$) that will cover the majority of the distribution, as shown in figure 4.6.

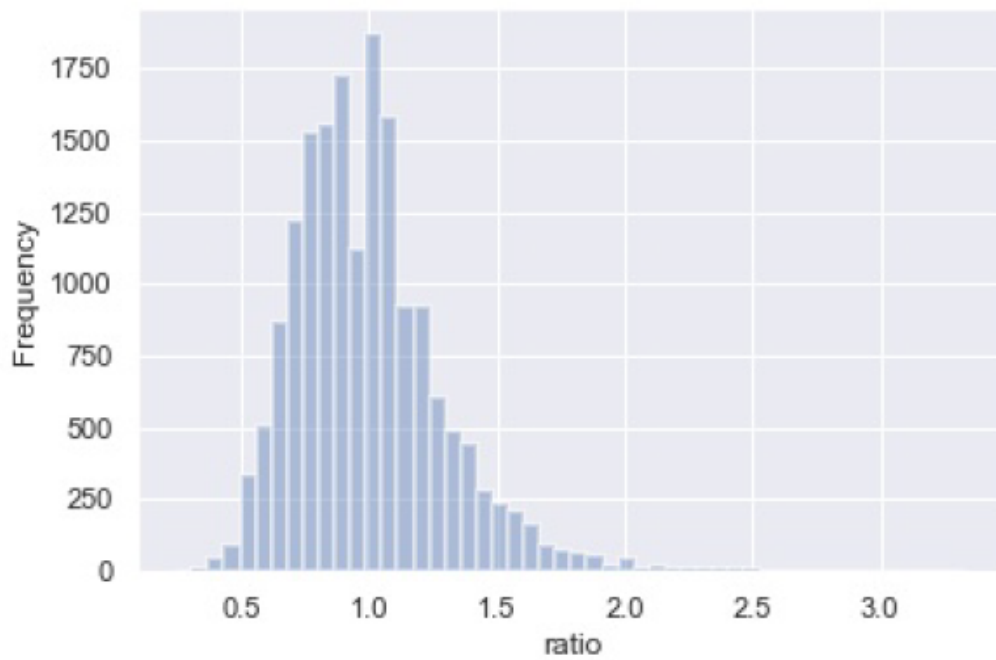


Figure 4.5: Frequency of cells according to their ratio

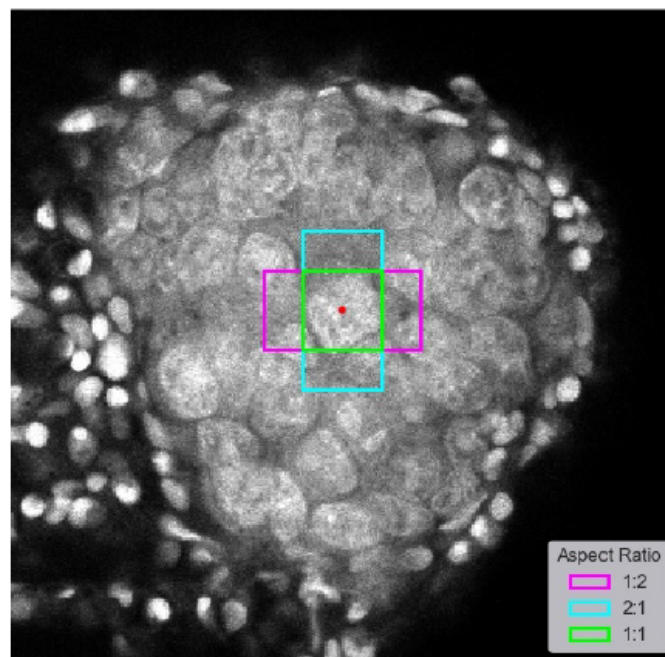


Figure 4.6: Possible aspect ratios taking into account the ratio distribution

The magenta bounding box will be able to cover aspect ratios of 0.5, the cyan one ratios of 2 and finally the green one ratios of 1.

Prepare the data to work in TensorFlow

In order to be able to train an object detection model in tensorflow, an open source machine learning library, the data to be considered as input should take the following considerations:

1. If images are not labeled, a graphical annotation tool, like labelImg ¹ should be used in order to create bounding boxes around every object of interest in the images we are going to use to train the model, evaluate and test. The labelImg will extract for each image a xml file with the coordinates of each bounding box and the corresponded label associated, in PASCAL VOC format.
2. TensorFlow works with TFRecord files, a binary file format, where all the information stored in all the xml files corresponding to each partition will be rearranged and stored in one single file in the appropriate format. All the individual xml files obtained using labelImg should be converted to this format.
3. Create a .pbtxt file that will contain the id and name for each of the possible labels in the dataset.

The diagram presented in figure 5.1 presents a high overview of the workflow to train an object detection model in Tensorflow Object Detection API, a open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models.

¹<https://github.com/tzutalin/labelImg>

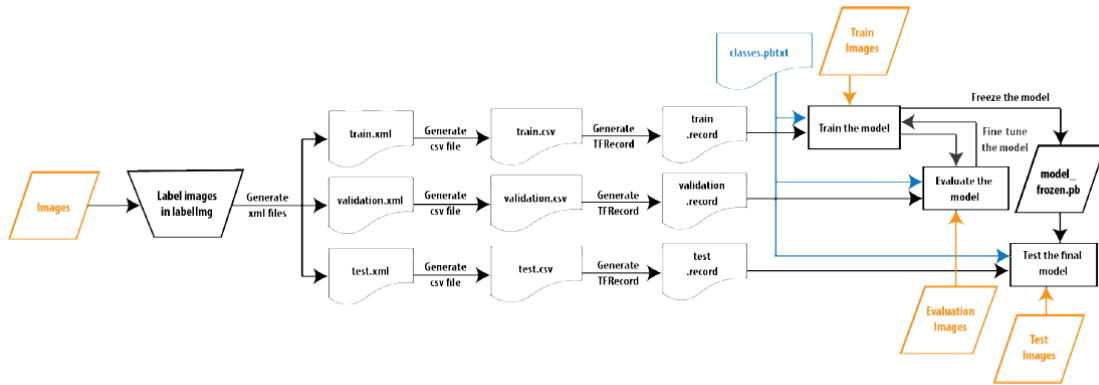


Figure 5.1: Basic flow diagram

5.1 Default parameters

The default parameters for the application of the object detector in the tensorflow Object Detection API has as object of study a multiclass problem where the number of objects in each image is relatively small. Taking into account the particularities of this project, the following parameters were updated:

1. Number of classes (num_classes) - Defined as 1, since the background should not be considered as a class while using tensorflow API.
2. Number of maximum proposals (first_stage_max_proposals) - By default, the value assumed is 300. Since the maximum number of cells in one single image was identified as more than 600, this parameter was changed to 2000 proposals.
3. Number of maximum detections per class (max_detections_per_class) - By default, this parameter is defined as 100. Concerning the maximum number possible of cells in one single image of the dataset, it was updated to 700.
4. Number of total detections (max_total_detections) - By default, this parameter is defined as 100. Since we are dealing with a one class problem, and for the reasons of the previous point, this parameter was updated to 700.
5. Score converter (score_converter) - By dealing with a uniclass problem, this parameter was changed from "SOFTMAX" to "SIGMOID".

6. Classification loss in the second stage (`second_stage_classification_loss`) - The classification loss was changed to a weighted sigmoid since we are dealing with a binary problem. Other options are explored further.

6.1 Hardware used

All considered tests in the following sections were carried out using Azure cloud computing resources, namely four Tesla K80 GPU (2 physical cards) with 32 GiB GPU memory, in a Virtual Machine with 24 vCPU, 224 GiB of memory and 1.44 TB SSD as temporary storage.

6.2 Dealing with a small dataset

The main constraints faced in the current project were associated with two essential aspects: the need for massive computational resources and the small number of samples of the provided dataset. Taking into account those limitations, the experiments described in this section were restructured the following way:

1. The first set of experiments had as a primary goal to conceive the most accurate architecture and establish the more appropriate parameters for the problem in hand. In this way, nearly 80% of the input data was used for training and 13 images for validation. The tested models were evaluated initially by mAP, and the fine-tuning was guided by this metric, except on specific cases (as documented on the experiments), where there was an evident difference on the behavior of the model regarding losses or overfitting situations.
2. Hereafter, and since the majority of the models at a certain point started to achieve similar mAP values, each model was frozen, and the inference was applied on the validation images, as a way to compute regression metrics, namely RMSE, MAE

and MEDAE (see section 6.11). The purpose was to obtain a small set of models using the different insights to test in a smaller training dataset and to achieve more accurate statistic results. In this way, six finalist models were selected by taking into account different approaches of analyzing the results.

3. The finalist models were trained during more epochs (until achieving an overfitting situation) in a way to check if the performance would increase (see section 6.12). Again, those models were evaluated by mAP and the already mentioned regression metrics, and these last applied to two distinct validation datasets: the complete dataset and a dataset previously defined as medium complexity. The need of evaluation in a medium dataset comes from the fact that it was perceived on an early stage that models could achieve better results on images where the number of cells present was around the average of the population distribution. In this step, two models were chosen due to the particularity of achieving distinct results in the regression metrics for the "medium" level and to the whole dataset.
4. In the last step, a new stratified data partition was done, consisting on 53 images for training, 13 for validation and 30 for testing, and applied on both models selected on the previous step (see section 6.13). As expected, the mAP of those models had decreased significantly due to the reduction of the training dataset. The two models were frozen, and a new division of the level of image complexity was defined using the interquartile related to the number of cells. Furthermore, the inference was applied on the validation images to define the best threshold of probability to be applied on considering a detection as a human cell or not, i.e., defining the confidence level, and to choose which was the best model for the different pre-defined levels.

6.3 Choosing the object detection model

With the aim of defining the most appropriate object detection architecture, the main pondering condition reflects on the preference of an algorithm that offers speed or primes per accuracy. Being accuracy the primary goal of this project, and as reported by [20], Faster R-CNN should be the one to go further, especially when dealing with objects of small size. Therefore, the performance of different meta-architectures went tested in

pre-trained models with COCO dataset ¹.

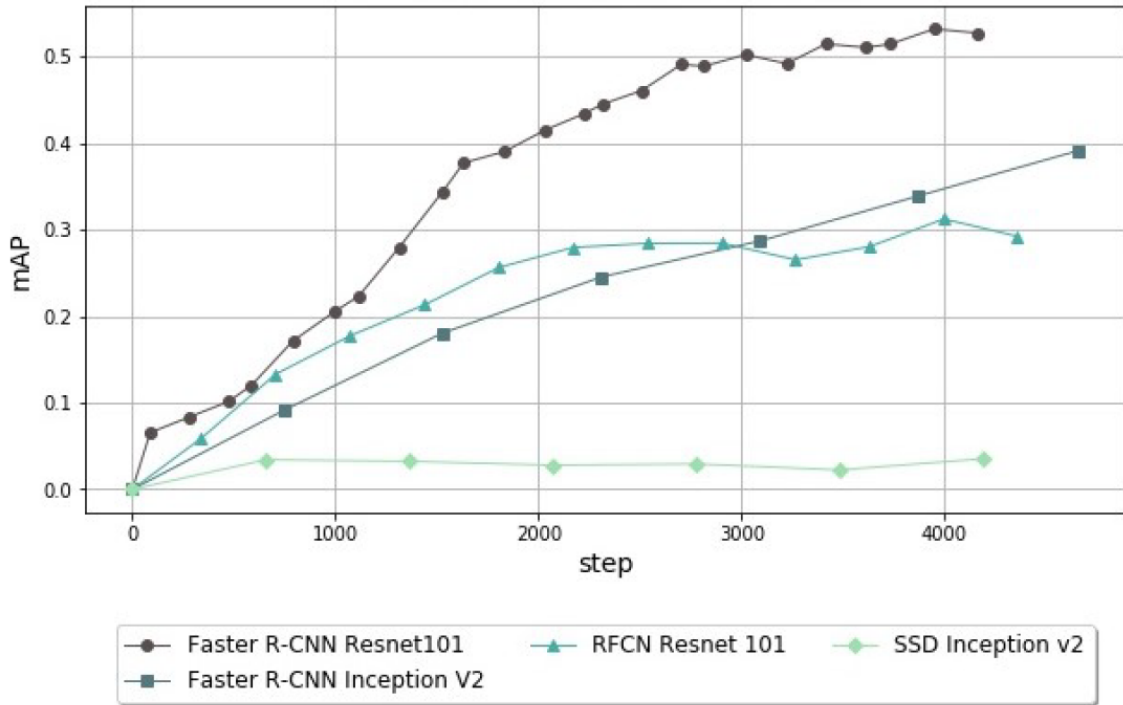


Figure 6.1: mAP using different meta-architectures

As expected, and as can be seen in figure 6.1, Faster R-CNN was the meta-architecture associated with the best performance, followed by RFCN and finally SSD. Furthermore, training speed connects strongly with the meta-architecture used.

Table 6.1: mAP and speed using different meta architectures

Meta-Architecture and Feature Extractor	mAP		Speed/epoch (sec)
	4000 steps	Maximum	
Faster R-CNN Resnet101	0.53	0.53	47.08
RFCN Resnet101	0.31	0.31	13.19
Faster R-CNN Inception v2	0.34	0.39	6.17
SSD Inception v2	0.04	0.04	1.13

It is possible to verify in table 6.1 that SSD is the fastest approach, taking around one second to perform one epoch (with a batch size of 24), while Faster R-CNN is the slowest one, taking almost fifty seconds to finish an epoch. However, it should keep in mind that the speed of training is also highly related to the feature extractor chosen. In this way, one can conclude that Faster R-CNN is the meta-architecture to be chosen when accuracy

¹https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

is the primary goal of the problem, while SSD should be the one to use if speed is more important. RFCN gives a balanced option between speed and accuracy.

6.4 Choosing the feature extractor

In this experiment, it was tested the performance of different feature extractors using a pre-trained model on COCO dataset.

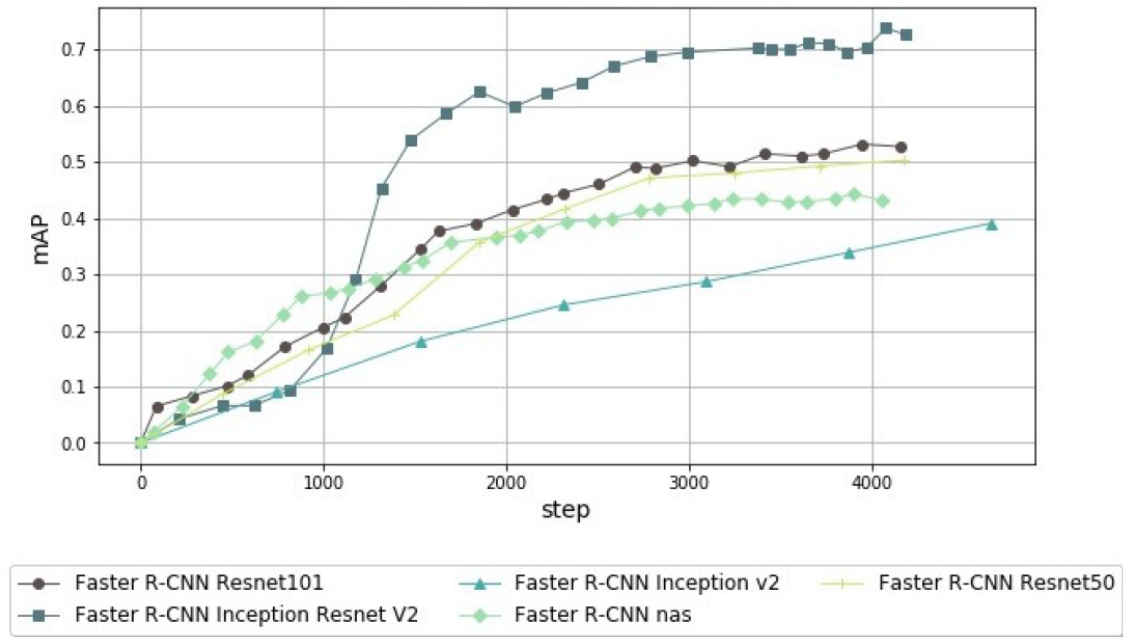


Figure 6.2: mAP using different feature extractors

As seen in figure 6.2, the model with the best performance is the Inception Resnet v2, as defended on [20]. Concerning the speed of training, the more complex the network, the slowest the training is, as expected, and confirmed in table 6.2.

Table 6.2: mAP and speed using different feature extractors

Feature Extractor	mAP		Speed/epoch (sec)
	4000 steps	Maximum	
Resnet50	0.50	0.50	10.34
Resnet101	0.53	0.53	47.08
Inception v2	0.34	0.39	6.17
Nas	0.44	0.43	96.36
Inception Resnet v2	0.71	0.74	45.55

Taking into consideration the results of the previous experiments associated to the selec-

tion of the meta-architecture and feature extractor, the chosen model for the remaining experiments has been defined as the Faster R-CNN with Inception Resnet v2, whose architecture figures in 6.3.

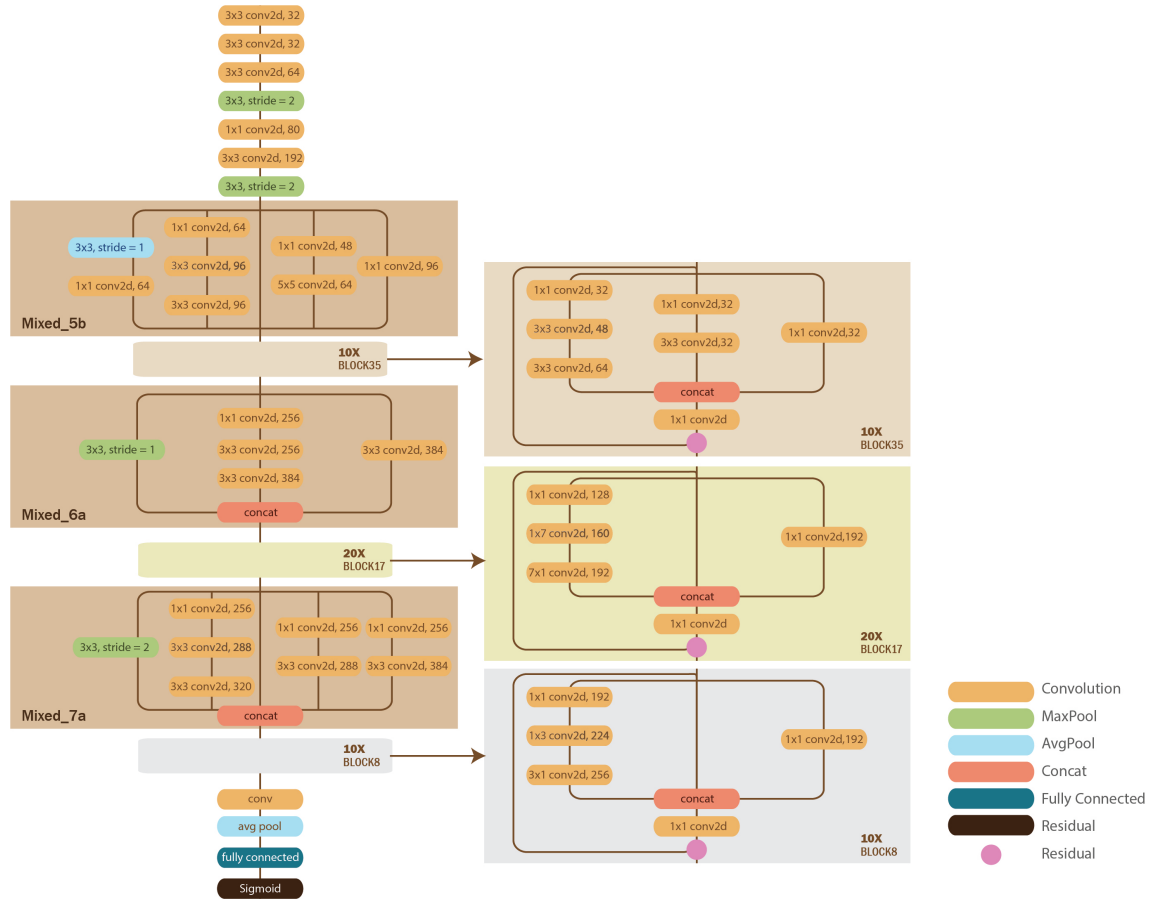


Figure 6.3: Architecture of the Inception Resnet v2 applied on the present project

6.5 Using a pre-trained model

Using a pre-trained model or model transfer can be an efficient technique when dealing with small datasets and if we want to boost the speed of model training. However, since the dataset in this project is not similar to the datasets of the available pre-trained models, some considerations should be taken into account. The performance of pre-trained models were evaluated in three different conditions, as can be seen in figure 6.4:

1. Uploading only the initial weights of the pre-trained model and do not freeze any layer
2. Freezing the first set of layers before the first branch of the architecture
3. Freezing the layers associated with the first stage allowing the second stage layers to train freely

Comparing the results from the previous approaches and adding one where there was no transfer learning, the results can be seen in the table 6.3:

Table 6.3: mAP and speed by using transfer learning

Transfer Learning	mAP (3K steps)	Speed/epoch (sec)
(1) Only weights	0.70	45.55
(2) Freeze first set of layers	0.69	44.49
(3) Freeze the feature extractor	0.04	17.25
(4) No transfer learning	4.7e-3	46.03

Considering the results, not using transfer learning makes the efficiency of the model almost null. While this situation could be partially solved by fine-tuning the parameters of the network, this would demand several epochs of training in order to achieve reasonable results.

The best solution in the approaches addressed seem to be using only the weights of the model pre-trained in order to initialize the network with optimized parameters, achieving a performance of 0.70 mAP in 3000 steps (around 150 epochs). By using a pre-trained model, the considerations regarding the hyperparameter initializer in the convnets will not have any effect, and so they are disregarded to further experiments.

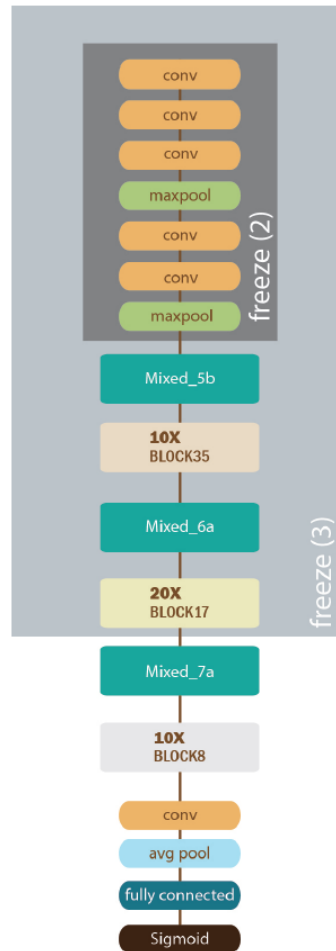


Figure 6.4: Freezing the layers in transfer learning: in freeze(2)(dark gray area), it was tested the performance of the model by freezing only the initial layers of the CNN, as suggested by several bibliographies. In freeze(3)(light gray area), all the process of feature extractor was frozen and only the variables associated to the second stage of the Faster R-CNN were allowed to train.

6.6 The preprocessing

In this phase, it was tested the influence of the dimension of the input images and the resize methods in the performance of the detector.

6.6.1 Image resizing

Since we are dealing with objects of small size and taking into account the limitations known of object detectors algorithms, one of the possible approaches to deal with this situation is to increase the size of the input images. In order to test the effect of the resize of the images on the performance of the algorithm, experiments for different sizes and resize methods were done. Due to hardware limitations, the maximum resolution to process the inputs was 1100 by 1100 pixels, taking into account that the higher the number of pixels, the more significant the number of parameters will be.

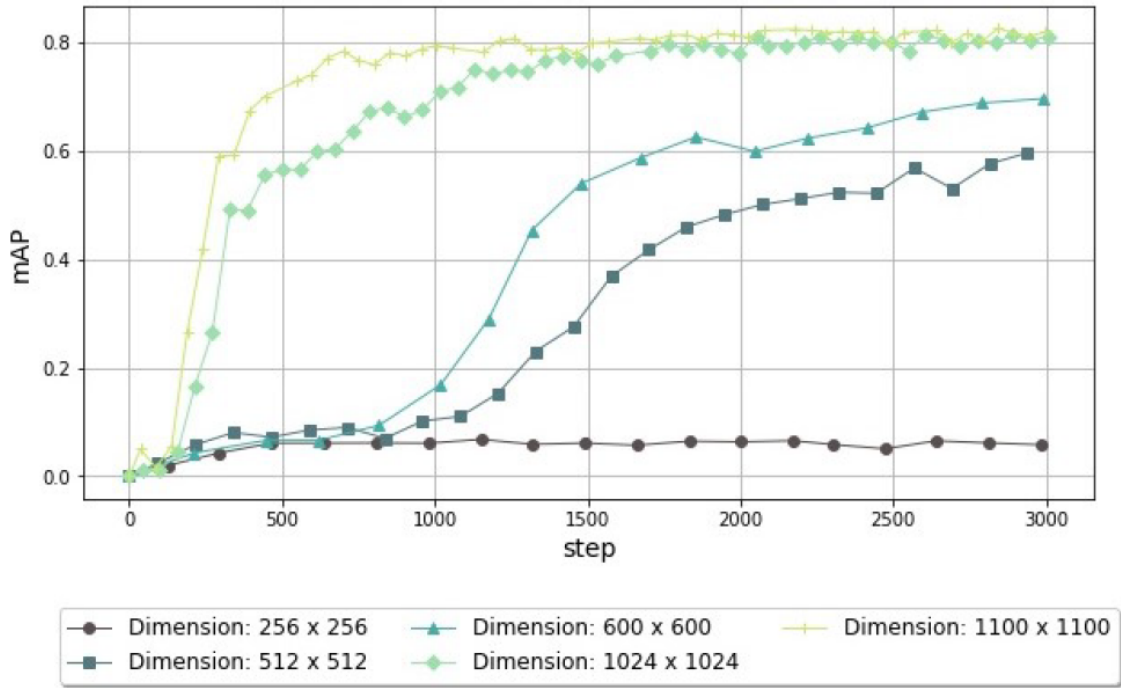


Figure 6.5: mAP depending on the dimension of the input image

It is clearly visible in figure 6.5 that the dimension of the image influences significantly the accuracy of the model, probably because of the presence of small objects in the input data. This fact is confirmed by the results presented in table 6.4, where it is possible to verify a decrease in the RPN localization loss and RPN objectness loss on the images with

higher resolution.

Table 6.4: Total losses, RPN losses in validation, mAP and speed of training for 3000 steps using different input images dimensions

Size	Total Loss		RPN Validation Loss		mAP	Speed/epoch (sec)
	Validation	Training	Localization	Objectness		
256x256	3.47	4.68	2.92	0.63	0.06	28.22
512x512	3.19	2.79	1.82	0.29	0.57	38.80
600x600	2.61	3.31	1.65	0.16	0.70	45.55
1024x1024	1.86	2.41	0.89	0.07	0.81	83.77
1100x1100	2.28	2.33	0.82	0.06	0.82	94.13

Furthermore, the dimension will also affect the convergence of the model: while using the raw dimension (512x512), the model will start to stabilize only at 3000 steps, by using the maximum dimension a convergence state will be achieved at around 1000 steps, as can be seen in figure 6.5. Moreover, it is possible to conclude that dealing with images of low resolution does not allow the model to improve the performance during training. At last, and as expected, by increasing the dimension, the time each epoch will take will also increase.

The following experiments will be carried on using input images resized to the dimension 1100x1100 pixels.

6.6.2 Resize method

To analyze the effect of the resize method on the model performance, different interpolation resizing approaches were tested, namely bilinear, nearest neighbor, bicubic and area.

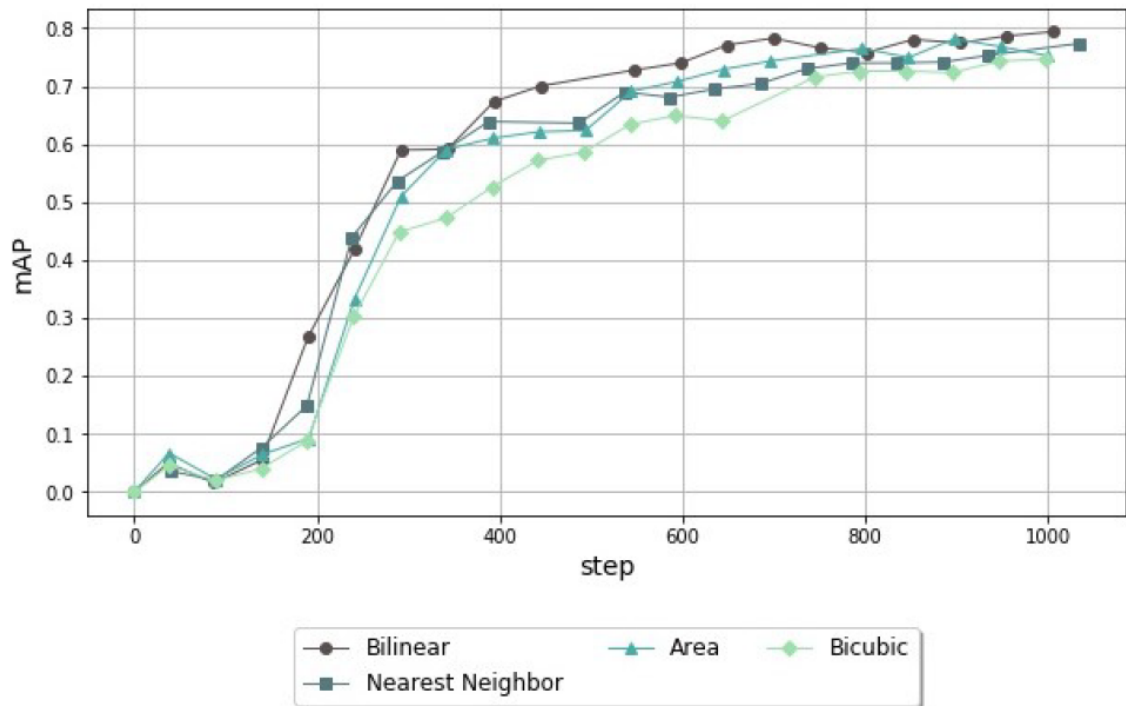


Figure 6.6: mAP using different resize methods

For 1000 training steps, the results in figure 6.6 show that, while not having significant differences on accuracy, the bilinear approach give slightly better results than the other methods (0.79 mAP), followed by Nearest Neighbor (0.77 mAP), Area (0.75 mAP) and finally Bicubic (0.75 mAP). There are no apparent differences regarding the losses and the speed of the training using the different techniques. The following tests will apply a bilinear resize method to the input images.

6.6.3 Using normalization

Image normalization could be a crucial step while applying CNNs for image classification. With the purpose of evaluating the effect of image normalization on the model, this scaling was applied to input images, and training loss had been analyzed for both situations.

Table 6.5: Training loss and speed using image normalization

Normalization	Train Total Loss	Speed/epoch (sec)
False	2.27	94.13
True	4.36	97.44

Considering the results exposed on table 6.5, it is possible to assume that, in this particular project, image normalization is not an approach to be used since training loss is higher when normalizing images. This can happen due to the fact of the use of a pre-trained model where non-normalized images were used. Concerning speed, applying normalization leads to slightly slower training.

Normalization will not be applied for the remaining tests.

6.6.4 Using grayscale

Similarly to normalization, converting images to grayscale can influence the behavior of the model.

Table 6.6: Training loss and speed using grayscale images

Grayscale	Train Total Loss	Speed/epoch (sec)
False	2.27	94.13
True	3.83	93.71

The results listed in table 6.6 lead to the conclusion that, alike normalization, converting images to a grayscale tone lead to worse results in the loss. Again, this could be associated to the fact of recurring to a pre-trained model where no grayscale conversion was applied. The speed decreases slightly since only one channel for each image is processed. Thus, image conversion to grayscale will not be used on the remaining experiments.

6.7 The training process

6.7.1 Batch size

As mentioned previously, the batch size is highly associated with the memory available on the machine. By resizing the image to 1100x1100 pixels, the training was limited to a stochastic approach. For future works where the memory available could be higher and as a matter of reference, it was tested the influence of the batch size for an input image of 600x600 pixel, using a stochastic process versus using a batch size of two (per GPU).

Table 6.7: mAP and speed using different batch sizes

Batch Size	mAP by Epoch								Speed	
	25	50	100	150	200	250	300	350	step (sec)	epoch (sec)
1	0.07	0.17	0.60	0.70	0.70	-	-	-	2,25	45
2	0.05	0.06	0.44	0.62	0.65	0.67	0.68	0.70	3,71	37.1

According to table 6.7, the results show that by using a stochastic training, the model achieves 0.7 mAP during the 150th epoch / 3000 steps, which takes around 113 minutes to train. On the other hand, by training with a batch size of 2, the model achieves the same accuracy only in the 350th epoch / 3500 steps, after 216 minutes of training. Taking into account those values, we can assume that, and for this particular case, a smaller batch seems to speed up the converge of the model, even if the computational speed is higher by using a larger batch.

6.7.2 The optimizer

Adam is the state of art of optimizers in object detection. The nature of the algorithm itself, joining characteristics of RMSProp and Momentum make it outperform the previous ones. While in the original Faster R-CNN the optimizer applied was momentum, three optimization algorithms were tested as a way to check the impact on them on the model itself. It was verified that Adam should be the one to be used on further experiments.

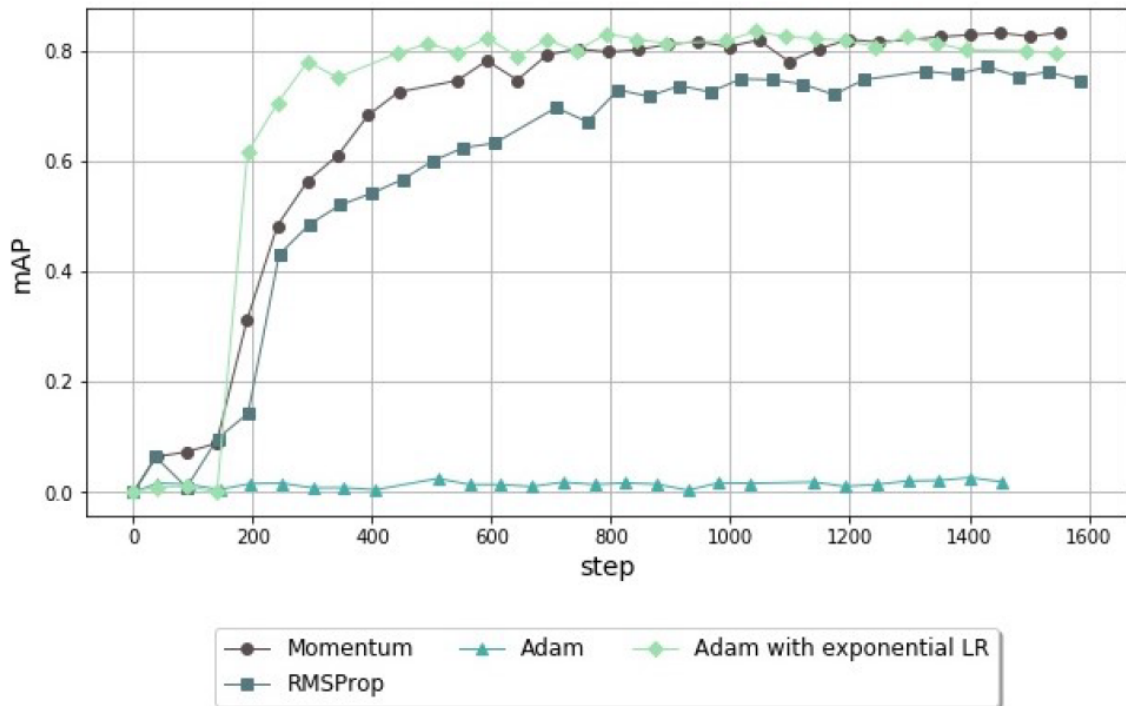


Figure 6.7: mAP using different optimizers

In figure 6.7, we can verify that the optimizer Adam with exponential LR and Momentum are the ones that achieve better results, but Adam has the advantage of speeding up the convergence of the model.

Table 6.8: mAP and speed using different optimizers

Optimizer	Best mAP	step best mAP	0.8 mAP step	Speed/epoch (sec)
Momentum	0.843	2865	744	95.04
Adam	0.027	1402	-	91.59
RMSProp	0.795	1739	-	93.16
Adam w/ Exp_LR	0.835	1043	493	95.70

There is no significant evidence in this case that the speed of training is influenced by the optimizer used, as opposed to the models' convergence. We can also verify based on the table 6.8 that all the optimizers can achieve reasonable rates of accuracy, but they should be fine-tuned regarding the learning rate and possibly other parameters in order to optimize their behavior. Clear evidence on this is that by using Adam optimizer with the default values, the model cannot increase their accuracy for more than 0.027, but when the initial learning rate and the type of decay were adjusted, this value increased to a mAP of 0.835.

6.7.3 The learning rate

Assuming Adam as the best optimizer to proceed with the experiments, it was tested the impact that different initial learning rates and decay steps could have on the model performance. After some initial tests, not exposed here for the sake of brevity, it was verified that the best initial LR range was between 0.0001 and 0.0005.

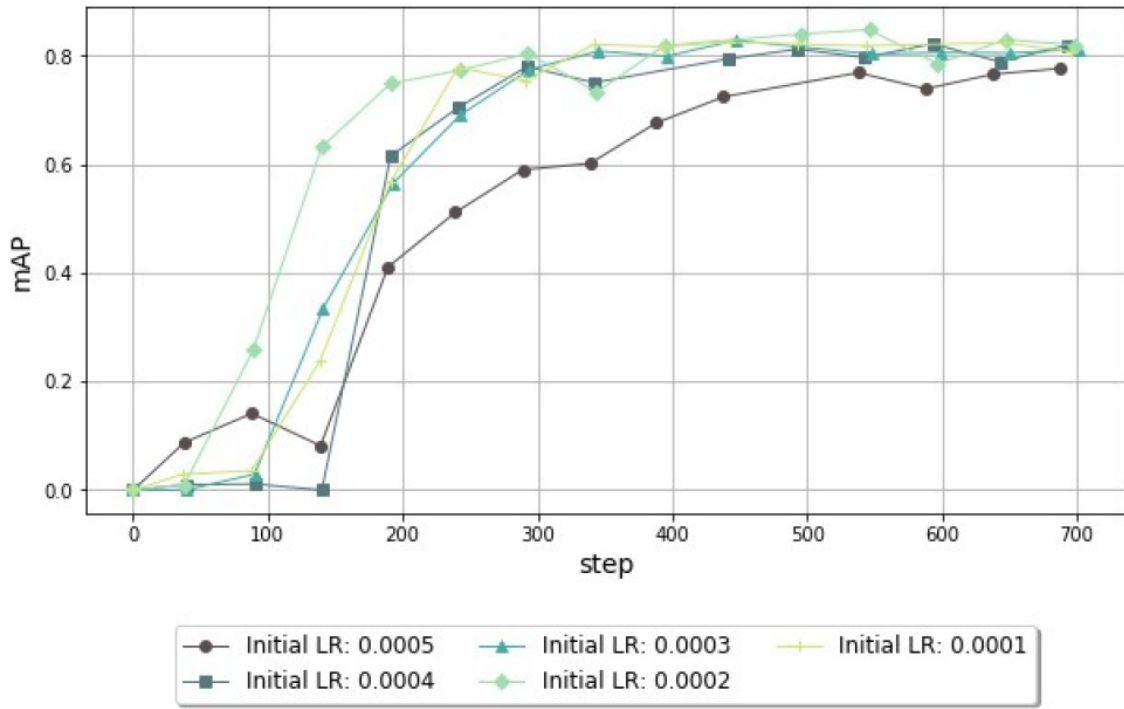


Figure 6.8: mAP using different initial learning rates (100000 decay steps)

In the figure 6.8 we can verify that an initial rate of 0.0002 seems to be the one that give better accuracy and boost the performance of the model in earlier stages, compared to the other values.

Table 6.9: mAP according to the decay steps (initial LR of 0.0002)

Decay steps	mAP at step				
	100	200	300	400	500
25000	0.06	0.39	0.74	0.81	-
50000	0.59	0.73	0.82	0.81	0.82
100000	0.26	0.75	0.80	0.82	0.84

By increasing the decay steps, the decrease of the learning rate will be slower, and a decay step of 100000 is proven as one good option regarding this parameter.

6.7.4 Data Augmentation

Data augmentation is a set of techniques that can have positive effects on the overall accuracy in situations where those techniques enhance the current dataset. To test that effect, a set of experiments was carried out using different techniques individually by 400 steps (20 epochs). The results are shown in table 6.10.

Table 6.10: mAP using different data augmentation techniques

Augmentation	mAP at step			
	100	200	300	400
(00) None	0.12	0.32	0.51	0.59
(01) Horizontal Flip	0.51	0.75	0.81	0.82
(02) Vertical Flip	0.12	0.57	0.78	0.81
(03) Pixel Value Scale	0.40	0.76	0.78	0.81
(04) Image Scale	0.72	0.79	0.80	0.80
(05) RGB to Gray	0.47	0.79	0.81	0.81
(06) Adjust brightness	0.68	0.82	0.82	0.80
(07) Adjust contrast	0.01	0.02	0.29	0.76
(08) Adjust hue	0.02	0.01	0.05	0.58
(09) Adjust saturation	0.02	0.11	0.66	0.78
(10) Distort color	0.25	0.75	0.80	0.81
(11) Jitter boxes	0.76	0.77	0.82	0.79
(12) Crop image	0.67	0.78	0.76	0.80
(13) Pad image	0.00	0.00	0.00	0.04
(14) Crop pad image	0.00	0.01	0.00	0.12
(15) Crop to aspect ratio	0.36	0.80	0.82	0.81
(16) Black patches	0.02	0.40	0.75	0.80
(17) Rotation 90	0.04	0.45	0.76	0.80
(18) Top4: (01)(06)(11)(15)	0.68	0.78	0.78	0.81
(19) All but (08)(13)(14)	0.51	0.74	0.81	0.82
(20) All	0.53	0.75	0.78	0.80

It is possible to verify that applying data augmentation can increase the performance of the model significantly, in most cases having a boost of over 30% of the accuracy comparing to not using any type augmentation. Still, augmentations using padding (13)(14) reduce the efficiency of the model to a state near zero accuracy. Adjusting the hue also seem to have a negative impact on the final mAP.

The summary of the results shows that the augmentation techniques that seem to have more effect in a short term of the training are the image scale, the jitter boxes, the brightness adjustment and cropping the image. The saturation and the hue adjustment prove their efficiency only after more than 300 steps of training. After checking what were the techniques more worthwhile to the performance of the model individually, it was

tested three different situations: create a set of the techniques that gave the best result (0.82 mAP) individually, named as Top4 (18), test all augmentation techniques applied simultaneously(20), and finally test the set with all the techniques that gave good results, removing the variables that influence in a high level the performance of the model, including the adjust hue technique to the techniques to be excluded, since the results of that technique were not undoubtedly good(19). While the differences in the results are only slightly different, we can verify that using a set with techniques that have been proven individually as efficient can give better results than using a technique in an isolated way, using all the techniques possible (20) and improve significantly the model that do not use any type of data augmentation.

The model that achieves better mAP during the first 400 steps was the model (19) with a mAP of 0.83 in step 342.

6.7.5 The gradient clipping

By default, the gradient clipping parameter has a value of 10. By testing the values of 5 and 15, the results were the ones presented in the table 6.11.

Table 6.11: mAP according to the gradient clipping value

Gradient clipping value	mAP at step				
	100	200	300	400	max(step)
10 (default)	0.51	0.74	0.81	0.82	0.83(342)
5	0.71	0.80	0.82	0.81	0.81(244)
15	0.64	0.80	0.82	0.81	0.83(348)

Gradient clipping technique will allow preventing vanishing and exploding gradient problems and according to the results shown in table 6.11, the performance of the model does not seem to be affected significantly by changing this parameter.

6.7.6 The number of steps

The number of steps is conditioned by several aspects, namely the convergence of the model, the need for applying early stopping due to overfitting, time limitations, among others. This parameter is going to suffer adjustments after the fine-tuning process.

6.7.7 The maximum number of boxes

By default, it was defined the maximum number of boxes as 700, since the maximum value of cells in an image in our dataset had 661 cells. The minimum value for this parameter should be at least the maximum value of objects present in an image on the dataset used.

6.8 Configurations on feature extractor

6.8.1 The stride

The output stride of the extracted RPN feature map was tested for 8 and 16 strides.

Table 6.12: mAP, speed and RPN losses depending on the stride of the feature extractor

Stride	mAP at step					Speed/ epoch(sec)	RPN Loss	
	100	200	300	400	max(step)		Objectness	Localization
8 (default)	0.51	0.74	0.81	0.82	0.83(342)	94.99	0.06	0.87
16	0.20	0.24	0.33	0.27	0.33(306)	46.44	0.33	3.49

As presented in table 6.12, while increasing the stride will reduce by more than half the time of training, the accuracy of the model is profoundly affected with this parameter change, where the maximum mAP achieved for 400 steps was 0.33. It is also possible to verify that this difference concerning the accuracy of the model is mainly due to the loss increase in the RPN stage.

6.8.2 Batch norm trainable

In this experiment, it was checked if updating the batch norm parameters during training would have an impact on the performance of the model. The updating is advisable especially in cases where we deal with large batch sizes.

Table 6.13: mAP and speed depending on the use of the batch normalization

Batch Norm	mAP at step					Speed/ epoch (sec)
	100	200	300	400	max(step)	
FALSE (default)	0.51	0.74	0.81	0.82	0.83(342)	94.99
TRUE	0.0	0.0	0.0	0.0	0.11 (37)	92.05

By turning this parameter to True, we can verify in the table 6.13 that the model after the first steps stops learning, which could be related to a case of vanishing gradient.

6.9 First stage of Faster R-CNN

6.9.1 Anchor generator

The grid anchor generator can be defined by the stride, the scales, and the aspect ratios. The following experiments show the behavior of the training according to some changes to those parameters. As a note, at any given location, i.e., in each anchor, the quantity of the scales times the number of the aspect_ratios anchors are generated with all possible combinations of scales and aspect ratios.

Aspect ratio In chapter 4, during exploration of the dataset and the images features, it was possible to conclude in figure 4.5 that the cells ratio range between 0.5 and 2.0. Those values are not affected by the resizing of the image from 512x512 pixels to 1100x1100 pixels considering that the aspect ratio was kept.

Table 6.14: mAP and speed depending on the aspect ratio of the anchors

Aspect ratio	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
[0.5, 1.0, 2.0] (default)	0.51	0.74	0.81	0.82	0.83(342)	94.99
[0.5, 1.0, 1.5, 2.0]	0.74	0.80	0.80	0.82	0.83(139)	96.00

The results in table 6.14 show that adding one more aspect ratio to the anchor generator do not improve the performance of the model in the long term. While the mAP rises faster using this additional value, we can verify that using only three aspect ratios are enough to detect the objects in our image, because eventually the bounding boxes are readjusted to the ground truth boxes using the losses regarding localization in the first and second stage of the object detector.

Scales The image resizing to 1100x1100 pixels will affect the value of the scales to be used in the anchor generator. In this way, the values presented in chapter 3 regarding the values of width and height of the cells and their distribution should suffer adjustments to the new dimensions. The table 6.15 show those values already scaled for the new reality, and the descriptive statistics associated.

Table 6.15: Width and height of the cells rescaled to the new image resolution

Dimension	Mean	Std	Min	25%	50%	75%	max
Width	34.2	9.8	9	28	32	41	95
Height	32.6	9.4	6	26	32	39	97

To calculate the best scales to use for the anchors, and since the size basis for the anchor is 256 pixels, it was scaled the value of the pixels presented on table 6.15 to this size. In this way, we can say that the mean of the cells in width and height is around 0.13 of the basis anchor size, the minimum is around 0.03 and the maximum 0.38. Taking into account those values, with was experimented different scales to check the behavior of the mAP during 400 steps.

Table 6.16: mAP and speed according to different scales used in anchor generator

Scale	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
[0.25,0.5,1.0,2.0] (default)	0.74	0.80	0.80	0.82	0.83(139)	96.00
[0.05,0.15,0.3,0.5]	0.58	0.76	0.79	0.83	0.83(400)	97.11
[0.15,0.3,0.5]	0.76	0.82	0.81	0.82	0.83 (339)	95.74
[0.15,0.3,0.5,1.0]	0.67	0.82	0.82	0.85	0.85 (400)	96.77
[0.3]	0.01	0.02	0.01	0.01	0.02(194)	92.82

According to the results shown in table 6.16, we can verify that limit the scales to only approximate values similar to the cell size increases overfitting, leading to poorer performance. However, reducing the lowest value of the scale improves the model since it is more adjusted to the reality of the problem. Regarding speed, there is a slight difference in the training time, where we can verify that the less number of scales, the lower will be the step time.

By testing a single scale (in this case it was tested the value 0.3), we can corroborate that the detection of the objects is profoundly affected. After concluding that the scales list [0.15,0.3,0.5,1.0] as the best achieved on the previous experiments, the mAP was re-evaluated, using the default list of aspect ratios in order to conclude what would be the best configuration to continue the experiments.

Table 6.17: mAP and speed according to different scales and aspect ratios used in anchor generator

Aspect ratio	Scale	mAP at step					Speed/ Epoch(sec)
		100	200	300	400	max(step)	
[0.5,1.0,2.0]	[0.15,0.3,0.5,1.0]	0.73	0.76	0.80	0.81	0.82 (238)	95.24
[0.5,1.0,1.5,2.0]	[0.15,0.3,0.5,1.0]	0.67	0.82	0.82	0.85	0.85 (400)	96.77

Joining scales and aspect ratio information, we can confirm in table 6.17 that the combination of the aspect ratio [0.5,1.0,1.5,2.0] and scales [0.15,0.3,0.5,1.0] is the one that gives better results concerning mAP. By following this configuration, around each anchor is going to be defined 16 different bounding boxes.

Stride The stride parameter will define the space between the anchor centers in the image. By default, this value is defined as 8 in Faster R-CNN. It was tested the influence of it by changing this value to 16. In order to apply a value of 16 in this parameter, the previous parameter already explored regarding the stride of the feature extractor also needs to be updated to 16.

Table 6.18: mAP and speed according to the different strides applied on anchor generator

Stride	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
8 (default)	0.67	0.82	0.82	0.85	0.85 (400)	96.77
16	0.76	0.80	0.80	0.82	0.82 (400)	46.01

According to the results in table 6.18, using a smaller stride give a better performance to the model as expected. Regarding the training speed, and as seen already in previous experiments, multiplying the stride by two allow to cut off the time of training for more than half.

6.9.2 Atrous rate

The atrous rate appears as a strategy that allows detecting smaller objects in object detection models. By default, the atrous rate is being applied on the previous experiments, but in order to evaluate the effect on this parameter on the mAP, it was tested the model without an atrous rate (i.e., atrous rate = 1).

Table 6.19: mAP and speed depending on the use of atrous rate

Atrous Rate	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
2 (default)	0.67	0.82	0.82	0.85	0.85 (400)	96.77
1	0.63	0.75	0.82	0.84	0.84(400)	96.77

Table 6.19 shows that using an atrous rate during training give slightly better results than not using atrous rate at all.

6.9.3 Network hyperparameters

Regularizer The regularizer can be an efficient approach in order to deal with overfitting situations and can improve the performance of the model in the validation set.

Table 6.20: mAP and speed using different regularizers during the first stage of the object detector

Regularizer	Weight	mAP at step					Speed/ Epoch(sec)
		100	200	300	400	max(step)	
None	-	0.67	0.82	0.82	0.85	0.85 (400)	96.77
l2	0.5	0.71	0.81	0.82	0.79	0.82 (286)	96.00
l2	1.0	0.59	0.68	0.78	0.8	0.80(400)	99.65
l1	0.5	0.01	0.01	0.36	0.61	0.61(400)	96.00
l1	1.0	0.36	0.24	0.42	0.63	0.63(400)	96.00

We can verify in table 6.20 that with the present configurations, applying a regularizer decrease the model performance, and should be avoided on this stage of the algorithm.

Initializer Due to the decision of using the weights from a pre-trained model, the effect of the initializer was not tested.

Activation By default, Faster R-CNN uses the RELU activation on the first stage of the system. It was tested the influence of RELU6 and also the non-utilization of an activation function in order to test the performance of the model.

Table 6.21: mAP and speed according to the activation used in the first stage

Activation	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
NONE	0.73	0.80	0.81	0.82	0.82(400)	96.00
RELU (default)	0.67	0.82	0.82	0.85	0.85 (400)	96.77
RELU6	0.69	0.80	0.82	0.84	0.84(400)	97.56

The results on table 6.21 prove that while there are no significant differences by using different types of activations, the RELU activation seems the one that gives slightly better results.

Batch normalization The batch normalization was tested as a hyperparameter of the convolutional neural networks in the first stage.

Table 6.22: mAP and speed depending on the use of batch normalization in the first stage

batch normalization	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
True	0.72	0.77	0.78	0.78	0.79(322)	106.89
False (default)	0.67	0.82	0.82	0.85	0.85 (400)	96.77

As seen in table 6.22, using batch normalization tend to lead to worst performances of the model, decreasing the same by more than 0.06 mAP.

Regularize depthwise Regularizing depthwise is also a technique in order to prevent overfitting situations. It was tested the model performance by turning on this parameter, and the results can be seen in table 6.23.

Table 6.23: mAP and speed by using depthwise regularization on the first stage

Regularize depthwise	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
True	0.75	0.79	0.76	0.78	0.80(233)	97.56
False (default)	0.67	0.82	0.82	0.85	0.85 (400)	96.77

Taking into account that the model is not suffering overfitting problems in the first stage of the algorithm, applying regularization on a depth level will not improve the performance of the model.

6.9.4 Box predictor

The box predictor can be configured based on two different parameters: the kernel size and the depth.

Kernel size The kernel size will define the filters size used for the convolutional operation immediately before the RPN box predictions.

Table 6.24: mAP and speed according to the kernel size of box predictor

Kernel size	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
2	0.77	0.81	0.82	0.82	0.82(238)	95.24
3 (default)	0.67	0.82	0.82	0.85	0.85 (400)	96.77
4	0.70	0.78	0.83	0.82	0.84(357)	104.35

As listed in table 6.24, changing the kernel size to distinct values than the default one generate slightly weaker results. A kernel size of 3 was defined as the value to proceed on further tests.

Depth The depth will define the output depth for the convolution operation immediately before the RPN box predictions.

Table 6.25: mAP and speed according to the depth of box predictor

Depth	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
256	0.66	0.80	0.80	0.81	0.83(375)	93.07
512 (default)	0.67	0.82	0.82	0.85	0.85 (400)	96.77
1024	0.53	0.79	0.79	0.80	0.80(400)	99.74

By cutting the depth by half, and doubling the value, we obtain lower values concerning mAP, as shown in table 6.25, which lead to the decision of maintaining the value of 512 for this parameter in the following experiments.

6.9.5 Minibatch size

Changing the size of the batch to use for computing the first stage objectness and location losses lead to a lower loss regarding RPN localization and objectness when using 512 batches, as can be seen in table 6.26. However, since the mAP is still higher using the default value for the parameter, the following tests maintain the batch size as 256.

Table 6.26: Total loss, RPN Validation Loss, mAP and speed depending on the size of the minibatch

Size	Total Loss		RPN Validation Loss		Best mAP	Speed/ Epoch(sec)
	Validation	Training	Localization	Objectness		
128	2.82	3.07	0.87	0.06	0.83	96.00
256 (default)	2.75	2.61	0.95	0.05	0.85	96.77
512	2.59	2.15	0.54	0.04	0.83	96.00

Using a minibatch size of 128 came with a higher loss in training and validation, while the speed of each training epoch does not suffer significant changes.

Positive balance fraction The positive balance fraction will define the admitted percentage of positive examples per image for the RPN. In the original Faster R-CNN implementation, this parameter assumes the value of 0.5.

Table 6.27: Total loss, RPN Validation Loss, mAP and speed depending on the positive balance fraction

Positive balance Fraction	Total Loss		RPN Validation Loss		Best mAP	Speed Epoch(sec)
	Validation	Training	Localization	Objectness		
0.25	2.53	1.67	0.41	0.04	0.83	97.30
0.5 (default)	2.75	2.61	0.95	0.05	0.85	96.77
0.75	2.49	2.36	0.43	0.04	0.81	96.59

By cutting this parameter by half, and doubling the value, we can verify in table 6.27 that localization loss in RPN is affected, and training total loss is highly reduced in the first situation. In spite of that, and since the best mAP value is associated with a 0.5 value for this parameter, it was decided to maintain this rate for the following experiments.

6.9.6 NMS

IoU threshold The IoU threshold will allow establishing the threshold used to define what should be considered overlapping bounding boxes during the non-max suppression technique. It was evaluated the performance of the model under various IoU thresholds, ranging from 0.3 to 0.9. Results are shown in table 6.28.

Table 6.28: Total loss, Box Validation Loss, mAP and speed according to the IoU threshold used

IoU threshold	Total Loss		Box Validation Loss		Best mAP(step)	Speed Epoch(sec)
	Validation	Training	Localization	Classification		
0.3	1.15	2.13	0.16	0.17	0.84(317)	102.4
0.4	1.36	1.98	0.19	0.40	0.84(400)	98.09
0.5	1.61	1.88	0.33	0.49	0.84 (329)	97.30
0.6	2.17	2.78	0.58	0.72	0.83(332)	96.26
0.7 (default)	2.75	2.61	0.74	1.02	0.85 (400)	96.77
0.8	2.13	2.95	0.57	0.73	0.83(337)	96.40

By decreasing the IoU threshold in non-max suppression, the number of accepted proposals will be higher which lead to an increase on the training time, as it is possible to verify in table 6.28. On the other hand, the decrease in the IoU threshold seems to affect the validation losses on the second stage regarding classification and localization highly. The option to choose the IoU threshold as 0.4 is a decision where time and losses insights are combined, even if the mAP is slightly worse than using a threshold of 0.7.

6.9.7 Maximum proposals

The number of maximum proposals should always be in a minimum of the maximum number of ground truth boxes present in the input images. In this way, it was tested the influence of the number of proposals on the performance of the model, and the results are tabulated in 6.29.

Table 6.29: Total loss, Box Validation Loss, mAP and speed based on the number of proposals

Maximum Proposals	Total Loss		Box Validation Loss		Best mAP(step)	Speed/ Epoch(sec)
	Validation	Training	Localization	Classification		
1000	1.67	2.39	0.39	0.40	0.83(340)	95.49
1500	1.34	1.73	0.25	0.32	0.83(381)	97.56
2000 (default)	1.36	1.98	0.19	0.40	0.84(400)	98.09
2500	1.29	1.73	0.16	0.33	0.83(400)	101.05
3000	1.15	1.73	0.11	0.25	0.85(310)	104.34
3500	1.12	1.75	0.12	0.20	0.84(213)	106.80

As shown in table 6.29 , increasing the number of proposals will raise the training time. While the final mAP does not show clear differences between the number of proposals, the model seems to perform better when this parameter is defined in 3000 proposals. We can conclude that the number of proposals generated can impact the speed of training

efficiently without any significant effects on the accuracy of the model. On the other hand, the losses associated with the second stage of the algorithm show a significant reduction by the increasing of proposals. Even though, that shrinkage is not visibly reflected on final mAP. For further experiments, the number of proposals is defined as 3000, since it shows the highest value regarding mAP.

6.9.8 Loss weights

Localization and Objectness In this step of experiments, the goal was to evaluate how changes on the weights of the losses of the first stage would affect the final performance of the model. By default, the faster R-CNN system applies a double loss weight in the localization compared to objectness loss. The results are summarized in table 6.30.

Table 6.30: mAP and speed changes using different combinations of localization and objectness loss weights

Localization	Objectness	mAP at step					Speed/ Epoch(sec)
		100	200	300	400	max(step)	
2.0	1.0 (default)	0.59	0.83	0.85	0.85	0.85(310)	104.34
1.0	1.0	0.75	0.80	0.84	0.82	0.84(314)	103.50
1.0	2.0	0.74	0.80	0.84	0.83	0.84(312)	103.78

According to the results in table 6.30, changing the weight of the losses of the first stage do not affect the performance of the model significantly, but slightly better results are achieved when the default configuration is used.

6.9.9 ROI pooling

After the RPN phase, and having the final proposals selected and filtered during NMS technique, the Region of Interest pooling is performed. In this phase, the proposals should be redimensioned in order to proceed to the second phase.

Initial crop size The crop size will define the size to crop the final region proposals. This parameter was changed to different values in order to perceive the effect of it in the model performance.

Table 6.31: mAP and speed according to the initial crop size

Crop size	mAP at step					Speed/
	100	200	300	400	max(step)	Epoch(sec)
14	0.73	0.83	0.84	0.79	0.84(376)	98.90
17 (default)	0.59	0.83	0.85	0.85	0.85(310)	104.34
20	0.74	0.82	0.83	0.82	0.83(299)	107.97

According to the results shown in table 6.31, the default configuration is the one to choose between the options tested. There is no evident difference between the several validation losses of the algorithm. Concerning speed, the lower the crop size, the faster will be the training process. The higher mAP using the default configuration defined the value of this parameter to further tests.

Maxpool Kernel size and stride In ROI pooling phase, the values corresponding to kernel size and the stride should be defined. By default, it is applied a 1x1 pooling. Other combinations were tested, and the results are exhibited in table 6.32.

Table 6.32: mAP and speed according to the maxpool kernel size

Maxpool kernel		mAP at step					Speed
size	stride	100	200	300	400	max(step)	Epoch(sec)
1	1 (default)	0.59	0.83	0.85	0.85	0.85(310)	104.34
3	3	0.72	0.83	0.83	0.85	0.85(400)	104.76
3	1	0.66	0.80	0.82	0.81	0.84(339)	92.04

With no evident differences in losses, the different configurations result in similar performances of the model, with a significant difference regarding speed when applying a kernel 3x3 with a stride of 1.

6.10 Second stage of Faster R-CNN

The second stage in Faster R-CNN will allow to refine the location of the bounding boxes and ultimately, classify the objects identified. The remaining layers of the CNN should be tuned to complete the training.

6.10.1 Box predictor network hyperparameters

Regularizer The second stage demands the hyperparameters evaluation of the remaining network. As so, it was tested the impact of adjusting the regularizer for this phase, as can be seen in table 6.33.

Table 6.33: mAP and speed using different regularizers during the second stage of the object detector

Regularizer	Weight	mAP at step					Speed/ Epoch(sec)
		100	200	300	400	max(step)	
l2	0.0 (default)	0.59	0.83	0.85	0.85	0.85(310)	104.34
l2	0.5	0.52	0.78	0.82	0.82	0.83(310)	104.92
l2	1.0	0.74	0.82	0.82	0.84	0.84(400)	103.79
l1	0.5	0.71	0.82	0.85	0.84	0.85(311)	104.35
l1	1.0	0.60	0.81	0.83	0.81	0.83(357)	104.07

As table 6.33 indicates, the speed of training is not affected by the application of regularizers. Further, the mAP does not seem to improve by the use of regularizers, and there are no notable differences between the several losses evaluated along the process. Thus, the remaining tests for parameter tuning were made with no resource to any regularizer.

Initializer Due to the decision of using the weights from a pre-trained model, the effect of the initializer was not tested.

Activation The activation function used was examined for the second stage of the algorithm. Results are presented in table 6.34.

Table 6.34: mAP and speed according to the activation used in the second stage

Activation	mAP at step					Speed/ Epoch(sec)
	100	200	300	400	max(step)	
NONE	0.77	0.83	0.83	0.82	0.84(221)	106.16
RELU (default)	0.59	0.83	0.85	0.85	0.85(310)	104.34
RELU6	0.41	0.78	0.82	0.85	0.85(400)	103.78

Again, results show that changing the activation function do not affect in a significant way the performance of the model. Maintaining the RELU as the option to the next experiments, it will be tested in the final finetuning of the model the use of RELU_6, since this activation gives better results when applying the regression metrics.

Batch normalization The influence of batch normalization was analyzed, and results can be observed in table 6.35.

Table 6.35: Total Loss, Box validation loss, mAP and speed depending on the use of batch normalization in the second stage

Maximum Proposals	Total Loss		Box Validation Loss		Best mAP(step)	Speed/ Epoch(sec)
	Validation	Training	Localization	Classification		
False (default)	1.15	1.73	0.11	0.25	0.85(310)	104.34
True	1.87	2.15	0.20	0.89	0.81(400)	104.35

The results defend that batch normalization should not be applied again on this stage since it will decrease the mAP value for more than 0.04. Furthermore, batch normalization has a significant negative impact on the classification loss, increasing this value from 0.25 to 0.89.

Regularize depthwise Regularization was applied at a depth level at the layers of the CNN in the second stage of Faster R-CNN.

Table 6.36: Total Loss, Box validation loss, mAP and speed by using depthwise regularization on the second stage

Maximum Proposals	Total Loss		Box Validation Loss		Best mAP(step)	Speed/ Epoch(sec)
	Validation	Training	Localization	Classification		
False (default)	1.15	1.73	0.11	0.25	0.85(310)	104.34
True	1.28	2.11	0.14	0.22	0.84(263)	105.21

While no meaningful differences can be found comparing both approaches, as can be seen in 6.36, differently from the first stage, applying this type of regularization at this step seem to improve the accuracy evaluated by the regression metrics, as exposed on 6.45.

6.10.2 Dropout

The use of dropout can be a handful technique when dealing with overfitting situations. While there was no evidence of an overfitting situation on the model fine-tuned until this moment, the dropout technique was applied in order to assess the effect of it on the network evaluation.

Table 6.37: Total Loss, Box validation loss, mAP and speed by using dropout

Maximum Proposals	Total Loss		Box Validation Loss		Best mAP(step)	Speed/ Epoch(sec)
	Validation	Training	Localization	Classification		
True (0.5)	0.80	1.94	0.14	0.28	0.84(355)	104.35
True (0.6)	0.80	1.70	0.13	0.29	0.84(353)	105.20
True (0.7)	0.84	2.08	0.15	0.26	0.83(307)	105.49
True (0.8)	0.80	1.71	0.13	0.33	0.83(349)	104.00
True (0.9)	0.81	2.33	0.14	0.22	0.83(400)	105.88
False (default)	1.15	1.73	0.11	0.25	0.85(310)	104.34

Comparing the results exposed in table 6.37, there is no clear indication of the benefit of using dropout during training in this case. Furthermore, the speed of training does not seem to suffer significant modifications, as defended in the bibliography. However, we can verify that applying dropout will drive to more significant differences between validation and training loss since dropout is activated only when training, and not applied during validation.

6.10.3 Post processing - Batch Non-Max suppression

Similarly to the first stage of Faster R-CNN, a Non-Max suppression is implemented again in the second stage in the detections as a way to avoid having more than one bounding box for each object. Thus, the IoU threshold should be adjusted, and the number of maximum detections should be defined.

IoU threshold To test the repercussion of adjusting the IoU threshold, different values for this parameter were tested in a range between 0.4 and 0.8.

Table 6.38: Total Loss, Box validation loss, mAP and speed according to the IoU threshold

IoU Threshold	Total Loss		Box Validation Loss		Best mAP(step)	Speed/ Epoch(sec)
	Validation	Training	Localization	Classification		
0.4	1.16	1.82	0.13	0.24	0.83(351)	105.43
0.5	1.20	1.49	0.13	0.25	0.83(400)	105.79
0.6 (default)	1.15	1.73	0.11	0.25	0.85(310)	104.34
0.7	1.24	2.00	0.14	0.31	0.84(307)	105.97
0.8	1.27	1.98	0.14	0.32	0.83(307)	105.78

In table 6.38, the default value for the IoU threshold is the one that gives better results involving mAP. The speed of training does not seem to suffer depending on the change of this parameter. Values higher than the default one seem to converge to worse results regarding total loss for validation and training, but not significant ones.

Maximum detections per class and total detections The maximum detections per class and total detections parameter are always limited by the highest value of objects in the input images.

Table 6.39: Total Loss, Box validation loss, mAP and speed according to to maximum detections

Maximum Detections	Total Loss		Box Validation Loss		Best mAP(step)	Speed/ Epoch(sec)
	Validation	Training	Localization	Classification		
700 (default)	1.15	1.73	0.11	0.25	0.85(310)	104.34
1000	1.24	2.12	0.14	0.25	0.83(400)	106.25
1500	1.40	1.74	0.12	0.44	0.86(303)	111.84

In order to understand if a higher number of detections would influence the final mAP, the model was tested using the value of 700, 1000 and 1500, as shown in table 6.39. The results defend that increasing the number of maximum detections imply a higher classification loss. However, using 1500 detections increases the mAP for the value of 0.86. Due to the loss increase mentioned previously, it was selected the default value of 700 to continue the tests. However, the version of 1500 detections is tested afterward as the magenta model in section 6.12. The speed in training is also affected by the number of detections applied: the higher the number of detections, the slower will be the process.

6.10.4 Loss weights

Localization and Objectness Having two different losses in this stage, the impact of changing the loss weights during the process was assessed.

Table 6.40: Total Loss, Box validation loss, mAP and speed changes using different combinations of localization and classification loss weights

Localization & Classification loss weights	Total Loss		Box Validation Loss		Best mAP(step)	Speed Epoch(sec)
	Validation	Training	Localization	Classification		
2.0 & 1.0 (default)	1.15	1.73	0.11	0.25	0.85(310)	104.34
1.0 & 1.0	1.16	1.39	0.07	0.23	0.83(315)	106.25
1.0 & 2.0	1.37	1.77	0.07	0.50	0.83(310)	106.53

Giving a double weight to localization loss lead to the best model in terms of mAP, as can be seen in table 6.40.

6.10.5 Classification loss

The classification loss can be changed to different options in order to adapt better the results for our problem. In this way, besides the weighted sigmoid, the bootstrapped and focal sigmoid were tested as options to the classification loss.

Table 6.41: Total Loss, Box validation loss and mAP using different classification losses

Classification Loss	Total Loss		Box Validation Loss		Best mAP(step)
	Validation	Training	Localization	Classification	
Weighted sigmoid (default)	1.15	1.73	0.11	0.25	0.85(310)
Bootstrapped sigmoid	1.03	1.36	0.13	0.12	0.84(400)
Sigmoid Focal	1.07	1.84	0.13	0.13	0.83(400)

Comparing the results in table 6.41, the best mAP is still achieved using weighted sigmoid. Nonetheless, using Bootstrapped classification grant lower total losses.

6.11 Evaluating the regression metrics of the models

Choosing the model in the previous steps was made taking into account mostly the mAP obtained after training. In a way to try to improve the model the best way possible, the different models were evaluated using the regression metrics RMSE, MAE, and MEDAE on the dataset with medium complexity. The results are shown in table 6.42, for the

use of different parameterizations to a faster R-CNN model with a feature extractor Inception Resnet v2, where the transfer learning applied is "only weights", and without normalization or the use of grayscale images.

Table 6.42: Regression metrics and mAP for different configurations in training and data augmentation parameters, where the symbol * represents the default model for Faster R-CNN

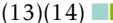
Parameter	Configuration	RMSE	MAE	MEDAE	mAP
Dimension	256x256	117.2	94.7	83.5	0.07
	512x512	52.9	33.4	17.5	0.60
	600x600*	21.1	15.5	11.5	0.74
	1024x1024	29.5	20.3	13.5	0.81
	1100x1100 	13.5	11.4	9.5	0.82
Resize Method	Bilinear* 	13.5	11.4	9.5	0.82
	Nearest Neighbor	15.6	11.5	8.5	0.77
	Bicubic 	14.0	10.6	7.0	0.78
	Area	17.8	15.9	13.5	0.75
Optimizer	Momentum * 	7.0	5.8	4.5	0.84
	Adam	139.2	129.8	122.0	0.03
	RMSProp	15.0	12.2	13.5	0.79
	Adam with exp. LR 	9.6	8.5	9.0	0.83
Initial Learning Rate	0.0005	11.1	8.9	9.5	0.80
	0.0004	9.6	8.5	9.0	0.83
	0.0003* 	6.1	5.4	5.0	0.83
	0.0002 	11.5	9.9	10.0	0.84
	0.0001	7.1	6.2	7.0	0.83
Decay steps	25000	17.0	13.0	7.5	0.33
	50000	13.9	11.1	10.0	0.83
	100000 	11.5	9.9	10.0	0.84
Data Augmentation	None	35.1	27.9	23.5	0.59
	Horizontal Flip *	12.0	9.1	7.5	0.82
	Vertical Flip	16.6	13.6	13.5	0.82
	Pixel Value Scale	12.3	9.9	9.5	0.81
	Image Scale	15.1	12.1	13.5	0.80
	RGB to GraY	8.5	7.8	9.0	0.81
	Adjust brightness	12.7	12.0	13.5	0.82
	Adjust Contrast	37.2	27.2	20.5	0.58
	Adjust Hue	15.5	12.6	9.0	0.78
	Adjust Saturation	12.5	8.4	5.0	0.81
	Distort color	16.6	13.6	12.5	0.83
	Jitter Boxes	13.0	9.9	6.0	0.76
	Crop Image	21.4	16.1	15.0	0.80
	Pad Image	135.3	122.9	116.5	0.04
	Crop pad Image	70.6	52.6	36.0	0.12
	Crop to aspect ratio	11.8	10.7	10.0	0.82
	Black Patches	17.9	14.0	13.0	0.80
	Rotation 90	15.1	11.9	9.0	0.80
	Top4 	7.2	5.9	6.5	0.81
	All but (08)(13)(14) 	15.5	11.9	11.5	0.83
	All	16.1	13.1	12.0	0.80

Table 6.43: Regression metrics and mAP for different configurations in some of the first stage parameters, where the symbol * represents the default model for Faster R-CNN

Parameter	Configuration	RMSE	MAE	MEDAE	mAP
Gradient Clipping	5 	12.0	9.5	8.5	0.81
	10 *     	15.5	11.9	11.5	0.83
	15	21.3	17.2	16.5	0.83
Feature Extractor Stride	8 *     	15.5	11.9	11.5	0.83
	16	30.7	24.6	18.0	0.33
Feature Extractor Batch Norm	TRUE	139.2	129.8	122	0.11
	FALSE *     	15.5	11.9	11.5	0.83
Aspect Ratios and Scales	[0.5,1.0,1.5,2.0]&[0.15,0.3,0.5,1.0]     	11.2	8.7	6.0	0.85
	[0.5,1.0,2.0]&[0.15,0.3,0.5,1.0]	11.3	9.0	6.5	0.82
Atrous rate	1 (FALSE)	11.3	9.0	6.5	0.84
	2 (TRUE) *     	11.2	8.7	6.0	0.85
1st Stage Regularizer	No regularizer *     	11.2	8.7	6.0	0.85
	l2 0.5	12.7	9.6	9.0	0.82
	l2 1.0	17.3	14.2	9.0	0.80
	l1 0.5	33.4	28.4	30.5	0.61
	l1 1.0	41.0	34.2	29.5	0.63
Activation	NONE	14.9	11.7	8.5	0.82
	RELU *     	11.2	8.7	6.0	0.85
	RELU_6	14.7	11.4	11.0	0.84
Batch Normalization	TRUE	25.2	17.4	11.0	0.79
	FALSE *     	11.2	8.7	6.0	0.85
Regularize	TRUE	73.1	38.9	20.0	0.80
Depthwise	FALSE *     	11.2	8.7	6.0	0.85

Table 6.44: Regression metrics and mAP for different configurations for the remaining first stage parameters, where the symbol * represents the default model for Faster R-CNN

Parameter	Configuration	RMSE	MAE	MEDAE	mAP
Kernel Size	3*	11.2	8.7	6.0	0.85
	2	11.7	8.5	6.5	0.82
	4	18.3	13.8	9.0	0.84
Depth	256	13.2	11.6	10.0	0.84
	512*	11.2	8.7	6.0	0.85
	1024	17.5	14.8	13.0	0.83
Minibatch Size	128	13.7	11.5	11.0	0.83
	256*	11.2	8.7	6.0	0.85
	512	13.1	9.0	5.0	0.83
Positive Balance Fraction	0.25	9.9	7.8	6.5	0.83
	0.5*	11.2	8.7	6.0	0.85
	0.75	18.5	15.0	11.5	0.82
NMS IoU threshold	0.3	14.2	11.4	8.0	0.84
	0.4	15.0	12.1	10.0	0.84
	0.5	11.5	10.1	10.0	0.84
	0.6	14.5	11.5	7.5	0.83
	0.7*	11.2	8.7	6.0	0.85
	0.8	13.7	10.9	9.5	0.83
Maximum Proposals	1000	11.9	8.9	6.5	0.83
	1500	15.3	11.2	8.5	0.83
	2000*	15.0	12.1	10.0	0.84
	2500	11.1	8.6	6.0	0.83
	3000	21.1	15.5	11.5	0.85
Loss weights	Localization 1.0 & Objectness 1.0	17.5	15.7	13.0	0.84
	Localization 2.0 & Objectness 1.0*	21.1	15.5	11.5	0.85
	Localization 1.0 & Objectness 2.0	11.0	9.7	8.5	0.84
Initial crop size	14	11.3	8.7	7.0	0.84
	17 *	21.1	15.5	11.5	0.85
Maxpool kernel & stride	Kernel 1 and Stride 1 *	21.1	15.5	11.5	0.85
	Kernel 3 and Stride 3	10.4	8.9	10.5	0.85
	Kernel 3 and Stride 1	11.3	8.8	8.5	0.84

Table 6.45: Regression metrics and mAP for different configurations for the second stage parameters, where the symbol * represents the default model for Faster R-CNN

Parameter	Configuration	RMSE	MAE	MEDAE	mAP
2nd Stage Regularizer	No regularizer *	21.1	15.5	11.5	0.85
	l2 0.5	13.8	12.0	11.0	0.83
	l2 1.0	10.7	8.3	6.0	0.85
	l1 0.5	10.6	9.2	8.5	0.85
	l1 1.0	9.3	7.6	5.5	0.83
Activation	NONE	17.4	13.2	12.5	0.84
	RELU *	21.1	15.5	11.5	0.85
	RELU_6	11.3	9.7	9.5	0.85
Batch Normalization	TRUE	10.9	8.8	7.0	0.81
	FALSE *	21.1	15.5	11.5	0.85
Regularize depthwise	TRUE	8.7	7.9	7.5	0.85
	FALSE *	21.1	15.5	11.5	0.84
Dropout	FALSE *	21.1	15.5	11.5	0.85
	0.5	17.1	14.1	15	0.82
	0.6	13.7	10.8	11.0	0.83
	0.7	17.3	13.1	8.5	0.81
	0.8	17.6	13.7	13.0	0.82
	0.9	10.0	8.8	8.0	0.83
NMS IoU threshold	0.4	10.3	9.2	8.0	0.83
	0.5	9.3	7.8	7.5	0.83
	0.6*	21.1	15.5	11.5	0.85
	0.7	18.8	14.3	8.5	0.84
	0.8	11.9	10.5	9.0	0.83
Maximum detections	700 *	21.1	15.5	11.5	0.85
	1000	9.4	7.2	7.5	0.83
	1500	22.6	17.3	15.0	0.86
Loss weights	Localization 1.0 & Classification 1.0	7.9	5.7	3.5	0.83
	Localization 2.0 & Classification 1.0 *	21.1	15.5	11.5	0.85
	Localization 1.0 & Classification 2.0	12.6	9.5	7.0	0.83
Classification Loss	Weighted sigmoid *	21.1	15.5	11.5	0.85
	Bootstrapped sigmoid	12.7	11.5	11.0	0.84
	Focal sigmoid	14.5	12.2	11.0	0.83

After the evaluation of the results of the tables 6.42, 6.43, 6.44 and 6.45, five best models are going to be chosen, and all are going to be run till 2000 steps in order to check the performance of them, namely:

1. **Fine-tuned model** - Takes into account the model fine-tuned from section 6.3 to 6.10. It is represented by the blue color.
2. **Best regression** - Takes into account the model with the best conditions regarding obtaining best regression metrics. It is represented by the yellow color.
3. **Mix regression and fine-tuned** - As a mix between the two last options, it will

choose the best regression metric results only if the mAP do not change more than 0.01 than the best mAP obtained on the present configurations. It is represented by the green color.

4. **Best mAP** - The model that obtained the highest score concerning mAP during the experiments. It is represented by the magenta color.
5. **Best RMSE&MAE** - The model that obtained the highest performance regarding regression metrics during the experiments, namely in RMSE and MAE. It is represented by the orange color.
6. **Best MEDAE** - The model that obtained the highest performance in terms of regression metrics during the experiments, namely in MEDAE. It is represented by the gray color.

6.12 Choosing the best model

In order to assess the best model, each configuration was trained for 600 steps (around 30 epochs) for the models using Adam optimizer (namely fine-tuned, best regression, mix regression&fine-tuned and best mAP) and for 2000 steps the models using momentum optimizer (namely Best RMSE & MAE and best MEDAE). The results for 600 steps regarding mAP are figured in 6.9. In table 6.46 are presented the main results to the five final models chosen.

Table 6.46: Main results of five final models

Model	best mAP	Level "Medium"			All levels		
		RMSE	MAE	MEDAE	RMSE	MAE	MEDAE
Fine-tuned	0.8457	9.8	9	9.5	64.5	30.8	10
Best regression	0.7705	122.0	112.1	99.5	223.4	160.4	100.0
Mix regression&fine-tuned	0.8300	18.7	15.4	13.5	81.0	41.2	15
Best mAP	0.8558	11.2	9.4	10.0	45.12	24.8	11
Best RMSE & MAE	0.8293	8.8	6.4	3.5	52.7	23.7	7.0
Best MEDAE	0.8426	10.0	7.5	6.0	24.8	17.4	13.0

As we can verify in table 6.46, the model with the highest mAP is not necessarily the one with the best regression results. Furthermore, due to the diversity of the images, different levels could have better results using different models.

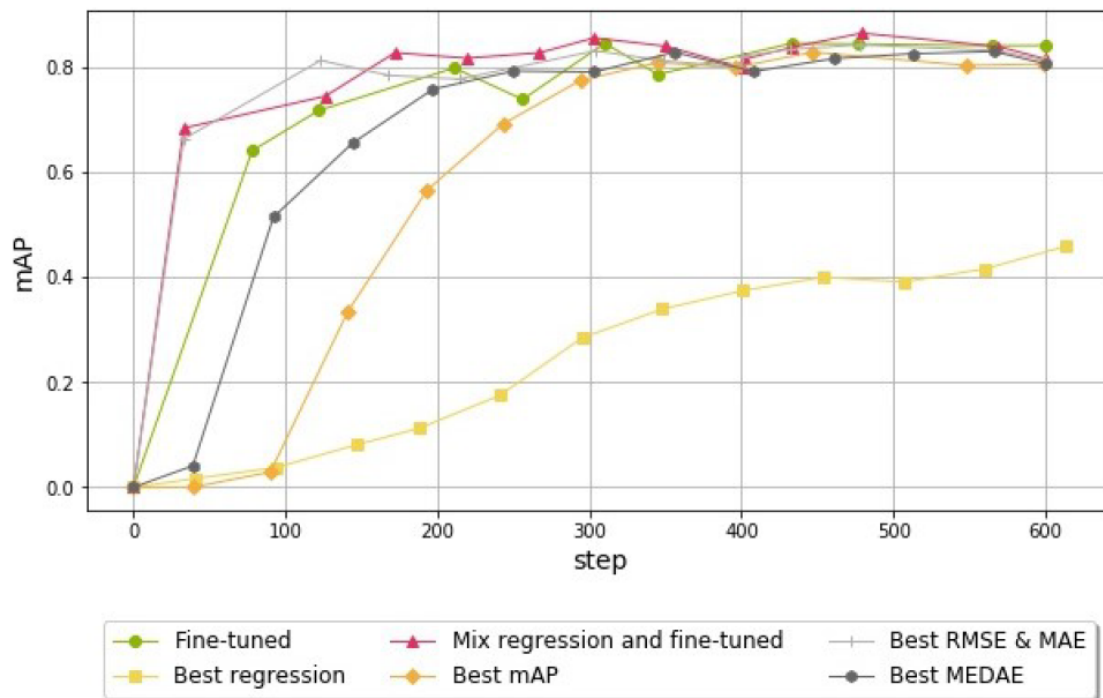


Figure 6.9: mAP of the five final models

6.13 Use the final models in a new dataset partition

Since the results in the section 6.11 gave ambiguous insights for different levels of image complexity, both models were considered to apply on a new data partition, as exposed in table 6.47 and trained again. The purpose was to increase the test dataset in order to have more robust statistics regarding to the results of the model.

Table 6.47: Final data partition

Dataset	#cells	%cells	#images	%images	Avg. #cells per image
Train	9729	56.8	53	55.2	180.2
Validation	2301	13.4	13	13.5	177.0
Test	5098	29.8	30	31.3	169.9
Total	17128	100	96	100	176.6

As expected, the fact of training with a smaller training dataset lead to worst results in terms of mAP, as shown in figure 6.10 for model "Best MEDAE" and in figure 6.11 for model "Best RMSE & MAE".

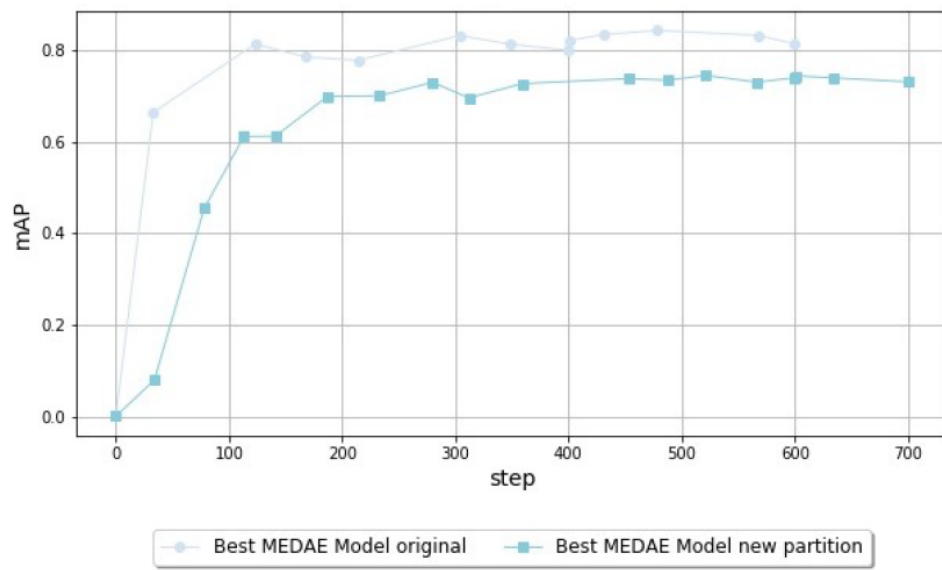


Figure 6.10: mAP of the model "Best MEDAE" using the two partitions

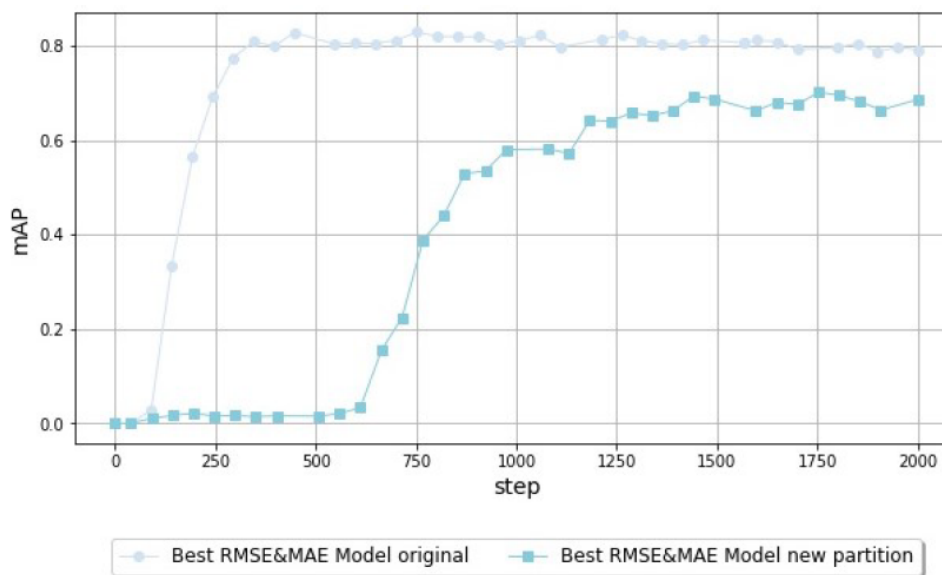


Figure 6.11: mAP of the model "Best RMSE & MAE" using the two partitions

Final model

In this section, it is presented the several losses for the final models, but also the losses for the models with the original partition, as a way to demonstrate that the model could achieve better results and lower losses if more data was available.

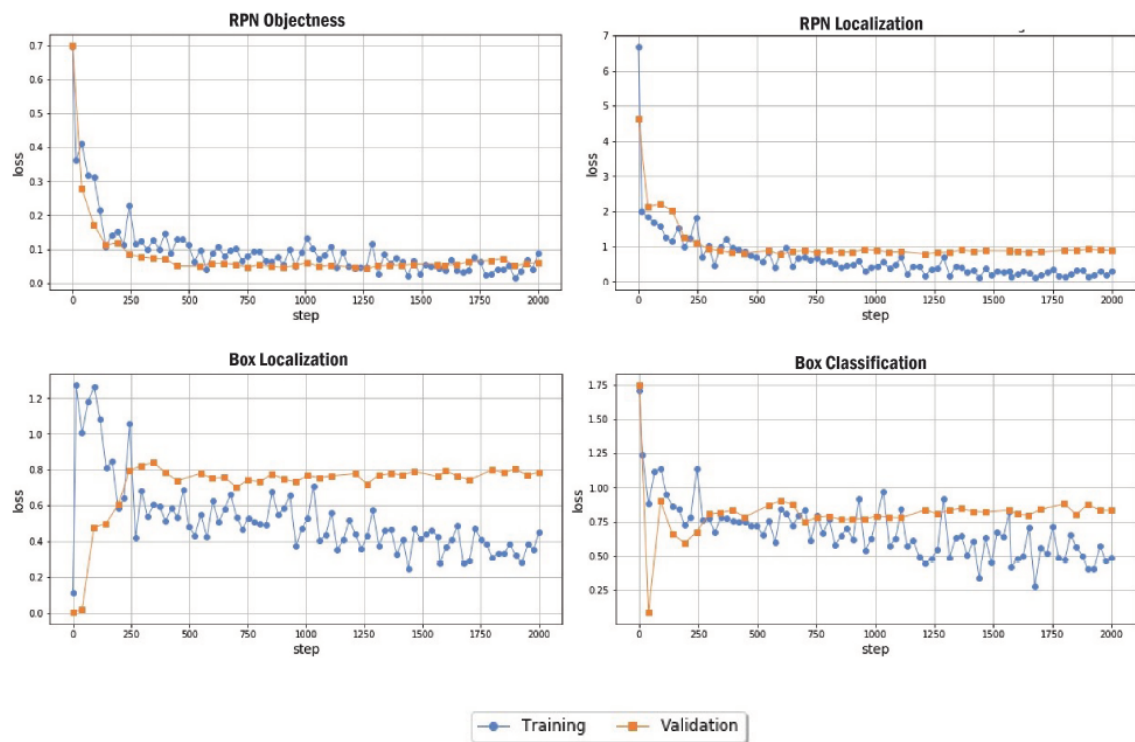


Figure 7.1: Losses for the model "Best RMSE & MAE" with the original partition

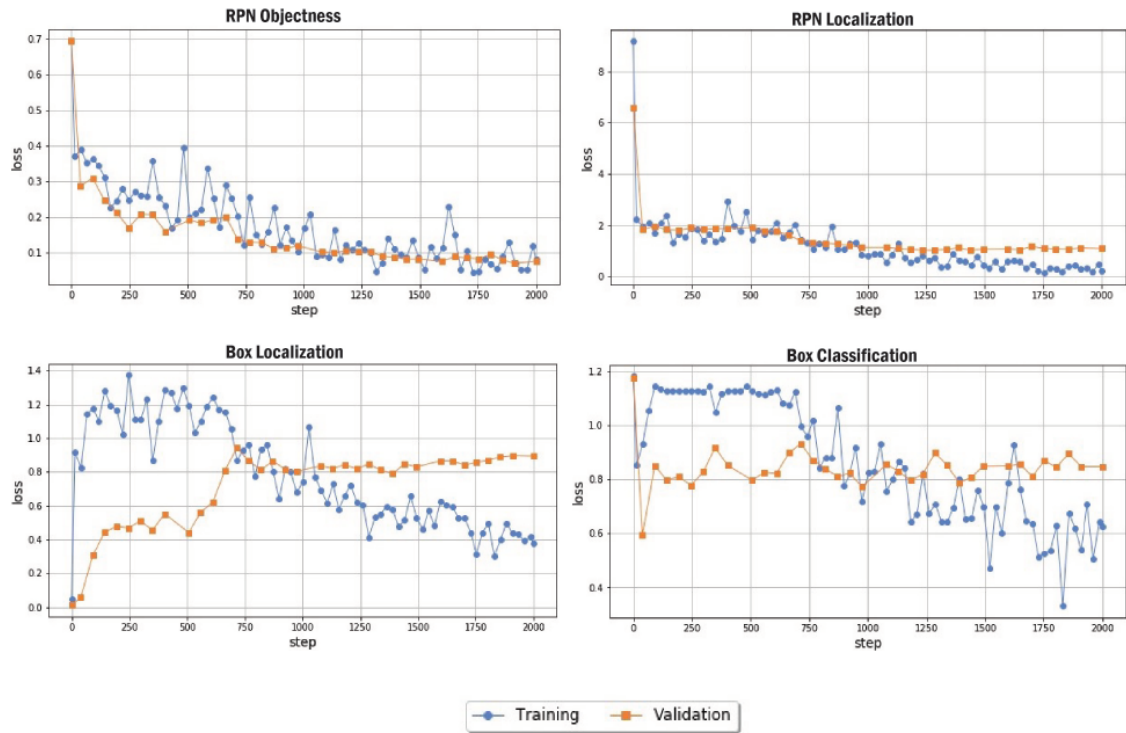


Figure 7.2: Losses for the model "Best RMSE & MAE" with the final partition

By comparing the results in figure 7.1 and in figure 7.2, it is clear that the reduced size of training lead the model to overfit on the second stage of the Faster R-CNN algorithm at around 1100 steps, that corresponds to a mAP near to 0.6, as seen in figure 6.11. Concerning the model "Best MEDAE", the overfitting is not so evident as in the model "Best RMSE & MAE", being most significant in RPN Localization loss as shown in figures 7.3 and 7.4.

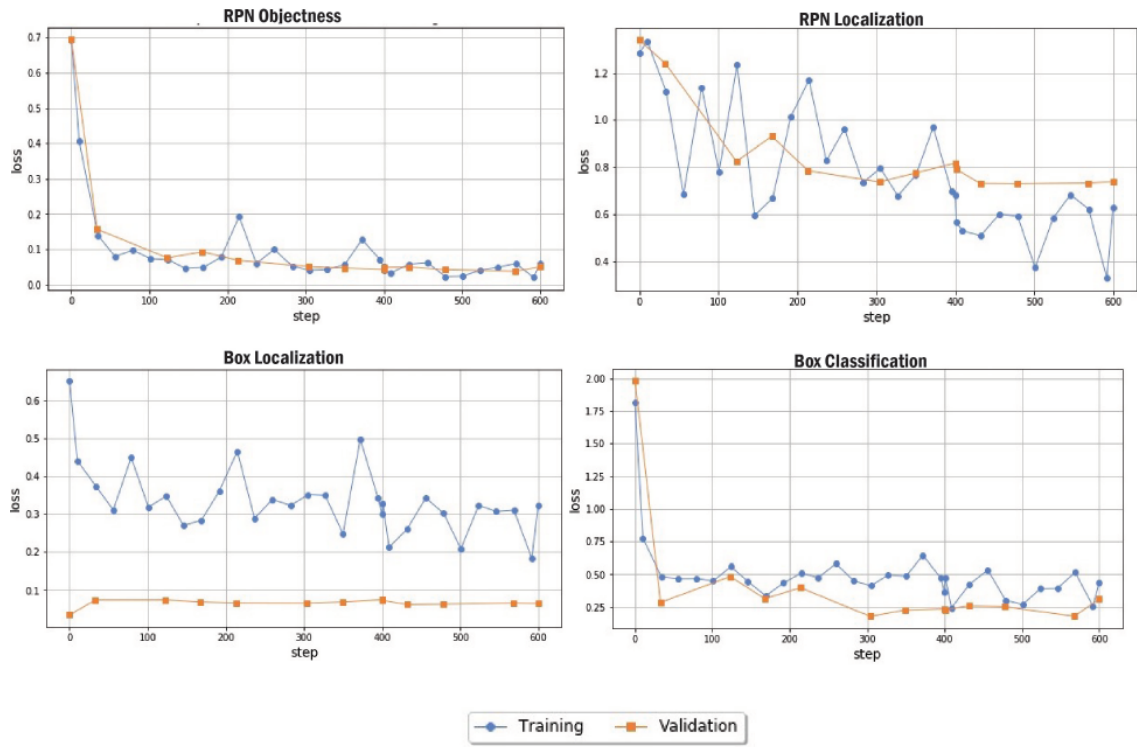


Figure 7.3: Losses for the model "Best MEDAE" with the original partition

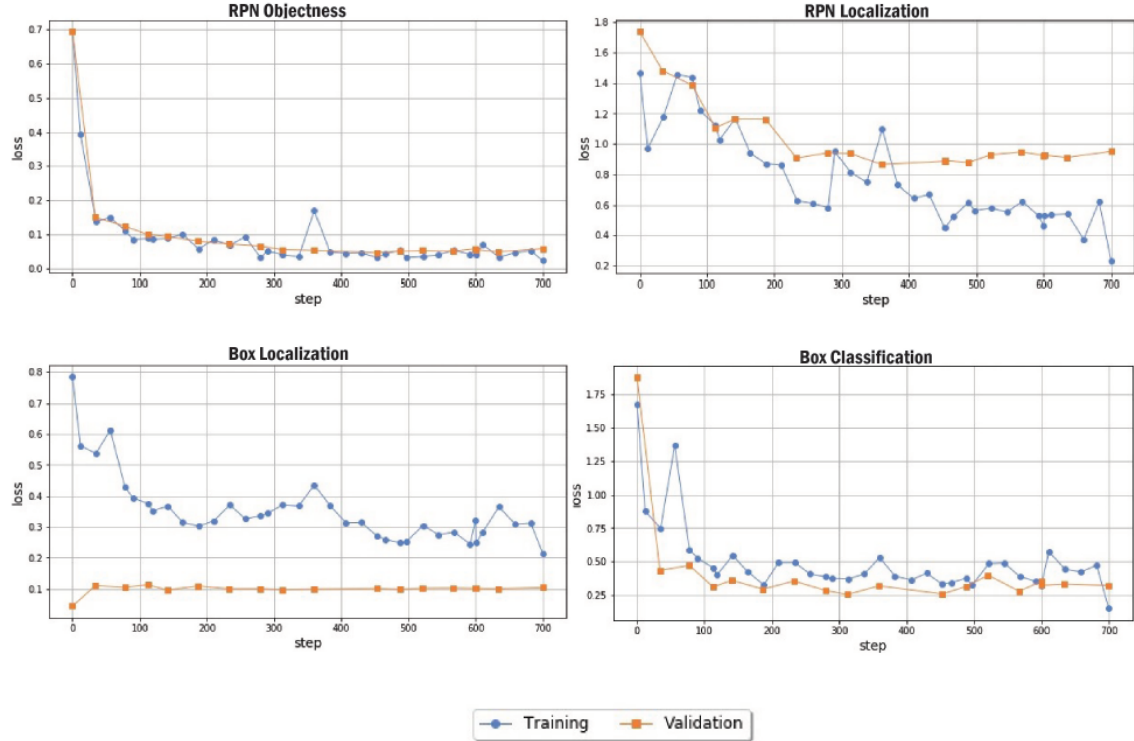


Figure 7.4: Losses for the model "Best MEDAE" with the final partition

In order to identify the best model with the new data partition for each complexity level, each model was frozen at the step where overfitting starts to be evident, and regression metrics were evaluated using validation dataset. Results are summarized in tables 7.1 to 7.6.

An early stopping criterion to avoid overfitting was selected in this phase to freeze the models with the new partition, with the purpose to maintain the characteristics of the original model. However, better results could be achieved if techniques like dropout and regularization were used in this step.

Table 7.1: Inference and Regression metrics for model "Best RMSE & MAE" - Low complexity"

Image	Value predicted per Score threshold									GT
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	
Image4	65	48	42	36	36	35	31	30	28	39
Image2	119	81	66	57	49	44	40	31	24	59
Image5	175	129	110	96	81	75	65	56	32	83
RMSE	65	30	16	8	6	10	16	23	36	-
MAE	59	26	12	6	5	9	15	21	32	-
medae	60	22	7	3	3	8	18	27	35	-

Table 7.2: Inference and Regression metrics for model "MEDAE - Low complexity"

Image	Value predicted per Score threshold									GT
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	
Image4	174	114	92	72	66	56	48	42	37	39
Image2	329	221	167	136	114	98	81	60	44	59
Image5	432	307	246	205	172	156	139	109	77	83
RMSE	266	165	117	85	62	49	35	15	9	-
MAE	251	154	108	77	57	43	29	10	8	-
medae	270	162	108	77	55	39	22	3	6	-

Comparing the results obtained for the Low complexity dataset, in tables 7.1 and 7.2, we can verify that the best results according to validation were achieved by using the model "Best MEDAE", using a score threshold of 0.9.

Table 7.3: Inference and Regression metrics for model "Best RMSE & MAE" - Medium complexity

Image	Value predicted per Score threshold									GT
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	
Image10	230	174	140	124	105	93	74	62	39	105
Image7	193	149	137	125	111	99	84	72	42	121
Image11	271	181	143	118	102	86	69	58	28	124
Image3	252	192	160	134	109	94	80	64	33	155
Image12	313	228	185	167	146	128	116	85	49	168
Image6	281	198	168	152	136	120	101	81	28	201
RMSE	115	48	23	23	35	48	63	80	115	-
MAE	111	42	21	17	28	42	58	75	109	-
MEDAE	111	47	18	13	22	39	54	75	108	-

Table 7.4: Inference and Regression metrics for model "Best MEDAE" - Medium complexity

Image	Value predicted per Score threshold									GT
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	
Image10	674	466	358	307	261	226	190	160	126	105
Image7	480	333	263	212	175	158	139	123	97	121
Image11	700	581	427	324	256	212	180	149	100	124
Image3	700	534	393	295	245	208	164	128	74	155
Image12	700	605	466	375	308	254	218	180	130	168
Image6	585	406	330	283	247	215	189	168	135	201
RMSE	502	356	237	162	111	75	47	31	48	-
MAE	494	342	227	154	103	67	38	26	42	-
MEDAE	539	370	246	170	111	70	34	26	31	-

As presented in tables 7.3 and 7.4, the results obtained for the medium complexity dataset show that, according to the validation dataset, it should be chosen the model "Best RMSE & MAE", using a score threshold of 0.4.

Table 7.5: Inference and Regression metrics for model "Best RMSE & MAE" - High complexity

Image	Value predicted per Score threshold									GT
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	
Image1	372	297	271	253	232	207	178	142	87	278
Image8	366	296	272	250	214	194	169	118	68	349
Image9	539	454	416	379	330	291	244	190	95	506
RMSE	58	44	69	94	131	158	192	239	308	-
MAE	48	41	58	84	119	147	181	228	294	-
MEDAE	33	52	77	99	135	155	180	231	281	-

Table 7.6: Inference and Regression metrics for model "MEDAE" - High complexity

Image	Value predicted per Score threshold									GT
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	
Image1	700	700	573	486	426	379	339	304	237	278
Image8	700	587	484	421	378	350	314	279	223	349
Image9	700	700	700	627	565	520	475	417	330	506
RMSE	336	301	218	145	93	59	44	67	127	-
MAE	322	285	208	134	79	39	42	62	114	-
MEDAE	351	238	194	121	59	14	35	70	126	-

Comparing the results obtained and presented in 7.5 and 7.6 for the high complexity dataset, we can verify that the best results according to the validation dataset were obtained by using the model "Best RMSE & MAE", with a score threshold of 0.2.

Test evaluation

In order to evaluate the performance of the final model, a set of 30 images were defined as test during the dataset partition, as defined in table 8.1.

Table 8.1: Test Dataset for levels "Low", "Medium" and "High"

Dataset	#cells	%cells	#images	%images	Avg. #cells per image
Low	399	7.83	7	23.33	57.00
Medium	2315	45.41	16	53.33	144.69
High	2384	46.76	7	23.33	340.57
Total	5098	100	30	100	169.93

From those 30 images, and after freezing the model, we can infer the number of cells present in each image. The results for each level are presented on tables 8.2, 8.3 and 8.4, where y stands for the ground truth value and $\hat{y}$ for the predicted value:

Table 8.2: Results on test dataset - predicted vs. ground truth for low level using model "Best MEDAE" for a threshold of 0.9

Level	Image	y	$\hat{y}$	$ \hat{y} - y $
Low	Image_037	30	24	6
	Image_169	35	45	10
	Image_091	44	44	0
	Image_085	57	51	6
	Image_151	65	55	10
	Image_193	77	80	3
	Image_094	91	97	6

Table 8.3: Results on test dataset - predicted vs. ground truth for medium level using model "Best RMSE & MAE for a threshold of 0.4

Level	Image	y	$\hat{y}$	$ \hat{y} - y $
Medium	Image_289	95	101	6
	Image_097	103	109	6
	Image_124	110	89	21
	Image_019	115	111	4
	Image_196	119	111	8
	Image_010	124	127	3
	Image_181	129	118	11
	Image_286	134	138	4
	Image_154	137	114	23
	Image_076	154	155	1
	Image_205	157	133	24
	Image_100	159	153	6
	Image_103	177	174	3
	Image_064	185	186	1
	Image_007	198	153	45
	Image_241	219	175	44

Table 8.4: Results on test dataset - predicted vs. ground truth for high level using model "Best RMSE & MAE for a threshold of 0.2

Level	Image	y	$\hat{y}$	$ \hat{y} - y $
High	Image_004	236	235	1
	Image_238	245	256	11
	Image_250	284	227	57
	Image_226	298	320	22
	Image_268	343	350	7
	Image_223	407	328	79
	Image_115	571	503	68

Table 8.5 presents the results regarding regression metrics for the test evaluation. As can be seen, and comparing as an example the results obtained for the medium level and presented on 6.46, the new data partition had led the performance of the model to significantly worse behavior, where the error measured by the RMSE, MAE and MEDAE went from 8.8, 6.4 and 3.5 to 22.04, 18.06 and 16.50, respectively.

Table 8.5: Regression metrics results on test evaluation

Level	RMSE	MAE	MedAE
Low	9.25	7.71	6.00
Medium	22.04	18.06	16.50
High	53.29	43.71	37

Finally, descriptive statistics were inferred from the final results, as shown in table 8.6, and conclusions can be made.

Table 8.6: Descriptive statistics associated to the error encountered on the different levels of complexity

Level	Min	Max	Mean	STD	1st Qrt	2nd Qrt	3rd Qrt
Low	0	10	5.86	3.58	3	6	10
Medium	1	45	14.37	13.12	3.25	6	22.5
High	1	79	35	32.13	7	22	68
All	0	79	16.53	20.96	3.75	6.5	22.25

According to the results exposed in table 8.6:

1. For the level low, the error found in samples was in the range of [0,10] units, with a standard deviation of 3.58. Half of the samples had less than 6 units or error.
2. For the medium level, the error range was between [0,45], with a standard deviation of 13.12. Half of the samples had less than 6 units of error and 75% less than 22.5 units of error.
3. For the high level, the error ranged between [1,79], with a standard deviation of 32.13 and half of the samples had less than 22 units of error.
4. Overall, the samples error range was between [0,79], with a standard deviation of 20.96, and half of the predicted values had an absolute difference to the real values lower than 6.5 units.

In figure 8.1 is presented the inference to images of the three levels as an example, side by side to the original labeled ones.

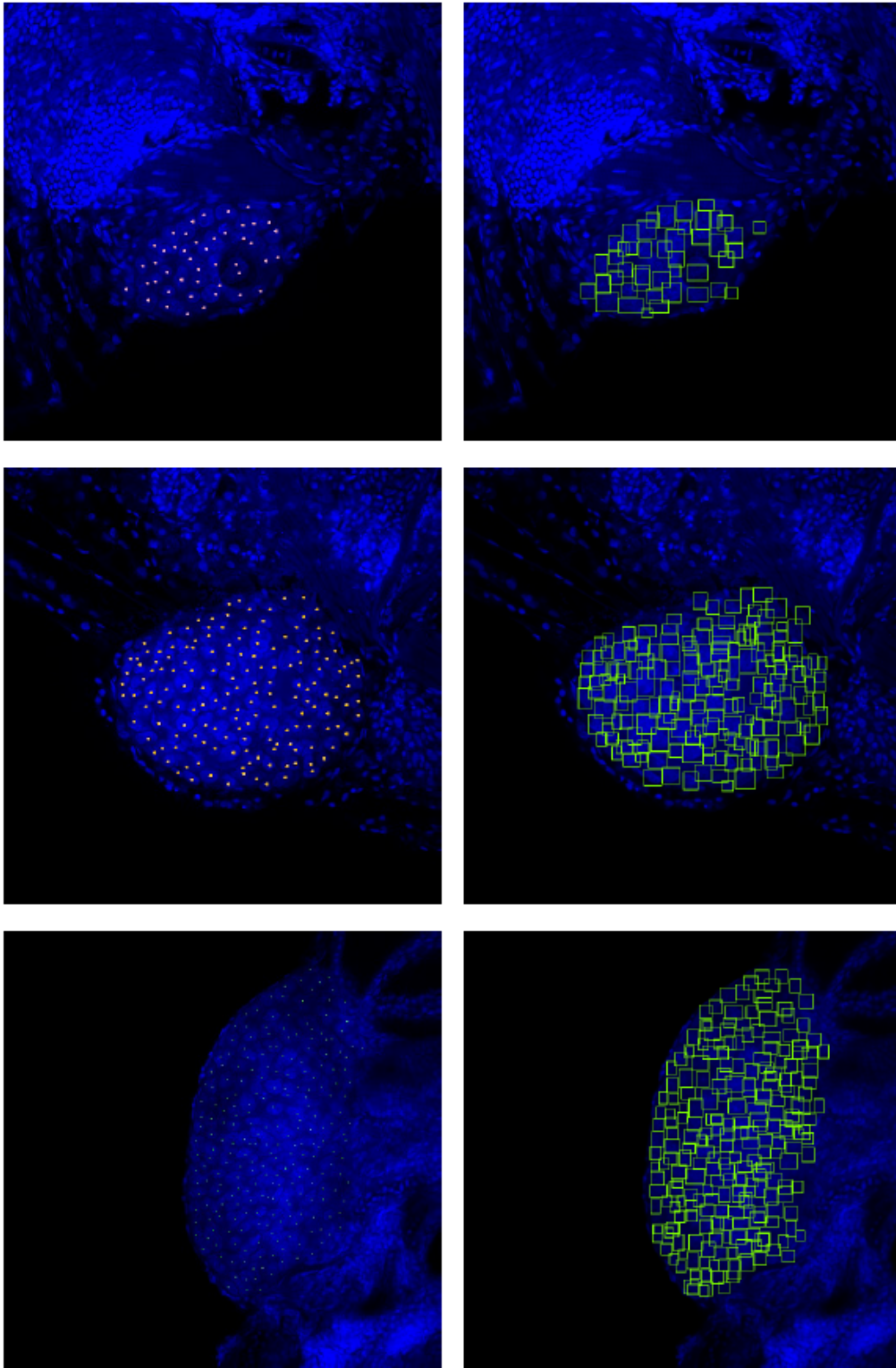


Figure 8.1: Ground truth labeling and inferred detections for images with Low, Medium and High complexity

In the first row, we have a test example with 44 cells. The model proposed predicts 44 cells in the image. In the second row, a test example is presented with 159 cells. The model predicts that 153 cells are present in the image. In the lower row, a test sample with 236 cells is shown. The model infers the cell counting as 235 cells.

Final application

The final application aims to deliver an executable easy-friendly to work on in order to make inferences on images similar to the ones used to build the final model.

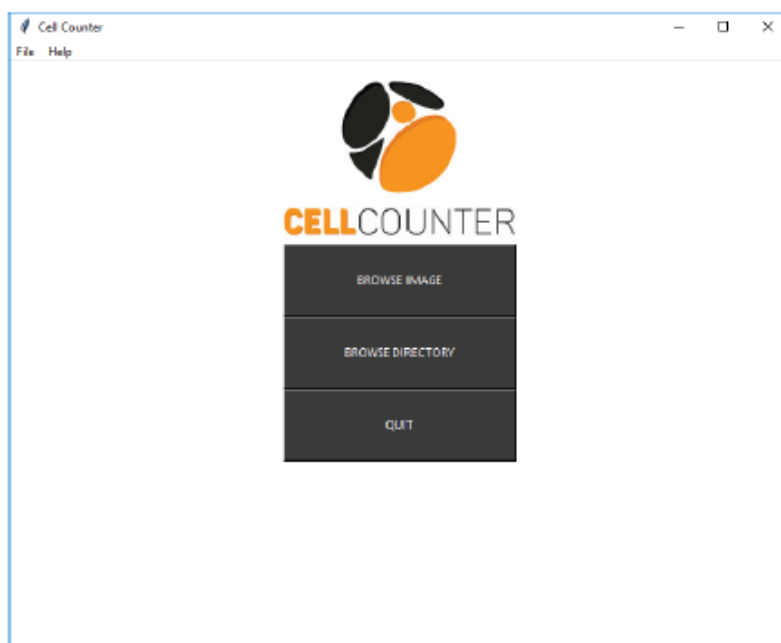


Figure 9.1: Initial screen of the application "CellCounter"

As a simple and concise tool, users can select two main options:

1. Browse Image - The user can infer the detection in an isolated image, and compare both raw and image with detections side by side, as also the number of detections predicted.
2. Browse directory - The user can select a directory and make the inference to all the

images included in the folder.

In both situations, and since two different models compose the final solution according to the complexity of the images, the user should define at first the level of complexity of the images that are going to be analyzed so the right model can be uploaded, as shown in figure 9.2.

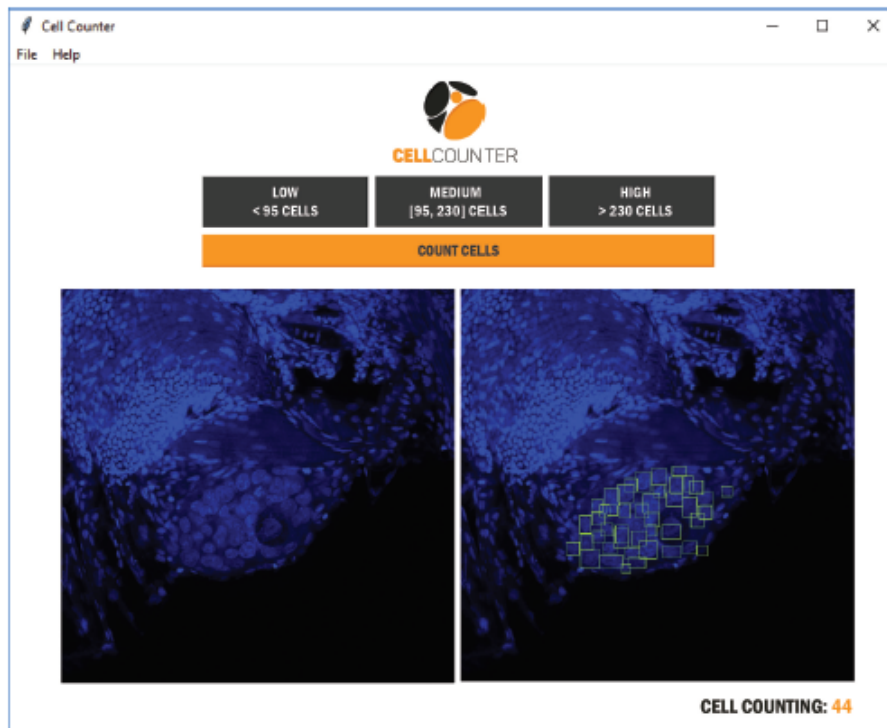


Figure 9.2: Making inferences in the application "CellCounter"

10.1 Conclusions

Two different models using object detection algorithm Faster R-CNN with Inception Resnet V2 were proposed in this report as a possible approach to cell quantification in zebrafish xenografts. Based on the insights achieved on the exposed experiments, several conclusions can be made during this project, namely:

1. From the meta-architectures analyzed, Faster R-CNN is the one that allows obtaining better results concerning performance, but it also leads to a slower training process;
2. The feature extractor should be chosen according to the complexity of the problem that is being dealt. In the present project, the Inception Resnet V2 gives the best results regarding mAP;
3. Using transfer learning can be an essential step in order to achieve viable results in the short term. Since the problem being addressed uses input images with no similarity to the existent datasets, gathering only the weights of the pre-trained model lead us to better results than freezing some of the layers, with the repercussion of a slower training process;
4. As a result of dealing with images where the objects are significantly small, the resolution of the input images can have a huge impact on the models' performance. The resize method should also be taken into account since different approaches can affect the final result of the model. In this way, resizing the images to a fi-

nal dimension of 1100x1100 pixels using a bilinear technique lead us to the best option;

5. Adaptive learning optimizers, like Adam, allows the faster convergence of a model, and the estimation of an optimal learning rate is one of the most important hyperparameters to tune;
6. Data Augmentation can be an essential technique to turn models more robust to variances and when in the presence of small datasets, to boost the size of it artificially.
7. Dealing with overlapped objects implies a careful parameterization on Non-Max suppression method used during object detection since this will establish when overlapping bounding boxes should define one or more objects. A careful threshold should be considered taking into account the nature of the images trained;
8. Defining the scales and aspect ratios of the proposals can affect the performance of the model significantly. Previously to training, the dimension of the objects in the images should be carefully analyzed as a way to define the best values for those parameters.

Furthermore, and from an overall perspective, we can assume that:

1. CNNs, and in particular, object detection algorithms, appear as a good approach to perform medical imaging recognition and perform cell quantification;
2. Regarding clumping and overlapping cells, the fine-tuning of the architecture and some pre-processing steps can have a high impact to deal with those constraints;
3. Changing parameters and hyperparameters can affect the performance of an object detection model significantly and should be evaluated and adjusted to the problem in hand;
4. Even in situations where the number of objects to detect is significantly high, an object detection technique can lead to good results if careful adjustments are made during the implementation of the model training.

Concluding, and regarding precision, the results were promising. It was shown that object detection technique could be a valid and competitive strategy concerning cell counting, even on the presence of overlapped objects, and more research on applying object detectors in cell quantification should be encouraged.

10.2 Limitations

While the results seem to suggest the proposed solution as a good approach to deal with the problem at hand, there are some limitations in this work that should be taken into consideration before making strong assumptions.

Due to the several parameters tested and the demanding computational and time resources needed, cross-validation, an essential technique for assessing the performance of a model and the generalization capability of the same to an independent dataset, was not applied during the experiments. This could lead to biased results, and to overestimate the behavior of the model.

Moreover, the dataset size created several constraints during training. Training with a bigger dataset leads us to poor statistical conclusions while evaluating the performance by training with a smaller number of samples implied a significant decrease in the performance of the model trained.

10.3 Future work

Future work is highly associated with the limitations exposed in section 10.2.

The models with better mAP results should be re-evaluated using cross-validation in order to assess the predictive performance of the models to new samples, and to minimize the likelihood of unbiased results.

Applying the insights obtained during this project to a new larger dataset will allow, eventually, to obtain more accurate models. Moreover, at least three different labeled datasets with a significant amount of images should be defined taking into account the complexity of the images. As perceived during the experiments, the performance of the model to infer a counting is highly associated to the clump of cells present on the images, and this fact should be taken into consideration in future works.

Thirdly, the definition of the ground truth bounding boxes should be applied per an expert on the analysis of images of this nature. While the dot annotation was made on this

way, the ground truth bounding boxes were defined during this project by a non-expert, based on the original labeling, and this can lead to misleading labels concerning the precision of the limits of a cell. Furthermore, the cells on each image should be quantified by more than one expert, in order to avoid inter-observer variability.

Moreover, a random search as a hyperparameter optimization method should be applied, where the main conclusions obtained on this work could be used to define thresholds. This technique will eventually lead to a model better adjusted, and could be a significant step to achieve superior performance.

Finally, other techniques besides object detection should be applied and tested, as a way to obtain benchmarking information and evaluate the potential of the approach developed.

An interesting topic for further research would be trying to combine the strengths of genetic algorithms(GA) with the presented approach, by implementing GA on the parameters and hyperparameters definition.

Bibliography

- [1] Nikhil Buduma and Nicholas Locascio. *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms*. " O'Reilly Media, Inc.", 2017.
- [2] Michael George Bulmer. *Principles of statistics*. Courier Corporation, 1979.
- [3] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.
- [4] Sebastian De Boodt, Ahmad Poursaberi, Jan Schrooten, Daniel Berckmans, and Jean-Marie Aerts. A semiautomatic cell counting tool for quantitative imaging of tissue engineering scaffolds. *Tissue Engineering Part C: Methods*, 19(9):697–707, 2013.
- [5] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.
- [6] Rita Fior, Vanda Póvoa, Raquel V Mendes, Tânia Carvalho, António Gomes, Nuno Figueiredo, and Miguel Godinho Ferreira. Single-cell functional and chemosensitive profiling of combinatorial colorectal therapy in zebrafish xenografts. *Proceedings of the National Academy of Sciences*, 114(39):E8234–E8243, 2017.
- [7] Kent Gauen, Ryan Dailey, John Laiman, Yuxiang Zi, Nirmal Asokan, Yung-Hsiang Lu, George K Thiruvathukal, Mei-Ling Shyu, and Shu-Ching Chen. Comparison of visual datasets for machine learning. 2017.

- [8] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.
- [9] Pascal Getreuer. Linear methods for image interpolation. *Image Processing On Line*, 1:238–259, 2011.
- [10] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [12] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.
- [13] Rafael C. Gonzales and Paul Wintz. *Digital Image Processing (2Nd Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.
- [14] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [15] Tongfan Guan and Hao Zhu. Atrous faster r-cnn for small scale object detection. In *2017 2nd International Conference on Multimedia and Image Processing (ICMIP)*, pages 16–21. IEEE, 2017.
- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [18] Carlos X Hernández, Mohammad M Sultan, and Vijay S Pande. Using deep learning for segmentation and counting within microscopy data. *arXiv preprint arXiv:1802.10548*, 2018.

- [19] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
- [20] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, volume 4, 2017.
- [21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [22] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.
- [23] Alboukadel Kassambara. *Machine Learning Essentials: Practical Guide in R*. STHDA, 2018.
- [24] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.
- [25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [26] Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7), 2010.
- [27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

- [29] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*, 2018.
- [30] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [31] Erik Meijering. Cell segmentation: 50 years down the road [life sciences]. *IEEE Signal Processing Magazine*, 29(5):140–145, 2012.
- [32] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.
- [33] João C Neves, Helena Castro, Ana Tomás, Miguel Coimbra, and Hugo Proença. Detection and separation of overlapping cells based on contour concavity for leishmania images. *Cytometry Part A*, 85(6):491–500, 2014.
- [34] Andrew Ng. Why does batch norm work? <https://pt.coursera.org/lecture/deep-neural-network/why-does-batch-norm-work-81oTm>. Online; accessed 11-October-2018.
- [35] Daniel Onoro-Rubio and Roberto J López-Sastre. Towards perspective-free object counting with deep learning. In *European Conference on Computer Vision*, pages 615–629. Springer, 2016.
- [36] Randall C O’Reilly, Y Munakata, MJ Frank, TE Hazy, et al. *Computational cognitive neuroscience*. PediaPress, 2012.
- [37] DPVV Pankaj S Parsania and PV Virparia. A comparative analysis of image interpolation algorithms. *Advanced Research in Computer and Communication Engineering*, 5, 2016.
- [38] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.
- [39] Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner’s Approach*. "O’Reilly Media, Inc.", 2017.

- [40] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.
- [41] Maryam Rahnemoonfar and Clay Sheppard. Deep count: fruit counting based on deep simulated learning. *Sensors*, 17(4):905, 2017.
- [42] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [43] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.
- [44] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [45] Mikel Rodriguez, Ivan Laptev, Josef Sivic, and Jean-Yves Audibert. Density-aware person detection and tracking in crowds. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2423–2430. IEEE, 2011.
- [46] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [47] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.
- [48] Mohit Sewak, Md Rezaul Karim, and Pradeep Pujari. *Practical Convolutional Neural Networks: Implement advanced deep learning models using Python*. Packt Publishing Ltd, 2018.
- [49] Rajalingappaa Shanmugamani. *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using Tensorflow and Keras*. Packt Publishing Ltd, 2018.
- [50] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. Don’t decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

- [51] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [52] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.
- [53] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [54] Weidi Xie, J Alison Noble, and Andrew Zisserman. Microscopy cell counting and detection with fully convolutional regression networks. *Computer methods in biomechanics and biomedical engineering: Imaging & Visualization*, 6(3):283–292, 2018.
- [55] Mingliang Xu, Zhaoyang Ge, Xiaoheng Jiang, Gaoge Cui, Pei Lv, and Bing Zhou. Depth information guided crowd counting for complex crowd scenes. *arXiv preprint arXiv:1803.02256*, 2018.
- [56] Yao Xue and Nilanjan Ray. Cell detection in microscopy images with deep convolutional neural network and compressed sensing. 2018.
- [57] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 833–841, 2015.
- [58] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017.

2018

Title: Convolutional Neural Networks for Cell Detection and Counting

Subtitle: A case study of Human cell Quantification in Zebrafish Xenografts using DL Object Detection Techniques

Student full name: Carina Isabel Andrade
Albuquerque

MAA

