Automatic Assessment of Honey Bee Cells Using Deep Learning

Thiago da Silva Alves - 38684

Thesis presented to the School of Technology and Management
to obtain the Master's Degree in Information Systems

Work supervised by:

Pedro João Rodrigues

Arnaldo Candido Junior

Maria Alice Pinto

Pedro Luiz de Paula Filho

Bragança

11/2018

Automatic Assessment of Honey Bee Cells Using Deep Learning

Masters in Information Systems

School of Technology and Management

Thiago da Silva Alves - 38684

11/2018

The School of Technology and Management is not responsible for the opinions expressed in this document.

Acknowledgements

Thank God for helping me get here. To my parents Paulo Henrique Alves and Margarida da Silva Alves for having supported me at all times until today. To my siblings, Robson and Raquel who have always been available to help me.

Special thanks to the professors Pedro João Rodrigues, Alice Pinto, Arnaldo Candido Junior and Pedro Luiz de Paula Filho, who guided me throughout the thesis and served as inspiration in various aspects of my life. I am also grateful for the two institutions that provided the opportunity to study my bachelor's and master's degree, Federal University of Technology - Paraná and the Polytechnic Institute of Bragança.

I would also like to thank all those who were close to me during the development of the thesis and gave me support so that I could do my best, Daiane, Leonan, Daniela and Carlos.

I am also grateful to all those who helped in the images gathering and those who tested the software in its test phase and gave us feedback, Paulo Ventura, Cátia Neves, Ana Rita Lopes, Nuno Capela, Artur Sarmento, Yoko Luise Dupont, Per Kryger, Cecilia Costa and Marco Lodesani.

This research was conducted in the framework of the project BEEHOPE, funded through the 2013-2014 BiodivERsA/FACCE-JPI Joint call for research proposals, with the national founders FCT(Portugal), CNRS(France), and MEC(Spain).

Finally, my sincere gratitude to all the people who somehow helped me during my academic period.

Abstract

Temporal assessment of honey bee colony strength is required for different applications in many research projects, which often involves counting the number of comb cells with brood and food reserves multiple times a year. There are thousands of cells in each comb, which makes manual counting a time-consuming, tedious and thereby an error-prone task. Therefore, the automation of this task using modern imaging processing techniques represents a major advance. Herein, we developed a software capable of (i) detecting each cell from comb images, (ii) classifying its content and (iii) display the results to the researcher in a simple way. The cells' contents typically display a high variation of patterns which make their classification by software a challenging endeavour. To address this challenge, we used Deep Neural Networks (DNNs). DNNs are known for achieving the state of art in many fields of study including image classification, because they can learn features that best describe the content being classified by themselves. Our DNN model was trained with over 70,000 manually labelled cell images whose cells were separated into seven classes. Our contribution is an end-to-end software capable of doing automatic background removal, cell detection, and classification of cell content based on an input comb image. With this software, colony assessment achieves an average accuracy of 94% across the seven classes in our dataset, representing a substantial progress regarding the approximation methods (e.g. Lieberfeld) currently used by honey bee researchers and previous techniques based on machine learning that used handmade features like colour and texture.

Keywords: Honeybee Conservation, Image Classification, Comb Segmentation, Convolutional Neural Networks, Cell Classification.

Resumo

A análise temporal sobre a qualidade e força de colônias de abelha melífera (*Apis mellifera L.*) é necessária em muitos projetos de pesquisa. Ela pode ser realizada contando alvéolos com alimento (pólen e néctar) e criação. É comum que ela seja feita diversas vezes ao ano. A grande quantidade de alvéolos em cada favo torna a tarefa demorada e tediosa ao pesquisador. Assim, frequentemente essa contagem é feita forma aproximada usando métodos como o de Lieberfeld. Automatizar este processo usando técnicas modernas de processamento de imagem representa um grande avanço, pois resultados mais precisos e padronizados poderão ser obtidos em menos tempo. O objetivo deste trabalho é construir de um software capaz de detectar, classificar e contar alvéolos a partir de uma imagem. Após, ele deverá apresentar os dados de forma simplificada ao usuário. Para tratar da alta variação de padrões como textura, cor e iluminação presente nas alvéolos, usaremos Deep Neural Network (DNN), que são modelos computacionais conhecidos por terem alcançado o estado da arte em várias tarefas relacionadas a processamento de sinais e imagens. Para o treinamento desses modelos utilizamos mais de 70.000 alvéolos anotadas por um apicultor experiente, separadas em sete classes. Entre nossas contribuições estão métodos de pré-processamento que garantem uma alta taxa de detecção de alvéolos, aliados a modelos de segmentação baseados em DNNs que asseguram uma baixa taxa de falsos positivos. Com nossos classificadores conseguimos uma acurácia média de 94% em nosso dataset e obtivemos resultados superiores a outros métodos baseados em contagens aproximadas e técnicas de análise por imagem que não utilizam DNNs.

Contents

List of Tables

List of Figures

Acronyms

AHE Adaptive Histogram Equalization. 15

AI Artificial Intelligence. 20

API Application Programming Interface. 138

BN Batch Normalization. 35

CCE Categorical Cross-Entropy. 46

CeDRI Research Centre in Digitalization and Intelligent Robotics. 5, 138

cHT Circle Hough Transform. 11

CIMO Mountain Research Center. 5

CLAHE Contrast-Limited Adaptive Histogram Equalization. 16

COCO Common Objects in Context. 37

CPU Central Processing Unit. 43

CREA Consiglio per la ricerca in agricoltura e lánalisi delléconomia agraria. 5, 58

DA Data Augmentation. 43

DL Deep Learning. 28

DNN Deep Neural Network. viii

EFSA European Food Safety Authority. 2

EU European Union. 1

FN False Negative. 45

FP False Positive. 44

GPU Graphical Processing Unit. 43

HE Histogram Equalization. 15

HLT Hough Line Transform. 18

HT Hough Transform. 17

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 28

IoU Intersection over Union. 47

IP Image Processing. 12

IPB Instituto Politécnico de Bragança. 6, 138

JSON JavaScript Object Notation. 138

LED Light-emitting diode. 49

ML Machine Learning. 20

MSE Mean squared error. 46

NN Neural Networks. 22

RANSAC Random sample consensus. 140

ROI Region of Interest. 88

TN True Negative. 45

TP True Positive. 44

UTFPR Federal University of Technology - Paraná. 6

VGG Visual Geometry Group. 32

Chapter 1

Introduction

In this chapter, we will show some basis for this work. We will first explore the background and the reasons that make this work relevant to researchers in apidology (Section 1.1). Next, we will present the objectives sought with this proposal (Section 1.2). Lastly, we will mention the contributions produced by the development of this work (Section 1.3) and present the structure of the document (Section 1.4).

1.1 Background

With an average production of 250,000 tons of honey per year, the European Union (EU) is the second largest producer in the world, only behind China [1]. To achieve this production, the EU has more than 600,000 beekeepers, who together own more than 16 million hives [1]. In addition to producing honey, propolis, royal jelly, pollen and wax as final products for human use, the honey bee (*Apis mellifera L.*) also provides pollination service which is fundamental for natural and agricultural ecosystem functioning. Even the estimated honey production value of nearly 320 million dollars in 2017 [2] is little when compared to the 11-15 billion dollars estimated annual pollination services in the United States [3], [4]. The plants that benefit from the bee pollination are mainly Angiosperms, which play a critical role in providing food and shelter to wildlife and humans. In addition to the benefits brought to humans from the consumption of pollinated fruits and

vegetables, these plants are increasingly used for the production of fuels, which makes society in many ways dependent on the services provided by the bees [3].

Recent events are negatively affecting the health of bees and other insect pollinators. The mortality of honey bee colonies across the planet is occurring at an alarming rate. Although an annual mortality of 10-15% per colony is accepted, Europe accounts for losses of more than 30% in countries such as Belgium and the United Kingdom [5], being an important cause of this increase is a worldwide phenomenon known as Colony Collapse Disorder. Despite the many studies that have been conducted on Colony Collapse Disorder, a consensus among researchers on its underlying causes has not been achieved. The main suspects are the increased use of pesticides, parasite proliferation, particularly the mite *Varroa Destructor* and climate change [4], [6].

In order to protect and develop the beekeeping sector, initiatives are being implemented. Among the EU's efforts are the establishment of national beekeeping programs [7], development of methods for monitoring bee mortality [8], supporting research projects such as EurBeSt [9], SmartBees [10], Swarmonitor [11] and BEEHOPE [12] which this thesis is part of, and also the regulation of the use of pesticides.

The *Regulation (EC) No 1107/2009* [13] establishes clear criteria related to the approval of the use of substances for plants protection, such as pesticides. According to this regulation, approval of pesticides will only be done if analyses are carried out and if it is proven that their use will affect an insignificant number of bees or that there are no negative effects related to the survival and development of colonies.

The European Food Safety Authority (EFSA) developed a guide on how to carry out the risk assessment of pesticides in colonies [14]. This guide compiles the methods to be used in order to prove that new plant protection products will not cause bee mortality and thus can be released for sale and use. Among the described methods, are estimation of colony strength. These methods includes colony weighing proposed by Costa et al. [15], which allows estimation of colony size, and the Lieberfeld method [16] which assists in estimating colony size by visually measuring the amount of brood and food reserves in the combs.

Using the Lieberfeld method it is possible to approximately calculate the number of cells of a given class in a frame. In this approach, the frame is divided into eight squares and then the number of cells occupied by brood, food reserves, or bees is estimated [14], [17]. An example of the application of the method is shown in figure 1.1.

Figure 1.1: Beekeeper using the Lieberfeld method to assess the comb.

While the Lieberfeld method is standard for assessing colony strength and it is easier to perform compared to exhaustive cell counts, it has drawbacks. In the guide developed by EFSA it is considered as less accurate than the hive weighing [14] due to being a method that involves subjectivity and is highly dependent on the experience of the observer. Because it is performed visually, its results may diverge between observers; thus, the same result may not always be reproduced. In Delaplane et al. [18] it is recommended to have at least two observers to assess the combs. The lack of reproducibility is also discussed in Lee et al. [19]. Other points that the author addresses are the need to have people trained to perform the count and the fact that this task is slow and tedious.

In order to minimise the difficulties mentioned above, a software to automatically perform the counting and classification of the content of cells in comb images was developed herein. This tool can mean a breakthrough for bee research due to reduction of human labour during the data gathering process. Acquisition using software like this does not require specialised observers, it will be less time-consuming, it will produce more accurate results and the photographs will serve as raw material for the reproducibility of the experiments.

The nature of the classification problem is complex due to the high variability of

shapes, colour and illumination that can occur on the images. Another factor that impacts the algorithm's decision about what is inside a cell is the number of classes being considered. In the present study, seven classes were considered: capped brood, larvae, eggs, pollen, nectar, honey and others. The large variety of patterns makes it difficult to manually craft features that best separate one class from another so that a classifier can be made. Therefore, we will use DNNs to build the classifiers.

DNNs are computational models that have become popular in recent years because they have the ability during their training to learn which patterns of their input data most impact on the correct prediction of their outputs. This capability gave these models great results such as overcoming humans in the task of classifying specific images [20], approaching the human level in identifying people in images [21], winning the world champion in the Go board game, [22] and achieving the state of the art in various tasks [23]–[26].

In addition to the creation of DNN models for cell classification, in this work we also developed new preprocessing approaches for cell detection, honeycomb segmentation methods to prevent false cell detections, and a software capable of showing results to users in a simple way, also allowing them to correct predictions according to their needs.

More information about the proposal of this work are in the appendix A.

1.2 Objectives

The main objectives of the study are:

1. To develop image processing techniques to detect cells in comb images;

2. To develop computational models capable of reliably classifying the contents of comb cells from images;

3. To develop software that retrieves in a simple and visual way the number of cells occupied by brood and food reserves present in comb images.

1.3 Contributions

With the development of this work, we have been able to produce several contributions to different communities such as researchers in machine learning, image processing, apidology and beekeepers.

For the researchers in machine learning, image processing and other software developers we contributed with (i) a method in the segmentation process to automatically readjust the scale of the images driven by the size of the cells, which as far as we can know is an original solution, (ii) comparisons among different neural network architectures related to performance and quality of results, (iii) datasets so they can test new methods, and (iv) making available our source codes so they can reproduce our results. We communicated our project to these communities during the PyConCZ 2018 which took place in Prague, Czech Republic, at this conference we presented the talk *HoneyBee Conservation with Python*.

For apidology researchers and beekeepers, we have contributed with a new methodology to assess combs automatically by images. With the software we have named DeepBee, they can automatically evaluate a set of comb images, edit the predictions made by the software if necessary, and produce an Excel file for further analysis. We communicated our project with these communities at the EurBee8 event that took place in Ghent, Belgium. In addition to our work *Assessment of Honey Bee Cells using Deep Learning* [27], we were also invited to present it orally.

Even before our presentation on EurBee8, our software was being used by research groups from several nations. Among them are researchers from the University of Coimbra (Portugal), *Consiglio per la ricerca in agricoltura e lánalisi delléconomia agraria (CREA)* (Italy) and Aarhus University (Denmark). After our presentation, there was an interest of more institutions. We have already contacted and started the partnerships with them.

It is also worth mentioning the contributions that this work generated for the research groups Mountain Research Center (CIMO) and Research Centre in Digitalization and

Intelligent Robotics (CeDRI) from Instituto Politécnico de Bragança (IPB). Having supervisors belonging to both, it was possible to extract contents from the bioinformatics software developed that benefited each one.

We believe that this work has also contributed to reinforce the bonds between the institutions IPB and Federal University of Technology - Paraná (UTFPR), principally the Dual Diploma program that enabled a student from both institutions to develop this work as his thesis. The quality of this work was also confirmed when the oral presentation *Análise Automática de Alvéolos Em Favos Usando Deep Learning* (Automatic Analysis Of Cells In Honeycombs Using Deep Learning) won the award for best presentation in the area of Electrical Engineering and Information Technology in the *The Double Diploma Summer School & Symposium 2018* that took place in IPB. An additional partnership created between the institutions was the foundation of a group of studies in Deep Learning[1] that happened in the first half of 2018, it was idealised and put into practice by the authors of this work.

While we conclude this thesis, we are developing two articles to be published in the journal *Computers and Electronics in Agriculture*. In the first will compile the methods we developed to detect cells invariant to scale and remove false detections (Chapters 3 and 6). In the second article, we will present our methodology to find the best DNN models for the cell contents classification, and comparisons with the literature early approaches (Chapters 4 and 6).

1.4 Document Structure

The current work was built on 9 chapters. In this chapter, first, we present the context in which the thesis is, the relevance of the chosen theme, the objectives to be worked out during the research and the contributions made. In Chapter 2, we will present state of the art and the theoretical basis for image processing and Deep Neural Networks.

[1] https://github.com/AI-IPB-UTFPR/Meetings/blob/master/IPB-Meetings.md

In chapters 3 and 4 we will present the bases of our developed methodologies. In Chapter 3 we will describe our setup for image capture, methods for invariant cell detection, and approaches to remove false detections. In Chapter 4 we will present the techniques developed for the cell annotation, the definition of best input for the classifier and the methodologies used to find the best CNN architecture for our problem.

In chapters 5 and 6 we will present the results of our methods. In chapter 5 we will show the results of our cell detector, compare the different approaches to remove false detections, compare results obtained with different versions of OpenCV and lastly we will make comparisons between a previous method and the one proposed in this work. In Chapter 6 we will present the results obtained for the classification with different CNNs architectures, introduce techniques to improve the obtained results, and provide comparisons with methods proposed in the literature.

In Chapter 7 we will present the software created using the methods researched and developed in this work, we will also present ways to obtain additional resources for this work as source code and datasets. In Chapter 8 we will present approaches that can be tested and developed in future works. In Chapter 9 we will present conclusions and finalise the work.

Chapter 2

Context and Technologies

In this chapter, we will present the basis of our project. First, we will present approaches in the literature for the detection of cells in combs and classification of their content using images (Section 2.1). Next, we will present the fundamentals of image processing focusing on image enhancement and feature extraction techniques (Section 2.2). Afterwards, we will make explanations about artificial intelligence and machine learning until we explain more deeply deep neural networks and their applications in image classification (Sections 2.3, 2.4 and 2.6).

2.1 State of the art

Besides the methods of hive weighing and the Lieberfeld, another common method for assessing colony strength is “the acetate sheet" [28]. In this method, a transparent acetate sheet is used to assist in the analysis. The sheet is placed on the hive frame under assessment and one by one the cells are annotated with markers according to their content. Only one sample is extracted in this approach and based on it the researcher makes subsequent analyses [29]. While annotation of individual cells on the acetate sheet guarantees reproducibility, this method has a drawback: besides being a tedious task, it makes the frame staying outside the hive for a long time causing stress to the bees, which may negatively influence colony development [30].

As the analysis of cells from images reduces the “off-hive-time", softwares were developed to help researchers in the combs assessment from images. In [31], an approach was implemented to track the brood development, where the goal was to photograph a frame, to annotate the cells with eggs and to store their relative positions in the comb. After a time of brood growth, new photos are taken, and the previous detections are used to analyse the current state of each cell. In this work, the authors used the software Fiji[1] where detections were made using a threshold that sought to highlight the contours of the cells. Using this method, the authors reported that the disturbance in the hives decreased about 50% as compared to the acetate sheet method due to the speed of the process.

Classification of comb cells in different classes is a complex task, so the first approaches using digital images tried only to count the number of capped brood [32]–[34]. In 2006, Emsen [32] developed a semi-automatic method to measure the area of capped brood in the comb. The segmentation was done mainly using the selection tools from the software *Adobe Photoshop®CS2 9.0*. This method did not provide the exact number of cells, but the percentage of the comb covered by capped cells. Yoshiyama et al. [33] developed an approach similar to that of Emsen [32], in their work is also necessary a tool like *Adobe Photoshop®* to create the semi-supervised segmentation with capped cells. The novelty of this work was the plugin called *LarvaeArea* created by Yoshiyama et al. [33] for the image processing software ImageJ[2]. With this plugin the user can open the previously segmented image and calculate automatically the area with capped and uncapped cells.

Cornelissen et al. [35] developed a semi-automatic method as an ImageJ plugin and compared its performance with the Lieberfeld method. The first comparison was related to time. The authors calculated 26 seconds per frame to collect the data with the Lieberfeld method and 19 seconds to take a photo. However, the image needs to be segmented and processed on a computer, so 30 seconds had to be added. The Lieberfeld method does not need this additional time. Thus, the Lieberfeld method proved to be more efficient. The digital method could be faster if the digital count of the cells were fully automatic.

[1]https://fiji.sc/
[2]https://imagej.nih.gov/ij/index.html

In the second comparison, Cornelissen et al. [35] found that using the segmented area with the aid of the software generates results with a higher correlation with the actual number of cells than when using the Lieberfeld method (0.99374 against 0.90848).

The first digital method able to detect individual cells was developed by Lee et. al [19]. The authors used pre-processing methods to highlight image edges and then applied Circle Hough Transform (cHT) to detect the cells. The aim of this work was to develop a method for detecting the cells individually. As a result, they obtained a detector with a cell detection rate of 82.6%. Herein, we also developed a method for detecting cells automatically. The method is described in section 3.3 and compared with that developed by Lee et. al [19] in section 5.5.

Rodrigues et al. [34] developed a method for automatic detection and counting of capped brood cells. The process uses a sliding circle; This circle has the same size as the cells and using a loop method it moves over the image pixel by pixel, in each position it stops, it calculates the contrast between the pixels of its edge and its interior. From this contrast, it is possible to know whether there is a cell in the analysed position and, according to established thresholds, it is possible to know if the analysed cell is capped or not. Rodrigues et al. [34] calculated precision and recall metrics to measure how the method performed on capped cells. They obtained 99.04% and 97.2% in these metrics.

In a conference poster, Wang and Brewer [36] presented some information and tests carried out using the commercial software *HoneybeeComplete*. The authors did not report the method to detect the cells and stated that the classification of cell contents was done using a combination of pattern recognition algorithms. According to a diagram shown in the poster, the colour and contrast of the cells were the two features used in the classifier. Given that the features were seemingly chosen by hand and given the year of publication (2013), it is likely that Wang and Brewer did not make use of more recent techniques of Deep Learning. The authors obtained a classification rate of capped cells of 97.4%, a number that increased to 99.5% when a pre-selected region in the image made by a user was used.

Although the work of Höferlin et al. [37] was converted into a commercial software

(*HiveAnalyzer*), details on the approach and results were made available in their publication. The images were captured in a circular chamber with LEDs to have a better light distribution over the frame. Detection of the cells was performed using the cHT method. The authors did not mention preprocessing methods before implementing cells detection. Content classification was carried out with a cascade of classifiers based on linear Support Vector Machines. The method has several stages, and in each one of them examines specific features, if the image has the features, it is defined that the image belongs to a particular class. Otherwise, the method searches for new features in a new stage. The cell receives the label uncategorised if it passes through all stages and does not fit in any of them. Features used in classifications are based on colour and texture descriptors such as Haralick, Local Binary Patterns, Colour Histograms, as well as shape descriptors such as Histogram Oriented Gradients and Template Matching. This work covered more cell classes than the previous works described above, namely: Empty, Egg, Young Larva, Old Larva, Capped Cell, Nectar and Pollen. The authors analysed the accuracy of the classifier under cells classified with high confidence (78% out of a total of 20,000 cells) the precision calculated on this set was 94% [37].

To our knowledge, none of the studies published so far have employed Neural Networks such as CNNs. Only one publication makes the classification of cells with contents that are not brood (e.g. Nectar and Pollen) [37], and none has produced a free software for the automatic detection and classification of cells with brood and food reserves. As such, this represents an excellent opportunity for innovation using novel methods to solve an important problem in honey bee research and there is a large group of users worldwide who need reliable software that are freely accessible.

2.2 Image Processing

Image Processing (IP) refers to a set of operations applied to images seeking to extract information from it. One interpretation of IP is a kind of signal processing where the input is an image and the output can be an image or features extracted from it [38]. The IP field

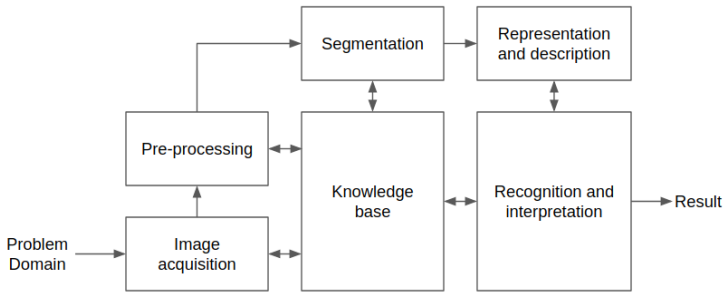


Figure 2.1: Image processing pipeline

studies the processing of digital images by computer and had one of its first applications when was necessary to reduce an image size to transport it by submarine cables from London to New York [39]. Some steps are common but not mandatory in several IP systems, we present them in figure 2.1, following we will make a brief explanation of each one of them.

- **Image acquisition:** performed by physical devices endowed with sensors sensitive to specific spectra of electromagnetic energy, they capture the energy and transform it into electrical signals. The acquisition process consists of three steps: light reflected by the object, an optical system focuses the light and a sensor measures the amount of energy, this data is transformed into a digital file [40]. In our work, we present our image acquisition setup in section 3.1;

- **Pre-processing:** at this phase, it is possible to increase considerably the quality of the features to be extracted in later stages, because it works with light corrections, focus adjustment, noise removal and edge enhancement. These operations can be understood analogously to the process of normalisation of data that is carried out in the statistics [41]. In our work we discuss our pre-processing methods in section 3.3;

- **Segmentation:** the goal of segmentation is to group regions of an image that belongs to the same context, that is, regions with similar surfaces as objects or parts of objects [42]. These methods use the texture or structuring elements present

in the image to divide it between regions or polygons [41]. In section 3.5.3, we present a segmentation method for combs, an important task to reduce the number of false cell detections;

- **Representation and description:** is common to perform this step after segmentation. Feature descriptor algorithms extract characteristics from the image such as colour, texture and shape [43]. The representation of an image can be given by the outline of the segmentation or by its internal characteristics. When its outline is chosen, edge descriptors such as length, joint orientation, and endpoints are used. When the internal representation is chosen, features such as colour and textures are sought. There are cases where both the contour and its internal features are part of the representation [39]. Because we use CNNs to classify cell contents, we do not need to explicit for the classifier the features that each class has, only to tell which class is present in each image during the training;

- **Recognition and interpretation:** the last phase of the process is recognition and interpretation, recognition consists of assigning a label to the object found according to its characteristics. The interpretation has the purpose of giving meaning to a set of recognised objects [44]. Herein the recognition phase is made using CNNs, in chapter 4 is explained how we applied recognition in our problem.

- **Knowledge base:** all stages of the process are connected to the knowledge base. This structure guides the communication of data between each stage [44].

In the next section, we will present some image enhancement methods. These pre-processing methods are essential for our work, as they allow the standardisation images to be treated by the same algorithms in the following phases.

2.2.1 Image Enhancement

The enhancement of an image can be performed using a set of techniques oriented to improve some details from the image based on a pre-defined purpose. This purpose may

be, image analysis by a user, or by computational methods such as feature extractors or segmenters. Image enhancement methods are usually problem specific, so its difficult the combination of methods applied to a problem, to solve a different one [39].

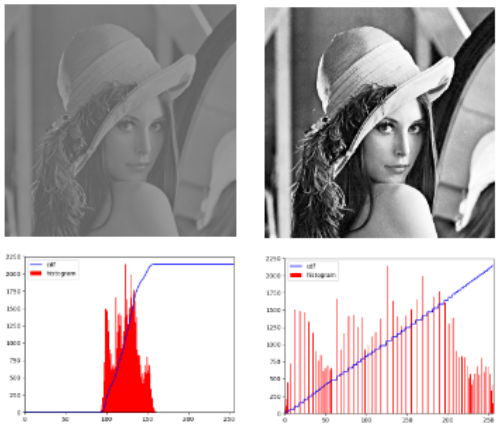
Enhancement methods are mostly separated into two categories, methods applied to the spatial domain and methods applied to the frequency domain. Methods related to the spatial domain work directly on the image pixels, operations performed by these methods can take into account the value of a single pixel or a pixels neighbourhood. Methods applied to the frequency domain work manipulating the image histogram. A histogram is a graphical representation of the distribution of pixel intensities on an image. Each intensity (from 0 to 255) has a vertical bar with height proportional to its frequency on the image.

Regarding image enhancement for feature extraction, it is common to use techniques such as colour reduction, edge enhancement to better separate image segments, contrast increasing to better define objects and reduction or removal of noise. We will explore different image enhancing techniques in the following.

- **Histogram Equalization (HE):** is a technique applied in the frequency domain, it seeks to improve the image contrast. This result is obtained from a better distribution of the most frequent intensities along the histogram [39], [45]. Examples of images that benefit from using this technique are those with foreground and background content with similar tones. We show an example of the HE application in figures 2.2(a) and 2.2(b).

  This method produces better results when the histogram has few peaks, and they are grouped in small regions. Images with many spread peaks can produce overexposed or underexposed results. One way around this problem is to use an adaptive histogram equalisation;

- **Adaptive Histogram Equalization (AHE):** this method calculates several histograms for the image, each one representing a different region. Separating the image into regions makes histograms with closer peaks. The most common way to



(a) (b)

Figure 2.2: (a) Example with low contrast, (b) example with HE applied.

separate the image is splitting it into small tiles, a typical size for them is 8×8px [46]. With the tiles extracted, HE is applied to each one of them. After this processing, the neighbouring tiles are recombined using bilinear interpolation so that the divisions became less evident;

Because AHE increases contrast in small regions, it also increases the noise. To reduce this problem, a contrast limiter can be applied as proposed in the Contrast-Limited Adaptive Histogram Equalization (CLAHE) method [47]. Using CLAHE will result in a image with less noise, but it will not remove all. According to the desired application other methods of noise removal may be necessary;

- **Smoothing Filters:** they are methods applied in the spatial domain in order to reduce noise in images [39]. Noise are abrupt changes that happen in pixels belonging to regions that would ideally be uniform in the original scene [48]. Poor quality of the photographic equipment and poor illumination at the time of image capture can lead to noisy images;

  Because noise is often a discrepant intensity of a few pixels in a region, if the value of each of them is recalculated based on the intensity of its neighbours, the noise tends to disappear. The simplest way to perform this value readjustment is by summing

the pixels values of a small region, divide that result by the number of pixels used in the calculation and assign this value to all the pixels in the region.

Noise reduction is usually made by blurring image. This action has a drawback which is the loss of features such as the objects contour. To bypass this problem it was developed a process called bilateral filter [49]. It was built from the junction of two filters. One takes into account the geometric distance between a central pixel and its neighbours and the other considers the intensity difference between the analysed pixel and each one of its neighbourhood. Using both filters, the bilateral filter can preserve contours while removing noises [50].

- **Canny:** is a method developed for edge detection in images. It is very sensitive to noise, so it is crucial to apply smoothing filters before using it. In order to find edges, firstly image gradients are calculated in the horizontal and vertical directions for each pixel. Based on the magnitude of a gradient in a pixel it is possible to know if there is a sudden change from colours, if there is, this pixel may be part of an edge. Pixels that are unlikely to be edges are removed in the second phase. Finally is performed the step called Hysteresis Thresholding, it defines which edge candidate are real and which are not. For this, two thresholds are used, $minVal$ and $maxVal$. Edges with gradient values higher than the $maxVal$ are kept, edges with values smaller than the $minVal$ are removed. Those that fall between the two thresholds are classified as edges only if any of their pixels have connectivity with pixels classified as an edge [51], [52].

In the following section, we will present the Hough Transform. This method has great importance in our work because we use it to perform cells detection.

2.2.2 Hough Transform

Hough Transform (HT) is a feature extraction technique. It can be used find features of a particular shape within an image. Duda and Hart developed the classical HT method

used today in 1972 [53]. In the beginning, the main application of the method was for detecting lines in images, but after new studies, it was discovered that the method also could identify arbitrary shapes, such as circles and ellipses.

HT is currently used for image processing in different areas such as astronomy for satellite tracks removal [54], detection of supernovae and galaxies [55]; robotics to help agents see the environment they are in [56], and also to help the decision-making of doctors about medical imaging results [57], [58].

Following we will present two principal applications of HT, they helps with the detection of lines and circles in images.

- **Hough Line Transform (HLT):** there are different ways to represent lines mathematically. Among the most common ways is the slope-intercept (Equation 2.1) where a stands for the slope and b is the point where the line intersects the y axis [59].

$$y = ax + b \tag{2.1}$$

  This form of representation has a negative side which is the value that slope a receives when the line becomes vertical, in this case, the value of a becomes infinite. The HLT method does not use the slope-intercept representation, but rather the polar representation (Equation 2.2), because it does not have this disadvantage.

$$\rho = xcos(\theta) + ysin(\theta) \tag{2.2}$$

  A perfect line represented in the Cartesian space becomes a point in the θ, ρ space (parameter space). A sine curve represents in the parameter space all lines passing through a point in the Cartesian space, a set of points that form a imperfect line in the Cartesian space tends to form a set of curves in the parameter space and all these curves have a point in common. Given these premises it is possible to understand the operation of the HLT method. A transformation is made from a set

of edge points in the Cartesian space, and then a set of curves is generated in the parameter space, points in this parameter space that have more curves intersecting it have a higher probability of representing a line in the xy space.

The HLT method uses an accumulator matrix-shaped, each position represents a point in the parameter space, +1 is added to each point that curve passes though, at the end of the processing a threshold is used to filter the points with more votes. In figure 2.3 is presented the comparison of a line representation in Cartesian and parametric space [59];

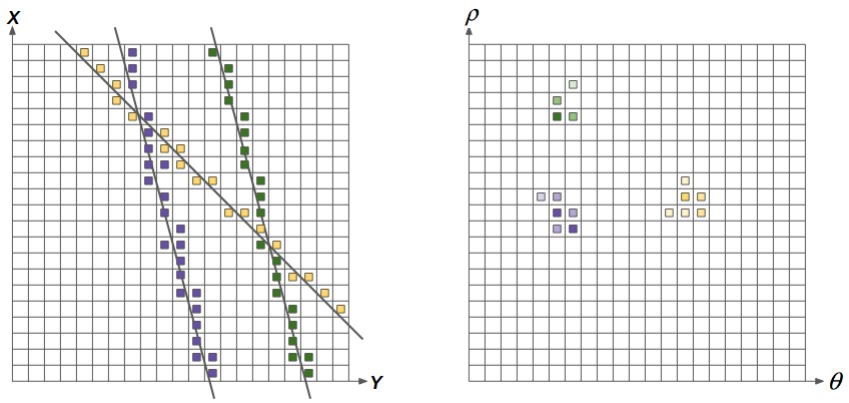


Figure 2.3: Lines represented in the Cartesian (left) and parameter (right) spaces.

- **Circle Hough Transform (cHT):** developed in 1975 [60], cHT works in a manner roughly analogue to HLT. In its most straightforward implementation, the accumulator is built based on the parametric equation presented in equation 2.3, where a and b represent the circle center and R its radius. Because the parametric equation has three coordinates in the parameter space (a, b and r), the complexity of the algorithm starts to increase and makes necessary to use a 3D accumulator [61].

$$(x - a)^2 + (y - b)^2 = r^2 \tag{2.3}$$

Using a three-dimensional accumulator increases memory usage and reduces the

performance. To work around this problem the *Hough Gradient Method* was developed, this method is the one implemented in the OpenCV library [62]. It uses the gradient information of edges calculated using methods like Canny and Sobel to perform the circle search, this trick increases the performance and reduces the memory needed [62].

2.3 Artificial Intelligence

This term was created by John McCarthy in 1955 [63]. Artificial Intelligence (AI) was defined by him as "the science and engineering of making intelligent machines". Starting from the premise that humans are intelligent, an AI system can be understood as a computational model that exhibits intelligent behaviours such as those performed by humans [64]. AI systems manifest intelligent behaviours often in specific tasks. Some behaviours are the ability to reason, discover meaning in data, generalise knowledge learn learn from experience [65].

AI study overlaps several different fields such as robotics, systems control, data mining, speech recognition, logistics and image processing. The result of an AI system is often a software. This a subset of AI as software, capable of learning autonomously, is called Machine Learning [66].

2.4 Machine Learning

The term Machine Learning (ML) refers to the automated detection of meaningful patterns in data [67]. ML is one of the fastest growing areas of computer science, one reason for this result is its capacity of being inserted into different contexts and problems. The society lives surrounded by ML technologies, among them are the services offered by big companies like Amazon, Facebook, Google and IBM. ML is so important to Amazon, they stated, "Without ML, Amazon.com couldn't grow its business, improve its customer experience and selection, and optimise its logistic speed and quality." [68].

Examples of tasks to be solved by ML algorithms are: (i) segmentation of markets and clients, for problems like this it is common the use of clustering algorithms, they can find structures and group data unsupervised, that is, without labels in the data; (ii) Sales forecast based on current data, it is typical that in problems like this the desired result is a continuous value, supervised techniques of regression can map a set of inputs into a continuous output, it can be an excellent choice for this kind of problem; (iii) Classification of images, classification problems often have a set of input examples that need to be fitted in a set of discrete and finite outputs, this kind of problem can be solved by ML models for classification, they can learn from a labelled training set and then generalise the solution to examples not seen yet [69].

ML techniques are tipically embedded softwares when the code to be developed needs adaptability, to produce results based on their "experiences", or to reproduce capacities simple to be performed by humans but complex to be described (programmed) using conventional methods [67].

Humans can perform tasks such as recognising Arabic numbers without significant efforts. For this task, the brain uses a sequence of visual cortices with millions of neurons and tens of billions of connections. The problem of number recognition is difficult, but the evolution our brain over the ages made process be done unconsciously, because we feel this task as automatic we usually do not pay attention to its complexity. The real complexity is discovered when one wishes to formally describe the features of numbers in images using a programming language. What was once easy, becomes complicated because of the different shapes that can be used to represented numbers visually. Even when coding many features, it will always be exceptional cases not convered [70].

Frustrated attempts to represent knowledge by classical methods in computers made studies with agents capable of acquiring knowledge from their own experiences gain space [71]. One approach to develop this capability in computers is using Artificial Neural Networks.

2.5 Neural Networks

Neural Networks (NN) have their foundation based on observations made about the human brain functioning. These mathematical models can build a knowledge base composed of rules that connect a set of inputs with the expected outputs during a process called training. Its can also maintain associations when new data is processed (generalisation) [72].

Haykin [72] presents some advantages for the use of NNs, they are: capacity to deal with non-linear problems; ability to adapt their synaptic weights based on environmental conditions; ability to show the confidence on its decision and fault tolerance. The distributed nature of the information in an NN allows it to be robust with faults, even with some damaged neurons, it still has the ability to detect patterns. Significant damage is required to break its functioning; its brain-like structure allows NNs to make decisions using parallel processing, often faster than sequential methods of processing.

Because of the similarities between NNs and the human nervous system, we will give a succinct presentation of the nervous system functioning following.

2.5.1 Functioning of the Nervous System

The adult human brain has about 10^{11} neurons interconnected by approximately 10^4 connections each. Figure 2.4 shows the representation of a neuron. Neurons has three main components they are: dendrites, cell body and axon. Dendrites are ramifications of neurons. They carry electrical and chemical signals into the cell. The cellular body sums up the received stimuli, and if they reach a threshold, it is sent to another neuron. The axon is an extension that carries the signal from one neuron to another. The point of connection between the dendrite of one cell and the axon of another is called synapse [73]. Synapses are a fundamental part of the neurons communication structure. In traditional descriptions of neuronal functioning, synapses are considered connections that may impose excitation or inhibition properties on the recipient neuron [72].

NNs have so far not reached the complexity of a human brain, but there are still two

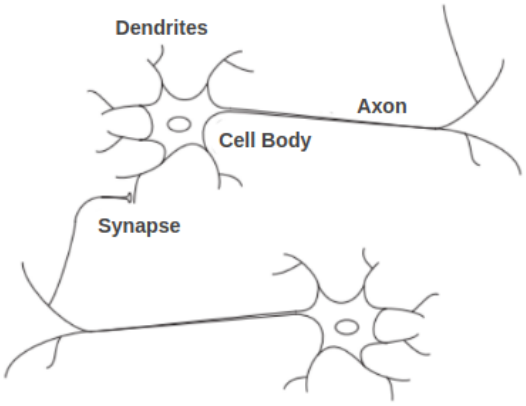


Figure 2.4: Simplified representation of a biological neuron.

fundamental similarities between the biological brain and the artificial brain. First, both are composed of blocks of neurons (although artificial ones are simpler) interconnected. Second, the connections between a network of neurons defines their function [73].

Biological neurons are typically six orders of magnitude slower than logic gates. Events that happen in nanoseconds in silicon circuits may take milliseconds to occur in neurons. Even so, the brain is capable of performing various tasks at speeds higher than conventional computers. For this to be possible, the brain uses a massively parallel processing structure [72], [73].

2.5.2 Historical Notes on Neural Networks

The first applications of mathematical models for the study of the nervous system were made by the Russian mathematician Nicolas Rashevsky between the years 1936 and 1938 [69]. The modern era of studies in Neural Networks began in 1943 with the works of McCulloch and Pitts. In this classic publication the authors described a model that connected the studies of neurophysiology and mathematical logic [72], [73].

The work of McCulloch and Pitts was continued by Donald O. Hebb. He summarised two decades of research in the book *The Organization of Behavior* in 1949. In this book, he presented a proposal on how synaptic modifications affect learning. Hebb also emphasises how neural connections are continually adjusted during the training process [72], [73].

Russell and Norvig [69] mention the work of Marvin Minsky published in 1951 as the likely first application of NNs in hardware. Haykin presents the work made by Rochester et al. in 1956 [74] as the first application of NNs in a computer. In the same article by Rochester et al. results showed the necessity of inhibitions for the theories proposed by Hebb to work. Still in the 1950s, Uttley demonstrated that an NN could learn to classify simple binary sets into corresponding classes [72].

Frank Rosenblatt, introduced in 1958 a new NN model called Perceptron, adjustable synapses were added in it [75]. In addition to the new model, Rosenblatt also presented a training algorithm, with this algorithm the network could learn to perform certain types of functions. Because Perceptron is able only to classify linearly separable data, in 1969, Minsky and Papert pointed out in the book *Perceptrons - an introduction to computational geometry* that this could be a problem. Another argument made in the book was that there was no guarantee of convergence when using more than one hidden layer in the model. Due to these arguments, the connectionist approach stopped for about a decade [76].

With the advances of microelectronics the availability of computational resources increased, allied to this factor, discoveries in NNs were responsible for the resumption of studies in the area in the 1980s. Another reason for the strengthening of the researches was the discovery of the *backpropagation* algorithm for training Multilayer Perceptron networks. It was discovered independently by several researchers, but the most influential publication was the *"Learning representations by backpropagating errors"* by Rumelhart et al. in 1986 [73]. The algorithm showed that the Minsky and Papert's view of the perceptron was indeed misleading about multilayered training.

2.5.3 Basic Structure of an Artificial Neuron

The artificial neuron is a fundamental unit in an NN. Figure 2.5 shows a representation of a neuron [72]. It can also be represented mathematically as in equations 2.4 and 2.5 where v_j represents the potential activation of the neuron, $\phi(.)$ stands for the activation

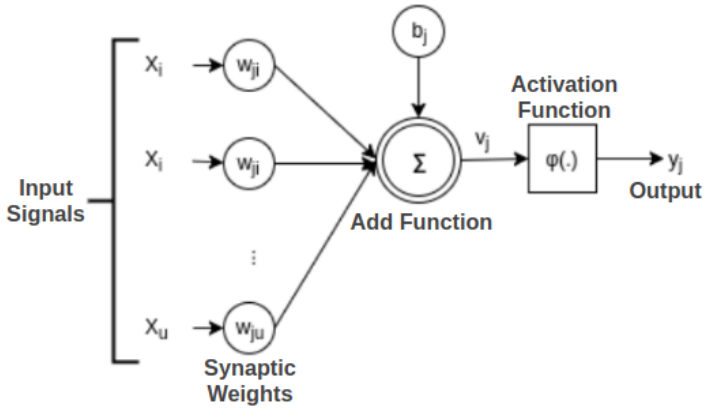


Figure 2.5: Artificial neuron.

function, u represents the total of synapses, and y_j is the output of the neuron j.

$$v_j = b_j + \sum_{i=0}^{u} w_{ij} x_i \tag{2.4}$$

$$y_j = \phi(v_j) \tag{2.5}$$

For Haykin an artificial neuron has three basic elements, they are [72]:

1. **Synapses:** each synapse is composed by an input signal X_i connected to a neuron j. Before the signal is sent to the neuron, it is multiplied by a weight w_{ji}. Unlike the synapse of the brain, the weights that make up the artificial neuron can assume positive or negative values;

2. **Adder:** calculates the sum of the input signals, weighting them according to the synaptic weights;

3. **Activation function:** has the objective of restricting the amplitude of the output of the neuron to a finite interval. It is common to use [0,1] or [-1,1] as the output range.

Another element present in figure 2.5 is the bias b_j. Bias is a synaptic weight similar to others and it can be omitted in some situations because it can be represented as a

constant input of value 1 [73]. bias has the effect of increasing or decreasing the net input in the activation function, depending on its value [72]. According to Nielsen [70], bias can be understood as how easy it is for an artificial neuron to have one as output (using the Heaviside activation function), i.e. a neuron with the high bias value will fire easily, the inverse rule is also applicable.

Activation functions can be represented by linear or non-linear functions. They are chosen to satisfy some specification of the problem that the neuron is trying to solve [73]. Some examples of activation functions are:

- **Signal Function (Heaviside):** this function transform negative or zero values into a null output and positive values into a unitary output, according to equation 2.6. A neuron that applies this activation function is known as the McCulloch-Pitts model. The name was attributed in tribute to the pioneering work of McCulloch and Pitts [77]. Many activation functions deviated from the McCulloch and Pitts one, a characteristic present in several of them is the ability to generate outputs with values different from only zero or one;

$$y_j = \begin{cases} 1, \text{if } v_j > 0 \\ 0, \text{if } v_j <= 0 \end{cases} \tag{2.6}$$

- **Linear and Ramp function:** given by $y_j = v_j$, has the property of output its own input. The Linear Function can be limited to generate values within a certain range [-β,+β], in this case, its name changes and it is called the Ramp Function. Equation 2.7 presents its mathematical representation [76].

$$y_j = \begin{cases} +\beta, \text{if } v_j > \beta \\ v_j, \text{if } -\beta < v_j < +\beta \\ -\beta, \text{if } v_j <= -\beta \end{cases} \tag{2.7}$$

- **Sigmoid Function:** to perform the training of an NN, in many cases, it is necessary

to ensure that small adjustments in the weights and *biases* of neurons generate small changes in the output. Classical Perceptron networks do not have this guarantee, regardless of the changes in weights the output will be only 0 or 1. A solution to overcome this issue is to use the activation function sigmoid in the network. It has the same structure as a Perceptron, but instead of their output being either 0 or 1, they can output any value in that range. An example of the sigmoidal function is the so-called logistic function. It is defined mathematically in equation 2.8 where ϕ stands for the slope of the function. Varying this parameter curves more or less accentuated are obtained [76], [77]. It may also be desirable to have a sigmoidal function that generates outputs between -1 and $+1$. In these circumstances, one can use the hyperbolic tangent function (denoted by *tanh*) [77]. Equation 2.9 defines it.

$$y_j = \frac{1}{1 + e^{-\phi v_j}} \tag{2.8}$$

$$y_j = tanh(\frac{v_j}{2}) = \frac{1 - e^{-v_j}}{1 + e^{-v_j}} \tag{2.9}$$

- **Rectified Linear Unit (ReLU):** this function has become popular in recent years. It computes the max function (Equation 2.10), so it turns all the negative values of v_j to zero and keep the value of v_j for the others. Krizhevsky et al. presented positive points for using the ReLU activation function [78]. The advantages are, the acceleration of convergence by a factor of six compared to the performance of the *tanh* function, in addition to being less computationally costly;

$$y_j = max(0, v_j) \tag{2.10}$$

- **Softmax Function**: this activation function is used mostly in neural networks for classification. With it, is assured that the sum of all neurons in the same layer

will be one. Because it is generally implemented in the output layer, this function output can be interpreted as the probabilities of an input belonging to each of the pre-defined classes [79]. Equation 2.11 represents the calculation required for the Softmax function, where j stands for a neuron belonging to the set C (e.g. output neurons).

$$y_j = \frac{e^{y_j}}{\sum_{j \in C} e^{y_j}} \tag{2.11}$$

2.6 Deep Neural Networks and Deep Learning:

Figure 2.6 shows the structure of a multi-layered NN. NNs with many hidden layers (between the first and last) are called Deep Neural Networks (DNN). Typically, DNNs have a set of sensory units in their input layer with the purpose of bringing external information to the inner layers. Hidden layers extracts characteristics from the input data to help in the prediction made in the output layer. The output layer is designed to represent the prediction made by the network over the input.

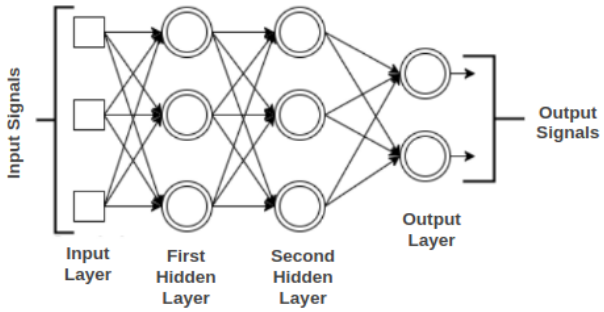


Figure 2.6: Deep Neural Network architecture.

Using DNNs is possible to build solutions using Deep Learning (DL). The DL has gained space in recent years mainly after Alex Krizhevsky et al. work in 2012 [78]. In this work, DNNs were used to classify images in a competition called ImageNet Large Scale Visual Recognition Challenge (ILSVRC). As a result, they achieved state of the art in the task of classifying images into 1000 different classes. Deng and Yu [80] defines DL

as being a technique belonging to the machine learning class of algorithms. It explores several layers of non-linear information processing for the extraction and transformation of characteristics into predictions [80].

The development of solutions using DNNs increased after the creation of more powerful computers and the formulation of a method known as *Backpropagation*. It was discovered independently by several research groups during the 70s and 80s [81]–[84]. The method is used to perform the adjustment of weights in a DNN, in order to reduce the errors made by its predictions.

The *backpropagation* method consists of two steps [72]. In the first, the input values are propagated through the layers in order to generate a set of outputs. In this step, the network weights remain unchanged. In the second step, also known as *backpropagation*, the network weights are adjusted based on the difference between the output obtained is the expected. A loss function generates this difference. With the *backpropagation* method, it is possible to know how much each weight of the network influenced in the computed loss. Based on this information the values of the weights are changed. After a sequence of weight adjustments, it is expected that the value of loss will decrease and converge. With the model converged is expected from it to be able to generate predictions over inputs not seen during training, this capacity is called generalisation.

DNNs can be built with different architectures. An architecture is defined by the way neurons are arranged, by the direction of their synapses, by the number of neurons in each layer and by the activation functions used in different layers.

2.6.1 Convolutional Neural Networks

CNNs are a category of deep neural networks that have recently been shown to be effective in image recognition and classification, receiving state of the art in several subareas [85]. The most recent models of CNNs are based on the studies made by Fukushima with its image processing model Neocognitron [86] and LeCun with its implementation of LeNet in 1998 for the recognition of characters [82]. Vargas et al. describe CNNs as a

variation of DNNs, which similarly to the computational vision processes can apply filters to images, maintaining the close relationship between the pixels along the network [85]. CNNs have particular layers for the features extraction, reduction of data dimensionality and classification. They are presented as follows:

- **Input:** receives an image from the external environment and sends it to the following layers;
- **Convolution:** is the process of calculating the intensity of a pixel according to the intensity of its neighbours. The example of the calculation performed is shown in figure 2.7. This type of layer is fundamental for the extraction of features from the image because according to the values of the kernels used, certain features are enhanced. The features can be, for example, lines, points and curves (Figure 2.8). The values of kernels are learned during the *backpropagation* process, so the model learns which features best contribute to generate correct predictions;

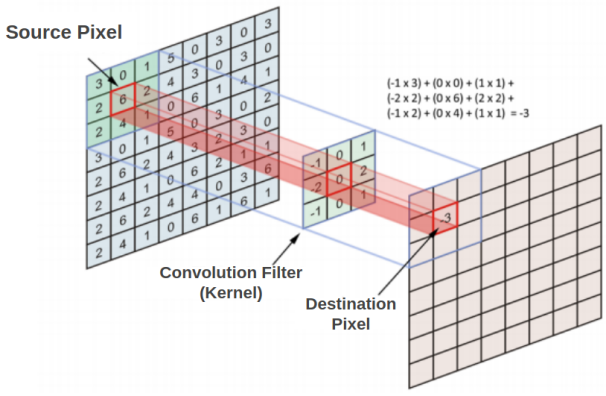


Figure 2.7: Example of convolution [87].

- **Activation Function (ReLu):** the same presented in section 2.5.3 or some variant of this function. It is applied in each pixel, in order to remove negative values and add non-linearity to the model [89];
- **Pooling:** with this layer, it is possible to reduce the dimensionality of the image being processed, thus reducing the number of parameters and calculations performed

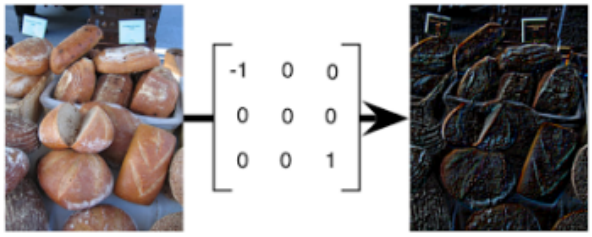


Figure 2.8: Convolution applied in an image [88].

by the network, besides assisting in the control of overfitting [89]. Another effect obtained by the use of the pooling layer is the selection of features that are invariant to transformations, this increases the generalisation of the network.

Figure 2.9 shows an example of the use of the pooling layer. In it, the size of the matrix representing the image is reduced by half. The reduction is made using a 2×2 rectangle (kernel) that moves through the image every two pixels (step), at each position it stops, the largest value is found and then transported to a new matrix [89];

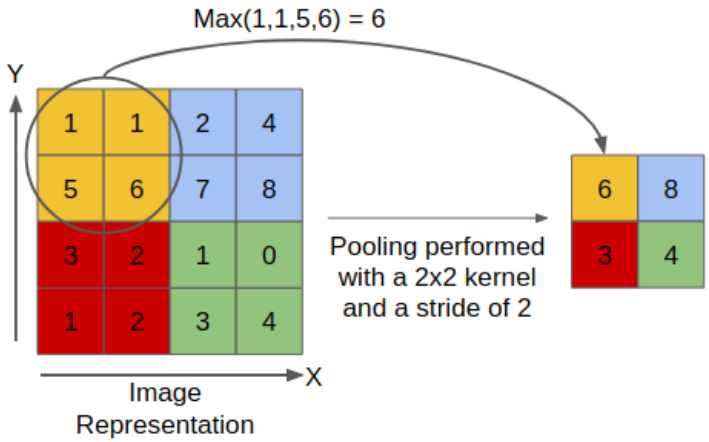


Figure 2.9: Pooling applied in an image.

- **Fully Connected:** the name “fully connected" implies that all the neurons of a previous layer are connected with all the neurons of the immediately next layer [89], [90]. The objective of this layer is, from the output of the convolution and pooling layers, to classify the input image in the classes belonging to the training set (in

classification problems). The probabilities of the image belong to a specific class are defined by the Softmax activation function in the last fully connected layer [89].

We have shown common layers to be used in CNNs. Now we will present some specific architectures. We will show them chronologically and some advances of each one will be highlighted.

2.6.2 Recent CNN Architectures

In this section, we make a historical comparison of novel architectures. New contributions made by each one will be highlighted. We present a comparison among some of these models in the task of classifying comb cells in section 6.

- **AlexNet**: known as the first large-scale CNN to win the ILSVRC image classification competition in 2012, AlexNet [78] has a large part of its architecture LeNet-inspired [82], the inspiration can be found in the use of convolution layers followed by pooling layers, although AlexNet is deeper. Among the major contributions of this work are the use of ReLU activation functions in its hidden layers seeking to create nonlinearities, the use of normalisation layers and data augmentation techniques to generate virtual examples. In 2013 the competition was won using the ZFnet model [91], this architecture was based on the AlexNet and had hyperparameters such as convolution filters size and strides better optimised.

- **VGG**: GoogleNet architecture won the 2014 ILSVRC in the image classification category, but another great architecture called VGG16 and VGG19 was close to the victory, besides gaining in the category of classification + localisation of objects in images. Visual Geometry Group (VGG) from the University of Oxford developed these architectures. Values 16 and 19 refer to the number of weight layers. Among the major contributions brought by this architecture is the use of 3x3 filters in all convolution layers, it allows the creation of deeper architectures. With a deeper model, more nonlinearities come out during training, this helps the model fitting the dataset.

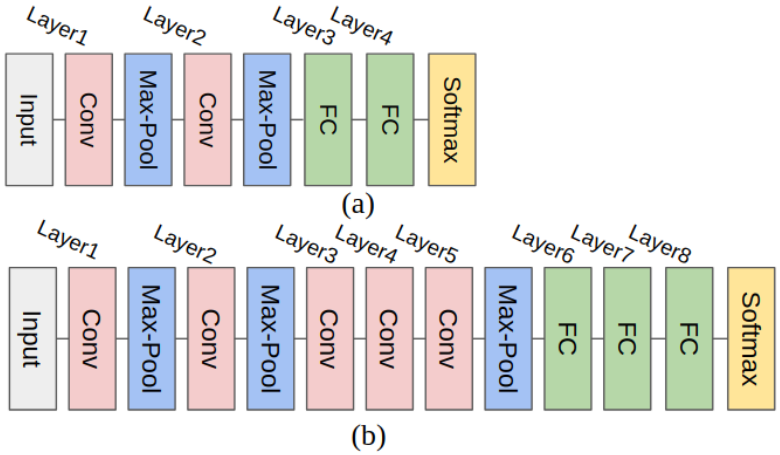


Figure 2.10: Comparison between the architectures LeNet (a) and AlexNet (b). Layers of normalisation and dropout were omitted in the AlexNet architecture to facilitate the visualisation.

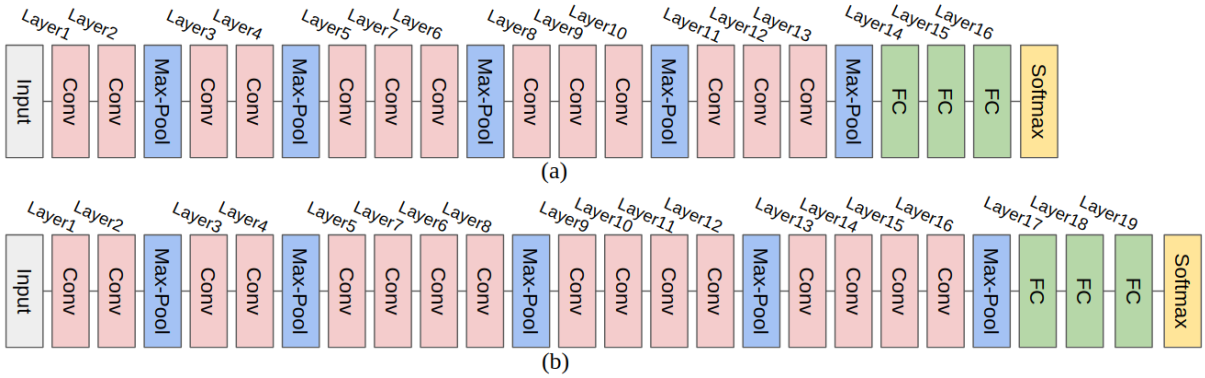


Figure 2.11: VGG16 (a) and VGG19 (b).

- **GoogleNet (InceptionV1)**: the architecture named Inception won the competition ILSVRC 2014 in the image classification category and reached state of the art in that year. Its main brand is the best use of computing resources within the network. With a carefully designed scheme, it was possible to increase the depth and width of the network while still reducing by about 12 times the number of parameters when compared to AlexNet. In addition to optimising the use of resources, especially processing and memory use, this architecture also optimised the quality of the results, among the main factors is the insertion of new module called Inception (Figure 2.12).

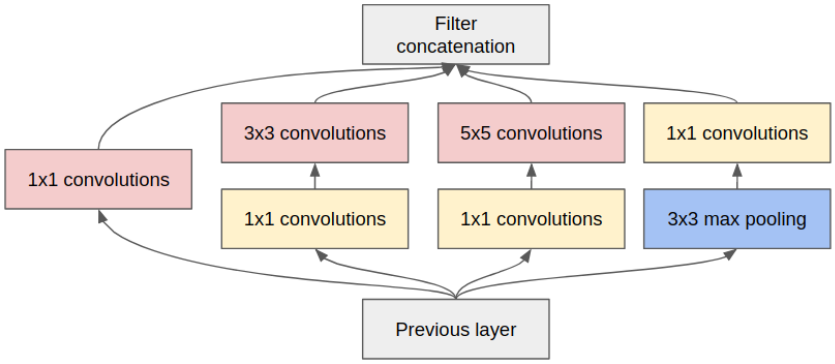


Figure 2.12: Inception module, the convolutions 1x1 in yellow, called bottleneck, reduces the dimensionality of the tensors coming from the previous layers. In the max-pooling layers, the bottleneck reduces the tensor's dimension before sending them to the next layer. This process reduces the total number of parameters in the network and makes possible to stack many modules.

One can interpret the Inception module as a network within a network [92]. This interpretation is due to the application of filters of different sizes in parallel on the results of the previous layer. These different filters process visual information in multiple scales and aggregate them so that the next layer abstracts characteristics at different scales simultaneously. Another contribution of this work is the addition of auxiliary classifiers in the hidden layers (Figure 2.13), they help in the generation of additional error signals during the training. This approach reduces the effects of a problem called vanish gradients, it occurs when the error calculated at the end of the network weakens along the layers during the backpropagation phase, this causes small adjustments in the previous layers, slowing the convergence. During the inference phase, these auxiliary classifiers are removed.

- **InceptionV3**: in the work *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* [93] is argued that during the training of a DNN model, different input examples can create different value distributions in the model's weights. This constant variation of distributions between the layers increases the training time because it becomes necessary to use lower learning rates

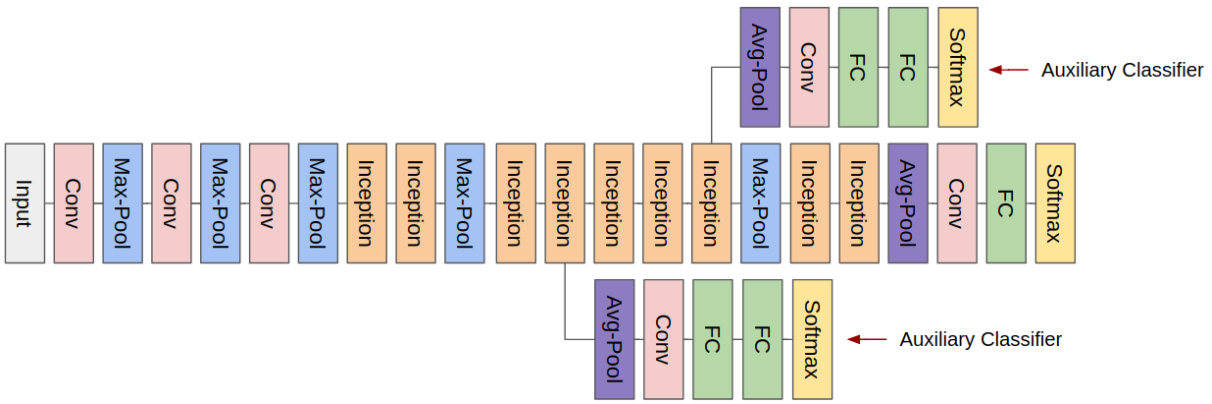


Figure 2.13: GoogleNet architecture, the first implementation using Inception modules.

and a more careful parameters initialisation. This effect called *covariance shift* becomes even more significant while the model becomes deeper due to the accumulated shift over the layers. A new layer called Batch Normalization (BN) was developed to address this problem. BN is applied during the training process after the convolution layers and before non-linear activation unities. The BN-Inception architecture was developed from this new approach, with this architecture, it was possible to achieve the same performance of the previous model with only 7% of the training steps.

InceptionV2 [94] was developed after BN-Inception and introduced the concept of convolution factorisation. This approach meant that large spatial filters such as 7x7 and 5x5, expensive in terms of computation, could be broken down into smaller ones. Examples of the filter factorisation are the replacement of a 7x7 convolution by two layers of 5x5 convolutions with stride 2 or three layers of 3x3 convolutions with stride 1. It can go further replacing 3x3 convolutions by the sum of two of size 3x1 and one 1x3. The figure 2.14(a) shows the Inception module with its 5x5 convolutions factored, in (b) a new factorisation was made on the convolution 3x3. The 7x7 convolution at the beginning of GoogleNet was also factored in this new version. Whole InceptionV2 is 42 layers deep and the computational cost is 2.5 times higher than GoogLeNet.

In this work we will not perform tests with the BN-Inception or InceptionV2 architectures, but with the InceptionV3 model derived from them [94]. This architecture uses layers called BN-auxiliary. They are Batch Normalisation-based layers applied to the fully-connected layers of the auxiliary classifiers.

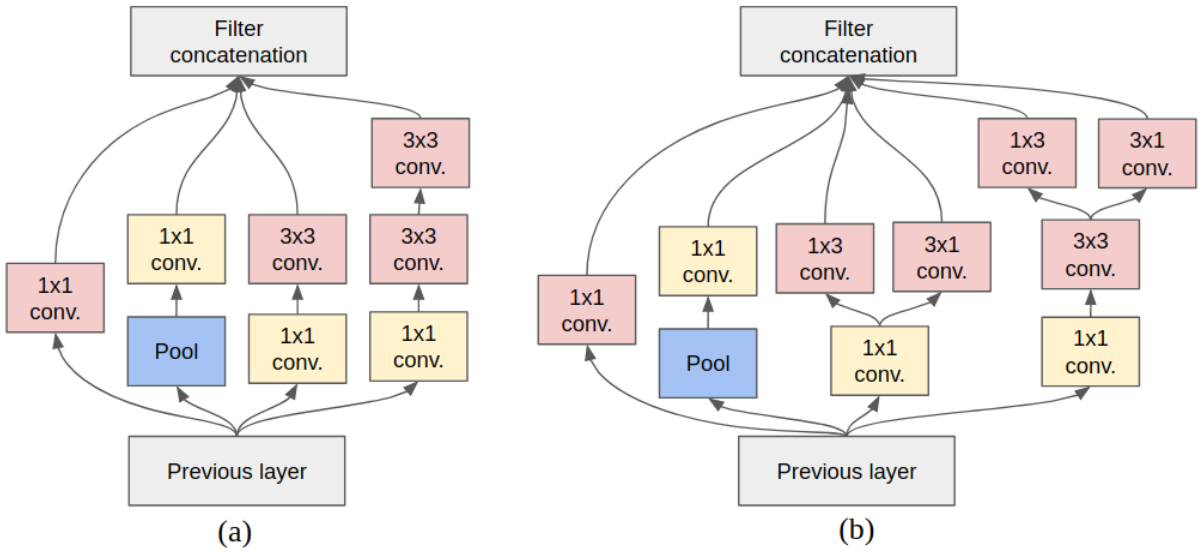


Figure 2.14: Inception modules factorised. In (a) the 5x5 convolutions are divided and in (b) the 3x3 are splitted.

- **ResNet**: previous architectures shows that using more layers improves the results obtained by the model.This statement is correct in parts, because stacking more layers in an architecture can lead to problems such as the difficulty of training the model. Kaiming He et. al. [95] reaffirmed previous tests that argue the degradation of results when the network becomes extremely deep. In their tests, they observed that architectures with 56 layers generated inferior results than the same models with only 20 layers, not only in test sets that could prove a possible overfitting, but also in the training sets. They also argue that deep models should at least have similar results to the shallow ones. This could happen if the first layers of the model acted as a shallow model and the remaining as identity functions (input=output, $F(x) = x$). The residual block was developed by Kaiming He et. al. to promote the creation of these identity functions during training. Being x the input of a layers

sequence $F(x)$ and y the output, they hypothesised that it would be more difficult for a model to learn the mapping $y = x$, having just $F(x)$ as a path than having $F(x)+x$. From this thought they developed the residual block presented in 2.15(b).

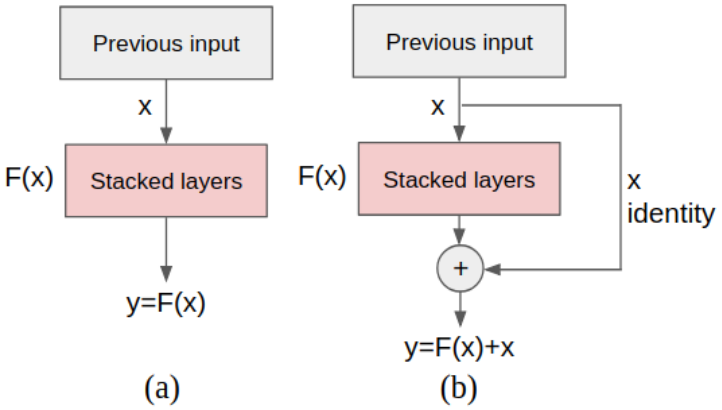


Figure 2.15: Standard plain block (a), residual block (b).

Architectures called ResNet [95] were developed and trained with depths of 50, 101 and 152 layers using stacked residual blocks. These models became known after winning the first place in the task of detection and localisation of objects in images in the competition ILSVRC 2015 and detection and segmentation in the contest Common Objects in Context (COCO) in the same year.

It was later discovered that the ResNet architectures are resistant to layer removal, an action that generates error probabilities close to 1 when applied to sequential architectures such as VGG [96], this is due to the large number of independent paths that are created (Figura 2.16). The ResNet architecture can be interpreted as ensembles of shallower networks [96]. Even with great achievements with these architectures, they still had a downside that was the time necessary for their training. Huang et al. [97] developed the method called *Deep Networks with Stochastic Depth* to mitigate this problem. With this technique, some layers are randomly ignored during training regarding a probability, and all layers are used during the test phase. This method is similar to the Dropout [98] that randomly ignores some neurons during training. Using this approach, it was possible to train ResNet architectures

in less time. In some tests, Huang et al. [97] were able to train architectures with more than 1200 layers.

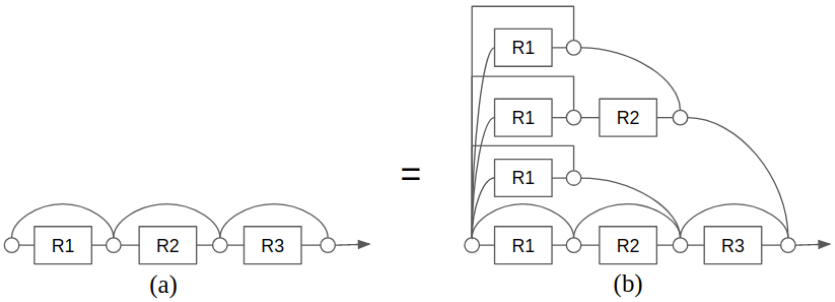


Figure 2.16: (a) Conventional 3-block residual network, (b) Unraveled view of (a).

- **DenseNet**: after studies with ResNets, it was observed that neural networks could be substantially deeper, more accurate and more efficient using shorter connections between layers. Using this knowledge, DenseNet was developed by Huang et al. [99]. DenseNet is an architecture where each layer is connected directly to all its previous layers. The advantages of using such an architecture are the reduction of the vanishing-gradient problem, encouragement of feature reuse, feature propagation increasing and a final classification based on all previous layers. Huang et al. observed that dense connections have a regularising effect, which reduces overfitting on tasks with smaller training set sizes. The figure 2.17 shows the representation of a DenseNet with three dense blocks, the layers between two adjacent blocks are referred to as transition layers and they reduces feature-map sizes via convolution and pooling.

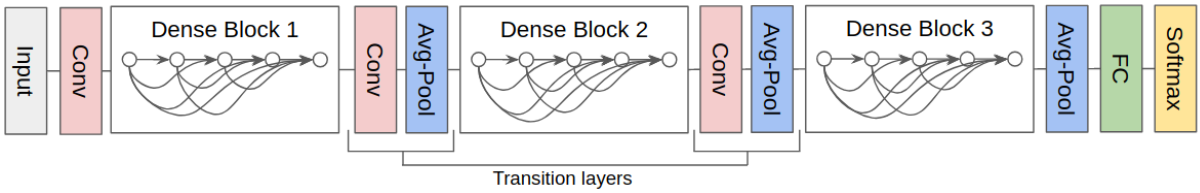


Figure 2.17: DenseNet representation with three dense blocks.

- **Xception**: Architecture developed by the creator of the Keras library, François

Chollet. It has the same number of parameters as InceptionV3, but still manages to surpass it using them in a more efficient way. Comparison tests were performed in the Xception paper using the datasets ImageNet and JFT, (Google's internal dataset with 350 million images and 17,000 classes). These results are possible because of the implementation of depthwise separable convolutions. This kind of convolution, that will be explained later in the MobileNet architecture, proved to be extremely efficient by enabling convolutions to be performed with fewer parameters. With less parameters, deeper and computationally more efficient architectures can be created [100].

- **InceptionResNetV2**: the InceptionResNetV2 [101] architecture was developed using the knowledge gained from the construction of the Inception and ResNet architectures. Although the authors of this paper question the statements that residual connections are inherently necessary for training very deep convolutional models [95], they agree that the use of residual connections improves the convergence time greatly, which is alone a great argument for the use [101]. With 572 layers and using the new Residual Inception Blocks, InceptionResNetV2 achieved better results in a subset of the ImageNet dataset when compared to the previous state of the art InceptionV3 [94], ResNet152 [95] and ResNetV2-200 [102].

- **NASNet**: creating new Deep Learning architectures is typically done by groups of engineers and scientists. This process is laborious because the search space of new architectures is combinatorially large, because of that great amount of time in development and tests is taken by those with experience in the area. To make this process more accessible, Google developed in the year 2017 approaches to automate the design of machine learning models called AutoML. Using evolutionary [103] and reinforcement learning [104] algorithms, a controller proposes an architecture that is trained and evaluated for a given task, the result obtained is informed to the controller that will use it in the next proposals. After executing this process thousands of times the controller develops the ability to assimilate which are the

best areas of the architecture space that creates better architectures.

After a few months, they discovered that AutoML was able to develop small neural networks comparable to the ones developed by human experts, but these results were limited to small academic datasets such as CIFAR-10 [105], and Penn Treebank [106]. Naively applying AutoML to larger datasets like ImageNet would require months of training. Changes were made in AutoML so that it could handle large-scale datasets, including optimising the search space for layers to find those that could be stacked multiple times, and then using the CIFAR-10 dataset to find the best architectures. With this methodology, they found modules that worked well not only in CIFAR-10 but also on ImageNet and the COCO object detection dataset. Among the modules discovered are the *Normal Cell* and the *Reduction Cell* (Figure 2.18) which combined made the NASNet architecture. This process was carried using 500 GPUs for 4 days [107].

Results presented in the NasNet publication says that it achieves the top-1 accuracy of 82.7% and 96.2% top-5 on the ImageNet validation set, which is 1.2% better than the best architectures developed by humans (published), including previous Inception architectures [107]. Comparing NASNet with the best architecture ever published in the web platform *arxiv.org*, SENet [108], it performs similarly but demands 28% less computational resources.

- **MobileNet**: MobileNets are a class of Deep Learning architectures with the goal of being computationally efficient so they can run on devices with fewer resources like smartphones and other types of embedded. The building of these lightweight neural networks is heavily based on the use of depthwise separable convolutions first presented in [109]. Usually, convolution layers apply filters (or kernels) across all image channels and each filter generates a single channel image as output (Figure 2.19(a)). MobileNets architectures also use standard convolutions, but only in their first layer, the remaining use depthwise separable convolutions that are the combination of two operations: depthwise convolution and a pointwise convolution (Figure 2.19(b)).

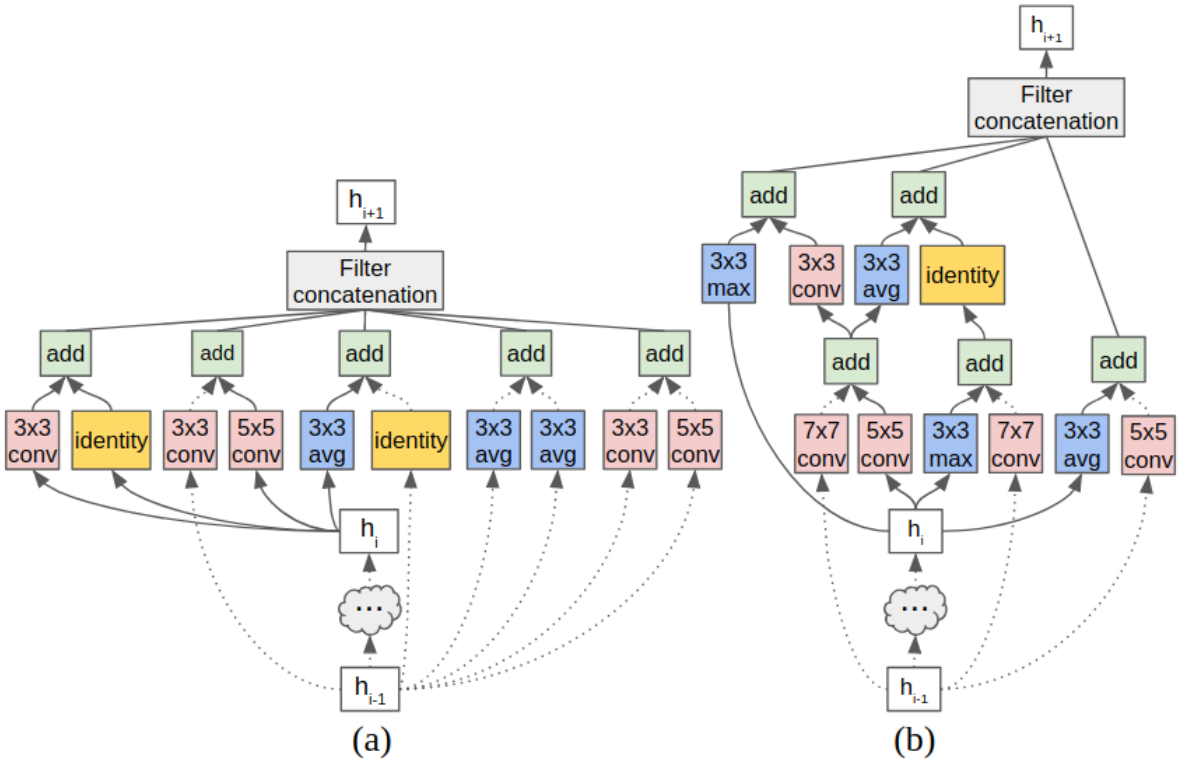


Figure 2.18: Best modules found with AutoML, (a) Normal Cell, (b) Reduction Cell.

These approaches are fairly similar, but a regular convolution needs more computing power to train its additional weights. In tests conducted with 3x3 filters, the depthwise separable convolutions performed about 9 times faster than traditional convolutions [110].

Its next version, MobileNetV2 [111],which will also be tested in this work, was developed on two new ideas, inverted residual blocks, and linear bottlenecks. While normal residual blocks compress the number of channels between the initial and final layer of the block, inverted residual blocks have the characteristic of being narrow-> wide-> narrow, these blocks have skip connections between the narrowest layers of the block (first and last). Linear bottlenecks are layers placed after the last convolution of the residual blocks, they have a linear output to compensate the information lost with compression and the ReLU functions that discard values smaller than zero. This architecture obtained top-1 accuracy 4.1% better than its

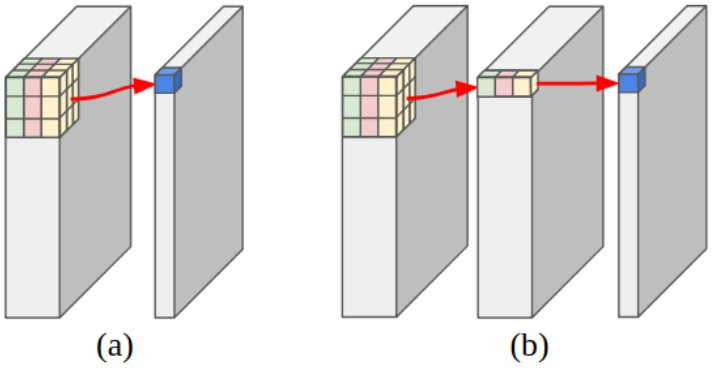


Figure 2.19: Comparison between a standard convolution (a) and a depthwise separable convolution (b).

previous model when trained on ImageNet (74.7%).

2.6.3 Methods to Accelerate CNNs Training and to Improve Generalisation

- **Transfer Learning**: Using this technique, it is possible to transfer the weights of feature extraction layers (e.g., convolutions) from a trained model over a dataset to another model that will be trained in a new dataset [112]. Because this new model received kernels trained to recognise specific features (e.g. lines and curves) it will be easier for it to learn the new dataset. If the new model did not use transfer learning, it would have to learn the features of its dataset from scratch and then learn to classify them, with the technique it is only necessary to learn how to classify the extracted features.

  It is typical for CNN architectures to be trained on huge datasets like ImageNet and then to use features learned during this process to train other models. Because ImageNet has 1,000 classes, it requires the model trained on it to be able to recognise the most various features present in objects, animals, plants and vehicles, for example [113]. This massive number of features learned can be applied not only to images belonging to ImageNet but also to new classes. Using pre-trained models can help accelerate the convergence of the model and enable it to be trained and

achieve excellent results, avoiding overfitting, even with datasets with few examples (hundreds).

- **Data Augmentation**: one way to prevent the model from memorising the training examples (overfitting) and being able to classify examples never seen is to perform the training with many examples (thousands). It is not always possible to obtain large datasets for the training of CNNs, either because of the difficulty in gathering images with the object or because of the difficulty in finding human resources to annotate the datasets. One way to enlarge the dataset to be worked on is by using the Data Augmentation (DA) technique [78]. By using DA, it is possible to create virtual examples from a set of images. Different transformations with random values are applied to these images, and new ones are generated. Examples of these transformations are changes in brightness, contrast, translates, rotations, zoom, and perspective changes. Because these transformations are not computationally intensive, it is possible that these new examples are created in the Central Processing Unit (CPU) while the model is being trained in the Graphical Processing Unit (GPU). Figure 2.20 shows some examples generated from an image using DA.

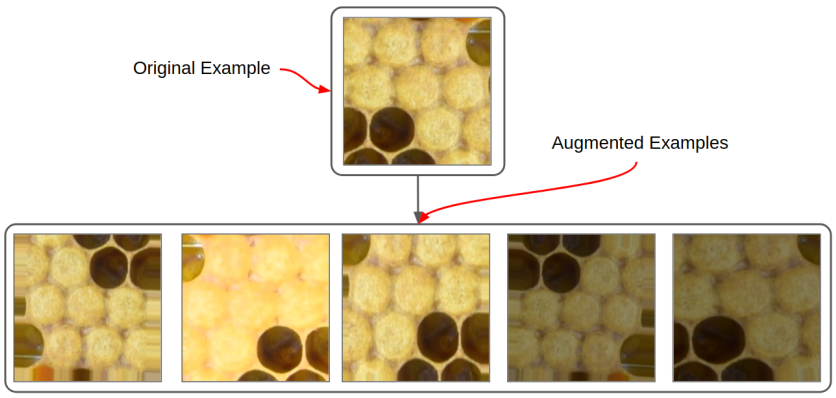


Figure 2.20: Example of Data Augmentation.

2.7 Metrics

The analysis of our results will be performed in a quantitative and qualitative way. The qualitative analyses will be based on the visual comparison of the results. The quantitative analyses will be based on the metrics explain following:

The confusion matrix is a visual way of describing the performance of a classifier in a data set. To use it, the actual values of the set must be known (ground truth). Although it is not a measure of performance as such, many metrics are derived from the values contained in it. Here the concept of confusion matrix will be explained using only two classes (positive and negative), but it can be expanded to problems with multiple classes. A confusion matrix is shown in figure 2.21, following its terms will be explained

- **Confusion matrix:** the confusion matrix is a visual way of describing the performance of a classifier in a data set. To use it, the actual values of the set must be known (ground truth). Although it is not a measure of performance as such, many metrics are derived from the values contained in it. Here the concept of confusion matrix will be explained using only two classes (positive and negative), but it can be expanded to problems with multiple classes. A confusion matrix is shown in figure 2.21, following its terms will be explained.

	Ground truth: Positive	**Ground truth: Negative**
Predicted: Positive	TP	FP
Predicted: Negative	FN	TN

Figure 2.21: Confusion matrix with two classes.

True Positive (TP): they are cases where the positive examples are predicted by the classified as positive.

False Positive (FP): false positive cases happen when an example should be classified as negative, but the classifier makes a mistake and predicts it as positive.

False Negative (FN): false negative cases occur when one or more examples that should be classified as positive are classified as negative.

True Negative (TN): true Negatives happen when negative class examples are classified as negative.

- **Accuracy:** in classification problems, the accuracy measures how many times the classifier has correctly predicted what it should predict regardless of the class. Accuracy is calculated with equation 2.12.

$$Accuracy = \frac{TP + FP}{TP + FP + TN + FN} \tag{2.12}$$

It is not prudent to use this measure in unbalanced datasets since the majority class will have a significant influence on the final result.

- **Precision:** this measure describes how many of the examples classified as positive were positive. It is calculated by equation 2.13.

$$Precision = \frac{TP}{TP + FP} \tag{2.13}$$

Precision gives us information about the model's performance related to false positives. Its results are appropriate in cases where the occurrence of false positives should be minimised. The disadvantage of using only this measure is that if the classifier captures only one positive example within several and classifies it correctly, the precision will be 100%.

- **Recall:** this measure describes from all positive samples how many were classified as positive. It is calculated by equation 2.15.

$$Recall = \frac{TP}{TP + FN} \tag{2.14}$$

This metric can be used when the objective is to minimise the number of false negatives.

- **Specificity:** this measure tells us the proportion of negative examples that were correctly classified as negative, so this measure is the opposite of the precision. Equation 2.15 presents the calculation for the specificity.

$$Specificity = \frac{TN}{TN + FP} \tag{2.15}$$

Because the measure of specificity is the opposite of the recall they have a similar drawback, if the classifier labels all the examples as negative the result of the specificity will be 100%.

- **F1-Score:** an overview of classifier performance can be obtained from the combination of analyses made with precision and recall measurements, one way to perform this combination is by using the F1-Score measurement. It is the harmonic mean of the precision and recall measurements and is calculated using the equation 2.16.

$$F1Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{2.16}$$

Unlike a simple mean between precision and recall, the F1-Score tends to be closer to the lowest value, thus giving the model a more appropriate result.

- **Loss:** this measure calculates the error between a set of predictions made by a model and its actual values. Loss functions are essential for the training of neural networks because their result is used to adjust the model weights. As the parameters are adjusted, smaller and smaller the loss results must be generated by the loss function. In equation 2.17 the loss function Categorical Cross-Entropy (CCE) is presented and in equation 2.18 the Mean squared error (MSE). In both equations N represents the number of samples being analysed, $\hat{Y}$ and Y respectively represent the

sets that have all the predictions and the ground truth. Finally, $\hat{y}_i$ and y_i represent an input result predicted and the ground truth of this input, respectively.

$$CCE(\hat{Y}, Y) = -\sum_{i=0}^{N} y_i \ln \hat{y}_i \tag{2.17}$$

$$MSE(\hat{Y}, Y) = \frac{1}{N} \sum_{i=0}^{N} (y_i, \hat{y}_i)^2 \tag{2.18}$$

- **Intersection over Union (IoU):** IoU is a measure used to calculate the quality of detection or segmentation. Object detection competitions such as the PASCAL Visual Object Classes Challenge [114] and the 2018 Data Science Bowl[3] use this metric to evaluate the results of their competitors. Its calculation is based on the area of intersection between the annotated and predicted segmentation, divided by the area of the prediction plus the annotation. Equation 2.19 shows the calculation, where A represents the predicted area and B the annotated area.

$$IoU(A, B) = \frac{A \cap B}{A \cup B} \tag{2.19}$$

[3]https://www.kaggle.com/c/data-science-bowl-2018

Chapter 3

Image Acquisition and Cells Detection

This chapter is subdivided as follows: In section 3.1 we will present the setup developed to obtain the frames images, data about the location of the captures and the number of images we got. In section 3.2 will be presented the first approaches developed, trying to enhance and detect the cells. These approaches were the basis for the our detection method. In section 3.3 are shown steps taken to develop our detector. In section 3.4 we demonstrate how to make the detections scale invariant. Finally, in section 3.6 we present ways to optimise the detector with different OpenCV versions and using GPUs.

3.1 Image Capture Setup

To guarantee the image's capture standardisation, we developed a wooden tunnel sealed for external light. Using it, all the frames could be photographed at the same distance and under the same lighting conditions. With a retractable structure, the tunnel measures 247cm when fully opened and 92cm when retracted, we choose this format to make the transportation easier. As shown in figure 3.1, the frames are placed in holders inside the tunnel, the holders have an angulation of 45 degrees, it helps to capture the cell interior. In the figure are also highlighted pair of Light-emitting diode (LED), they were placed

40cm from the end of the tunnel and were turned to the tunnel's sides at 45 degrees to provide a homogeneous illumination and to prevent shadows. The LEDs used had 7 Watts of power, the LED model chosen was the *Bestlight Phantom PT-C 204s*. More information about the tunnel building can be found in appendix B. In the figures 3.2(a) and 3.2(b) we show the tunnel being used.

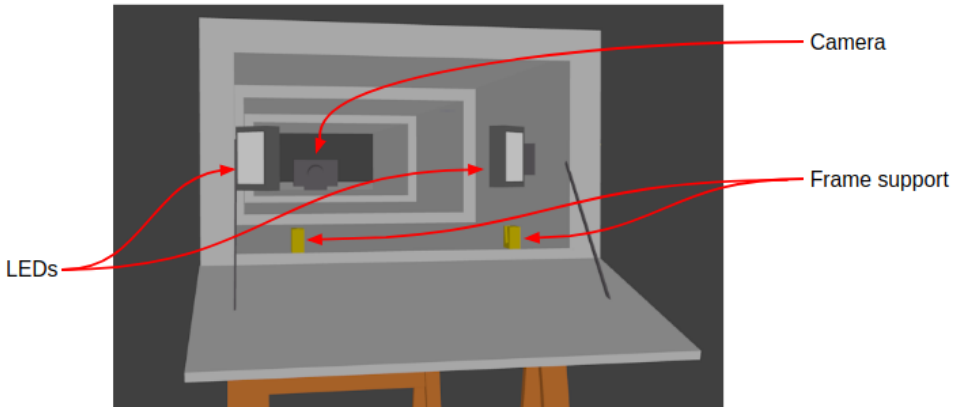


Figure 3.1: Details of the tunnel

(a) (b)

Figure 3.2: (a) Frame placed on holder before being photographed, (b) researcher adjusting the camera before shooting.

This tunnel was used to photograph in two apiaries in Portugal, one in Bragança city and another in the Algarve region. The photos were taken with a digital camera *Nikon D3300*, the lens used was an *AF-S DX VR Zoom-Nikkor ED 55-200mm F4-5.6G*. With this equipment we captured images with a resolution of 24MPixels (6000x4000px). The settings used were: Aperture, 10; ISO: 100; shutter speed: 1/60; auto focus: on; flash:

no; compression: JPEG and white balance: on. During the captures the tunnel was completely closed, we used an external trigger to activate the camera.

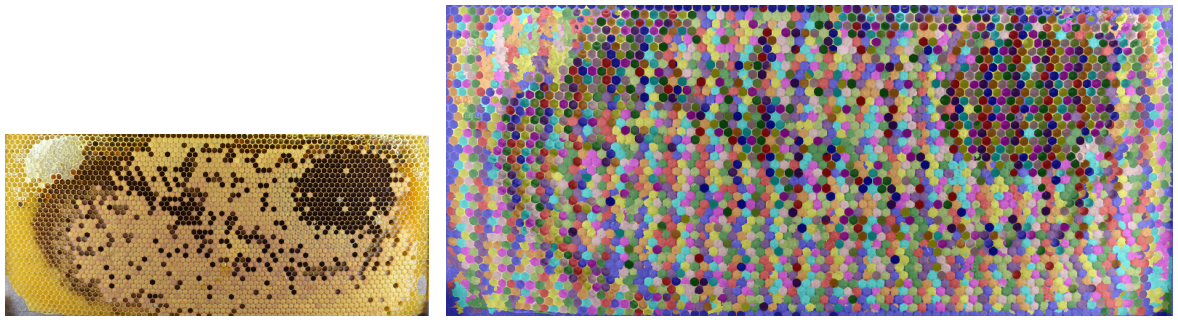
Altogether 1102 frames were photographed on both sides, creating 2204 images. With them, we created a dataset, it will be referenced in this work as DS-COMB-PT.

3.2 First Approaches to Cells Enhancement and Detection

In this section we will present the methodologies that we used in our first tests, they were centred on the exploration of methods to enhance features and to detect cells.

3.2.1 Watershed Transform for Segmentation

It is common for segmentation algorithms to have as their starting point the use of the watershed transform due to the quality of its results. With the implementation made by the scikit-image[1] library it is possible to clustered pixels that are similar and close together based on a predefined number of anchor points. This algorithm was one of our first approaches tried. Figure 3.3(b) shows the result obtained by applying this method on a comb image with dimensions 1500×640px (Figure 3.3(a)).



(a) (b)

Figure 3.3: (a) Frame image before applying the watershed transform. (b) Frame processed by the watershed transform.

[1] http://scikit-image.org/docs/stable/api/skimage.morphology.html#skimage.morphology.watershed

Before applying the watershed transform, we applied a Sobel filter to enhance the boundaries between the cells. We used the Sobel filter implemented in the library scikit-image[2]. We defined the number of anchor points as 6000 expecting each cell to be fitted in a cluster. After applying the transform, is returned a 2D matrix, where positions with the same value represents the same region. Results obtained by this approach are discussed in section 5.1.

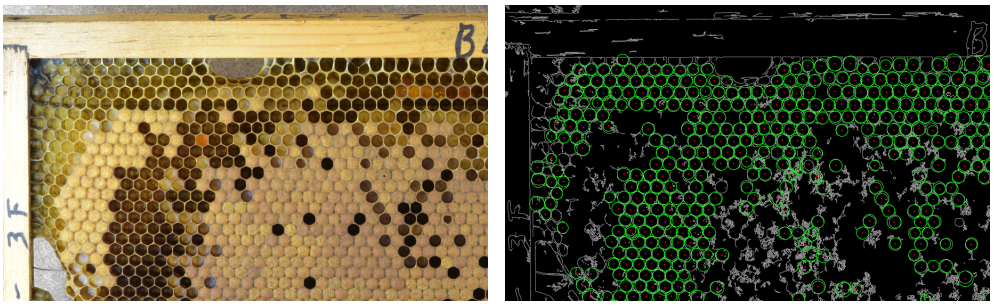
3.2.2 Circle Hough Transform for Cells Detection and Prediction

The first cell detection approach with interesting results we developed was using the cHT. In our first tests, we had in mind that it would be necessary to apply some filter to enhance the cells contour, after some tests with filters such as Sobel, Canny, and Laplace, we observed that the results obtained with the Canny filter were visually satisfactory. Up to this point, we had not yet noticed that the Canny method is implemented with the OpenCV cHT method and that the cHT parameter *param1* represented the Canny *maxVal*.

Therefore, in our first tests we used the Canny as a pre-processing method before detecting the cells with the method *cv2.HoughCircles()*. As can be seen in figures 3.4(a) and 3.4(b), at this time it was possible to find some uncapped cells, but there were still large areas without detections, most of them regions with capped cells and honey. Trying to cover these areas, we have developed methods to predict the position of cells based on some markers.

The markers were created from the detections made by the Hough Transform. They are formed by 7 points, the criterion chosen to define good markers was the six external points being equidistant from the central point, or the closest to it. Firstly the 5 best markers were selected, using them lines were drawn, we expected finding undetected cell in their intersections (Figure 3.5).

[2]http://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.sobel



(a) (b)

Figure 3.4: (a) Frame image before applying the Canny filter and the cHT, (b) Frame image processed by the Canny filter and the cHT.

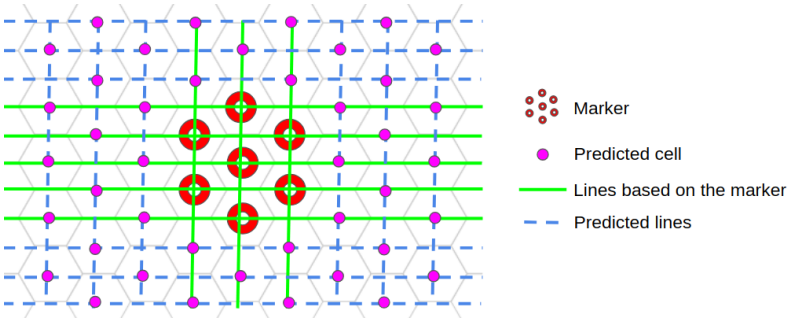
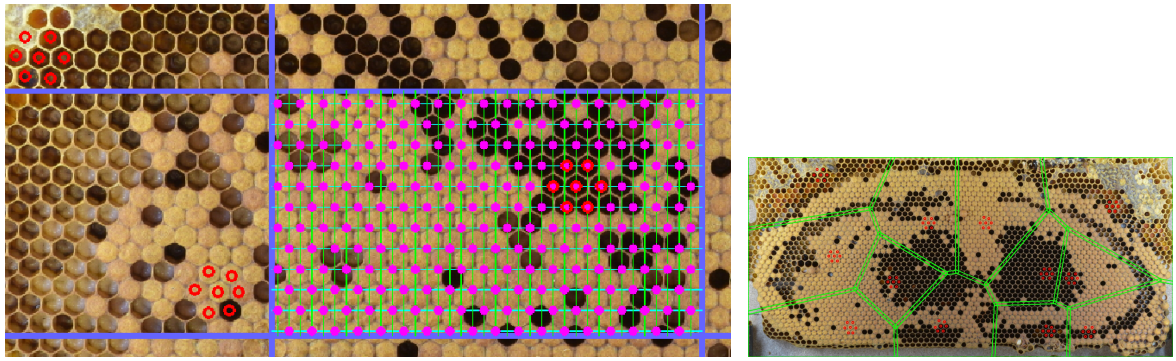


Figure 3.5: Prediction of cells positions using a marker.

Selecting the top five markers, in many cases, made them concentrated in a small region of the image. In this case, cells farther away from the markers were predicted with less precision. To mitigate this problem, we subdivided the frame into 12 squares and we defined that should have at least one marker in each square, this marker would be responsible for the predictions of the cells inside its region (Figure 3.6(a)).

Distortions caused by the curvature of the camera lens and cells with varying sizes caused the quality of the predictions to drop according to the distance between the predicted cell and its marker. Although using the grid made the detections more localised, it did not guarantee that the cells would have their positions predicted by their nearest marker. Then we used the Scipy library and we created a Voronoi diagram where it would be guaranteed that each marker would only be in charge of predicting the position of the nearest cells. In this approach, regions would be created between the marker areas, predictions made in this region would be removed if they were close to other prediction

(Figure 3.6(b)).



(a) (b)

Figure 3.6: (a) Cells prediction based on a marker. (b) Voronoi diagram created to ensure that each cell would be predicted by the closest marker.

Further developments of this approach have been interrupted because a new approach developed in section 3.3 was generating better results and had a simpler process.

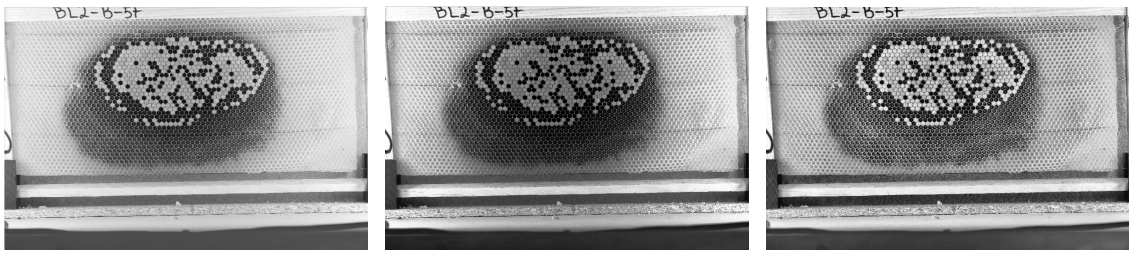
3.3 Cells Detection Improved

The results obtained using cHT so far were not satisfactory, but after studying the work made by Lee et al. [19] we had new ideas on how we could tackle this detection problem. Lee et al. developed a process with the following steps: crop manually the image to remove unnecessary parts such as the frame structure; converting the image to grayscale, in this step the RGB channels of the image are multiplied by scalars and then summed by the following formula $grayImage = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B$ [115]; Canny edge detection and finally, the detection of cells using cHT. The process developed in this work also suffered from no detections in areas with capped cells, but reproducing it was a good starting point to develop our method presented below.

While we were experiencing ways to convert the image to grayscale we did some tests with separate channels and mixing them using weighted formulas. As result, we observed that using only the Red channel we could achieve a greater contrast between the center and the edge of the cells. Thus, filtering only the Red channel was defined as the first

step to our method.

Then, we come across a peculiarity of our problem. During the honeycomb building process the bees starts at the center, which causes the cells in this area to be darker than the ones in the borders. This colour difference weaken the Canny detections, to mitigate this wider Canny thresholds can be used. Wider thresholds have a drawback, because they increase the number of false edge detections. One solution for this problem is to equalise the image histogram before applying Canny, is expected from this method to balance the frame border and center colours. We found that using the standard OpenCV histogram equalisation method, in some cases, accentuated the contrast between the border and the central area, because the equalisation is based on the entire histogram. To distribute the lighting more evenly over the image we used the CLAHE method present in the library OpenCV[3]. Using CLAHE was possible to perform a localised histogram equalisation. In this method we defined the tile size as 8×8px, the *clipLimit* used was 2.0. Figure 3.7 compares an image with only the filtered Red channel (a) processed by the global HE (b) and by the CLAHE (c). Processing the image with the CLAHE was the second step defined in our method.



(a) (b) (c)

Figure 3.7: (a) Image with only the Red channel, (b) Histogram Equalisation applied in the Red channel, (c) CLAHE applied in the Red channel.

Although the equalisation made by the CLAHE method improved the image illumination quality, it has a drawback which was the noise increasing, the noise can be controlled by the parameter *clipLimit*, but even tuning it there is still a considerable amount of noise after processing. Noise reduction can also be performed using filters such as the average

[3]https://docs.opencv.org/3.3.1/d6/db6/classcv_1_1CLAHE.html

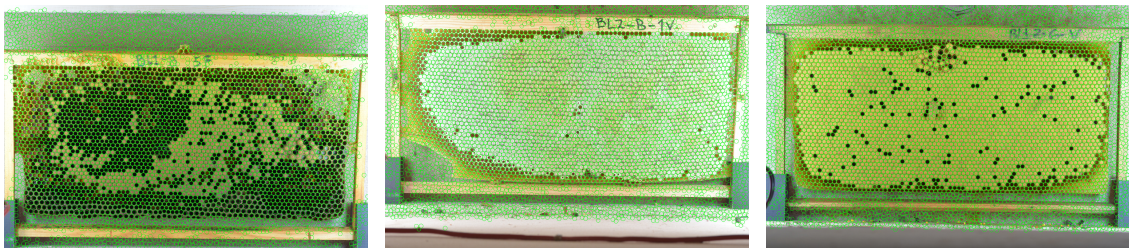
and the Gaussian. These filters are effective in many tasks, but the way they calculate the new value of each pixel based on its neighbourhood causes a blurring effect on edges, this effect is undesirable in our problem because information about the cells edges are important for their detections. Rodrigues et al. uses the bilateral filter in their work [34]. This filter is highly effective in noise reduction and has the characteristics that we need that is to keep the edges sharp. As a negative point, this method is slower compared to the other cited, this drawback was not important in our scenario. Therefore, we define the bilateral filter as the third step of our method. The parameters we choose were d (diameter of the neighbourhood to be analysed): 5, *sigmaColor*: 50, *sigmaSpace*: 50.

With the preprocessing methods defined, we seek the best parameters for the cHT. The parameters expected are: *image*: image in a grayscale; *dp*: size of the accumulator that will store intermediate cHT results; *minDist*: distance that must be kept between the center of two detections; *param1*: upper threshold to be applied to the internal Canny (the lower threshold is defined as twice smaller than param1); *param2*: number of votes that a circle must have in the accumulator to be set as true. Small values in this parameter may cause false detections; *minRadius*: minimum circle radius; *maxRadius*: maximum circle radius.

Finding optimal parameters manually would be a laborious process due to the number of possible combinations. Thus, we developed a grid search based algorithm to accomplish this task. At this moment we were using the OpenCV version 3.3.1 and in some cases, the cHT method took more than 30 seconds to process an image of size 6000x4000, because of this constraint it was necessary for us to develop heuristics and to select good possible parameter combinations before trying them in an image. The parameters that we used in the search and the values used in the combinations are described in the following: *dp* [2, 3], *minDist* [50, 51...65], *param1* [80, 82...120] (step 2) and *param2* [10, 11...40]. According to preliminary tests we kept the parameters *minRadis* and *maxRadius* fixed in 31 and 37 respectively. To measure the quality of each set of parameters we manually created three annotations drawing points in the centers of each cell in three comb frame images. After, we extracted the annotations and place them in a black image. At each

iteration of the algorithm a new set of parameters is generated and used by the cHT. The detected points are drawn in another black image. Then, we performed an *and* operation between the annotation and the detection. The result of this operation is an image with nonzero pixels only in positions with nonzero pixels in both images in the same place. In the end, we counted the number nonzero pixels and these results were stored in an output file. The idea behind this method was that resultant images with more nonzero pixels would came from input images with detections in similar positions.

We found analysing the output file that the best combination for our images was *dp*: 3, *param1*: 100 and *param2*: 25. Examples of detection made with these parameters can be seen in figure 3.8. The Hough transform was defined as the fourth step of our cell detection method. The chosen parameters have a drawback, they increase the number of false detections by allowing more detections to be accepted as a circle. But they also had a positive side, it predicts cells on areas with capped brood and honey even when is difficult to see the cells edges.



(a) (b) (c)

Figure 3.8: Cells detected in hive frame images using the method presented in this section.

3.4 Making the Detection Scale Invariant

Applying the solution presented in section 3.3 we can detect the cells in the hive frame images from the dataset DS-COMB-PT, because they were photographed with a standardised distance and most of their cells had a radius ranging from 31 to 37 pixels. Increasing the cell search range for different radius sizes can generate false or overlapped detections

(a) (b)

Figure 3.9: (a, b) Images from the dataset DS-COMB-CREA.

because of the lack of control we would have over the minimum distance between two detections. To generalise our method, we needed to make it able to detect cells in images photographed without distance standardisation. As example, we have two hive frames in figure 3.9, these frames were photographed using a smartphone camera by a team of researchers from the Italian governmental organisation *CREA*. Along with another images taken from them we created a dataset called DS-COMB-CREA. Although the frames in figure 3.9 have the same image size (3840x2160), the radius of their cells varies being approximately 13px in (a) and 18px in (b).

The process we developed to deal with cells detection invariant to scale is defined in four steps:

1. **Detect cells with radius belonging to different ranges**: in our method we defined that cells radius between 6 and 50px would be detected. In this first step we only look for detections made with high confidence by the cHT method (cells found even with cHT parameters chosen to restrict false detections). The cHT parameters we use in this step are *dp*: 2, *param1*: 145, *param2*: 55 and *minDist*: 12. In algorithm 1 we show that this step look for cells with radius ranging from 5 to 50 with a step of 5. After running this method it is returned a list of detections made for each radius, figure 3.10 shows a processed image, in figure 3.11 is shown the amount of detections made for each radius. More examples like the one shown in figure are presented in appendix D.

Algorithm 1 Find cells with different radius

```
1: procedure findCells(image)                          ▷ Preprocessed image
2:     allDetections ← [ ]
3:     i ← 5
4:     dp ← 2
5:     minDist ← 12
6:     param1 ← 145
7:     param2 ← 55
8:     while i <= 50 do
9:         minRadius ← i + 1
10:        maxRadius ← i + 5
11:        detec ← cHT(image, dp, minDist, param1, param2, minRadius, maxRadius)
12:        allDetections ← allDetections + detec
13:        i ← i + 5
14:    end while
15:    return allDetections                    ▷ Detections made in all ranges
16: end procedure
```

2. **Find the most frequent radius**: from the detections made on the image with different radius, we select the one which the radius repeats the most. It will be used for a second detection made with the parameters obtained in section 3.3.

3. **Seek what should be the minimum distance between two detections**: due to detections with different sizes and imperfections during honeycomb construction, it is necessary to choose values for the cHT *minDist* parameter smaller than $2 \times radius$. In images from the dataset DS-COMB-PT, on average, the cells radius is 35px and we use 55px for the *minDist* parameter. Using this values in images with smaller cells, as figure 3.12(a) can cause the detections to overlap. Figure 3.12(a) has low resolution and on average its cells have 9px radius. To obtain the result shown in figure 3.12(b) we tuned the parameter *minDist* until we find value of 12px.

   To automate the *minDist* calculation, we assumed it has a linear correlation with the radius. Considering that a 35px radius image has *minDist* as 55px and an image with 9px has 12px of distance between detections, it is possible to model an equation to find out which should be the minimum distance given a radius. Using the data cited, we modelled the equation 3.1. Solving it we have $x = \frac{43}{26}$ and $y = -\frac{75}{26}$. With

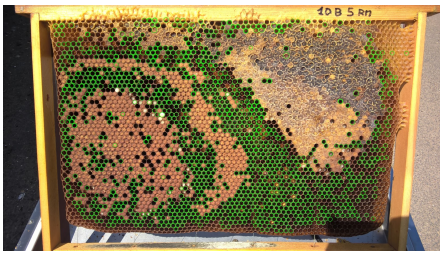


Figure 3.10: Cells detected with high confidence.

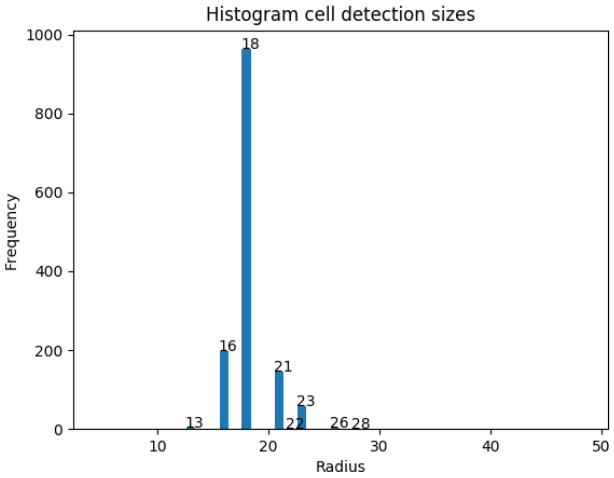
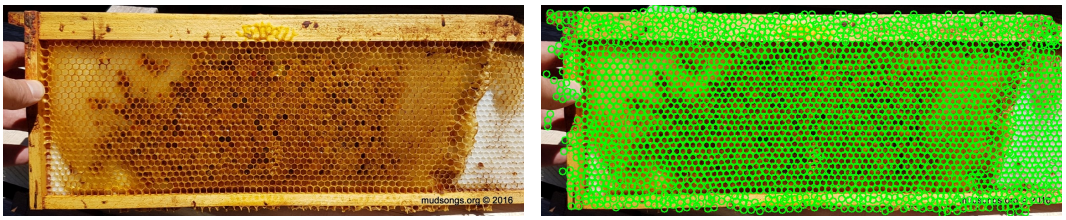


Figure 3.11: Histogram showing how many cells were detected with each radius.



(a) (b)

Figure 3.12: (a) Frame image with low resolution and small cells, (b) detections made in a frame image with small cells [116].

these values it is possible to create the equation 3.2, it is capable of defining the *minDist* given a radius r. Having as root ≈ 1.744, it fulfills our purpose that is to detect cells with a radius ranging from 5 to 50px. A plot generated by this linear function is shown in figure 3.13.

$$\begin{cases} 35x + y = 55 \\ 9x + y = 12 \end{cases} \tag{3.1}$$

$$minDist(r) = r\frac{43}{26} - \frac{75}{26} \tag{3.2}$$

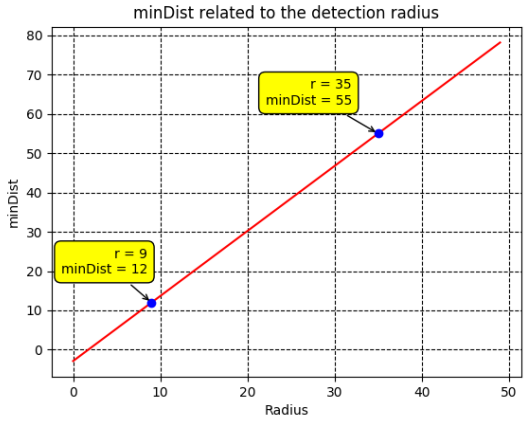


Figure 3.13: Function created to relate a radius to the *minDist* parameter of the cHT method.

4. **Given a radius find the parameters *minRadius* and *maxRadius***: as mentioned in section 3.3, we define that would be sought cells with a radius ranging from 31 to 37px in the images belonging to the dataset DS-COMB-PT. Being 34 the average of these values, we can define a range of $\pm 3px$. Using this range in small cells like the ones in figure 3.12 can produce undesirable results (such as undetections or overlapping). To deal with cells of different radius, we defined that the variation value (in pixels) would be $\pm 10\%$ of the radius found by equation 3.2, we also limit that the variation value would be at least 1 as shown in equation 3.3.

$$range(r) = \pm \begin{cases} 0.1r, & \text{if } 0.1r > 1 \\ 1, & \text{if } 0.1r <= 1 \end{cases} \tag{3.3}$$

5. **perform cHT with the parameters found**: after finding parameters *minDist*, *minRadius* and *maxRadius*, we process the image again with the cHT method, but this time, using the parameters *dp*, *param1* and *param2* found in section 3.3.

3.5 Removing False Detections

As stated in section 3.10, using less restrictive thresholds in cHT contributed to detecting areas with brood capped cells and honey, but as a drawback we had a high rate of false detections. To address this problem we have developed three approaches, we will present them below.

3.5.1 Segmentation based on Hough Lines (SEGHL)

In the dataset DS-COMB-PT, the frame structure is similar in most images. Therefore, in our first approach, we decided to use the Hough Transform once again, but this time for line detection in the frame structure. Before detecting the lines we perform an image preprocessing to highlight the edges, for that we use the adaptive threshold[4] and morphological operations belonging to the OpenCV library. We show in figure 3.14 three steps to preprocess the image. (i) Before we detect the frame structure edges we reduced the problem by resizing the image to 1500×1000px and extracted four areas. (ii) Based on trial and error we choose the following parameters for the adaptive threshold *maxValue*: 255; *adaptiveMethod*: ADAPTIVE_THRESH_GAUSSIAN_C; *thresholdType*: THRESH_BINARY; *blockSize*: 11 and *c*: -2. (iii) In the sub-images corresponding to the top and bottom of the frame, we applied one erosion and one dilation operation with a kernel of shape (1, 49). We perform the same operation on the side images, but this time with a kernel with the shape (33, 1). Both kernel values were found based on empirical tests.

With the preprocessing done, we applied the Hough Transform for line detection implemented in the OpenCV library. For the method HoughLinesP[5] we used the parameters: *rho*: 1; *theta*: $\frac{\pi}{180}$; *threshold*: 100; *maxLineGap*: 50px and *minLineLength*: 120px. After running this method, in some cases, more than one line was obtained, in these cases, we returned only the detection closest to the image center. The figure 3.15 shows a result of

[4]https://docs.opencv.org/3.3.1/d7/d1b/group___imgproc___misc.html
[5]https://docs.opencv.org/3.4.0/dd/d1a/group___imgproc___feature.html

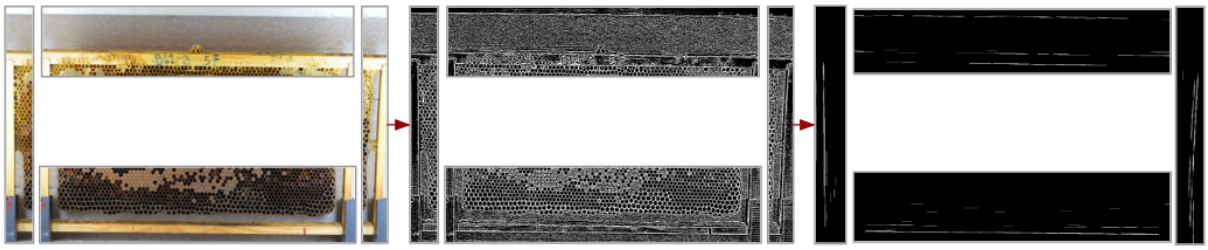


Figure 3.14: Frame enhancement pipeline. First, four regions are extracted from the original image. After, we apply an adaptive threshold to enhance the edges. As the last step, we enhance the frame structure using morphological operations (erosion and dilatation).

this approach. In it, only the detections within the detected area are kept, this reduces the number of false detections.

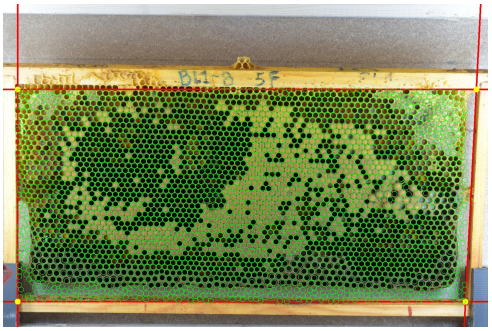


Figure 3.15: Cell detection area reduced to the region inside the comb structure.

We defined the name of this method as SEGHL because it makes the frame segmentation based on the Hough Lines method.

3.5.2 Segmentation Based on Cell Classification (SBOCC)

As will be discussed in chapter 5, the approach presented in section 3.5.1 has some drawbacks, among them, the number of false detections it keeps between the defined limits and the comb region. Therefore, in this second approach, we try to classify each cell detected as being real (cells detected over the comb hive) or not (cells detected in the frame structure and in the background), for that we use CNNs.

For the creation of the classification dataset, we labelled all cells detected in 17 images

contained in the DS-COMB-PT dataset. To assist in annotating the cells we developed a software where we could define the class of each detection using the mouse (Figure 3.16). Altogether 43,441 true and 24,738 false cells were labelled. These annotated cells were extracted from the frame images in three different sizes 42x42, 50x50 and 70x70, figure 3.17 shows examples of them. With these cells extracted we created three new datasets DS-SEG-CELL-42, DS-SEG-CELL-50 e DS-SEG-CELL-70. The goal of creating datasets with different image sizes is to check whether better results are obtained by analysing only the inner cell's contents or whether the external area is relevant as well. We split these datasets in two sets: train: 80%; validation: 20%.

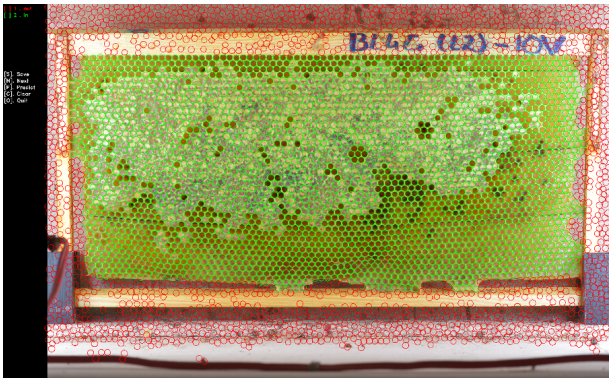


Figure 3.16: Cells labelled as true or false detection.

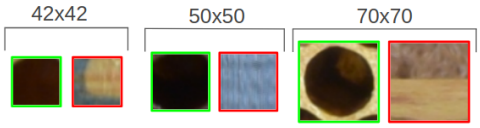


Figure 3.17: Cells extracted with different sizes to create the datasets.

The CNN architecture that we used for the training was the GoogLeNet-Inception[117]. In these experiments, we used the transfer learning technique, so the model only learned the weights of the last fully connected layers. For the feature extraction layers we used the parameters trained on the ImageNet[6] dataset. The training was performed with the aid of the platform NVIDIA DIGITS®[7] v.6. The Deep Learning framework used was the

[6]http://dl.caffe.berkeleyvision.org/
[7]https://developer.nvidia.com/digits

Caffe[8].

The following process was performed in each dataset. For the preprocessing phase, the average image calculated over the training set and then subtracted from each example to carry out the normalisation. The images were then resized to 224x224px (network input size). The model was trained throughout 30 epochs, the learning rate used was 0.01 and it was divided by 10 every 10 epochs. The optimisation was performed using Stochastic gradient descent and the loss was calculated using the MSE metric.

The configurations of the computer used for training are two GPUs: *NVIDIA GeForce GTX 1080 Ti* and *NVIDIA GeForce GTX 1070*; RAM: 16GB; processor: Intel® Core™ i7-7700K CPU @ 4.20GHz × 8; operating system: Ubuntu 17.10. This computer was used in all tests performed in this work.

Because this method removes false detections using classification we named it as *Segmentation Based on Cell Classification (SBOCC)*. Results for tests performed the SBOCC method are presented in section 5.3.2.

3.5.3 Comb Semantic Segmentation (CSS)

In the third approach we use the technique semantic segmentation to remove false detections. We call this method Comb Semantic Segmentation (CSS). Differently from the approach presented in section 3.5.1 the purpose of this approach is to segment the exact area of the comb, not just the content inside of the frame structure. In this method once again we use CNNs, but the result obtained by the processing of an image using this network is another image, different from the solution presented in section 3.5.2 which produced a class as a result (true or false detection).

- The **dataset building** for the training was performed manually. The annotations were created using the *Quick Selection Tool* from the software *Adobe Photoshop® CS6*. For the annotations we painted the comb area as white, while the background and frame structure were defined as black. Altogether 61 pictures belonging to the

[8]http://caffe.berkeleyvision.org/

dataset DS-COMB-PT were labelled, as selection criteria we seek images with a diversity of cells content such as honey and brood in different stages. The comb age also was taken in consideration to create the dataset, because as older it gets, darker the comb becomes, so we selected images to represent as much as possible these changes. Figure 3.18 shows some annotations made. We named this set of images and annotations as DS-COMB-SEG-FULL and split it in three sets: train: 85%; validation: 10% and test: 5%.

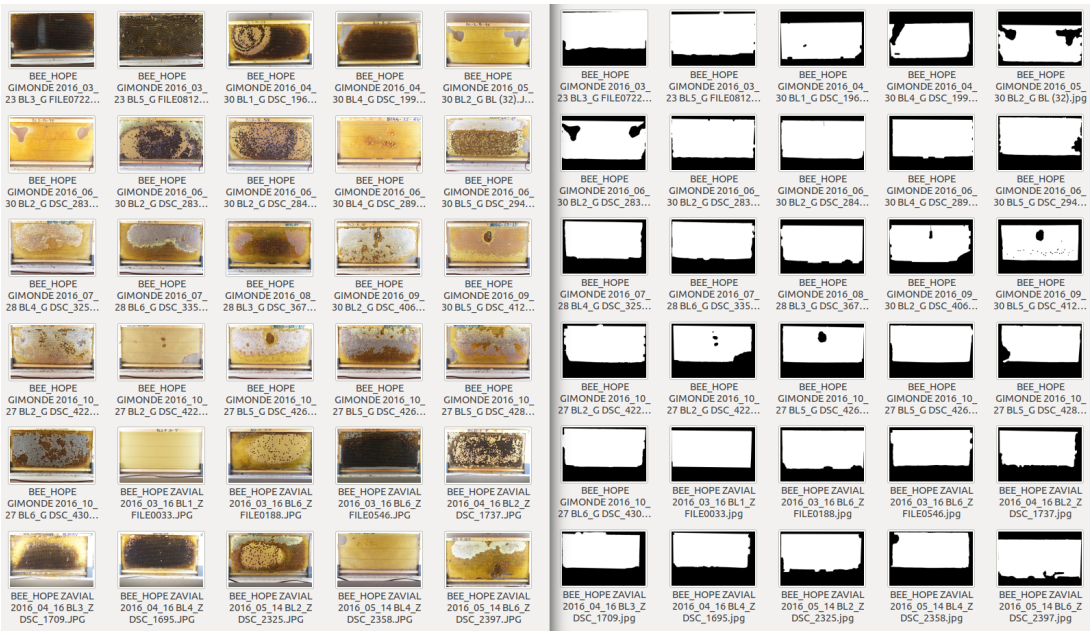


Figure 3.18: Dataset DS-COMB-SEG-FULL created for comb segmentation by neural networks.

Processing an image with this resolution by a CNN is discouraged due to the complexity of the model that would be required. Thus, we used the same approach as Ronneberger et al. [118] where the input image is divided into tiles. These tiles, in its turn, are processed by CNN and then the image is reassembled. Another strategy based on Ronneberger et al. [118] work is the creation of an offset with overlap among the tiles before the extraction, they do it in the input image and in the annotation. This process helps the result image reconstruction and also makes possible to process images of any size, without being limited by the amount of memory

available in the GPU.

Before making the tiles extraction we created a mirrored border in the images with the size top-bottom 184px and left-right 148px, we also reduced the images size by a factor of four. We defined the tile's size as 128x128 with 7px overlap between them as can be seen in figure 3.19. Thus, we extracted 117 tiles from each image, and created the dataset for the training. The total number of tiles extracted was 7137.

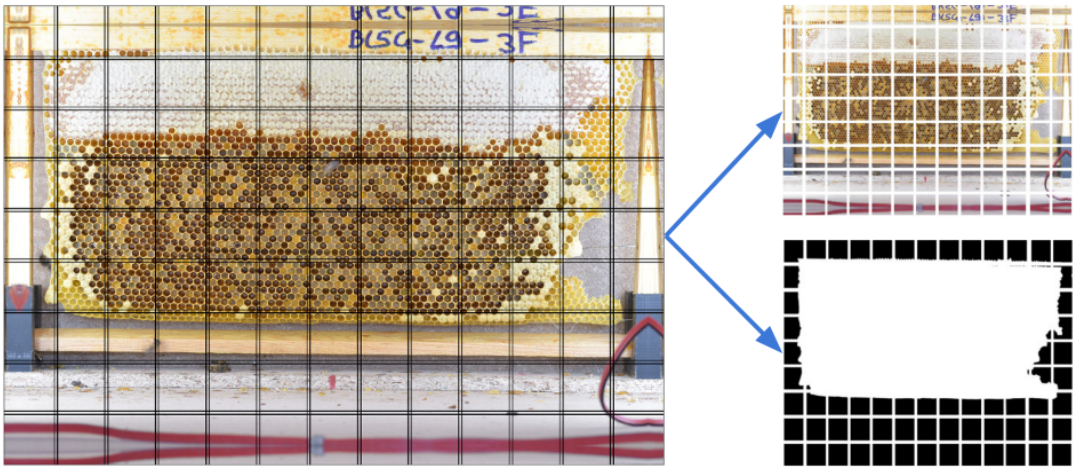


Figure 3.19: Creation of tiles made on the images of the dataset DS-COMB-SEG-FULL to facilitate the training.

- For **the training**, the CNN architecture was based on a model called U-Net [118]. This architecture is similar to a convolutional encoder-decoder. Using this approach, the network is fed by an image, so it is processed by a sequence of convolutions and pooling layers to extract features and after, it is reconstructed by a sequence of upsampling and deconvolutions trying to approximate the result image to the expected output in a supervised training [119]. This architecture has a differential from a simple convolutional encoder-decoder, it has skip connections between the encoder and decoder layers of the same level. This connection gives the network the ability to preserve spatial features, that would otherwise be lost in the encoding process [120].

  Our architecture has the depth of 5 convolutions with 3×3 filters and layers of max

pooling with 2×2 filters and stride of 2, as proposed by Ronneberger et al. [118]. Our modifications to its original model are: input image with 128×128 resolution; use of dropout [98] ranging from 0.1. to 0.3 between the convolution layers; use of Exponential Linear Units (Elu) activation function [121], and we used 16 filters (channels) in the first layer, doubling the amount at each inner level and returning to 16 filters in the penultimate layer, the last layer has only two dimensions. The architecture used is shown in figure 3.20.

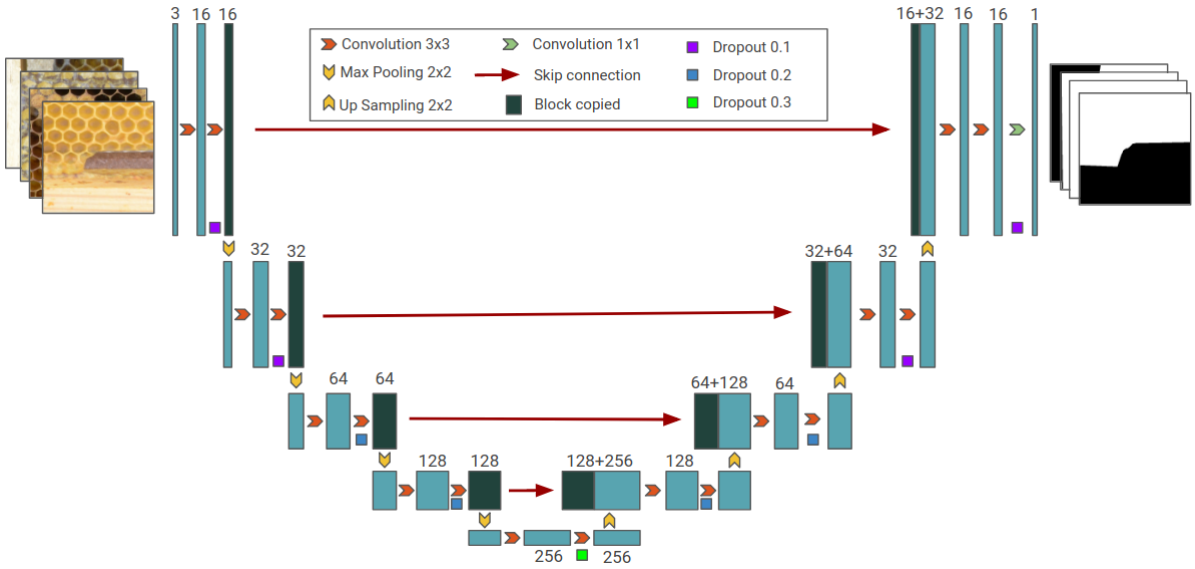


Figure 3.20: CNN architecture developed based on U-Net to handle honeycomb segmentation.

We normalised the input images dividing each pixel by 255. Given that we have only two classes, we used the binary cross-entropy as loss function. The network weights were initialised using the He Normal initialisation [122]. To the output layer we choose a sigmoid function, so we could keep the results in a range and easily transform the output in a binary image applying a threshold. The training was made using the Adam optimiser [123] with parameters $\beta 1 = 0.9$ and $\beta 2 = 0.999$. The learning rate chosen was of 10^{-3}, it was preserved during 50 epochs. The training can last less than 50 epochs due to the Early Stopping method we used, this method can stop the training if a chosen metric not improve after a pre-defined

number of epochs (six). The architecture was built using the framework Keras 2.1.4 with TensorFlow 1.4 as backend.

- A **post-processing** step is defined so that the processed image can be used to remove false detections. Seeking to have a binary output after the CNN computation, a threshold of 0.5 was applied in the result image. With the thresholding performed, the tiles are assembled together using the Python module Numpy. The reconstruction is made using the *vstack* and *hstack* methods, taking into account the offset removal before reassembling the image. With output image obtained, the outer borders are removed. Finally, the image is rescaled to the initial size.

3.5.4 Dataset to Test False Cell Detection Removal Methods

In order to compare our approaches to remove false cell detections in honeycombs, we need annotations that can be used by all three methods. In order to create these annotations, we have selected 10 images from the dataset DS-COMB-PT, different from the images in the DS-COMB-SEG-FULL and create a test dataset. We detected all the cells on these images using the scale-invariant cell detection method. We also manually annotated the honeycomb region in the image just as it was done for the DS-COMB-SEG-FULL dataset. We use the segmented comb areas to filter the detections that have occurred on the background or over the honeycomb. The detections made on the comb were defined as positive and the others as negative. With the cells annotations and the segmentations, we created the dataset DS-SEG-TEST. This dataset has a total of 46.333 cells, of which 28.676 were labelled as positive and 17.657 as negative.

In section 5.3 we present the results of processing the dataset DS-SEG-TEST using the methods SEGHL, SBOCC and CSS. From the results obtained, we created confusion matrices and extracted metrics such as Precision, Recall and F-Score. In this section, we also compare the time required to process an image using each method. Based on these results we will define which approach will integrate our method of detecting cells in honeycombs.

3.6 Tests to be Performed in Different OpenCV Versions

During development of this work different versions of the OpenCV library were developed and in several of them changes in the method *cv2.HoughCircles()* were done. Among the main changes are the parallelization of the method in the version 3.4.0 and optimizations made in the version 3.4.1.

Considering the above points, we developed comparisons between different OpenCV versions using implementations made for CPUs and GPUs in Python and C++ languages. In the tests we used the Python version 3.6. The OpenCV versions we will compare are 3.3.1, 3.4 and 3.4.1. As a result of these comparisons we will analyse the performance related to time, and the detections quality. The results for this comparison are in section 5.4.

Chapter 4

Cells Classification

In this chapter, we describe the work of cell annotation and classifiers training. First, we will specify how the annotation process was performed in section 4.1. Next, we will present a methodology for choosing the best input format for CNNs (Section 4.2) and better architectures for our problem of classifying honeycombs cells (Section 4.3). Lastly, we will define our dataset for conducting tests (Section 4.4) and our methodology to train classifiers using Data Augmentation (Section 4.5).

4.1 Annotations Creation

We developed methods to classify the internal contents of each cell detected using the methodology defined in Chapter 3. These contents were grouped into seven different classes being: Capped Brood, Honey, Pollen, Nectar, Larva, Egg, and Others. In figure 4.1 are shown examples of cells belonging to each class.

The dataset created for training should contain cells of different classes in different comb images. To get these annotations with the quality required, we had the help of an experienced beekeeper. The annotations were made using a software developed by the BEEHOPE team before the beginning of this thesis (Figure 4.2). In this software, it was necessary for the beekeeper to detect the position of the cell in the image and to assign a class to it. The software had some options to adjust brightness, contrast, and gamma in

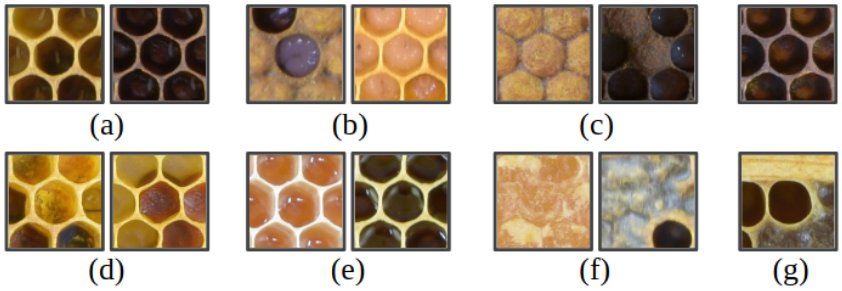


Figure 4.1: Classes: (a) Egg, (b) Larva, (c) Capped, (d) Pollen, (e) Nectar, (f) Honey, (g) Other.

order to facilitate the visualisation of the content in darker cells.

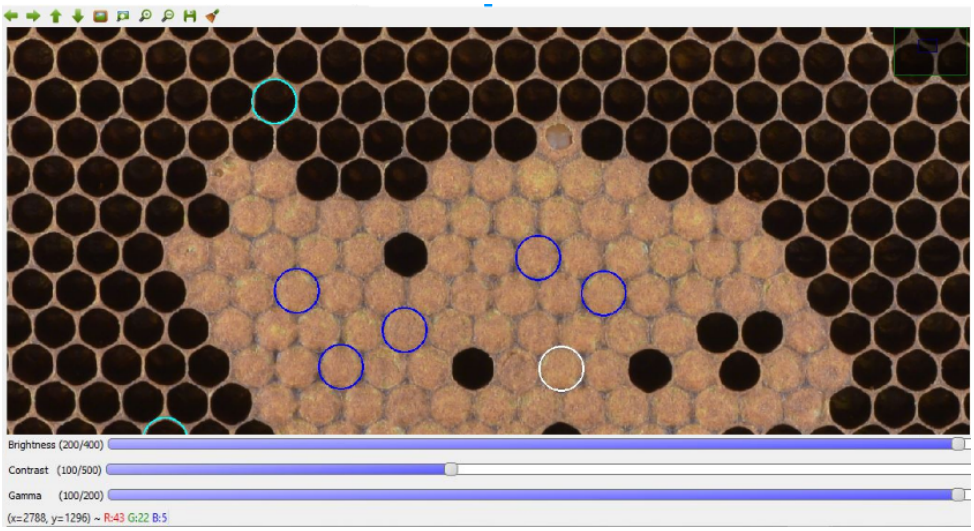


Figure 4.2: Software developed to create cells annotations.

We built the dataset using 1201 frame images belonging to the DS-COMB-PT. The number of cells annotated in each comb ranged from 1 to 375, on average were made 59.8 annotations per comb. In total the beekeeper labelled 71.915 cells, the amount of cells labelled for each class is presented in figure 4.3.

For each image, the software generated an output file with the annotated cells positions. Table 4.1 shows the format in which the annotations were stored. Each annotation is represented with a line in the output file, the line contains the class and four values, these values represent two points on the image, they form a square 50×50px that covers the annotated cell interior (Figure 4.4).

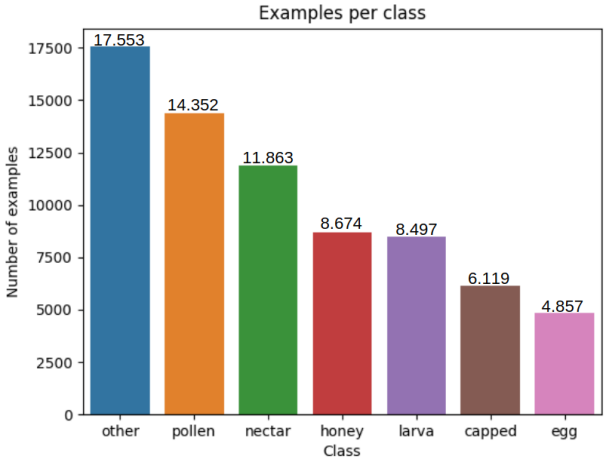


Figure 4.3: Number of examples per class.

Class	P1_x	P1_y	P2_x	P2_y
pollen	1077	1213	1127	1263
pollen	1110	1151	1160	1201
pollen	1144	1089	1194	1139
honey	1033	1156	1083	1206
nectar	1065	1093	1115	1143
larva	962	1156	1012	1206

Table 4.1: Example of an output generated from one image using the annotation software.

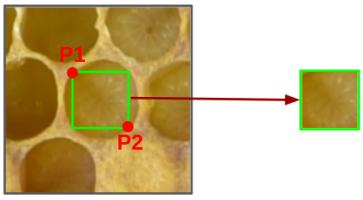


Figure 4.4: Annotations made using two points to represent the ends of a square sized 50×50px.

Some changes were made in the annotation files in order to facilitate their future use. First, we group them into one file. To keep the annotation-image reference we created a new column and stored the source image name. The fixed 50×50 size could make it difficult to us to create future experiments using annotations with different sizes, so we replaced the square outer points by two columns indicating the center of the cell. Until then we were using random names to the extracted images, this caused the reference loss

between the images and the annotations, we solved this problem creating a new column with a unique identifier. Summarising, the new columns created so far would be: a unique identifier, the x, y position of the cell center and the image name. The columns to be removed would be the points that represented the 50×50 rectangle.

During the development of the new annotation file, we identified that the annotations made by the beekeeper did not occur exactly at the same location as the ones detected by cHT. There were a few pixels between them, these differences could make the classifier not reach its full potential when it was in production, because it would be trained on the beekeeper annotations and during the tests it would classify cells detected automatically. To associate each annotated cell with its detected position we developed a script that (i) iterated over each annotated image, (ii) made the detections and (iii) found for each detection which was the closest annotation made by the beekeeper. To compare the distance tens of annotations with thousands of detections efficiently we use the method *spatial.distance.cdist()*[1] it belongs to the Python module Scipy. When we find annotations with a distance bigger than 20px we keep the beekeeper's choice. In figure 4.5 the positions of the annotated and predicted cells are compared. The positions of the detected cells were also placed into the new annotation file for future tests. Table 4.2 shows new annotations examples.

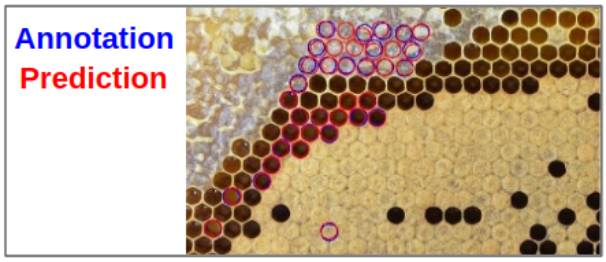


Figure 4.5: Comparison between the positions of the annotated and detected cells.

The annotation file was divided into other files for the creation of the training, test, and validation sets. Three phases were required for the creation of these sets. First, we shuffled the annotations inside the main file. Second, we separate the annotations of each

[1]https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html

Id	X_Annot	Y_Annot	Class	Img_Name	X_Det	Y_Det
64339	5289	1541	pollen	BEE...DSC_2944.JPG	5291	1536
64340	5219	1668	pollen	BEE...DSC_2944.JPG	5219	1671
64341	5252	1730	pollen	BEE...DSC_2944.JPG	5255	1728
4984	2337	2013	larva	BEE...DSC_2945.JPG	2336	2015
4985	2333	2135	capped	BEE...DSC_2945.JPG	2324	2144

Table 4.2: Annotations subset showing the data stored to each cell annotation.

class into seven files. Third, each class file was divided into three others respecting the following percentage of examples: training, 80% of the original set; validation, 20% of the training set and test with 20% of the original set. Thus, we created 7 files to each set. Following are the number of samples per set: training 46.032, validation 11.504 and test 14.379. We named this annotation subdivision as DS-COMB-CELL-PT, using it the classification datasets will be created in the next steps.

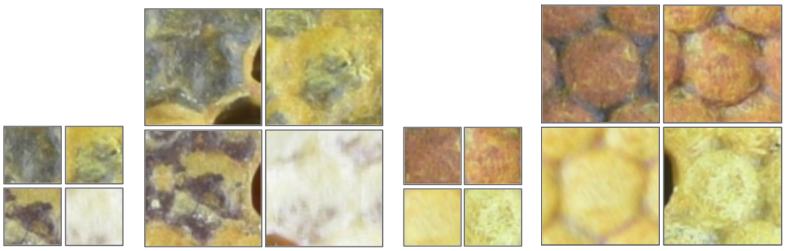
4.2 Defining the Best Annotation Formats

Before we define which would be the CNN architecture for the classifier, we first needed to find the images shape to be used in the training. To explore this question we developed two hypotheses.

The first hypothesis was that the annotations made from detections, rather than positions annotated by the beekeeper, would be better to classify cells outside the training, test and validation sets. Thus, we trained identical models to compare the results obtained when trained on human and automatic annotations.

In our second hypothesis, we tried to know if image cells larger than the cell area would generate better results due to the greater context provided to the CNN. In figure 4.6 an example is presented. In this example is difficult to visually define the cell class when only observing its interior, but when a larger area is observed this task is easy to be performed.

Tests needed to be performed to test both hypotheses, and for that, would be necessary the training of different models. Performing these training with all the images in the



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

Figure 4.6: Comparison between classes Honey (a, b) and Capped (c, d). In this caparison is hard to define the cell class observing just inside of it (a, c) than when is taken into consideration the surroundings of the cell too (b, d).

dataset would make this process time-consuming (two hours per model in preliminary tests). Then, we defined that each training would be done with only 10% of the DS-COMB-CELL-PT images. After reducing the annotation files to 10% of their original size, we begun extracting the cells and creating the datasets for the training.

We developed a script to extract the images of the cells from the annotations. It receives as input a set of annotations, full-size frame image and the size of the cells to be extracted. After, the script goes through all the annotation files, and for each of them loads and group the annotations by the column *image_name*. Following, it loads each image and extract the cells based on the centers annotate and the size chosen previously. In some cases, annotations are made close to the image boundaries, extracting them with large dimensions exceeds the boundaries and errors arise. We avoided this problem by adding borders to the images, as the border size we choose half of the extraction size. To create the borders we used the method *cv2.copyMakeBorder()* with the parameter *border type* defined as *cv2.BORDER_REFLECT*, this method belongs to the OpenCV library. We made the extractions using a multiprocessing pool, we extracted two versions of each cell, one made by the beekeeper and another by the detector (they were saved in different locations). Figure 4.7 shows the process developed for the extraction. The extracted images were placed in folders respecting the structures defined by NN frameworks such

as Caffe[2] and Keras[3] (Figure 4.8).

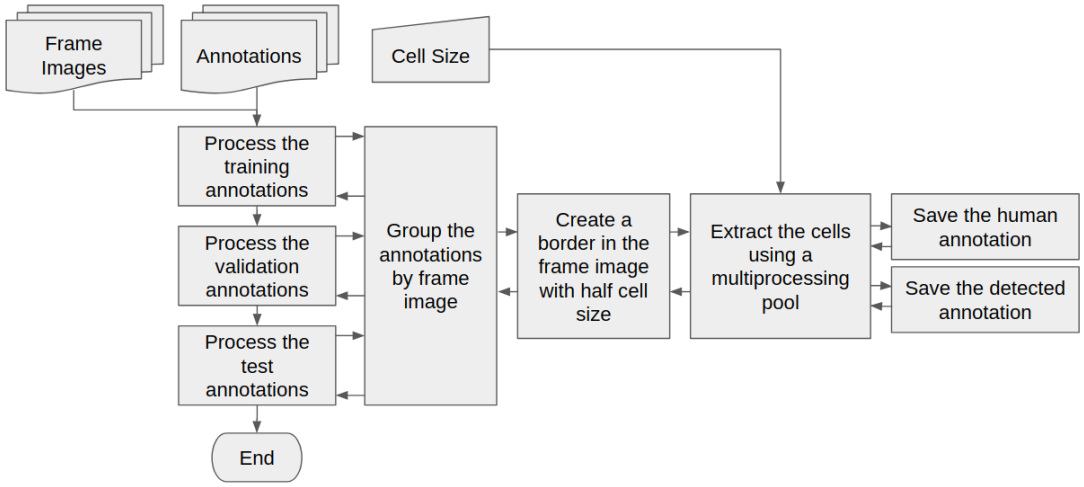


Figure 4.7: First process created to extract the cells from the original comb images.

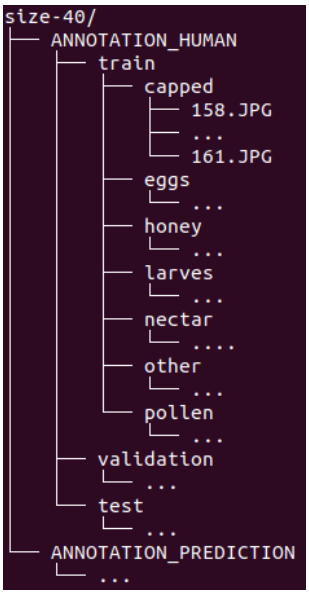


Figure 4.8: Folder structure to the extracted cells.

Using 10% of the annotations contained in the DS-COMB-CELL-PT and the developed scripts we created 34 datasets. Each dataset had the same images, 17 of them used the human annotations and 17 used the automatic annotations. Each one of the 17

[2]https://github.com/NVIDIA/DIGITS/blob/master/docs/ImageFolderFormat.md
[3]https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

datasets had different extraction sizes which were: 40, 50, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240, 256, 280, 298, 400 and 500.

We wanted to know which of these dataset best fitted in our purpose aiming at performance, resources needed and accuracy. To know the best input size and if the annotations made from detections were better than those made by humans we **trained classifiers** on the datasets. In the following we explain the training process.

The training framework chosen for the tests is the Keras version 2.2, it works on top of TensorFlow framework version 1.7, it aims to simplify the neural network architectures building. The architecture chosen is *Inception-v3*, we chose it due to its performance during the training and its good results obtained in image classification competitions such as ILSVRC [94]. For the feature extraction layers we used weights pre-trained on the ImageNet dataset using the transfer learning technique. The architecture and weights trained are provided by Keras library, they can be used calling the constructor *InceptionV3()*.

The constructor *InceptionV3()* requires some parameters, we have modified three of them. The first is the *input_tensor*, this tensor defines the input format of the network. As defined in the Keras[4] documentation, this architecture doesn't allow inputs smaller than 139x139. This is due to the pooling layers, they reduce the input size, if the image becomes very small it can not reach the end of the network. Datasets with images smaller than 139×139 were resized to the minimum input size, the remaining had their sizes kept. Considering *size* as the size (width and height) of the dataset images to be used in training, the parameter *input_tensor* was defined as *Input(shape=(size, size, 3))*, 3 is the number of channels in the input image. The second parameter to be defined was the *weights*, because we want to use the pre-trained weights on the ImageNet dataset, we chose the value *'imagenet'*. The third parameter, *include_top* receives *True* or *False*, by default its value is *True*, when defined like that fully-connected layers are automatically included at the end of the network, we chose *false* to be able to define our own classifier. In addition to the input layer we added three layers at the end of the architecture to create our classifier. The first was a *GlobalAveragePooling2D* to flatten the network output (convert

[4]https://keras.io/applications/#inceptionv3

to one dimension), after we included two *Dense* layers (fully-connected), in the first we reduced the number of neurons from 2048 to 1024 and the second from 1024 to 7, referring to the number of classes we have. The second *Dense* layer have *Softmax* as activation function. Figure 4.9 shows more details about the architecture.

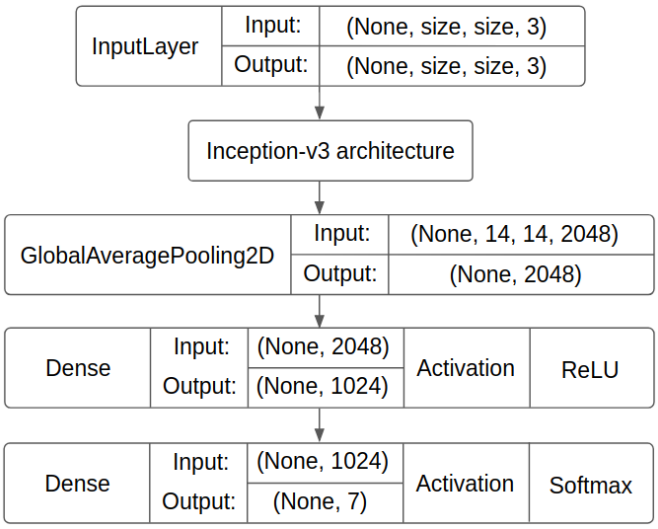


Figure 4.9: Architecture we developed from InceptionV3.

We choose initialisation seeds for both Numpy (*np.random.seed(42)*) and TensorFlow (*set_random_seed(42)*), this was made to ensure that all models had their last layers initialised with the same values. The classifiers were compiled using the *categorical_crossentropy* loss function, we choose Adam with the parameters $\beta 1 = 0.9$ and $\beta 2 = 0.999$ as the optimiser. We used loss and accuracy in the training and validation sets as metrics during training. The learning is guided by the loss, we used the accuracy to know the percentage of correct predictions the model had in each epoch.

The training started with the learning rate of 10^{-3} and we used a policy similar than the one adopted by Kaiming He et. al. [95] to reduce it during the epochs. This policy reduces the learning rate when there is no improvement in a metric chosen after a number of epochs, in Keras, its called *reduce_on_plateau*. We specify that the learning rate would be halved whenever the loss of the validation set did not improve after three epochs, being the minimum value 10^{-6}. We defined the maximum number of epochs as 50 but the total

number of epochs for training can be smaller because of the *Early Stopping* technique used. This technique limited the number of epochs without improvement in 5, after that, the training is aborted. We saved the model with the lowest validation loss in each dataset. Results and discussions on these tests are made in section 6.1.

4.3 Tests with Different CNN Architectures

Using the results obtained by the architectures trained using the section 4.2 methods, it is possible to know which input image size generates better results in our data and if the annotations made from detections produce superior results than those made by the beekeeper. Having this information, we then move on to the next stage where we train different CNN architectures to see which one produces better results on our dataset. We chose 13 distinct architectures to be trained, they were: DenseNet (121, 169 and 201); InceptionResNetV2; InceptionV3; MobileNet; MobileNetV2; NasNet; NasNetMobile; ResNet50; VGG (16 and 19) and Xception.

Each architecture was trained using all training and validation images from the DS-COMB-CELL-PT dataset. The images were extracted using the same extraction method proposed in section 4.2. All models used in these tests have implementations in the Keras module. Before starting each model training, we made some modification in the architectures. We added an input layer with the image shape found using the method of section 4.2, and in the end a layer of *GlobalAveragePooling2D* and two *Dense* layers such as the *InceptionV3* modifications in section 4.2. Before the images being processed by the model they were normalised using the *keras.applications.imagenet_utils.preprocess_input* method. The weights of the convolution and pooling layers were transferred from architectures trained on the ImageNet dataset.

The training was performed with batches of 40 images. We defined the initial learning rate as 10^{-3}, as in section 4.2 the *Early Stopping* and *reduce_on_plateau* methods are used. Their values are the same as those used in the previous section.

During and after training of each model we extracted data for future comparisons.

The data collected are: Model name; Total architecture parameters (weights); Accuracy and loss of each epoch in the training and validation sets; Accuracy and loss calculated in the test set DS-COMB-CELL-TEST defined in section 4.4; Time to process each epoch; Time to perform predictions on a batch of 40 images; F1-Score by class and overall; Recall by class and overall; Precision by class and overall and Confusion Matrix by class. The Confusion Matrix and the F1-Score, Precision, Recall measures were also calculated on the DS-COMB-CELL-TEST dataset. Results for the tests performed in this section are presented in section 6.2.

4.4 Test Dataset

At this time we had only the test set obtained from the DS-COMB-CELL-PT dataset split. In this dataset, just a few cells were labelled on each honeycomb. We wanted to be able to visualise a comb frame with all the cells classified and compare these results with the annotations made by humans. So, we created a new dataset, it has 13 images of honeycomb frames. In the pictures, all the cells were labelled by researchers in apidology.

To facilitate cell annotation, we have developed a software. It is able to detect all honeycomb cells automatically using the method described in section 3.3. Using keys 1-8, the researchers can select different classes (including None), and with the mouse, they can sign labels for each detection. We also put a trackbar to control the image brightness, this adjustment is necessary for viewing the contents of darker cells. Figure 4.10 displays a screenshot of the software.

From these annotations, we get more 39.533 labelled cells. The amount of cells annotated by class is: Capped: 7.138; Honey: 2.943; Nectar: 2.625; Pollen: 2.066; Egg: 2989; Larva: 4.528 and Other: 17.244. These cells added to the 14.381 present in the test split of the DS-COMB-CELL-PT dataset are part of the dataset of test called DS-COMB-CELL-TEST. Results of tests performed using this dataset are presented in chapter 6.

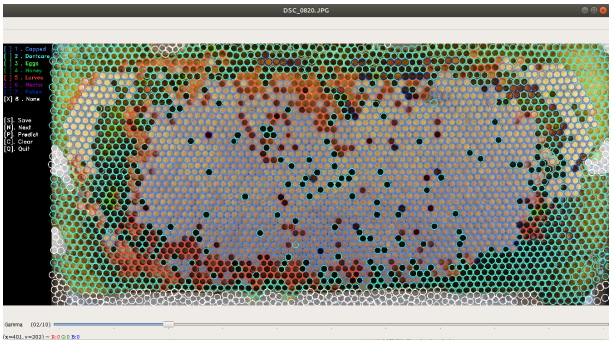


Figure 4.10: Software developed to help cell annotations.

4.5 Applied Data Augmentation in Architectures with Best Results

We decided to compare the results of tests in the DS-COMB-CELL-TEST dataset using models trained only using images from the DS-COMB-CELL-PT dataset with models trained in a larger dataset with more images acquired using DA.

The dataset building with DA was performed using only training images from the DS-COMB-CELL-PT dataset. Transformations were done randomly on the images. The transformations chosen are presented along with the Data Augmentation algorithm in appendix C. As a result, we obtained a training set where each class had between 35.714 and 35.715 samples totalling 250.000 images.

Training different architectures in a dataset with this amount of images could take days, so we trained on this dataset only the models with best results in the DS-COMB-CELL-TEST dataset. The training will be carried out using the same methodology used in the in section. We present the models used and the results obtained in section 6.2.4.

Chapter 5

Results for Cell Detection

In this chapter we will present the results obtained in our cell detection tests. They are divided into preliminary test results 5.1, results of our algorithm that uses the cHT and is scale invariant (Section 5.2), results for the false detections removal (Section 5.3), performance comparison between different OpenCV versions (Section 5.4) and results of the comparison made between our cell detection method and the one proposed in Lee et al. [19] (Section 5.5).

5.1 Results for the Cells Detection in Preliminary Tests

- **Watershed transform for segmentation**: the watershed transform presents modest results for our problem of cell detection. Using it, many cells are segmented individually, but still, a large number of them are grouped in the same cluster (Figure 5.1):

  To continue to use this approach we would need to solve the problem of clustering different cells together, remove clusters smaller than the size of a cell, and look for ways to define whether segmentations were made over a region with honey. As we show below we found better ways to deal with the detection problem, consequently,

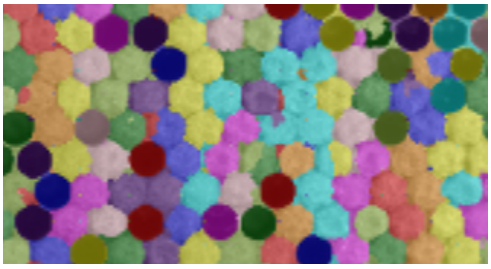


Figure 5.1: Result for the watershed transform for segmentation method.

we did not did further researches using this approach.

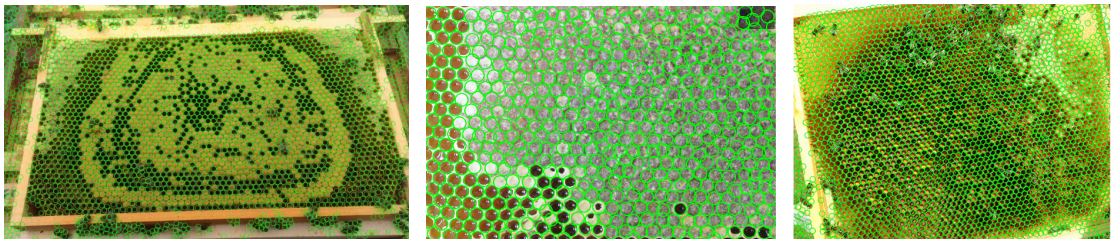
- **First approach Hough transform for cells detection**: before we search for the Hough Transform best parameters, we had in mind that this might not be the best approach to the cell detection problem. What made us think this way was a large number of false negatives in regions with honey and capped cells, based on these problems that we had developed the cell prediction techniques using markers in section 3.2.2. The marker-based detection technique had potential, adding more markers in the comb would produce fewer detection errors associated with marker distance.

  Based on three factors we gave up this approach. First was the complexity. Detecting all cells, creating markers, segmenting them using a Voronoi diagram, predicting cell positions, and removing close detections between regions are many steps. The complexity of this approach has made us look for simpler solutions. Second factor was that we found a simpler approach using only the Hough Transform (Section 3.3) over the pre-processed image. The last factor that made us abandon this approach is its performance, more than 20 seconds were needed to execute it, and it wasn't finished yet.

5.2 Results for the Scale Invariant Cell Detection Method

Using the approach for cell detection presented in section 3.3 it is possible to obtain a high detection rate of cells in all classes, including capped cells and areas with honey. We achieved this result allowing the cHT method to detect cells with less votes in its accumulator, different from the approach proposed by Lee Hung et. al [19]. Our approach generates a greater number of false detections compared to the Lee Hung et. al [19] method, but we work around this negative side by segmenting the comb as explained in section 3.5. Quantitative and qualitative comparisons between both approaches will be carried out in section .

We tested our scale invariant cell detection method with a set of images belonging to the datasets DS-COMB-PT, DS-COMB-CREA and some honeycombs images found on the internet. Among them there are a great variation of lighting, frame format, resolution and size of cells to be detected. Some images are shown in figure 5.2, the remaining images tested are in appendix D.



(a) (b) (c)

Figure 5.2: Detections made with different cells radius. (a) and (b) images have a cell radius of 18px, this value was obtained even with the image distortion in (a) and the amount of honey cells in (b). In (c) our algorithm found 13px as the most frequent cell radius and used it to make the final detection

With these tests, it is possible to observe that our scale invariant detection method works in the most different scenarios, even in images outside our datasets. The process for finding the average size of cells in an image makes 50 searches with radii between

Pre Proc. (s)	Radius Detection (s)	Cells Detection (s)	Total (s)	Im. Area (px)
0,482	1,345	22,348	24,176	24,000,000

Table 5.1: Time for each phase of the scale invariant cell detection algorithm.

12 and 50. The execution time of this process is less than the time spent in the second phase of the algorithm, where the search is based on the most frequent radius found. In the second search, the Hough Transform parameters are less restrictive, this makes the process slower. As shown in table 5.1, on average, to make the process invariant the scale increases by an average of (≈6%) seconds the time for detection, which makes this approach valid to be used in an application outside the academic environment, removing user's need to inform cells size. These tests were performed using OpenCV version 3.3.1.

5.3 False Detections Removal

This phase is of extreme importance for the development of our method, it is able to remove the large number of false detections that occur outside the comb region, these detections are caused by the less restrictive parameters that we use in the Hough Transform, the use of parameters this way also increases the capped cells detection rate.

For the false detections removal, we developed three approaches. The first one, SEGHL (Section 3.5.1), uses the Hough Lines Transform to find the wooden structure and then detects only for cells inside the frame. In the second, SBOCC (Section 3.5.2), a classifier was built to define whether each detection is real or not. In the third approach implemented, CSS (Section 3.5.3), a finer segmentation of the comb is made, again using CNNs, but this time with the semantic segmentation technique rather than the classification. We will present results of each one following, and compare the three approaches in section 5.3.4.

5.3.1 Results for the Segmentation based on Hough Lines (SEGHL)

To measure the quality of our approach SEGHL presented in section 3.5.1 we will use the DS-SEG-TEST annotations. First, we will process the images and compare the results with the ground truth, then we will use the result of this comparison to create a confusion matrix. From the matrix, we will extract the number of TP: true cells inside the segmented area, FP: false cells detected within the comb and the segmentation boundary, FN: true cells outside the segmented area and TN: false cells detected outside the segmented area. The figure 5.3 shows a result image after being processed, in addition, are shown examples of the four classification types. The figure 5.4 presents the confusion matrix generated for this approach.

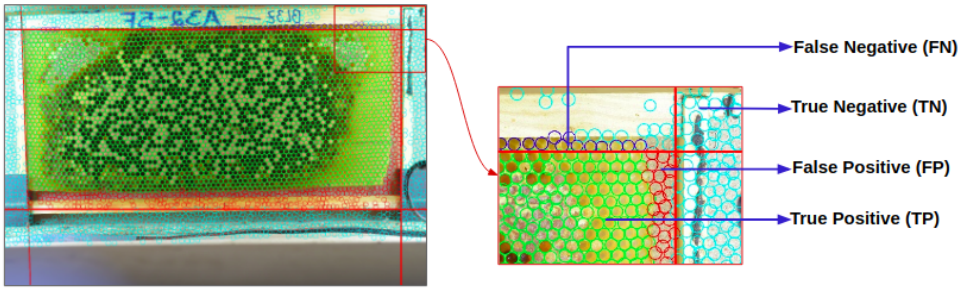


Figure 5.3: SEGHL method applied in a DS-SEG-TEST's image.

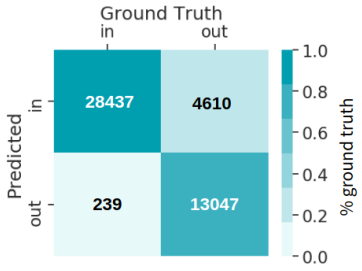


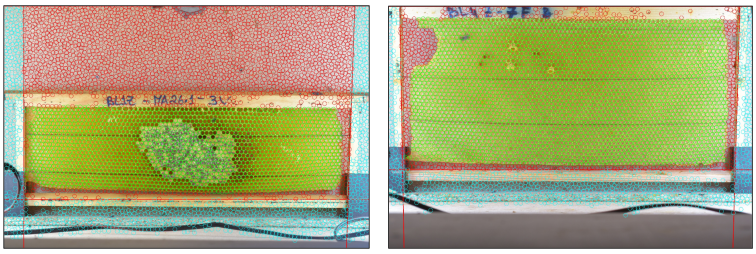
Figure 5.4: SEGHL confusion matrix.

We extracted some metrics from the confusion matrix, we present them in table 5.2. Analysing the results we observed that this model is able to filter the positive cells with high confidence, but it has drawback that is the great number of false cells classified as true, this can be seen both in the accuracy and specificity. We analysed the images with

the greatest number of false positives and among them were both presented in figure 5.5. Figure 5.5(a) is an image of a frame that is placed on top hive, that's why it is smaller. Its small size causes the top frame structure not to be present in the upper Region of Interest (ROI) we use to detect the top frame boundary with the HLT (Figure 3.14). Detections made between the comb and the boundary area were another factor that aggravated the false detections rate, we show an example of them in figure 5.5(b). These false detections are caused by the parameters we chose make possible the detections of areas covered by honey by the cHT method.

Method	Accuracy	Precision	Recall	Specificity	F1-Score
SEGHL	89,53%,	86.05%,	99.2%,	73,9%,	92.1%

Table 5.2: Metrics calculated using the method SEGHL.



(a) (b)

Figure 5.5: The drawback of detections using the SEGHL method

We also apply this method to images belonging to the DS-COMB-CREA dataset. In figure 5.6 we present some results. We can analyse in this figure that the ROIs we chose to find the frame borders based on the DS-COMB-PT do not fit perfectly in this dataset, mainly the left area. Therefore, adjustments to the ROIs sizes should be made to use this approach in different datasets, or the images should be standardised.

The positive side of this approach is its performance, in our tests the time to find the frame margins ranged from 24ms to 46ms, with an average of 32.9ms. This segmentation can also be used to create an ROI where only the cells within the ROI will be detected. The figure 5.7 compares the time required to detect cells using the entire image, the ROI, and presents the time distributions. We can observe on both density distribution

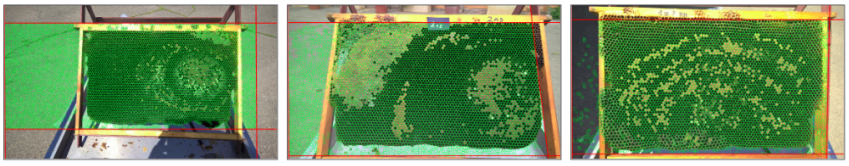


Figure 5.6: Bad results of the method SEGHL when applied in the dataset DS-COMB-CREA.

and mean values (dashed lines), that looking for cell inside the ROI increases detection performance.

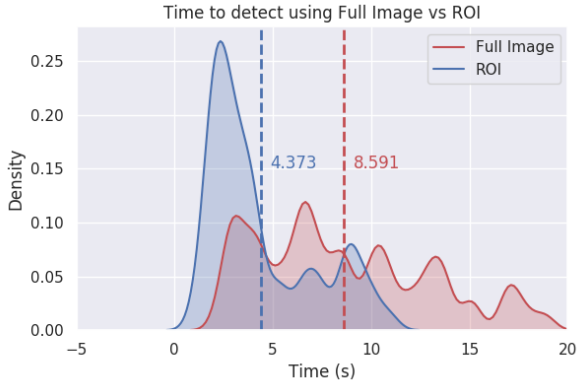


Figure 5.7: Time to detect all cells using a full image vs ROI obtained by the SEGHL method.

5.3.2 Results for the Segmentation Based on Cell Classification (SBOCC)

We obtained three models from the trainings made with NVIDIA DIGITS, for each one we saved the epoch with the highest accuracy. The metrics related to training are presented in table 5.3, it compares the best accuracy and loss obtained among the models.

We noticed that this was not a very complex problem to be solved by the GoogleNet architecture, considering that since the first epoch the model already had an accuracy above 90% (Figure 5.8). This good performance is also due to the use of a pre-trained model from the ImageNet dataset, with it, feature extraction filters were not learned from scratch. Figure 5.8 shows the evolution of the metrics over the 30 epochs. In this training,

Model	Accuracy on validation set (%)	Loss on validation set
SBOCC-42	98,64	0,0574
SBOCC-50	98,81	0,0501
SBOCC-70	**99,28**	**0,0275**

Table 5.3: Best loss and accuracy on validation set to each model trained using the method SBOCC.

the epoch 29 was the one that presented the best accuracy.

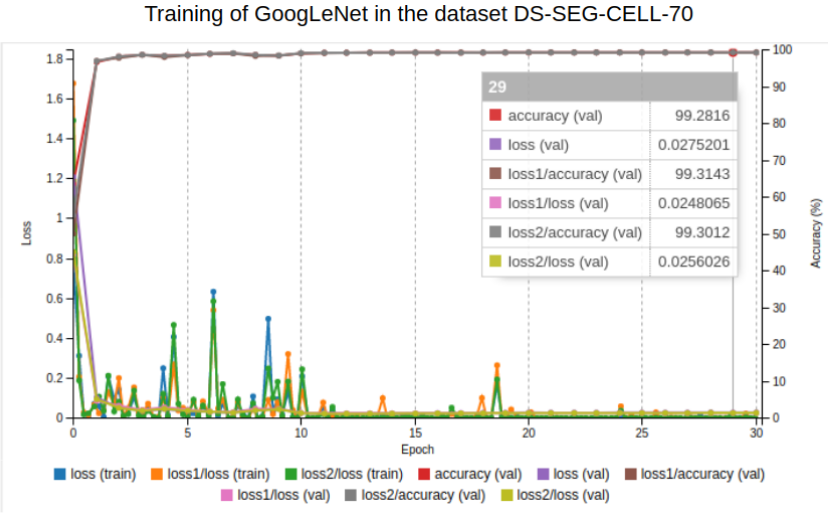


Figure 5.8: Evolution of the metrics loss and accuracy over 30 epochs.

We performed tests similar to those made in the previous section using the created models and DS-SEG-TEST. First, we generate the confusion matrices, from there we extract the metrics TP, FP, TN and FN, these rates are presented in table 5.4.

Model	TP	FP	TN	FN
SBOCC-42	25984	517	17140	2692
SBOCC-50	26768	283	17374	1908
SBOCC-70	**27625**	**161**	**17496**	**1051**

Table 5.4: Metrics calculated for the method SBOCC over the confusion matrix calculated using the DS-SEG-TEST dataset.

We calculated new metrics based on the results presented in table 5.4 seeking to compare each aspect of the models developed. We present these results in table 5.5. The model with the 70×70px input performed better than the others in the chosen metrics.

We noticed that the quality of the results increases as we increase the region extracted from the cell to be analysed, it can be seen in the graph shown in figure 5.10.

Model	Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	F1-Score (%)
SBOCC-42	93,07	98,05	90,61	97,07	94,18
SBOCC-50	95,27	98,95	93,95	98,40	96,07
SBOCC-70	**97,38**	**99,42**	**96,33**	**99,09**	**97,85**

Table 5.5: Metrics calculated using the method SBOCC.

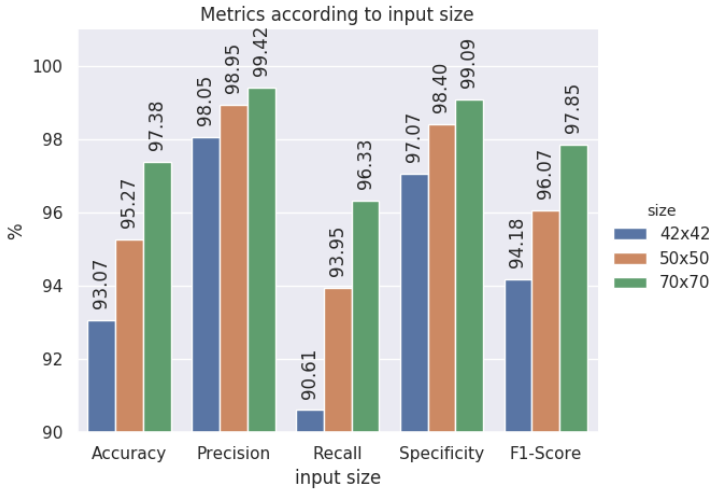


Figure 5.9: Metrics according to the extracted cell image size.

Using the best-trained model according to our metrics, we compared the average time needed to process all the cells found in a comb image. As shown in figure 5.10, the processing can be done in CPU or GPU, but related to performance, using GPU is on average 10 times faster than the same processing using our CPU. The similar result with different input sizes shown in figure 5.10 is due to the GoogLeNet input size, although the cells were extracted with different sizes, all of them were resized to 224×244px before being processed.

We did qualitative analysis of the method based on the examples presented in figure 5.11. We obtained these examples after processing some annotated images from the dataset DS-SEG-TEST using the model SBOCC-70. We can see in figure 5.11 that many false detections made within the wooden frame structure were classified as FP. If we had

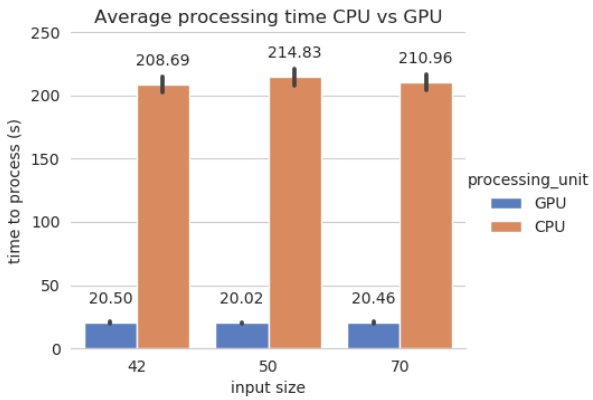


Figure 5.10: Average processing time for using GPU and CPU with different cells size.

used the method SEGHL, these detections would be classified as true, because they would be inside the segmentation boundaries. We also observed in the tests that in some images a large amount of capped cells are not classified as real cells.

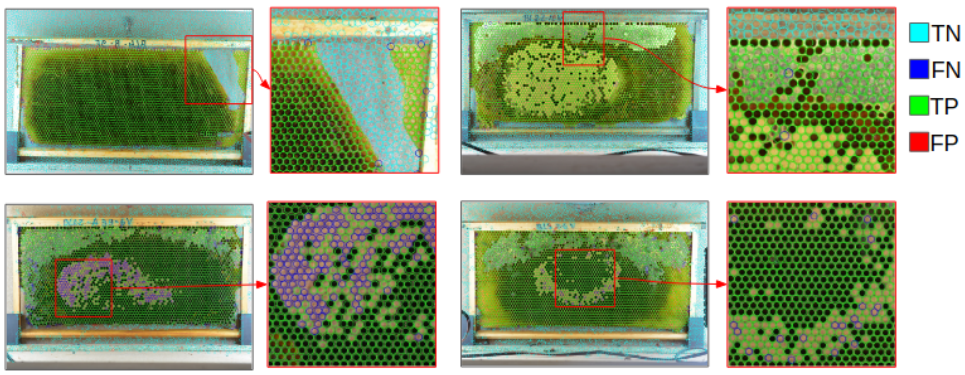


Figure 5.11: Results of DS-SEG-TEST dataset images processed by the SBOCC method.

We also performed analyses on images from the dataset DS-COMB-CREA using the model SBOCC-70. Some results are shown in figure 5.12. In this figure, it becomes even more evident the difficulty that this approach has classifying capped cells as real cells. Although this approach produces less FP compared to the SEGHL method, it produces a great number of FN detections, mainly on capped and honey areas. Carry out training with cells extracted with larger sizes than 70×70px, and more training examples may help reducing these number false classifications.

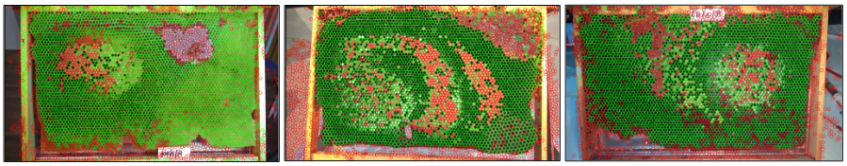
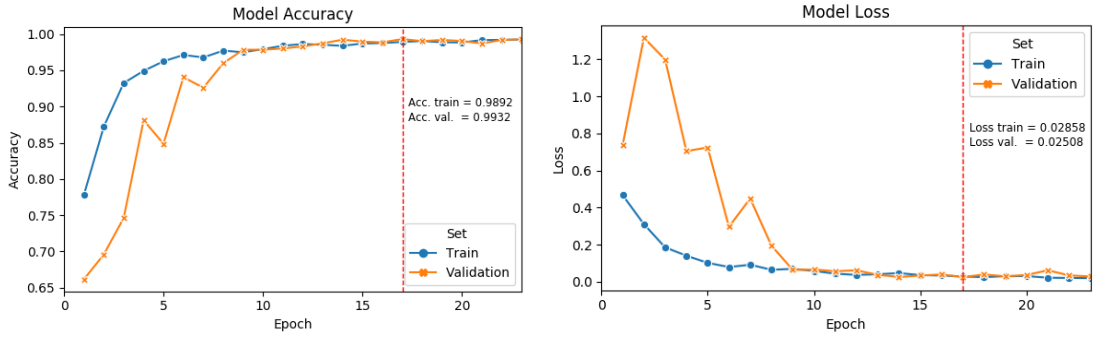


Figure 5.12: Results of DS-COMB-CREA dataset images processed by the SBOCC method. Red cells were detected as false detections and the green ones as true cells.

5.3.3 Results for the Comb Semantic Segmentation (CSS)

The training was carried out with 23 epochs in 3.45 minutes. The training was finalised before the 50th epoch due to the *Early Stopping*. The figures 5.13(a) and 5.13(b) show plots of the loss and accuracy metrics in the training and validation sets along the epochs. The dashed line represents the epoch with the lowest loss calculated, 17.



(a) (b)

Figure 5.13: Evolution of loss and accuracy during training of the honeycomb semantic segmentation model.

As output, this model produces a set of 128×128px images with values ranging from zero to one due to Sigmoid Logistic activation function in its last layer. This output is then processed using the Heaviside step function that defines values smaller than 0.5 as zero and the remaining as one. This process is shown in figure 5.14.

Using the images from the training and validation sets from DS-COMB-SEG-FULL dataset and the test images from the DS-SEG-TEST, we extracted the accuracy and loss metrics using the method *model.evaluate()* from the Keras module. These results are presented in table 5.6. Both metrics were computed pixel-wise.

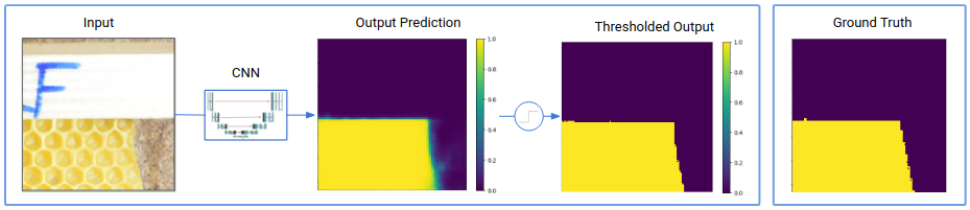


Figure 5.14: Post-process applied to an output image from the semantic segmentation model.

Set	Accuracy (%)	Loss
Train	98,92	0,02858
Validation	99,32	0,02508
Test	98,54	0,04393

Table 5.6: Accuracy and loss calculated pixel-wise in different sets.

To perform qualitative analysis on the CSS method we processed some images and compare their annotated and predicted segmentations. As a result, we obtained images with regions defined as TP, TN, FP, and FN, as shown in figure 5.15. The remaining images are presented in appendix E.

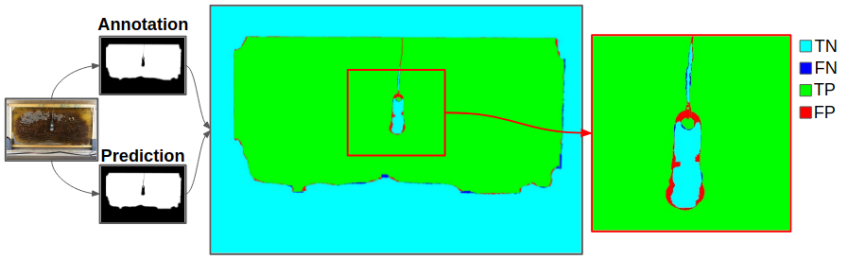


Figure 5.15: Comparison between annotated and predicted regions.

From these qualitative results, we observed that some images had regions inside the comb marked as negative and some areas outside the comb classified as positive as shown in figure 5.16.

Using methods present in the OpenCV library it is possible to carry out the removal of false segmented areas, as the example in figure 5.17 shows. For this removal we detect all contours in the image using the function *cv2.findContours()*[1]. From these contours we choose the one with the biggest area. Lastly, we draw the biggest contour in a black

[1]https://docs.opencv.org/3.0.0/d3/dc0/group___imgproc___shape.html

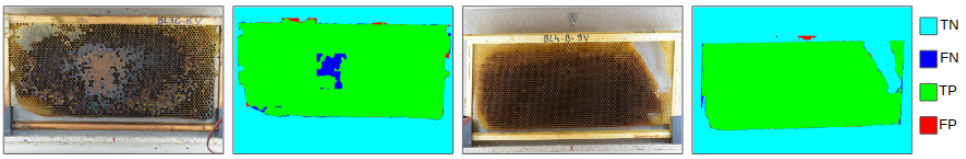


Figure 5.16: False negative areas inside the honeycomb and false positive regions on the background.

image using the function *cv2.drawContours()*, we use the *thickness* parameter as *-1* to fulfil the contour with a chosen colour (white). We name this approach as *Comb Semantic Segmentation using Largest Contour (CSS-LC)*.

Figure 5.17: Process to reduce the number false segmented areas (CSS-LC method).

CSS-LC has a drawback when the image contains some object in front of the comb, or if the comb is divided into two or more parts. In these cases the segmented object in front of the frame will be segmented as comb, and if the comb is divided, the smaller parts will be removed as we present in figure 5.18. Both images in the figure belongs to the DS-SEG-TEST. Although this drawback exists, it occurs mainly in images with anomalies. As can be seen in figure 5.19, the CSS-LC method remains robust even in honeycomb frame images photographed without a controlled environment nor a high-resolution camera (DS-COMB-CREA). It is worth remembering that only images from DS-COMB-PT dataset were used to train the CNN model used in this method.

To measure the quality of the predicted segmentation compared to the annotated we used the IoU metric. This metric was calculated on the DS-SEG-TEST dataset images. They were processed using the methods CSS and CSS-LC. We show in table 5.7 the comparison of the IoU results calculated using each approach.

In order to compare the CSS and CSS-LC approaches quantitatively with the other methods, we used the annotations from DS-SEG-TEST once again. As we did in the sections 5.3.1 and 5.3.2, we started by extracting the rates TP, TN, FP and FN (Table

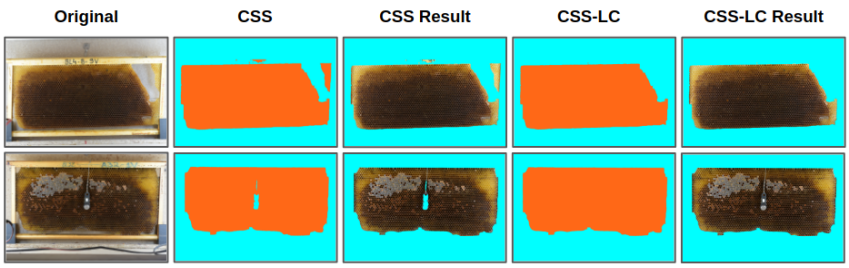


Figure 5.18: Visual comparison of results generated by CSS and CSS-LC methods on images from the DS-SEG-TEST dataset.

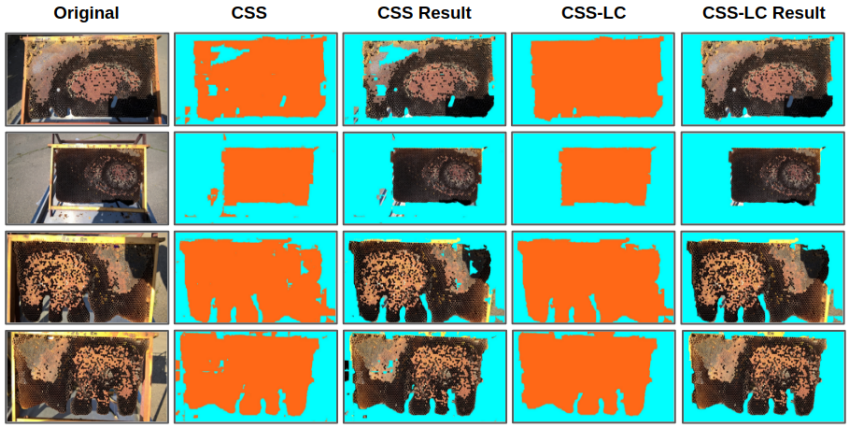


Figure 5.19: Visual comparison of results generated by CSS and CSS-LC methods on images from the DS-COMB-CREA dataset.

5.8) and then, generated metrics based on these values (Table 5.9).

In order to measure the performance of this approach related to processing time, we performed two tests. The first was a comparison of the time needed to split an image into tiles, perform inference, and reassemble the image using GPU and CPU only. In the second test, we compared the time needed to detect the cells of a full frame image and an image where the segmentation is used as ROI.

Figure 5.20 displays the results of our first test. As seen in the results obtained in section 5.3.2 with the method SBOCC, we see a higher performance of this method when the images are processed by the GPU compared to the CPU, but in this case, GPU performance was 33 times faster than the CPU. Probably the GPU performance increased because of the smaller input in the CSS method, 128×128px, compared to 224×224 using the method SBOCC, but further comparisos need to be performed.

Method	IoU (%)
CSS	97,60
CSS-LC	**97,74**

Table 5.7: Accuracy and loss calculated pixel-wise in different sets.

Method	True Positive	False Positive	True Negative	False Negative
CSS	28.319	**168**	**17.489**	357
CSS-LC	**28.378**	187	17.470	**298**

Table 5.8: TP, TN, FP and FN calculated for the methods CSS and CSS-LC over images from DS-SEG-TEST dataset.

In the second test, we compare the time needed to detect all cells present in 61 images from the dataset DS-COMB-SEG-FULL. First, we detect cells in the full images. In a second moment, we detect cells in all images again, but this time we reduce the search area as seen in figure 5.21. In this second approach, we multiply the original image with the segmentation and then, remove the excess area using a bounding rectangle calculated by the function *cv2.boundingRect(cnt)*, where the parameter *cnt* refers to the largest contour found in the image. Figure 5.22 shows the comparison of time density distributions using or not the ROI.

5.3.4 Comparison of All False Detections Removal Methods

In this section, we will compare the results obtained with our three approaches developed to remove false detections (SEGHL, SBOCC, CSS and CSS-LC). First we will compare the quantitative metrics of the three approaches, then we will compare the computational cost to accomplish the segmentation and finally, we will compare the time needed to detect the cells in full frame images and with ROIs generated by the approaches SEGHL and CSS-LC.

- **Quantitative comparison:** this comparison was made using the methods results over the DS-SEG-TEST dataset. For the SBOCC approach, only the model trained with the input images 70×70px will be used in the comparison, since it surpassed

Method	Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	F1-Score (%)
CSS	98,87	99,41	98,76	99,05	99,08
CSS-LC	98,95	99,35	98,96	98,94	99,15

Table 5.9: Accuracy, Precision, Recall and Specificity calculated for the methods CSS and CSS-LC.

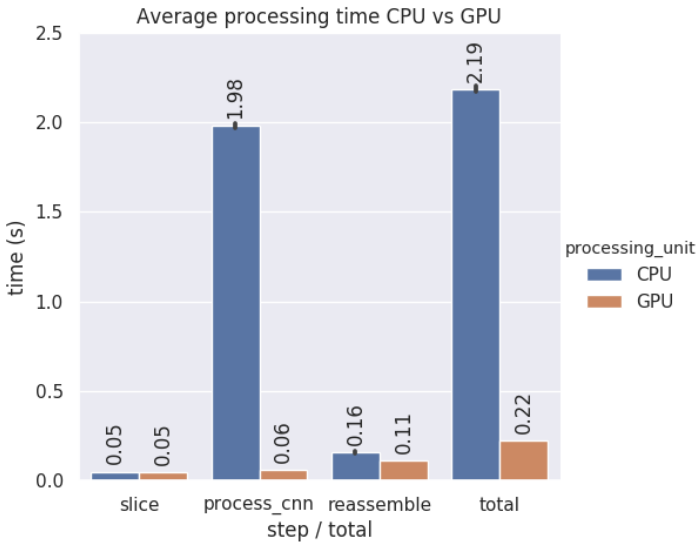


Figure 5.20: Comparison of average time to process an image using the CSS-LC technique with GPU or CPU only.

the other models in all metrics analysed. Table 5.10 presents a comparison of the methods results.

Method	Acc. (%)	Precision (%)	Recall (%)	Specificity (%)	F1-Score (%)
SEGHL	89,53	86,05	**99,2**	73,9	92,1
SBOCC-70	97,38	**99,42**	96,33	**99,09**	97,85
CSS	98,87	99,41	98,76	99,05	99,08
CSS-LC	**98,95**	99,35	98,96	98,94	**99,15**

Table 5.10: Comparison of metrics for false detection removal methods.

We observe in table 5.10, that there is considerable dispersion in the quality of the results between the methods. The recall measure, for example, although larger for the SEGHL method, can not be observed outside the context, since this model tends to accept a large part of the cells as positive, this increases the recall metric but in

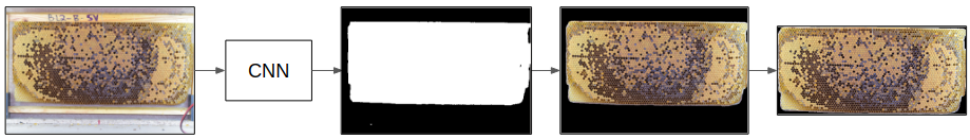


Figure 5.21: Process for creating the region of interest based on the segmentation performed by the CSS-LC method.

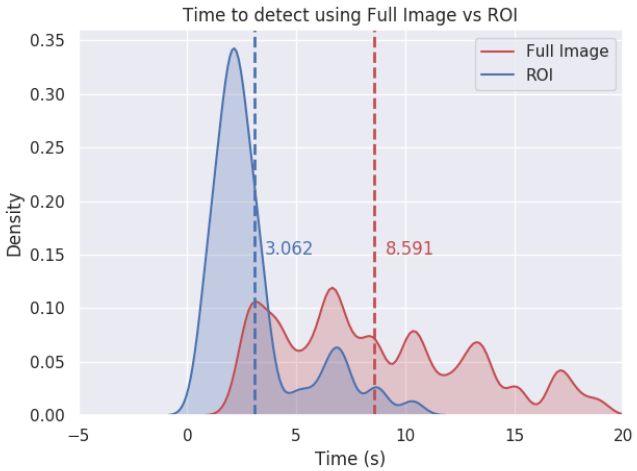


Figure 5.22: Comparison of time to detect the cells of a comb using or not a region of interest created from the CSS-LC method.

contrast, decreases the other metrics. The SBOCC-70 method also demonstrated two great results for both precision and specificity in the test set, but overall it performed poorly compared to the semantic segmentation methods, mainly in accuracy and F1-score metrics. It is worth remembering that the F1-score metric is calculated from the precision and the recall and different from a simple average it tends to the smaller value. Thus, the result of 99.15 for the CSS-LC method, allied with the IoU of 97.74% calculated in section 5.3.3 demonstrate that this approach is the one that best generalised during the training and generated better results on the test sets we are working on.

- **Comparison of computational cost:** in this test we compared the average time required to process an image of size 6000×4000px (reduced to 1500×1000px for the SEGHL method) with each of the three approaches we developed. For the SEGHL

method we only use the method *cv2.houghLinesP()* implemented for CPU, although there is also a version for GPU[2]. Related to the semantic segmentation methods, we made the comparison only using the CSS-LC. These results are shown in table 5.13.

Method	Time with CPU (s)	Time with GPU (s)
CSS-LC	2,19	**0,22**
SBOCC-70	210,96	20,46
SEGHL	**0,032**	-

Table 5.11: Comparison of time to run three approaches to remove false detections, using CPU and GPU.

The results in table 5.13 demonstrate that the SEGHL method performs better than the other two approaches, this is due to the simpler operations that it performs, besides not being dependent on neural models. We also noticed that the second best-placed model, CSS-LC, is about 90x faster in our environment when compared to the SBOCC-70.

- **Comparison of time needed to detect cells:** In this test, we compare the average time required to detect all the cells of an image in three different situations. In the first one, we detect cells in full images, in the second we detect cells with ROIs extracted using the SEGHL approach of section 5.3.1 and in the third, we also do the detection using ROIs, but this time it was obtained from the CSS-LC approach (section 5.3.3). These tests were performed in all 61 images of the dataset DS-COMB-SEG-FULL. Figure 5.23 displays the comparison among the methods. With the SBOCC-70 method, it is not possible to create an ROI to detect the cells, because it depends on the detections to work, that's why we present it along with the full image distribution.

  As can be seen in figure 5.23, using methods to create ROIs and using these regions to reduce the cells search space influences considerably the time needed for detections. Despite adding the time to segment the comb over the time to detect the cells, the

[2]https://docs.opencv.org/3.0-beta/modules/cudaimgproc/doc/hough.html

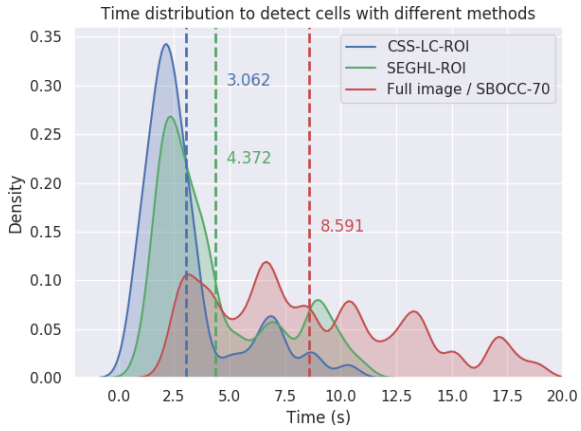


Figure 5.23: Time distribution to detect cells with different methods.

CSS-LC approach with ROI is faster when compared to only detecting cells using the entire image. It is also important mentioning that before each detection we used the method from section 3.4 to make the detections scale invariant. This process takes between 400ms and 1480ms, with an average time of 860ms, so the detections can be about 1 second faster if the radius of the cells is previously known.

5.4 Comparison of Different OpenCV Versions for Cells Detection

During the development of this work several versions of the OpenCV module were released (3.3.1, 3.4.0 and 3.4.1), in all these versions changes in the implementation of the method **cv2.houghCircles()** were made. We began developing our methods for detecting cells using the version 3.3.1. In this version the detections were made in a single processor core. After the pull request 10041[3] in the OpenCV master branch, the method of detecting circles started to be distributed among the available CPU cores and thus improved the method's performance, this new implementation is part of version 3.4.0. The implementation of the method added in the pull request 10232[4] brought new optimizations and

[3]https://github.com/opencv/opencv/pull/10041
[4]https://github.com/opencv/opencv/pull/10232

once again the performance was improved, but according to our analyses in honeycomb frames images, the quality of the detections worsened. Table 5.13 compares among the three versions the number of detections and the detection time in three different images.

Version	Image name	Cells found	Time (s)
3.3.1	img1.jpg	4.127	16,052
3.3.1	img2.jpg	4.250	20,333
3.3.1	img3.jpg	4.186	21,269
3.4.0	img1.jpg	4.148	3,369
3.4.0	img2.jpg	4.254	8,505
3.4.0	img3.jpg	4.193	9,359
3.4.1	img1.jpg	4.132	0,384
3.4.1	img2.jpg	4.341	0,428
3.4.1	img3.jpg	4.205	0,489

Table 5.12: Comparison of the time needed to detect the cells and the amount detected in different OpenCV versions.

In table 5.13 we can observe that there was a more than 2-fold increase in the performance of the cHT method between versions 3.3.1 and 3.4.0. In our analyses there were no significant changes in the quality of detections in our problem. Regarding the changes made between version 3.4.0 and 3.4.1, they were largely optimizations. These optimizations made the method run about 19 times faster. This improvement in performance came with a drawback to our problem, by analysing images processed by this implementation we saw that the quality of the results on the capped areas worsened when compared with the previous versions, figure 5.24 gives an example. We tried to search for other hyperparameters that could improve the results, but even using the search method proposed in section 3.3 we cannot get results with the same quality as those obtained in version 3.4.0.

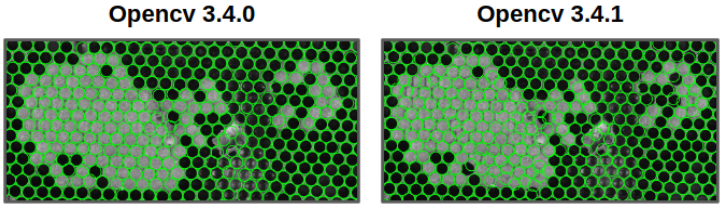


Figure 5.24: Comparison of results obtained by OpenCV versions 3.4.0 and 3.4.1.

OpenCV also provides an implementation of the HoughCircles method for GPUs. It is present in all OpenCV versions we work with. Although this method implementation is not yet available for the Python language, we also perform tests with this implementation to analyse the quality of its results using the C++ language. We performed tests using the three different OpenCV versions, in each of them we tested the same images we used in the tests with CPU, we show the results in table 5.13.

Version	Image name	Cells found	Time (s)
3.3.1	img1.jpg	4710	0,097
3.3.1	img2.jpg	7111	0,112
3.3.1	img3.jpg	7002	0,122
3.4.0	img1.jpg	4677	0,095
3.4.0	img2.jpg	7128	0,111
3.4.0	img3.jpg	6996	0,122
3.4.1	img1.jpg	4682	0,095
3.4.1	img2.jpg	7120	0,111
3.4.1	img3.jpg	6999	0,123

Table 5.13: Time to detect cells and amount detected using the GPU implementation of different OpenCV versions.

In the results presented in table 5.13 we can see that the time needed to detect the cells is on average 109ms, this result is better than those obtained using only the CPU (table 5.13). We observed in the result images that the detected points are very similar to those obtained using the CPU with version 3.4. One drawback of this approach is it does not take into account the *minDist* parameter completely, as it can be seen that the *CELLS FOUND* column, using the GPU method more detections are made when compared with the CPU only results. What happens is that the method implemented in the GPU, in all the versions tested, allows that multiple detections be made in the same point. The figure 5.25 presents the comparison between a detection performed with and without GPU, in the right image we can see that some cells have the circle drawn thicker due to the number of detections in the same place. We can conclude with this tests that using the GPU implementation produces faster detections, but to use it will be necessary to remove detections made in the same point, and if the Python language is being used

it is necessary to create an wrapper from the C++ implementation.

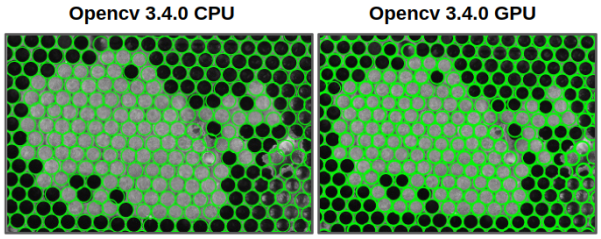


Figure 5.25: Comparison of detections made using CPU and GPU.

5.5 Detections Quality Comparison

In order to measure the quality of the detections we also performed tests similar to the tests proposed in Lee et al. [19]. Comparisons between our results and the results obtained by Lee et al. [19] were also made. For the tests 10 images were chosen, in our case, we used the images from the dataset DS-TEST-SEG. In these images the class of all cells are annotated manually, this result is then compared with the detections made by the detector. To detect the cells we use the scale invariant approach proposed in section 3.4. After the detections being made we used the CSS-LC method to remove the false detections. The task of labelling all the cells manually would require a lot of time, so we used the DS-TEST-SEG annotations and manually corrected them adding or removing cells when necessary. Table 5.14 shows the results obtained over the ten images.

It is shown in figure 5.26 one of the test images with a large number of false detections compared to the other images. In this image, we had a large number of false detections at the comb edges as well as observed in Lee et al. [19]. False cell-related or noise-related detections were not significant in our tests as they were in Lee et al. [19]. Another factor that had a negative impact on the results of Lee et al. [19] was the low contrast in some cells. How it is presented in the cell detection method (Section 3.3), we diminished this effect by applying the CLAHE filter to the images before detecting the cells. The comparisons made in the other nine images are in the appendix F.

Image Name	Manual Count	Automatic Count	TP	FP	FN
DSC_1940.JPG	3024	2949	2944	5	80
DSC_1992.JPG	2795	2742	2735	7	60
DSC_2832.JPG	2869	2833	2794	39	75
DSC_2839.JPG	3082	3062	3041	21	41
DSC_2864.JPG	2961	2982	2948	34	13
DSC_2951.JPG	2910	2889	2857	32	53
DSC_2443.JPG	2077	2088	2075	13	2
DSC_3475.JPG	2875	2876	2852	24	23
DSC_4326.JPG	3061	3092	3054	38	7
DSC_4496.JPG	3072	3056	3044	12	28

Table 5.14: Comparison between automatically detected cells and the ones automatically detected and manually corrected.

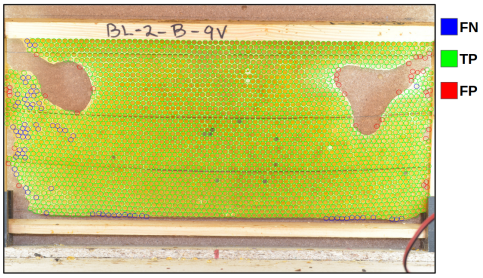


Figure 5.26: Results of the comparison between cells detected by our algorithm and by humans.

As in Lee et al. [19], we also calculated for each image the cells detection rate using equation 5.1. This metric is based on the total number of cells automatically detected excluding the falsely detected cells divided by the manual count.

$$CellDetectionRate = \frac{DetectionCount - FP}{ManualCount} \times 100\% \tag{5.1}$$

Table 5.15 shows the calculated cell detection rate for each image. The results obtained showed a great quality considering that the values varied between 97,35% and 99,9% compared to the results obtained by Lee et al. [19] which range between 73% and 93%. In table 5.16 we present some other comparisons made between our approach and the one developed by Lee et al. [19].

We can observe in table 5.16 that the [19] method has a lower FP rate than ours,

Image Name	Cell Detection Rate (%)
DSC_1940.JPG	97,35
DSC_1992.JPG	97,85
DSC_2832.JPG	97,39
DSC_2839.JPG	98,67
DSC_2864.JPG	99,56
DSC_2951.JPG	98,18
DSC_2443.JPG	99,90
DSC_3475.JPG	99,20
DSC_4326.JPG	99,17
DSC_4496.JPG	99,09

Table 5.15: Cell detection rate calculated for the method CSS-LC, per image.

Method	Avg FP	Avg FN	Avg Cell Detection Rate (%)
Lee et al.	**5,1**	274,8	82,6
Ours	22,5	**38,2**	**98,7**

Table 5.16: Comparison between the detection method CSS-LC and that proposed by Lee et al. [19], the numbers in parentheses represent the standard deviation

this is due to the more restrictive parameters they chose for cHT, this becomes more evident when we examine the false negatives rate, where our approach had higher results. Observing our results, we can verify that the method we developed was quite balanced related to the FP and FN rate and that their impact was not significant on the cell detection rate, since this was close to $\approx$99%.

Chapter 6

Results for Cell Classification

In this chapter, we will present the results related to cell classification. First, we will show the results associated with the search for the best input format for our CNN models (Section 6.1). Next, we will present findings related to comparisons made between different CNN architectures using or not DA (Section 6.2). Finally, we present comparisons made with previous methods proposed in the literature (Section 6.3).

6.1 Definition of the Best Input Format for the CNN Models

We defined in section 4.2 two hypotheses that could help choosing the best input format for the models. They consider the input size and the position of the annotated cells, these positions were divided as human annotated (HANOT) and automatically predicted (PRED).

After training the 34 models (17 different inputs sizes with positions annotated by humans and automatically) using the architecture *InceptionV3* we obtained the plot shown in figure 6.1. The plot was generated from the test set images. We use a dashed green line to represent the trend of accuracy according to the size of the input. Observing this line, we can verify that there is an increase in the accuracy along with the input size until

about 300px, after that the quality tends to decrease. This effect gives us the information that input images with sizes close to 300px tend to produce better results.

Regarding the position of the annotations used in the training, we noticed that 68,7% of the models trained using annotations with predicted positions surpassed the models with positions annotated by the beekeeper. One hypothesis to explain this effect is that the dataset DS-COMB-CELL-TEST, have their cells automatically detected and classified by humans. Thus, we have evidence that models trained using cells detected automatically and labelled by humans generate superior results in test sets that had their cells detected automatically and libelled by humans when compared to models trained on cells detected and annotated by humans. This information is relevant to our work because we do not expect human interaction for detecting the cells.

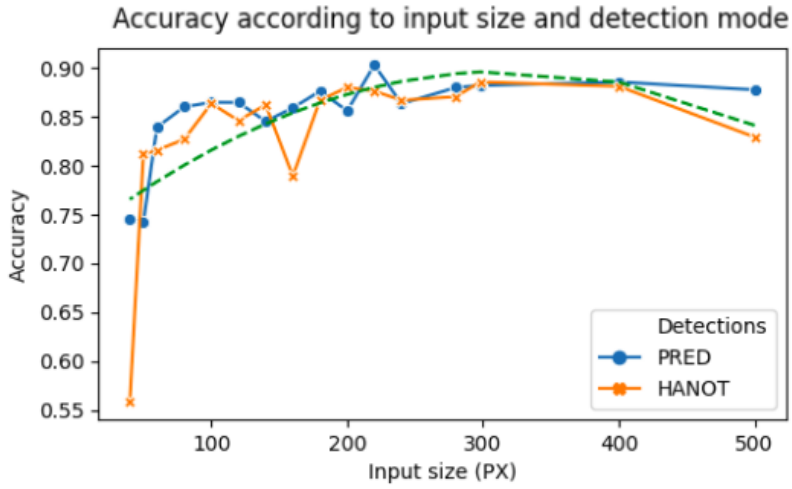


Figure 6.1: Test accuracy according to the input size and the detection mode. Models trained using the InceptionV3 architecture

Figure 6.2 presents the time required to process a batch of 100 images according to their size, using the models trained using *InceptionV3*. Looking at this plot we noticed that the time required for processing a batch tends to grow in a squared rate according to the input size, this value is expected because the image format is square. It is worth remembering that images with smaller than 139×139px were inputs to 139×139px due to restrictions of the *InceptionV3* architecture, this explains the small variation of time in

the first results in the plot.

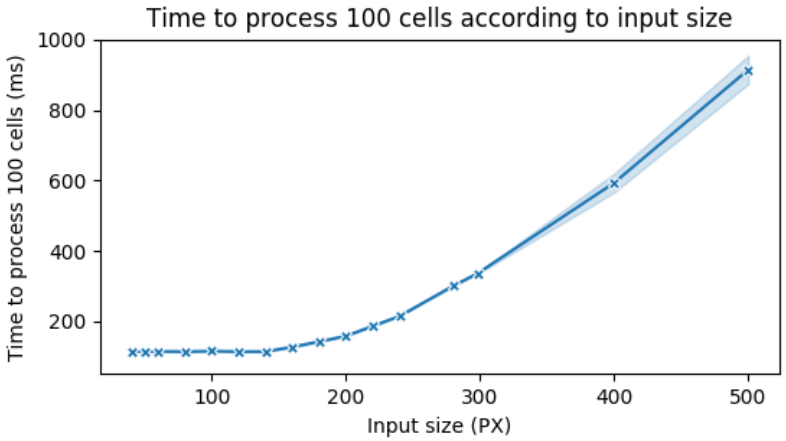


Figure 6.2: Time to process a batch of 100 images according to the image size.

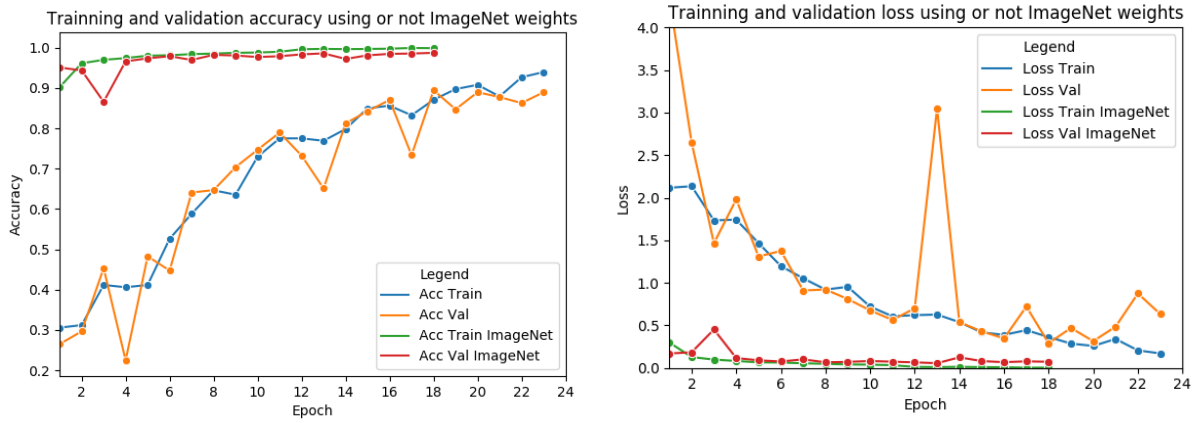
We chose as the default size for the input image in the next tests 224×224px. For this choice we consider the computational cost, this size has a moderate cost compared to the others, we also consider that the best result in the test set was the one trained with 220×220px inputs. The reason why we did not choose the input size 220×220px in the following tests is due to some architectures like MobileNetV2 that only have weights pre-trained in the ImageNet dataset for the following sizes (128, 160, 192, or 224)[1]. Since we did not want to compare trained models with different input sizes, we chose only 224×224px.

6.2 Comparison of Different CNN Architectures

Before we train the different architectures we decided to perform a sanity check to know if the models trained using the ImageNet weights do better than those trained from scratch. For this test, we use the *InceptionV3* architecture and the dataset DS-COMB-CELL-PT. This architecture was trained using the same methodology proposed in section 4.3. The figures 6.3(a) and 6.3(b) show the result we obtained. We can observe in the figures that using pre-trained weights during training helps the model to achieve better results

[1]github.com/keras-team/keras-applications/blob/master/keras_applications/mobilenet_v2.py

in less time, consequently, fewer epochs are necessary for convergence. Observing that the model trained from scratch took 5 extra epochs to converge and that each epoch took 253.6 seconds on average, we can conclude that it took about 21 minutes more to be trained, in addition to not achieving the same quality of the results of the other model trained using transfer learning.



(a) (b)

Figure 6.3: Comparison between models trained from scratch and using pre-trained weights from ImageNet in the DS-COMB-CELL-PT dataset.

After we have more confidence in using pre-trained weights in ImageNet, we began the training process of the different CNN architectures. The models we chose to compare were DenseNet (121, 169 e 201), InceptionResNetV2, InceptionV3, MobileNet, MobileNetV2, NasNet, NasNetMobile, ResNet50, VGG (16 e 19) e Xception. The training process was defined in section 4.3. In these training we used 224×224px input images with its positions detected automatically and labelled by humans (PRED).

During the training we noticed that the VGG 16 and 19 models were not converging, even after trying to use different LRs, cost functions and optimiser, then we decided to remove them from the tests. Another model that had problems during the training was the NasNet (Large), due to its complexity we had problems in saving it. This is a known bug in the community[2] and there are bypasses for it, we decided not to use them because

[2]https://github.com/keras-team/keras/issues/8711#issuecomment-354585187

we would have to train the model again and this process would take more than 18 hours. Although we have not been able to save the model, we were able to train it and extract some data that will be displayed next.

6.2.1 Results of Analyses Related to the Convergence Time and Size of the Models

The time required for the models convergence (5 epochs without improving on the *val_loss* metric) of the was one of the first observations we made for the trainings. Figure 6.4 presents the time needed to train each model. In the figure, we observed that the NasNet architecture proved to be more time-consuming to train, this is due in large part to its quantity of weights to be learned and its depth (Table 6.1).

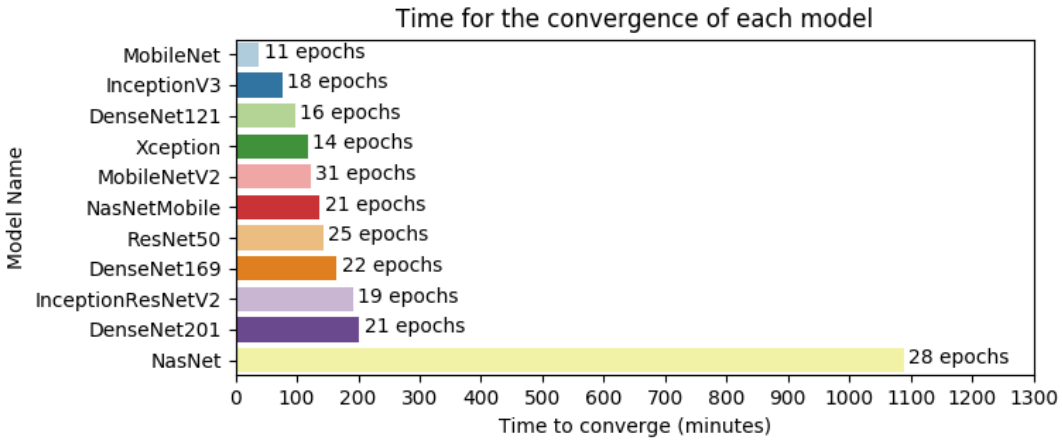


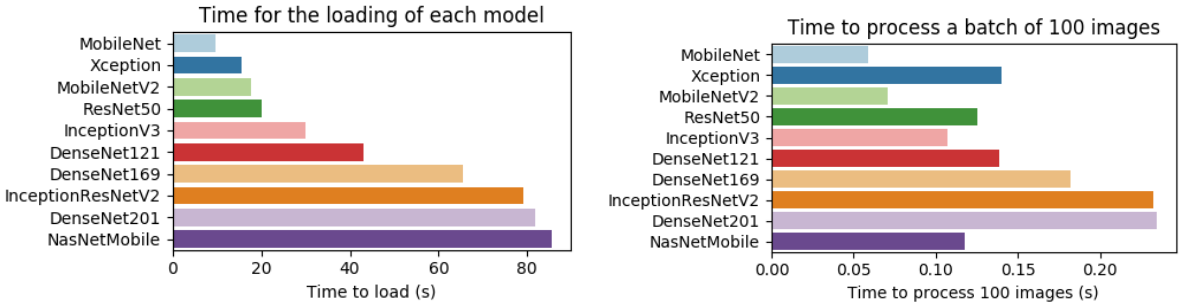
Figure 6.4: Time for the converge of each model during training.

Table 6.1 presents other perspectives regarding the computational performance of each model. In this table, we can observe that the MobileNet model had great result related to the number of epochs to converge and the average time of each epoch. Its number of weights is also among the lowest, only behind of its second version.

Two other perspectives that can be used to analyse the models are the time to process a batch with 100 images and the time for the model to be loaded into memory. These analyses are important to know how the models will behave when used after training. These

Model Name	Epochs to converge	Avg. time per epoch (m)	Num. of Weights
DenseNet121	16	362.77	8,094,279
DenseNet169	22	450.16	14,355,015
DenseNet201	21	577.14	20,296,263
InceptionResNetV2	19	606.31	55,917,799
InceptionV3	18	253.63	23,908,135
MobileNet	**11**	**211.58**	4,285,639
MobileNetV2	31	235.98	**3,576,903**
NASNet	28	2332.04	89,053,785
NasNetMobile	21	393.81	5,359,259
ResNet50	25	343.45	25,693,063
Xception	14	503.87	22,966,831

Table 6.1: Comparison between models in relation to training time and amount of weights.



(a) (b)

Figure 6.5: (a) Comparison between the models related to the time to be loaded in memory, (b) comparison between the models related to the time to process 100 images 224×224px.

two comparisons are presented in the figures 6.5(a) and 6.5(b). Highlights of these plots go to the MobileNet model which once again performed well compared to the others and the NasNetMobile model, even though it had fewer parameters than most models (table 6.1), it was still the one that took the most time to be loaded into memory. The time to load the model refers to the time required to execute the method *keras.models.load_model(filepath)* and the time to processing 100 images is calculated with the method *model.predict(batch)*.

6.2.2 Results Related to Overall Classifications Quality

In order to evaluate the quality of the classifications of each model, we first compare the Loss and Accuracy of each one in its best epochs. Table 6.2 shows this result. Some analysis can be done on this table, such as the ResNet50 model, which showed a great ability to predict the training set examples, but performed worse on the other sets, probably due to overfitting, the DenseNet201 model that had its highlight related to accuracy in the validation and test sets, and the MobileNet that was the model with the best Loss results in the validation and test sets. This shows that the MobileNet predictions were made with more confidence, even if it did not hit so many predictions as the DenseNet201 did.

Model Name	Loss train	Acc Train	Loss Val	Acc Val	Loss Test	Acc Test
DenseNet121	0.00818	99.75%	0.05213	98.56%	0.25716	93.71%
DenseNet169	0.00159	99.95%	0.06365	98.58%	0.37087	93.12%
DenseNet201	0.00115	99.97%	0.05990	**98.66**%	0.31397	**93.94**%
InceptionResNetV2	0.00425	99.89%	0.05986	98.55%	0.29882	93.45%
InceptionV3	0.00415	99.90%	0.05594	98.58%	0.27237	93.47%
MobileNet	0.01563	99.57%	**0.05106**	98.48%	**0.23944**	93.31%
MobileNetV2	0.00942	99.69%	0.06468	98.57%	0.37828	93.02%
NasNetMobile	0.00162	99.94%	0.07417	98.56%	0.37836	93.79%
ResNet50	**0.00033**	**99.99**%	0.08845	98.44%	0.39329	92.99%
Xception	0.01173	99.73%	0.06574	98.54%	0.36011	92.70%

Table 6.2: Comparison of loss and accuracy between models in different datasets.

As we presented in section 2.7, using metric accuracy to compare models trained over unbalanced datasets is not a good choice because the metric will consider more the majority class. Considering this problem, we calculated the Precision, Recall and F1-Score metrics for each model on the test set. These results were obtained using the function *precision_recall_fscore_support()*[3] from the the Sklearn library with the parameter *average* = "*weighted*". The result is shown in figure 6.6, the models are sorted by F1-Score. We can see in the figure that the *InceptionResNetV2* model had the best result balancing Accuracy and Recall, the DenseNet201 model that had the best accuracy in the test set is in seventh place when we consider the F1-Score metric, this may indicate that

[3]http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

it suffered from overfitting and favoured the majority class. Another point that must be observed in the figure is the scale of the X-axis, although there are differences between the results, they have been quite close. More detailed results of this comparison are in the appendix G.

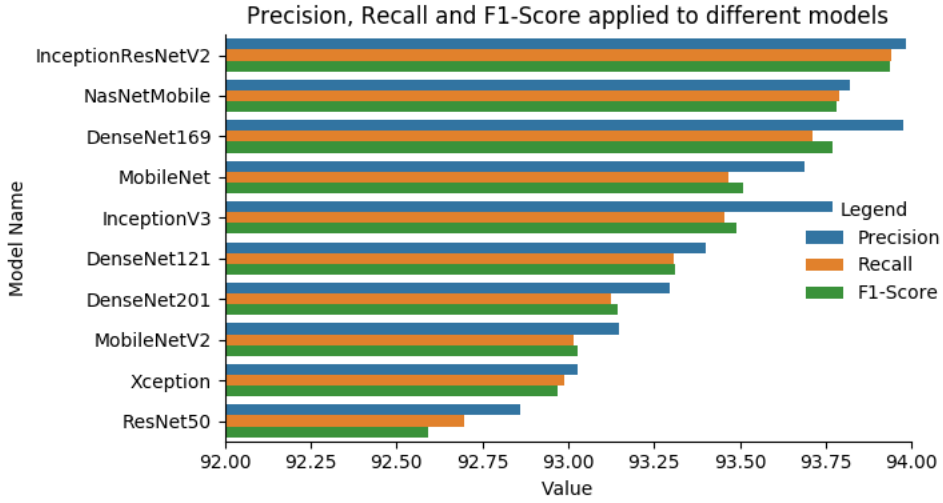


Figure 6.6: Precision Recall and F1-Score calculated using different models.

6.2.3 Results Related to Classification Quality by Class

Due to some models having evidence of overfitting, we decided to perform per-class performance reviews. In this analysis, we seek to know if any class has been favoured due to its number of examples. In order to collect this data, we processed all models in the test set and extract the F1-Score measure by class of each one, once again using method *precision_recall_fscore_support()*. We group the results in figure 6.7. We observed in this figure that the Egg class have more incorrect predictions, while the Capped class was very close to the 100% of F1-Score. To analyse the relation between the number of examples used in training and the result of F1-Score per class, we developed figure 6.8.

The figure 6.8 shows the relation between the number of examples and the average F1-Score obtained, by class. Observing the trending line we can see that the F1-Score tends to increase according to the number of examples. The Egg is an example of a class

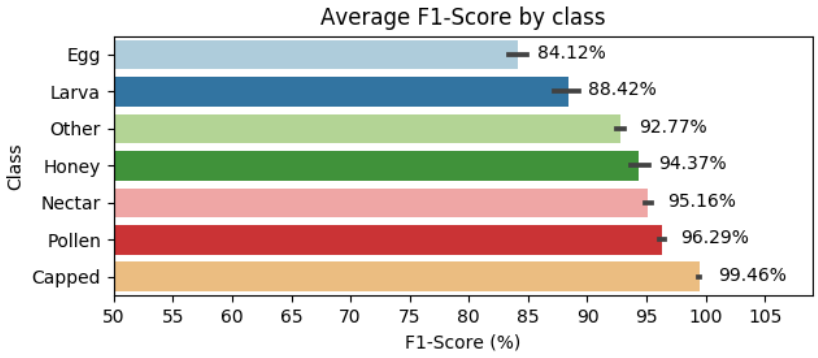


Figure 6.7: Average F1-Score per class.

that achieved low F1-Score when compared to the others, it is the class with less examples, it is probable that the Egg's result can be improved with a new training with more eggs examples. More examples are important for the Egg class, because it present geometries difficult to detect and can be easily confused with certain artifacts. The Capped and the Other classes can be interpreted as outliers, because the Capped class may have achieved a great F1-Score because of its easily distinguishable characteristics, and the Other class may have had a lower-than-expected result because it is the class with more variations.

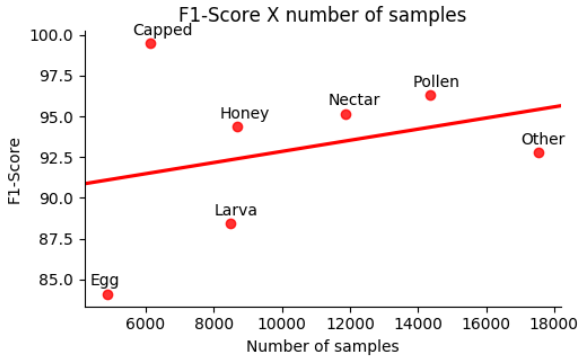


Figure 6.8: Average F1-Score per class.

6.2.4 Results for the Use of Data Augmentation

Following we will discuss the results of carrying out the training of models with the Data Augmentation technique explained in section 2.6.3. We present our methodology for this

test in section 4.5.

We chose four models to be trained using DA, they were: InceptionResNetV2 and NasNetMobile for having the best F1-Score calculated, MobileNet for having the lowest loss in the validation and test sets and DenseNet201 for having the best accuracy in the validation and test sets.

Figure 6.9 presents the comparison of the calculated F1-Score measure for the selected models trained on the normal data set and the dataset with DA. In this figure, we observed that outside the DenseNet201 model, all others models benefit from training using DA. Among the models, MobileNet had more surprising results outperforming more complex models, even by a small margin.

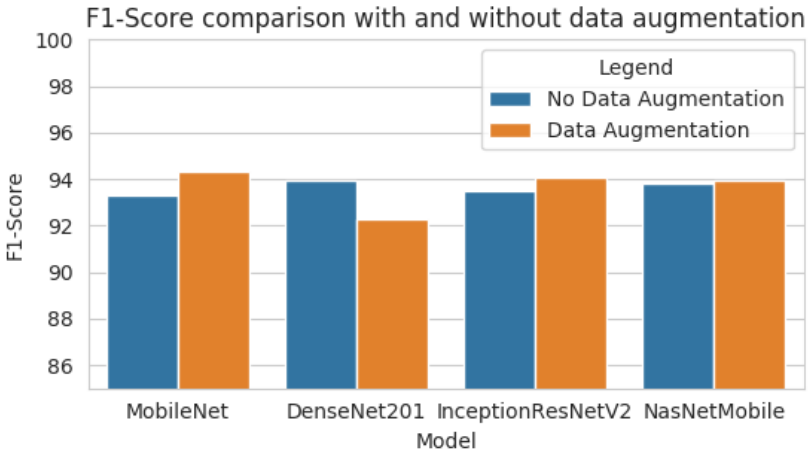
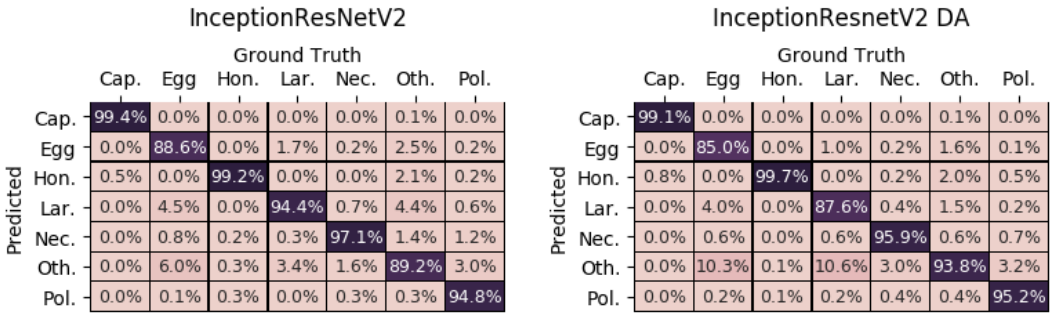


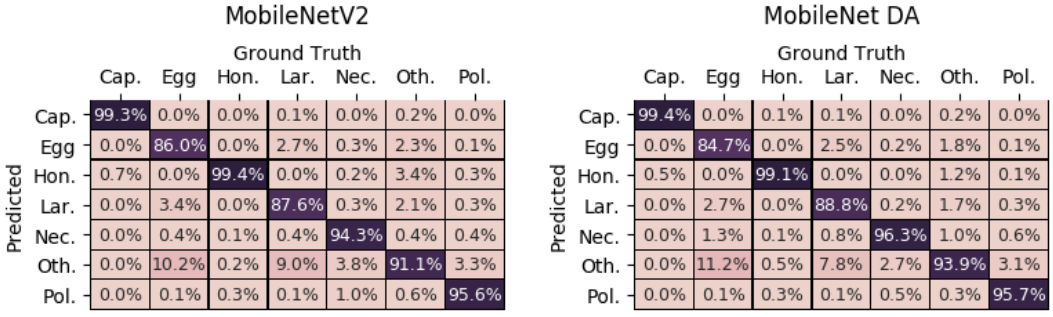
Figure 6.9: Comparison of models trained with and without Data Augmentation.

We also perform analyses by class to compare the models trained with the two different methods. In the figures 6.10 and 6.11 we present the confusion matrices for the InceptionResNetV2 and MobileNet models respectively. As we can observe in both images, there was a loss in the predictions quality of the minority class, Egg and a considerable increase in the precision of the majority class, Other. Regarding the InceptionResNetV2 model, the Larva class results also had a quality reduction. This may mean that DA has added non-representative characteristics to minority classes, in this case, more real examples may be needed to improve the results.



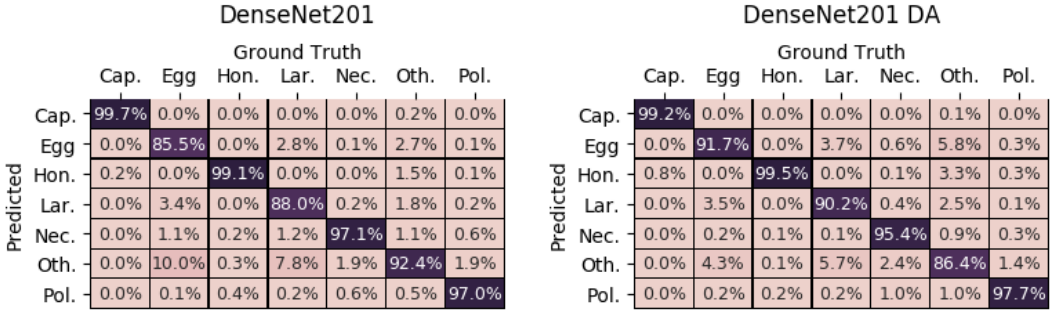
(a) (b)

Figure 6.10: (a) Confusion matrix InceptionResNetV2, (b) Confusion matrix Inception-ResNetV2 DA.



(a) (b)

Figure 6.11: (a) Confusion matrix MobileNet, (b) Confusion matrix MobileNet DA.



(a) (b)

Figure 6.12: (a) Confusion matrix DenseNet201, (b) Confusion matrix DenseNet201 DA.

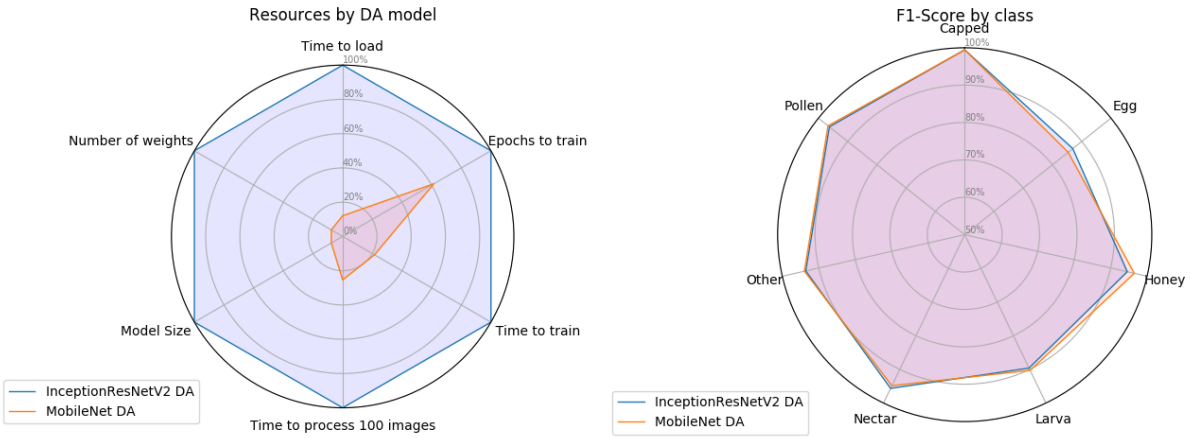
Another confusion matrix worth analysing is the one generated by the DenseNet201 model (Figure 6.12), which was the model with the best overall accuracy in the validation and test sets (table 6.2). Analysing its results without the use of DA it is possible to see that it had a high precision in the majority class, in this way it hit more examples of the test set, but had less capacity to predict classes with fewer examples such as Larva and Egg. This may be the reason for this model is the best placed when accuracy is considered (table 6.2), but do not having good results when F1-Score measurement is used (figure 6.6). After training the model with DA, there was a significant change among minority classes, which had best results, and the majority class that had a considerable reduction in its precision. It is possible to say with some caveats that the model trained using DA generalised better the dataset, or at least kept its results more balanced.

Confusion matrices were also generated for the other seven trained models without DA and NasNetMobile DA, these results are present in the appendix H. The data used to create these confusion matrices are presented in appendix I.

Being InceptionResNetV2 and MobileNet the models with best F1-Score metrics after training with DA, we decided to compare them based on the F1-Score by class and resources needed to train and use the models. These results are presented in figures 6.13(a) and 6.13(b). As the figure shows, the difference in the quality of the results of both models per class is not great, but related to the computational resources the MobileNet model was superior in all metrics analysed.

6.2.5 Results of Analyses Related to the Dataset and Classification of Cells with Different Content

While we were working with the datasets, we noticed some factors relevant to the results quality. These are the honeycomb positions most annotated by the beekeeper and the cells that had more than one content in their interior (e.g. Pollen and Egg). In the following, these factors will be explored and we will produce new insights on how to improve the classifications results.



(a) (b)

Figure 6.13: (a) Resources needed normalised by model, (b) F1-Score by class.

The task of classifying correctly 100% of cells in an image is not an easy task because of the wide range of colours, shapes, and textures that cells have. Besides these factors, we also identified during the development of the project that there are cells that have more than one content inside or are in a transition stage like Egg to Larva or Larva to Capped. This mixture of contents in the same cell makes the evaluation of the classifier less precise, since there may be cases where it hits one of the classes of the cell, but the other has been defined as the ground truth of the image, the figure 6.14 we present some examples.

In some image classification competitions this problem is handled with the use of Top-n accuracy [78]. With this methodology, the model earns credit for correctly classifying the image in its Top N guesses. For example, we can evaluate our model with the measure Top-2 accuracy, in case, if the correct class is between the two more likely predictions, it will be correct.

To perform this test, we recompile our models and add the metric *keras.metrics.top_k_categorical_accuracy()* with $k = 2$ and $K = 3$. Afterwards, we process the images from the test set and aggregate the results in the figure 6.15. In it, it is possible to observe that using the Top-2 accuracy is it possible to have an increase of 5% on average in the results.

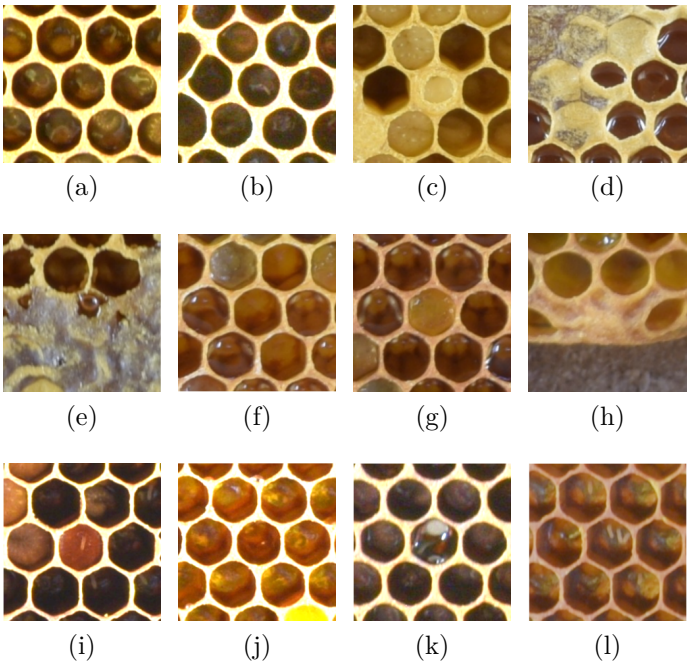


Figure 6.14: (a), (b) Transition between egg and larva; (c) transition between larva and capped; (d), (e) transition between nectar and honey; (f) cell with a little pollen; (g) cell with pollen and nectar; (g) defect in a honeycomb similar to an area of honey; (i) cell with pollen and an egg; (j) cell with nectar and appears to have an egg in its upper region; (k) cell with a larva, but with a bright similar to nectar; (l) cell with more than one egg, in case one first breaks this cell will have two classes.

Using this type of metric it is possible to have a broader notion of the quality of the model being analysed, this is due to the reduction of the negative impact that the cells with multiple contents generate on the result.

Another relevant factor for the quality of the results we discover during the development of this work is the inverse relation between the areas most frequently noted by the beekeeper and the areas where most incorrect predictions occur. Due to camera lens distortion, cells in different regions of the comb may display different areas of their interior, as can be seen in figure 6.16. We thought that this effect could impact the cell classifications if there were many annotations in certain regions of the picture and few in others.

In order to be able to analyse if areas with few annotations impact the results, we

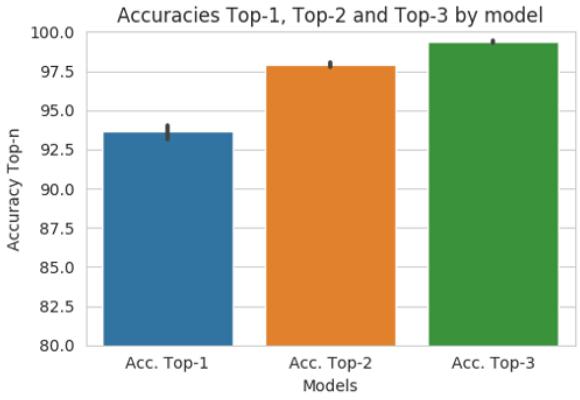


Figure 6.15: Accuracies Top-1, Top-2 and Top-3 by model.

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

Figure 6.16: (a) Cell extracted from the upper left region, (b) cell extracted from the upper region, (c) cell extracted from the central region, (d) cell extracted from the lower left region, (e) cell extracted from the lower right region.

should first know if the annotations were done in a well distributed way, or if they were concentrated in some regions. For this, we generate a plot to display the most annotated areas using the dataset annotations DS-COMB-CELL-PT (Figure 6.17), with it, we could see that most of the annotations were concentrated in the upper left part of the comb, following our hypothesis the models trained in this dataset would best classify cells of that region.

Next we needed well-distributed test annotations to see if more errors occur in areas with less annotations. For this we use the test set DS-COMB-CELL-TEST annotations. We processed the cells using the trained model InceptionResNetV2 and based on the incorrect predictions we created the figure 6.18(b), the figure 6.18(a) represents the most annotated regions. Comparing the results of both figures we can see that the lower-right regions were where most incorrect predictions occurred, this information is inversely related to regions where more annotations were made in most of the cases.

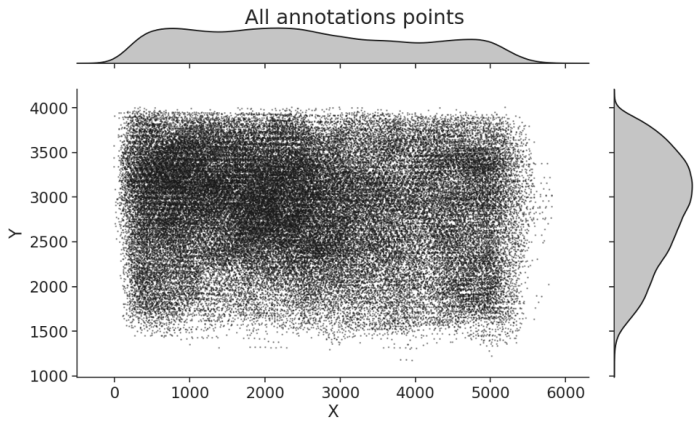
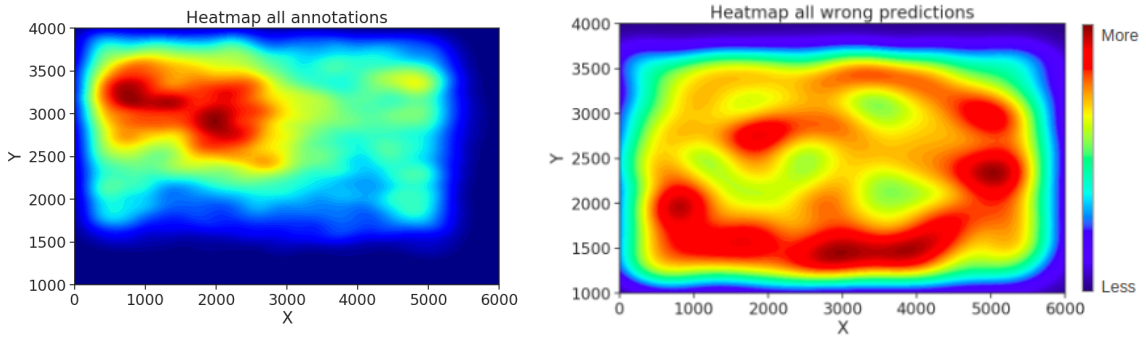


Figure 6.17: Distribution of all annotations made in the DS-COMB-CELL-PT.

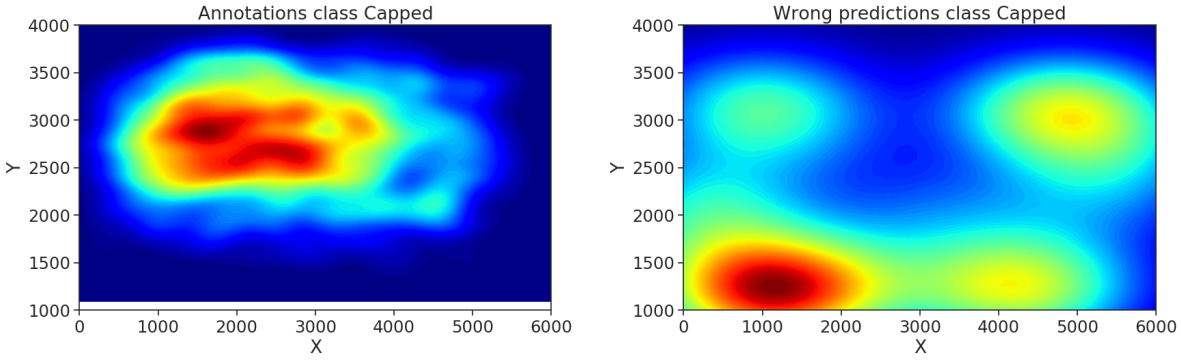


(a) (b)

Figure 6.18: Comparison between most annotated areas (a) and with more errors (b).

We also perform this analysis by class, the figures 6.19(a) and 6.19(b) respectively present the most annotated regions and the areas with more incorrect predictions for the Capped class. The others comparisons are in the appendix J. For the Capped class we can observe that most of the annotations were made in the comb central area, where the bee's eggs posture is most common, while most of the errors occurred at the edges of the combs, where there are commonly drone cells. It is important to the training dataset to have representations of drone cells because they have differences related to the common bee's cells, mainly in their size.

From these analyses, we can verify that for the training of a good classifier not only a large number of annotations are enough since they must be done homogeneously on all



(a) (b)

Figure 6.19: Comparison between most annotated areas (a) and with more errors (b).

regions of the comb, only then they can inform the model during the training the different angles that cells have, and help in generalisation.

6.3 Comparison with Previous Methods

In this section, we will compare the results obtained by the algorithms proposed in the literature with the results presented in this chapter. In the following, we will make comparisons with each approach described in the state of the art (Section 2.1).

- **Cornelissen et al.** [35] Compared its semiautomatic method of counting capped brood cells in comb images with the Lieberfeld method. As a result, they observed that annotations with the Lieberfeld method took 26 seconds on the apiary, while their approach took 19 seconds to capture each frame image and 30 seconds to process each image using the software. In their software, it is necessary to segment the capped area manually and then the software will be able to calculate the number of capped cells.

  Figure 6.20 presents the time distribution for each phase of our cell detection and classification process. This result was obtained by processing all 61 images of the

DS-COMB-SEG-FULL dataset, the detections were performed using the scaled invariant detection algorithm, and the classification was performed using the MobileNet model trained using DA. In the figure, we can observe that the time to completely process an image varies between ≈4 and ≈16 seconds, having the average value of 9.07s. Considering only the average value, the time to photograph a frame and process the image is 28.07s on our hardware. About 2 seconds slower than using the Lieberfeld, when the method is used only for the capped cells.

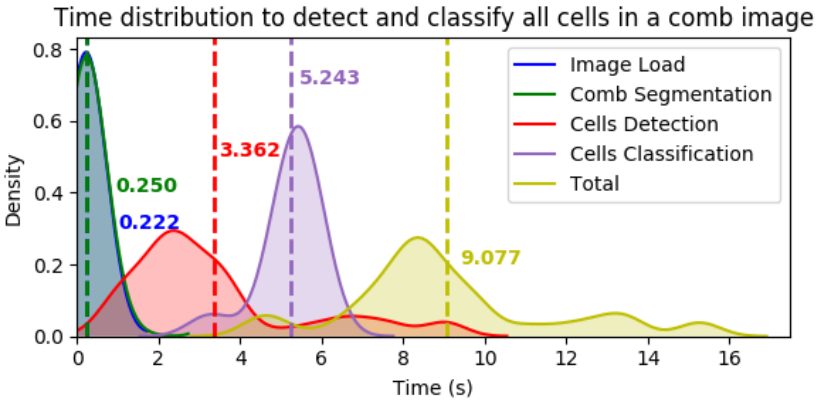


Figure 6.20: Time distribution to detect and classify all cells in a comb image.

Furthermore in the work of Cornelissen et al. [35] it is mentioned that the proposed method predicts correctly 99.37% of the cells against 90.85% when the Lieberfeld method is used. Using CNNs, we achieved 99.47% precision for the Capped class using the MobileNet DA model, using the InceptionResNetV2 DA model this value increased to 99.77%. More results calculated for other classes and models on the test set are presented in appendix I.

- **Rodrigues et al.** [34] analysed the result of their detector and obtained precision and recall of 99.04% and 97.2% respectively for the capped class. In our analyses, we got using the MobileNet DA model accuracy of 99.47% and recall of 99.41%. Other models such as InceptionResNetV2 have obtained superior results for the Capped class as presented in appendix I.

- **Wang and Brewer** [36] in their conference poster, they presented the commercial software for counting capped cells in combs named *HoneybeeComplete*. With this software, they achieved a 97.4% hit rate for the analysed class. This value goes to 99.5% when a region of interest is created by a user. Comparing with our results, we can observe that the result obtained by the MobileNet DA architecture is very close (99.47%) to the Wang and Brewer result when they use human aid.

- **Höferlin et al.** [37] developed the commercial software *HiveAnalyzer*. Unlike previous works, this software is able to classify the cells detected in seven different classes: Empty, Egg, Young Larva, Old Larva, Capped Cell, Nectar and Pollen. In their results, they obtained 94% accuracy on the cells classified with high confidence. From 20.000 cells they owned, 78% were classified with high confidence. So they had 94% accuracy over 78% of all cells in the dataset.

  Seeking to compare our results with the results presented by Höferlin et al. [37] we used 100% of our test dataset DS-COMB-CELL-TEST with 53.914 examples and MobileNet model DA. With them, we obtained an accuracy of 94.31%, very close to the F1-Score value, 94.3%. When we only filter the predictions with confidence higher than 99.6%, we got 42.410 examples or 78.66% of the dataset. The calculated accuracy on this set was 99.35%.

Chapter 7

Other Results and Contributions

In this chapter, we will present other results that we obtained during the development of this work. Namely, these results were a software with a graphical interface that we named DeepBee (Section 7.1) and a website where we shared the source code, the DeepBee, datasets and present more details about the project (Section 7.2).

7.1 DeepBee Software

From the methods presented in chapters 3 and 4 and the results shown in Chapters 5 and 6 we build software for end users. This software we named DeepBee by merging the names Deep Learning and Hive. It has compatibility with Windows and Linux and performs the classification of cells with CPU or GPU. It is subdivided into four modules. We present them below.

- **Detection end classification module:** we developed this module as a script. When executed, it searches for all images in a specified directory, performs the segmentation of the frame, detects cells invariant to scale and classifies each cell found. The cells are classified using the MobileNet model. As output, a serialised file is produced to each image.

- **Viewing and editing module:** in this module, we use the OpenCV and PyQT[1] libraries to build a user interface. When the script starts it scans for the serialised files and the corresponding images. Then, the first image found is displayed, and the detections are drawn on the image. They have colours based on their content (Figure 7.1).

Figure 7.1: Software developed for the interaction of the users with the detections made on their images.

On the left side of the screen it is possible to see the number of cells of each class found, besides some functions that can be carried out on the open image, they are:

  - *Class changing:* using the keys from 1 to 7 the user can choose each class. With a class selected, he can click on any detected cell and change its label. This function can be used when the automatic classifier makes some incorrect prediction;
  - *Move:* by pressing the *Space* key, the user can zoom and move the image without the cell labels being changed;
  - *Select area:* by pressing the *R* key, the user can create a region of interest (Figure 7.2). With a region selected only the cells within this area will be

[1] https://wiki.python.org/moin/PyQt

processed and taken into account. The user must press enter to confirm the region.

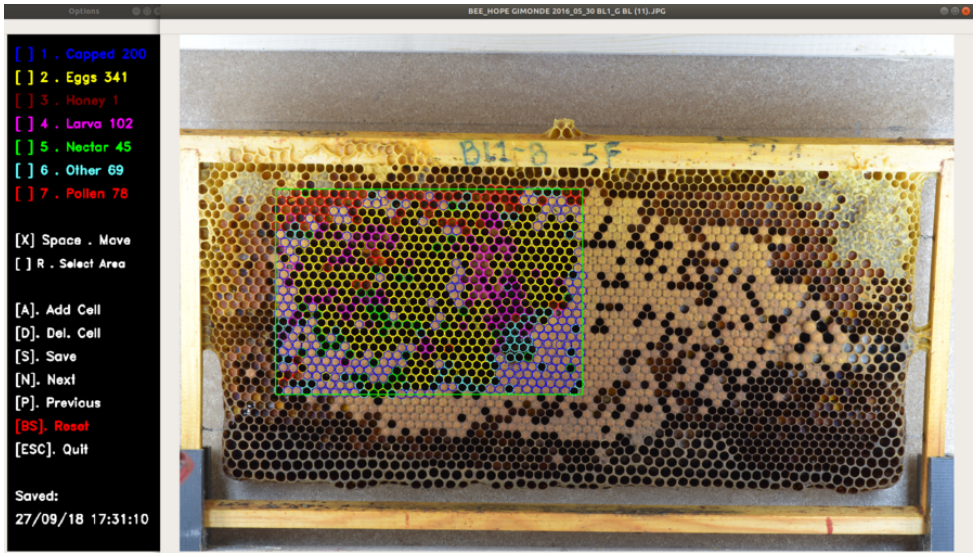
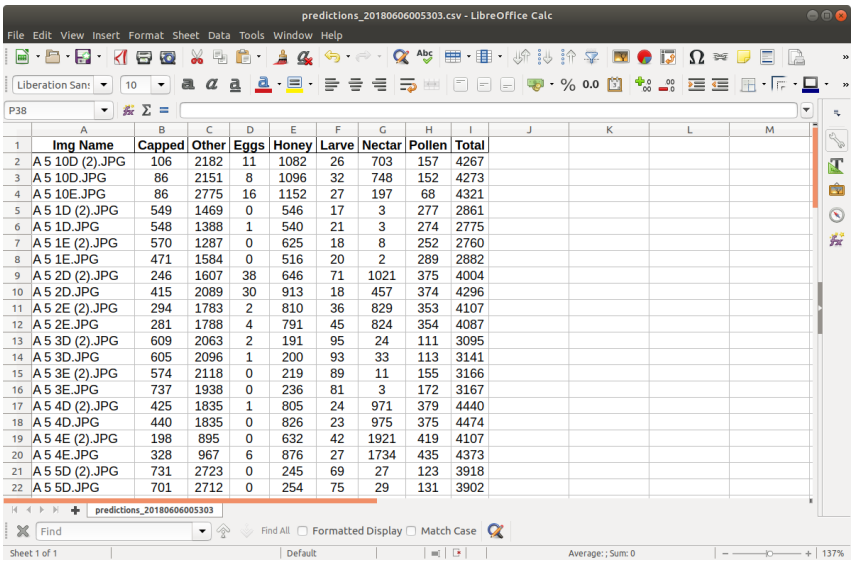


Figure 7.2: Region of interest created by the user.

- *Add Cell:* by pressing the *A* key, the user can add new cells to the image. This option can be used when the software has not detected one or more cells.
- *Delete Cell:* by pressing the *D* key, the user can remove cells from the image. This option can be used when the software has detected cells that do not exist.
- *Save:* by pressing the *S* key, the user-made adjustments will be saved in a serialised file.
- *Next:* pressing the *N* key will display the next image, if all have already been viewed the first will be displayed again.
- *Previous:* pressing the *P* key will display the previous image, the last one will be displayed if the current image is the first one.
- *Reset:* by pressing the *Backspace* key all changes made will be removed and the image will return to its original state.
- *Quit:* pressing the *Esc* key will close the software.

- **Exporting module:** after all the images are processed, and the detections are corrected, the user can use this module to export the results in CSV format. It receives as input a folder containing the files edited by the user and serialised and outputs a CSV file with the number of detections per class of each image. Figure 7.3 presents the results obtained on a set of images that we process.



	A	B	C	D	E	F	G	H	I
1	Img Name	Capped	Other	Eggs	Honey	Larve	Nectar	Pollen	Total
2	A 5 10D (2).JPG	106	2182	11	1082	26	703	157	4267
3	A 5 10D.JPG	86	2151	8	1096	32	748	152	4273
4	A 5 10E.JPG	86	2775	16	1152	27	197	68	4321
5	A 5 1D (2).JPG	549	1469	0	546	17	3	277	2861
6	A 5 1D.JPG	548	1388	1	540	21	3	274	2775
7	A 5 1E (2).JPG	570	1287	0	625	18	8	252	2760
8	A 5 1E.JPG	471	1584	0	516	20	2	289	2882
9	A 5 2D (2).JPG	246	1607	38	646	71	1021	375	4004
10	A 5 2D.JPG	415	2089	30	913	18	457	374	4296
11	A 5 2E (2).JPG	294	1783	2	810	36	829	353	4107
12	A 5 2E.JPG	281	1788	4	791	45	824	354	4087
13	A 5 3D (2).JPG	609	2063	2	191	95	24	111	3095
14	A 5 3D.JPG	605	2096	1	200	93	33	113	3141
15	A 5 3E (2).JPG	574	2118	0	219	89	11	155	3166
16	A 5 3E.JPG	737	1938	0	236	81	3	172	3167
17	A 5 4D (2).JPG	425	1835	1	805	24	971	379	4440
18	A 5 4D.JPG	440	1835	0	826	23	975	375	4474
19	A 5 4E (2).JPG	198	895	0	632	42	1921	419	4107
20	A 5 4E.JPG	328	967	6	876	27	1734	435	4373
21	A 5 5D (2).JPG	731	2723	0	245	69	27	123	3918
22	A 5 5D.JPG	701	2712	0	254	75	29	131	3902

Figure 7.3: CSV file generated.

From the data saved in this file the user can make more in-depth analyses on the health of their hives.

- **Training module:** with this module, also developed as a script, the user can use the corrections made by on the detections and train a new model. This new model will also use the MobileNet architecture. It is expected that the model will be more finely tuned for user comb frame images after this training.

7.2 Website to Host this Project

For this project to be accessible to more people, so that our results can be reproduced and so that third parties can add improvements, we have decided to make it openly available.

We have developed the website for this project using the free tool GitHub Pages[2], the link to access it is: *https://avsthiago.github.io/DeepBee/*. The homepage of the website is shown in figure 7.4.

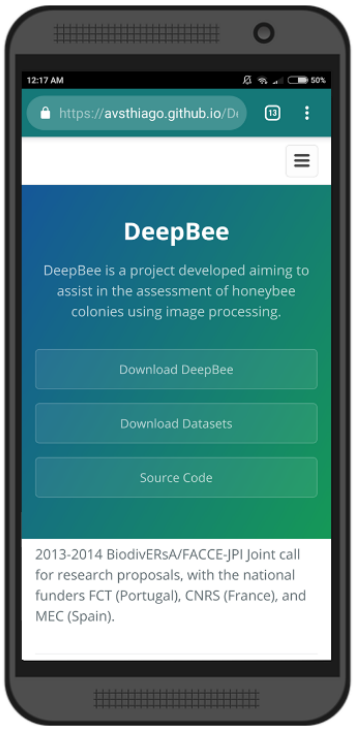


Figure 7.4: Website developed to share this project.

On this site we provide links to our datasets, the DeepBee software, and the source code hosted on GitHub.

[2]https://pages.github.com/

Chapter 8

Future Work

During the development of our work different approaches were developed and tested in order to solve the challenge of detecting and classifying cells in comb images. We have succeeded in many of our formulated hypotheses, and as a result, we have generated tools that are easy for researchers to use without in-depth knowledge of computing. However, as is common in research projects, our work has also opened new gaps to be closed with future works. The following will list some relevant points to be addressed in future developments. These include methods that can help improve the results from different perspectives (quality of classifications and processing time) and other software formats for the end user.

- **Get more examples for training:** not just get more examples, but new annotations made uniformly on the combs. As we have shown in section 6.2.5, an imbalanced number of annotations among comb regions tends to generate models with biases in better-classifying regions with more annotations. Thus, making annotations in diverse positions in the comb will help the model understand the particularities of each region and then generalise the results to all positions.

  One way to annotate cells uniformly in the combs is to annotate all of them. This task is tedious to do when the user needs to set the label for each cell in the image. Thus, it is possible to use the model that we previously trained and classify cells,

classifications with confidence lower than a threshold (e.g. 99%) can be defined as Uncategorized, and then the user just needs to define the class of those cells. From these new annotations, new models can be trained. The figure 8.1 shows an image of a comb with its labelled cells, the blank cells are those that were predicted with confidence less than 99.95%. The calculated accuracy for detections with confidence greater than 99.95% is 99.79%.

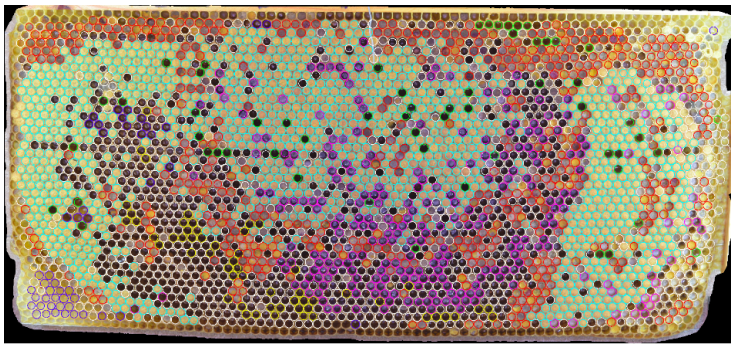


Figure 8.1: Cells predicted with low confidence.

Our DS-COMB-PT dataset has 2204 frame pictures, on these pictures were annotated 71.915 cells for the DS-COMB-CELL-PT dataset and 34.000 cells for the DS-COMB-CELL-TEST dataset. On average, a comb has 3000 cells. Therefore, we only use 1.6% of all cells present in our dataset. Although we have achieved quite impressive results with the number of cells we have annotated, we know that we have the possibility of obtaining better results by annotating more cells from our dataset and others in the future.

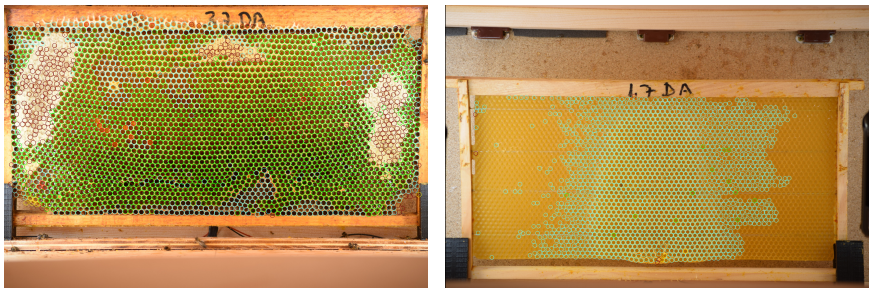
- **Get more examples for training the segmentation network :** another piece of our work that may benefit from a larger number of examples for training is the honeycomb segmenter. In our tests, we used only frames images belonging to the DS-COMB-PT dataset. All these images have the same background and lighting conditions. Although we have achieved significant results for segmenting images out of the DS-COMB-PT dataset (Section 5.3.3), we believe that they can be improved, especially with new annotations made with different backgrounds.

- **Test other input sizes for the segmentation network:** our first approach for combs segmentation was made with tiles of size 128×128px. With this approach, we managed to reduce to almost zero the number of false detections made outside the honeycomb region, but we did not test different sizes of tiles. It is possible that using larger tiles (e.g. 256×256 or 512×512) better results are obtained since more context will be given to the network.

- **Test ensemble classifiers:** an ensemble consists of a set of individually trained classifiers whose predictions are combined when classifying novel instances [124]. A simple way to improve the results of almost any classifier is to train many different models on the same data and then to average their predictions [125]. Based on these two statements, we can hypothesise that training different models to classify cells and to average their predictions may result in significant improvements in our results. Until now, all our tests were done using one classifier at a time. Specialised classifiers can be created for each class, or groups of classes and then combine their predictions. The output of the classifiers should be changed from Softmax to linear until to perform the average of the results since the Softmax will accentuate the difference between the classes. After merging the results, the Softmax activation function can be applied, and to transform the result into probabilities.

- **Filling areas without detections:**we noticed that in some specific cases where there is no cell in the comb (only wax), or there are large white areas with honey, there are cells that are not detected (Figure 8.2). Although these false detections are rare, this problem exists and is worth proposing a solution for it.

  We have analysed and found that even cells not being detected by our method proposed in section 3.3 are part of the honeycomb segmentation made by the method CSS-LC, in most cases. Although it is not possible to observe the exact contour of cells with honey in some cases, their positions can be predicted. Is possible to create this prediction in five steps:



(a) (b)

Figure 8.2: Some false detections.

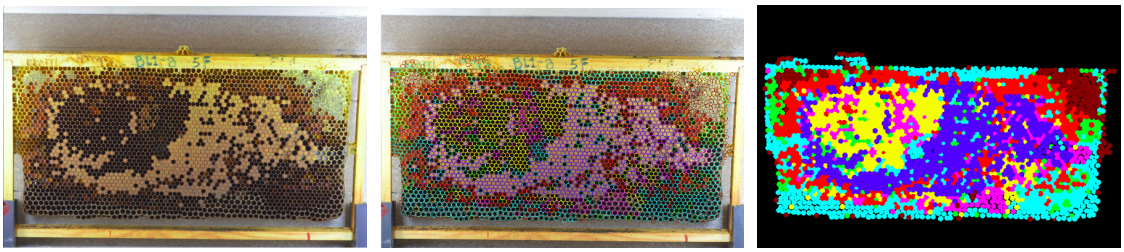
1. Clip only the comb from the image using the enclosing rectangle obtained from the segmentation;;
2. Create a new image with the same size as the clipped from in step 1, and fill it uniformly with circles with the same radius of the detected cells. The position of the created circles must be stored in an array;
3. Remove from the points array those that are not over the comb (use segmentation);
4. Compare distance of each detection made by cHT with the predicted positions. This task can be done efficiently with the cdist[1] method from SciPy library, in our preliminary tests it is possible to calculate the distance between all points of two arrays with 3000 positions in less than one second using this method;
5. Remove the predictions made with distances smaller than 2x cells radius, until some real detection.

- **Create an end-to-end network for this problem:** it is common to use end-to-end networks to solve problems that would otherwise require a complete pipeline. Examples of end-to-end solutions were made to automatically create captions for images in the work *DenseCap: Fully Convolutional Localization Networks for Dense Captioning* [126], and to drive cars autonomously in the publication *End to end*

[1]https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.spatial.distance.cdist.html

learning for self-driving cars [127]. We believe an end-to-end solution can be made for this project.

The complete replacement of our pipeline by a DNN could be made using the semantic segmentation technique, as we used in section 3.5.3, but rather than separating the comb from the background, segmentations with different colours for each class would be created. This new architecture can have as output an image with its segmentation, seven values corresponding to the number of cells per class or both results. The annotations of the regions can be made the drawing the results generated by our current methodology in a black image (Figure 8.3).



(a) (b) (c)

Figure 8.3: (a) original image, (b) image with detections, (c) detections drawn in a black image.

The instance segmentation technique can also be used. This technique will try to detect and create segmentation for cells individually [128]. Solutions for regions with honey need to be found to make this approach feasible.

- **Compare with human analyses:** in this work, we compared automatic annotated, and human annotated cell contents in images. Future work can compare the performance of our method with the approximate method of Lieberfeld and with the method of hive weighing.

- **Develop an API and offer our project as a service:** our pipeline has some points that require a considerable amount of computational processing. Especially older computers that do not have GPUs can take minutes to process each image.

Ensuring DeepBee works the same way on all available hardware is another challenge. There is a way to minimise these problems by having all the processing done online by a service. Changing the software to work in this way will allow the user to send images to a server, which in turn will detect and classify the cells and return a JavaScript Object Notation (JSON) containing the results. Then, the user can use the DeepBee viewing and editing module to change the predictions if necessary.

To create this API, one can use the tutorial *Deep learning in production with Keras, Redis, Flask, and Apache* created in the PyImageSearch[2] blog as a base. A server where it may be possible to install this service is the cluster that CeDRI has in IPB.

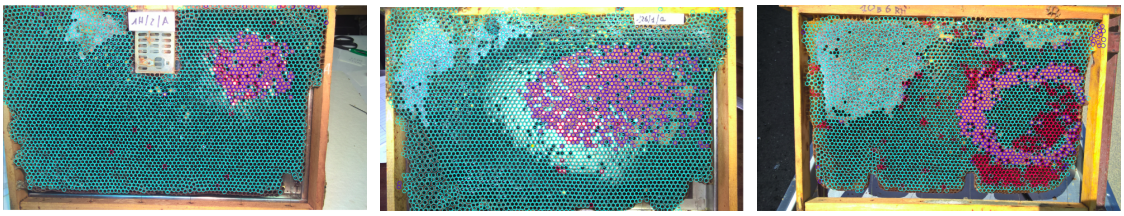
Another alternative to hosting this service is the Algorithmia[3] website. It supports algorithms developed with different DL frameworks including Keras and Caffe. It is possible to host an algorithm on it for free. In order for users to access the API, they must pay. Payment is made according to the processing time.

- **Develop a mobile application:** at the beginning of this project we had the idea of developing a mobile application (Appendix A), it would be an embedded system with the NVIDIA® Jetson TX2 module. With this embedded solution, the analyses would be done in the field. In the course of the project, we found that it was more worthwhile to do the processing using a computer after collecting the data, this advantage is due to the need of batteries that the embedded module would have and the difficulty that the users would have in acquiring the device.

  Unlike using an embedded device, an alternative to making this project more affordable is to develop a smartphone application. Although the MobileNet architecture is designed to work on smartphones, re-creating our entire pipeline as an app would result in a slow solution since the average number of cells to be classified in a comb is about 3000. Thus, it would be worthwhile for the mobile application to access an API like the one we described in the previous item.

[2]https://www.pyimagesearch.com/2018/02/05/deep-learning-production-keras-redis-flask-apache/
[3]https://algorithmia.com/

Another point to be analysed is the image quality of smartphones. It is increasing with the new models released, but the quality is still lower than that of professional cameras. In our preliminary tests we trained a model with only three classes (capped, larva and other), the images we used for training were taken using a professional camera. The 8.4 image shows the results we obtained by processing images from the DS-COMB-CREA dataset (made by smartphone images).



(a) (b) (c)

Figure 8.4: Images from the DS-COMB-CREA with its cells detected.

We can observe in figure 3 that the comb had few false cell detections. About the content classification, there were few confusions with the capped cells and many with larvae cells. Ways to improve these results would be by train a new model on annotations made in smartphone images, photographing the frames with more controlled light and a capturing the frames with a distance that allows the internal visualisation of the highest number of cells possible.

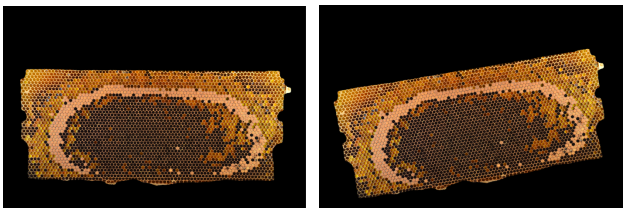
If the goal is just count capped cells, we believe it is possible to use the semantic segmentation technique. In this case, it is possible to fully process the image on the device.

- **Create a method to track the development of cells:** Another analysis can be done by researchers when the development of the cells is monitored over a period. With these data, it is possible to monitor the rate of mortality of brood and to examine the levels of food reserves in different moments, for example. For this monitoring to be done cell by cell in images, is necessary a method to associate the cells of one image with those of another. In the method proposed by Kimmel et

al. [31], Markers are placed at the edges of the frame, and from them, the cells of different images are aligned.

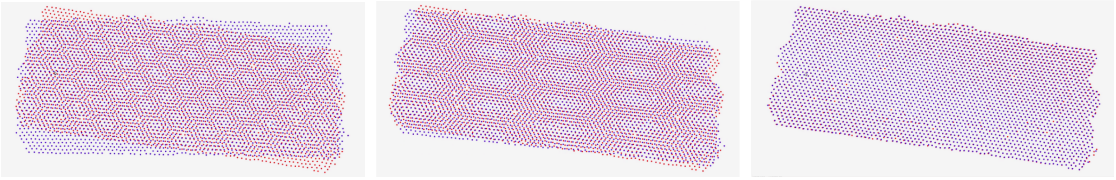
We believe it is possible to align photos without the use of physical markers. Using methods like Random sample consensus (RANSAC) can be found similar features in two images. After finding these features, a transformation matrix can be created to overlap the cells in two images.

We performed preliminary tests using a segmented honeycomb image in two orientations (Figure 8.5). After, we detect the cells of both images and save them. Using the Python-CPD[4] module we load the detected points, and with the iterative method called *Coherent Point Drift Algorithm* we find the correspondences between the two sets of points, figure 8.6 shows some steps of the algorithm until its final convergence. After convergence of the algorithm, it is possible to discover the new location of each cell in the new image and then make the analyses about what changed in the cell between the two images.



(a) (b)

Figure 8.5: Segmented images for testing with Coherent Point Drift Algorithm.



(a) (b) (c)

Figure 8.6: Different steps of Coherent Point Drift Algorithm until its convergence.

[4]https://github.com/siavashk/pycpd

These are some ideas that we believe are the next steps to improve our results. Because this project is open source, we hope that we will have new developers interested on it and continue with us. New ideas are always welcome in our repository: *https://github.com/AvsThiago/DeepBee-Resources*.

Chapter 9

Conclusion

Counting the number of comb cells with brood and food reserves allows analyses related to *Apis mellifera* colony strength to be made. Among the approaches to perform this count are the estimated methods of Lieberfeld and Acetate Sheet and the methods of assessment digitally using comb images. In this context, the present work aimed to develop a fully automatic method to (i) detect cells in comb images regardless of their scale, (ii) to classify the contents of each cell into seven different categories. Besides, it was also our goal to develop graphical software capable of presenting the results to the user and allowing them to make adjustments if cells were detected or classified incorrectly.

During the development of the method for detecting cells, we noticed that the cHT method provides a high detection rate of cells according to the parameters choice. Although the least restrictive parameters that we found generate a large number of false detections, they were fundamental for finding cells in regions with honey. In these regions, in some cases, it is impossible to locate the edges of the cells visually.

To remove the false detections generated by the cHT method, we developed three approaches. Among them, the one that showed superior results was the CSS-LC. Analysing the results produced by this approach we can conclude that using CNNs based on the U-Net model provides a very robust comb segmentation, even in images of combs photographed under conditions different from those used in training. With this methodology, we obtained a cell detection rate superior to that obtained by Lee et al. [19]. We believe

that our false detection removal method could have better results by training the model with larger tiles (e.g. 256×256px) and also using images from different datasets.

We can also verify that using CNNs for the classification of cells extracted from combs provides results with high precision and recall. During our studies, we found that extracting cells from our dataset images with size 224×224px produces superior results than other sizes tested (when accuracy is considered). Comparing 13 different architectures we noticed that for our classification problem the MobileNet and InceptionResNetV2 architectures have the quality of their results very similar, but concerning the resources needed, the MobileNet is more efficient in all analysed aspects. Regarding that we did not find previous works that used CNNs to classify comb cells and that our approach using CNNs generated results superior to the results obtained by prior methods, we can confirm the ability of CNNs to achieve the state of the art in classification tasks.

Lastly, we expect DeepBee software to become a standard software for colony assessment among apicultural researchers. For this to be possible, we hope new contributors to join us in the future developments of this open source project. We also expect that our source codes, models, datasets and insights made available will contribute fomenting the research in the apidology field using computer vision methods.

Bibliography

[1] R. Rossi, *The eu's beekeeping sector*. Oct. 2017. [Online]. Available: http://www.europarl.europa.eu/RegData/etudes/ATAG/2017/608786/EPRS_ATA(2017)608786_EN.pdf.

[2] U. D. of Agriculture National Agricultural Statistics Service, *Honey*, Mar. 2018. [Online]. Available: https://usda.mannlib.cornell.edu/MannUsda/viewDocumentInfo.do?documentID=1191.

[3] N. W. Calderone, "Insect pollinated crops, insect pollinators and us agriculture: Trend analysis of aggregate data for the period 1992-2009," *PLoS One*, vol. 7, no. 5, G. Smagghe, Ed., e37235, May 2012, ISSN: 1932-6203. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3358326/.

[4] B. Dennis and W. P. Kemp, "How hives collapse: Allee effects, ecological resilience, and the honey bee," *PLoS One*, vol. 11, no. 2, J. A. Marshall, Ed., e0150055, Feb. 2016, ISSN: 1932-6203. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4765896/.

[5] D. Cressey, *Eu states lose up to one-third of honeybees per year*, Apr. 2014. [Online]. Available: https://www.nature.com/news/eu-states-lose-up-to-one-third-of-honeybees-per-year-1.15016.

[6] N. Council, D. Studies, B. Resources, B. Sciences, and C. America, *Status of pollinators in north america*. National Academies Press, 2007, ISBN: 9780309102896.

[7] E. Commission, *Report from the commission to the european parliament and the council on the implementation of the measures concerning the apiculture sector of council regulation (ec) no 1234/2007*, Aug. 2013. [Online]. Available: https://eur-lex.europa.eu/legal-content/en/ALL/?uri=CELEX:52013DC0593.

[8] "Bee mortality and bee surveillance in europe - a report from the assessment methodology unit in response to agence francaise," *EFSA Journal*, vol. 6, no. 8, Aug. 2008. DOI: 10.2903/j.efsa.2008.154r. [Online]. Available: http://www.efsa.europa.eu/en/efsajournal/pub/rn-154.

[9] *Eurbest*. [Online]. Available: https://www.eurbest.eu/.

[10] *Smartbees - sustainable management of resilient bee populations*, Nov. 2014. [Online]. Available: http://www.smartbees-fp7.eu/.

[11] *Swarmonitor*, Nov. 2012. [Online]. Available: https://cordis.europa.eu/project/rcn/105847_en.html.

[12] *Beehope*. [Online]. Available: https://www.biodiversa.org/1077.

[13] *Regulation (ec) no 1107/2009 of the european parliament and of the council of 21 october 2009 concerning the placing of plant protection products on the market and repealing council directives 79/117/eec and 91/414/eec*, Jul. 2013. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32009R1107.

[14] E. F. S. A. (EFSA), "Efsa guidance document on the risk assessment of plant protection products on bees (apis mellifera, bombus spp. and solitary bees)," *EFSA Journal*, vol. 11, no. 7, pp. 219–220, 2013. DOI: 10.1002/efs2.2013.11.issue-7. [Online]. Available: https://www.efsa.europa.eu/en/efsajournal/pub/3295.

[15] C. Costa, B. Ralph, S. Berg, M. Bienkowska, M. Bouga, D. Bubalo, L. Charistos, Y. Le Conte, M. Dražić, W. Dyrba, J. Filipi, F. Hatjina, E. Ivanova, N. Kezić, H. Kiprijanovska, M. Kokinis, S. Korpela, P. Kryger, M. Lodesani, and J. Wilde, "A

europe-wide experiment for assessing the impact of genotype-environment interactions on the vitality and performance of honey bee colonies: Experimental design and trait evaluation," vol. 56, pp. 147–158, Jul. 2012.

[16] A. Imdorf, G. Buehlmann, L. Gerig, V. Kilchenmann, and H. Wille, "A test of a method of estimating brood areas and the number of worker honeybees in free-flying colonies," *Apidologie*, vol. 18, no. 2, pp. 137–146, 1987. DOI: 10.1051/apido:19870404. [Online]. Available: https://eurekamag.com/research/001/520/001520597.php.

[17] R. Schwarz, *Ein hobby in dadant*. [Online]. Available: https://dadant-imkern.blogspot.com/2015/04/liebefelder-schatzmethode.html.

[18] K. S. Delaplane, J. V. D. Steen, and E. Guzman-Novoa, "Standard methods for estimating strength parameters of apis mellifera colonies," *Journal of Apicultural Research*, vol. 52, no. 1, 2013. DOI: 10.3896/ibra/1.52.1.03. [Online]. Available: https://www.tandfonline.com/doi/ref/10.3896/IBRA.1.52.1.03.

[19] L. H. Liew, B. Y. Lee, and M. Chan, "Cell detection for bee comb images using circular hough transformation," in *2010 International Conference on Science and Social Research (CSSR 2010)*, Dec. 2010, pp. 191–195. DOI: 10.1109/CSSR.2010.5773764.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. eprint: 1502.01852.

[21] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 1701–1708. DOI: 10.1109/CVPR.2014.220.

[22] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van

den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–, Oct. 2017.

[23] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The microsoft 2017 conversational speech recognition system," *CoRR*, vol. abs/1708.06073, 2017.

[24] D. Teney, P. Anderson, X. He, and A. van den Hengel, "Tips and tricks for visual question answering: Learnings from the 2017 challenge," *CoRR*, vol. abs/1708.02711, 2017.

[25] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[26] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017.

[27] T. d. S. Alves, P. J. Rodrigues, A. Pinto, A. Candido Junior, P. L. de Paula Filho, P. Ventura, and C. Neves, *Assessment of honey bee cells using deep learning*, Sep. 2018.

[28] R. Becker and J. Lückmann, *Variability of brood termination rate in honeybee brood studies under semi-field conditions*, Mar. 2011.

[29] J. Pistorius, R. Becker, J. Lückmann, A. Schur, M. Barth, L. Jeker, S. Schmitzer, and W. von der Ohe, *Effectiveness of method improvements to reduce variability of brood termination rate in honey bee brood studies under semi-field conditions*, Jan. 2012.

[30] J. Maehl, *Standard methods for toxicology research in apis mellifera*, Jan. 2018. [Online]. Available: `http://www.coloss.org/standard-methods-for-toxicology-research-in-apis-mellifera/`.

[31] S. Kimmel, L. Jeker, J. Magyar, and T. Schmidt, *Evaluation of bee brood development using automated digital image processing and analysisd evaluation*, Sep. 2010.

[32] B. Emsen, "Semi-automated measuring capped brood areas of honey bee colonies," *Journal of Animal and Veterinary Advances*, vol. 5, no. 12, pp. 1229–1232, 2006. [Online]. Available: http://medwelljournals.com/abstract/?doi=javaa.2006.1229.1232.

[33] M. Yoshiyama, K. Kimura, K. Saitoh, and H. Iwata, "Measuring colony development in honey bees by simple digital image analysis," *Journal of Apicultural Research*, vol. 50, no. 2, pp. 170–172, 2011. DOI: 10.3896/IBRA.1.50.2.10. eprint: https://doi.org/10.3896/IBRA.1.50.2.10. [Online]. Available: https://doi.org/10.3896/IBRA.1.50.2.10.

[34] P. J. Rodrigues, C. Neves, and M. A. Pinto, "Geometric contrast feature for automatic visual counting of honey bee brood capped cells," in *EURBEE 2016: 7th European Conference of Apidology*, 2016.

[35] B. Cornelissen, S. Schmid, J. v. d. Steen, and T. Blacquière, *Estimating honeybee colony size using digital photography*, Oct. 2009.

[36] M. Wang and L. Larry, *New computer methods for honeybee colony assessments*. [Online]. Available: http://sesss08.setac.eu/embed/sesss08/Larry_Brewer_-_New_Computer_Methods_for_Honeybee_Colony_Assessments.pdf.

[37] B. Höferlin, M. Höferlin, M. Kleinhenz, and H. Bargen, "Automatic analysis of apis mellifera comb photos and brood development," *Apidologie*, vol. 44, p. 19, Mar. 2013. [Online]. Available: https://www.springer.com/cda/content/document/cda_downloaddocument/AGIB%20-%20Abstracts%202013_Final.pdf?SGWID=0-0-45-1417002-p174076256.

[38] G. Anbarjafari, *Introduction to image processing*. [Online]. Available: https://sisu.ut.ee/imageprocessing/book/1.

[39] R. C. Gonzalez and R. E. Woods, *Digital image processing*, 2nd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992, ISBN: 0201508036.

[40] T. Moeslund, *Introduction to video and image processing: Building real systems and applications*, ser. Undergraduate Topics in Computer Science. Springer London, 2012, ISBN: 9781447125037.

[41] S. Krig, *Computer vision metrics: Survey, taxonomy, and analysis*, ser. SpringerLink : Bücher. Apress, 2014, ISBN: 9781430259305. [Online]. Available: https://books.google.com.br/books?id=ktKuAwAAQBAJ.

[42] D. Fleet and A. Jepson, *Image segmentation*. [Online]. Available: http://www.cs.toronto.edu/~jepson/csc2503/segmentation.pdf.

[43] E. Spyrou, H. Le Borgne, T. Mailis, E. Cooke, Y. Avrithis, and N. O'Connor, "Fusing mpeg-7 visual descriptors for image classification," in *Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005*, W. Duch, J. Kacprzyk, E. Oja, and S. Zadrożny, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 847–852, ISBN: 978-3-540-28756-8.

[44] G. A. M. d. de Deus, D. P. Fernandes, and C. d. F. dos Santos, "Uma proposta de processamento digital de ovoscopia," *VIII ENACOMP*, vol. 8, p. 3, 2010. [Online]. Available: http://www.enacomp.com.br/2010/anais/artigos/resumidos/enacomp2010_36.pdf.

[45] J. Stark and W. Fitzgerald, "An alternative algorithm for adaptive histogram equalization," *Graphical Models and Image Processing*, vol. 58, no. 2, pp. 180–185, 1996, ISSN: 1077-3169.

[46] OpenCV, *Opencv: Histograms - 2: Histogram equalization*. [Online]. Available: https://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html.

[47] K. Zuiderveld, "Contrast Limited Adaptive Histograph Equalization," in, P. S. Heckbert, Ed., San Diego, CA, USA: Academic Press Professional, Inc., 1994, ch. Contrast limited adaptive histogram equalization, pp. 474–485, ISBN: 0-12-336155-9.

[48] J. C. Russ, *The image processing handbook (3rd ed.)* Boca Raton, FL, USA: CRC Press, Inc., 1999, ISBN: 0-8493-2532-3.

[49] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Jan. 1998, pp. 839–846. DOI: 10.1109/ICCV.1998.710815.

[50] X. Xia and R. Du, *Parallel bilateral filtering using opencl*, Dec. 2015. [Online]. Available: http://xidexia.github.io/Bilateral-Filtering/#.

[51] G. Bradski and A. Kaehler, *Learning opencv*. OReilly Media, 2015, pp. 151–161.

[52] U. Sinha, *Introduction the edge detector*. [Online]. Available: http://www.aishack.in/tutorials/canny-edge-detector/.

[53] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972, ISSN: 0001-0782. DOI: 10.1145/361237.361242. [Online]. Available: http://doi.acm.org/10.1145/361237.361242.

[54] M. Cheselka, "Automatic detection of linear features in astronomical images," in *Astronomical Data Analysis Software and Systems VIII*, vol. 172, 1999, p. 349.

[55] T. Butler-Yeoman, M. Frean, C. Hollitt, D. Hogg, and M. Johnston-Hollitt, "Detecting diffuse sources in astronomical images," *ArXiv preprint arXiv:1601.00266*, 2016.

[56] S. N. Gieser, J. Tompkins, A. Sharifara, and F. Makedon, "Using humanoid robot to instruct and evaluate performance of a physical task," *CoRR*, vol. abs/1805.02249, 2018. arXiv: 1805.02249. [Online]. Available: http://arxiv.org/abs/1805.02249.

[57] J. Climent and R. A. Hexsel, "Particle filtering in the hough space for instrument tracking," *Computers in Biology and Medicine*, vol. 42, no. 5, pp. 614–623, 2012, ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2012.02.007.

[Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010482512000455.

[58] F. Zana and J. C. Klein, "A multimodal registration algorithm of eye fundus images using vessels detection and hough transform," *IEEE Transactions on Medical Imaging*, vol. 18, no. 5, pp. 419–428, May 1999, ISSN: 0278-0062. DOI: 10.1109/42.774169.

[59] C. Shu and G. Roth, *Introduction to computer vision*. [Online]. Available: http://people.scs.carleton.ca/~c_shu/Courses/comp4900d/notes/lect10_hough.pdf.

[60] C. Kimme, D. Ballard, and J. Sklansky, "Finding circles by an array of accumulators," *Communications of the ACM*, vol. 18, no. 2, pp. 120–122, 1975.

[61] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, *Hough transform*, 2003. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm.

[62] G. Bradski and A. Kaehler, *Learning opencv: Computer vision in c++ with the opencv library*, 2nd. O'Reilly Media, Inc., 2013, ISBN: 1449314651, 9781449314651.

[63] M. Childs, *John mccarthy: Computer scientist known as the father of ai*, Oct. 2011. [Online]. Available: https://www.independent.co.uk/news/obituaries/john-mccarthy-computer-scientist-known-as-the-father-of-ai-6255307.html.

[64] T. Lozano-Pérez and L. Kaelbling, *Techniques in artificial intelligence*. [Online]. Available: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-825-techniques-in-artificial-intelligence-sma-5504-fall-2002/lecture-notes/Lecture1Final.pdf.

[65] B. Copeland, *Artificial intelligence*, Aug. 2018. [Online]. Available: https://www.britannica.com/technology/artificial-intelligence.

[66] W. L. Hosch, *Machine learning*, Sep. 2016. [Online]. Available: https://www.britannica.com/technology/machine-learning.

[67] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*, ser. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014, ISBN: 9781107057135. [Online]. Available: https://books.google.pt/books?id=ttJkAwAAQBAJ.

[68] B. Marr, *The key definitions of artificial intelligence (ai) that explain its importance*, Feb. 2018. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/02/14/the-key-definitions-of-artificial-intelligence-ai-that-explain-its-importance/#7cfb529f4f5d.

[69] S. Russell, S. Russell, P. Norvig, and E. Davis, *Artificial intelligence: A modern approach*, ser. Prentice Hall series in artificial intelligence. Prentice Hall, 2010, ISBN: 9780136042594. [Online]. Available: https://books.google.com.br/books?id=8jZBksh-bUMC.

[70] Nielsen and M. A., *Neural networks and deep learning*, Jan. 2017. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap1.html.

[71] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[72] S. HAYKIN, *Redes neurais - 2ed.* BOOKMAN COMPANHIA ED, 2001, ISBN: 9788573077186.

[73] M. Hagan, H. Demuth, and M. Beale, *Neural network design*. Martin Hagan, 2014, ISBN: 9780971732117. [Online]. Available: https://books.google.com.br/books?id=4EW9oQEACAAJ.

[74] N. Rochester, J. Holland, L. Haibt, and W. Duda, "Tests on a cell assembly theory of the action of the brain, using a large digital computer," *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 80–93, Sep. 1956, ISSN: 0096-1000. DOI: 10.1109/TIT.1956.1056810.

[75] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[76] A. de Pádua Braga, *Redes neurais artificiais: Teoria e aplicações*. LTC Editora, 2007, ISBN: 9788521615644. [Online]. Available: https://books.google.com.br/books?id=R-p1GwAACAAJ.

[77] Iyoda, E. Masato, and F. J. V. Zuben, "Inteligencia computacional no projeto automatico de redes neurais hibridas e redes neurofuzzy heterogeneas," PhD thesis, UNICAMP, 2000. [Online]. Available: http://repositorio.unicamp.br/jspui/handle/REPOSIP/259071.

[78] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[79] B. Reis, *Redes neurais – funções de ativação*, Jul. 2016. [Online]. Available: http://www.decom.ufop.br/imobilis/redes-neurais-funcoes-de-ativacao/.

[80] L. Deng and D. Yu, "Deep learning: Methods and applications," Tech. Rep., May 2014. [Online]. Available: https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/.

[81] P. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Harvard University, 1975. [Online]. Available: https://books.google.pt/books?id=z81XmgEACAAJ.

[82] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[83] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, p. 533, 1986.

[84] D. B. Parker, "Learning logic," 1985.

[85] A. C. G. Vargas, A. Paes, and C. N. Vasconcelos, "Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres," in *Proceedings of the XXIX Conference on Graphics, Patterns and Images*, 2016, pp. 1–4.

[86] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*, Springer, 1982, pp. 267–285.

[87] A. Durville, *Computer vision with convolution networks*, Feb. 2017. [Online]. Available: https://github.com/OKStateACM/AI_Workshop/wiki/Computer-Vision-with-Convolution-Networks.

[88] P. Franceus, *From a remote village*, Mar. 2007. [Online]. Available: http://fromaremotevillage.blogspot.com/2007/03/cocoa-application-with-custom-core.html.

[89] U. Karn, *An intuitive explanation of convolutional neural networks*, May 2017. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[90] S. Pio and F. M. Ribeiro, *Reconhecimento de caracteres em imagens com ruído*. 2014.

[91] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. arXiv: 1311.2901. [Online]. Available: http://arxiv.org/abs/1311.2901.

[92] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013.

[93] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[94] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015. eprint: 1512.00567.

[95] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. eprint: 1512.03385.

[96] A. Veit, M. J. Wilber, and S. J. Belongie, "Residual networks are exponential ensembles of relatively shallow networks," *CoRR*, vol. abs/1605.06431, 2016. eprint: 1605.06431.

[97] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," *CoRR*, vol. abs/1603.09382, 2016.

[98] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[99] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. arXiv: 1608.06993. [Online]. Available: http://arxiv.org/abs/1608.06993.

[100] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016. eprint: 1610.02357.

[101] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016.

[102] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *CoRR*, vol. abs/1603.05027, 2016.

[103] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *CoRR*, vol. abs/1703.01041, 2017.

[104] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017.

[105] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html.

[106] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, "The penn treebank: Annotating predicate argument structure," in *Proceedings of the Workshop on Human Language Technology*, ser.

HLT '94, Plainsboro, NJ: Association for Computational Linguistics, 1994, pp. 114–119, ISBN: 1-55860-357-3.

[107] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *CoRR*, vol. abs/1707.07012, 2017. eprint: 1707.07012.

[108] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *CoRR*, vol. abs/1709.01507, 2017. arXiv: 1709.01507. [Online]. Available: http://arxiv.org/abs/1709.01507.

[109] L. Sifre and S. Mallat, "Rigid-motion scattering for texture classification," *CoRR*, vol. abs/1403.1687, 2014. arXiv: 1403.1687. [Online]. Available: http://arxiv.org/abs/1403.1687.

[110] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[111] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. arXiv: 1801.04381. [Online]. Available: http://arxiv.org/abs/1801.04381.

[112] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2014.

[113] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 3320–3328. [Online]. Available: http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf.

[114] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[115] G. Jyothi, C. Sushma, and D. Veeresh, "Luminance based conversion of gray scale image to rgb image," *International Journal of Computer Science and Information Technology Research*, vol. 3, no. 3, pp. 279–283, 2015.

[116] Phillip, *Why i have pollen in my honey super*, Sep. 2016. [Online]. Available: `https://mudsongs.org/why-i-have-pollen-in-my-honey-super/`.

[117] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[118] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. eprint: `1505.04597`.

[119] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017, ISSN: 0162-8828.

[120] M. Ghafoorian, J. Teuwen, R. Manniesing, F. de Leeuw, B. van Ginneken, N. Karssemeijer, and B. Platel, "Student beats the teacher: Deep neural networks for lateral ventricles segmentation in brain MR," *CoRR*, vol. abs/1801.05040, 2018. eprint: `1801.05040`.

[121] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015.

[122] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[123] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[124] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, Jan. 1999. DOI: 10.1613/jair.614.

[125] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *ArXiv preprint arXiv:1503.02531*, 2015.

[126] J. Johnson, A. Karpathy, and L. Fei-Fei, "Densecap: Fully convolutional localization networks for dense captioning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[127] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *ArXiv preprint arXiv:1604.07316*, 2016.

[128] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. arXiv: 1703.06870. [Online]. Available: http://arxiv.org/abs/1703.06870.

Appendix A

Original Project Proposal

Proposta de tema para
Dissertação/Estágio/Projeto - Trabalho de Conclusão de Curso

Orientador da Instituição onde se realiza o trabalho:

Pedro João Soares Rodrigues	pjsr@ipb.pt

Instituição do orientador:

IPB	ESTiG

Co-orientador da Instituição parceira:

Pedro Luiz de Paula Filho	plpf2004@gmail.com

Instituição do co-orientador:

UTFPR	Câmpus Medianeira

Curso ou cursos da Instituição do orientador onde se propõe que o trabalho seja realizado:

Engenharia Informática.

Título do trabalho:

Classificação Automática do Conteúdo de Favos em Imagens de Quadros de Colmeias

Palavras chave:

Aprendizagem máquina; Deep Learning; Reconhecimento de padrões.

Objetivos:

O objetivo principal deste trabalho consiste em criar um modelo computacional para fazer corretamente a classificação/contagem do conteúdo de favos de colmeias. A criação do modelo assentará em mecanismos de aprendizagem máquina, nomeadamente deep learning, que será treinado através de padrões visuais de favos fotografados digitalmente.

Descrição adicional:

No âmbito de várias tarefas apículas existe uma que obriga o apicultor e/ou investigador a classificar e contar o conteúdo de cada favo dos dois lados de cada quadro de colmeia. Esta tarefa tem a finalidade de analisar e controlar a progressão da criação de abelhas e da produção do mel o que implica repetí-la múltiplas vezes a cada ano. Cada quadro contém milhares de favos o que leva a que a contagem, na maior parte dos casos seja feita de forma aproximada. Os favos podem conter: crias em diferentes fases de evolução, mel, néctar, pólen ovos ou podem estar vazios. Um sistema que consiga fazer essa contagem e classificação automaticamente e corretamente representa uma importante evolução na referida tarefa.
As arquiteturas de deep learning têm mostrado um bom potencial a classificar padrões que sofrem variabilidade, a vários níveis visuais, de caso para caso, aumentando a capacidade de generalização do sistema. Assim, a utilização deste método de aprendizagem máquina adequa-se promissoramente à complexidade e variabilidade visual dos padrões apresentados pelos favos.

Metodologia/Plano de trabalhos:

Estudo do problema;
verificação do estado da arte de soluções a este problema;
estudo de uma arquitetura de deep learning específica;
organização do conjunto de padrões de treino;
implementação do modelo de aprendizagem baseado em deep learning por observação dos padrões de treino;
implementações conjugadas entre diversos descritores de sinais e deep learning;
análise de resultados;
otimização da implementação;
conversão da solução para um sistema móvel;
elaboração do relatório.

Recursos necessários:

Sistema computacional de elevado desempenho baseado em placas gráficas NVIDIA.

Appendix B

Tunnel Schematics

This appendix provides additional information on the tunnel built to photograph the frames. Images in different angles and links to additional information are provided below.

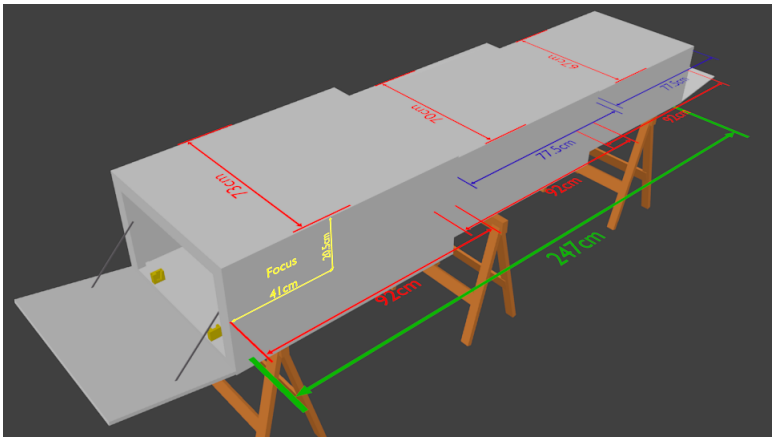


Figure B.1: First tunnel angle.

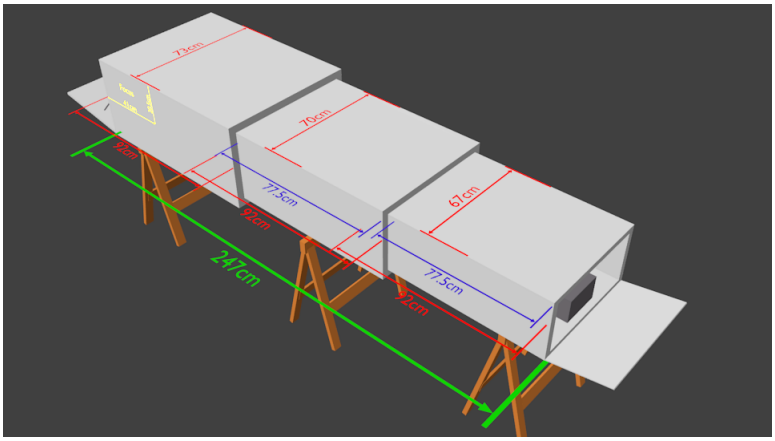


Figure B.2: Second tunnel angle.

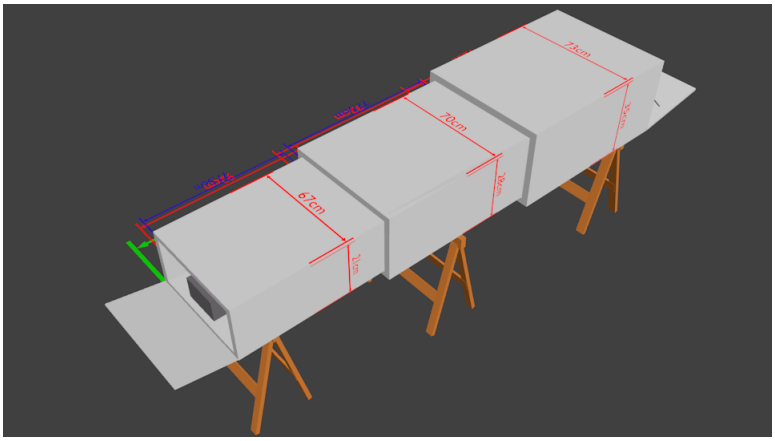


Figure B.3: Third tunnel angle.

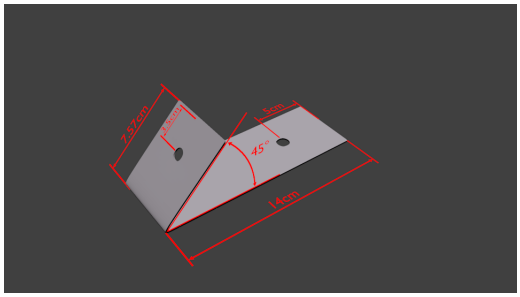


Figure B.4: LEDs holder schematics.

Information about the frame holder and additional 3D models of the tunnel are in the following repository: https://github.com/AvsThiago/DeepBee-Resources/tree/master/Tunnel3DModel

Appendix C

Data Augmentation Algorithm

This appendix presents the algorithm we developed to perform Data Augmentation in our training datasets.

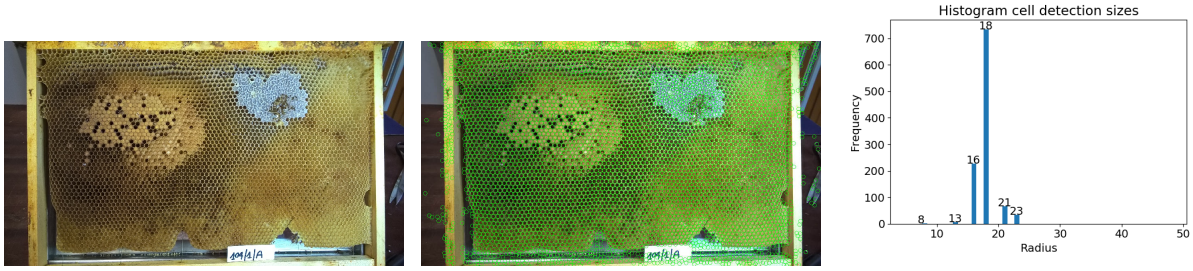
```
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import cv2
# pretty progressbar
from tqdm import tqdm

def brightness_adjustment(img):
    # turn the image into the HSV space
    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    # creates a random bright
    ratio = .5 + np.random.uniform()
    # convert to int32, so you don't get uint8 overflow
    # multiply the HSV Value channel by the ratio
    # clips the result between 0 and 255
    # convert again to uint8
    hsv[:,:,2] =  np.clip(hsv[:,:,2].astype(np.int32) \* ratio, 0, 255).astype(np.uint8)
    # return the image int the BGR color space
    return cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

# creates an image generator - https://keras.io/preprocessing/image/
img_generator = ImageDataGenerator(preprocessing_function=brightness_adjustment,
                                   rotation_range=3, width_shift_range=0.02,
                                   height_shift_range=0.02, shear_range=0.02,
                                   zoom_range=0.03, channel_shift_range=4.,
                                   horizontal_flip=True, vertical_flip=True,
                                   fill_mode='nearest')
# check here for more details
# https://keras.io/preprocessing/image/#imagedatagenerator-methods
images_path = 'images/path'
aug_iter = img_generator.flow_from_directory(images_path,
                                             target_size=(224, 224),
                                             shuffle=True,
                                             batch_size=1)
# number of images to be generated
n_images = 35714
# path where the generated images will be stored
path_out =  'output/path'
for j,i in tqdm(enumerate(range(n_images)), total=len(range(n_images))):
    img = next(aug_iter)[0].astype(np.uint8)[0]
    cv2.imwrite(path_out +'aug_'+ str(i) + '.png', img)
```

Listing 1: Data Augmentation algorithm.

Appendix D

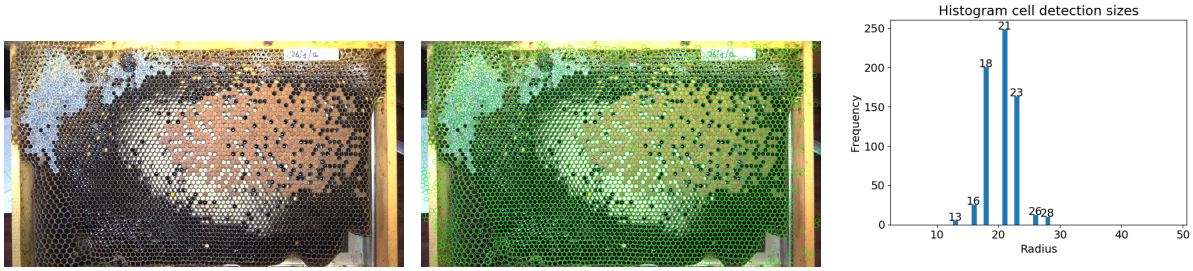
Scale Invariant Cell Detection Results

This appendix presents images processed by the scale invariant cell detection algorithm proposed in section 3.4.



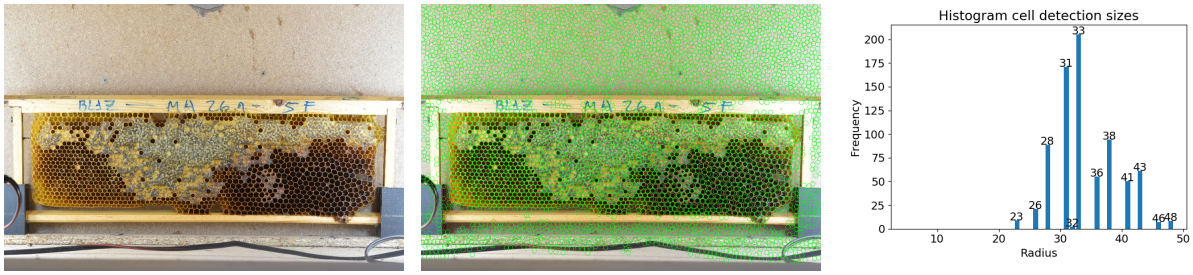
(a) (b) (c)

Figure D.1: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-CREA



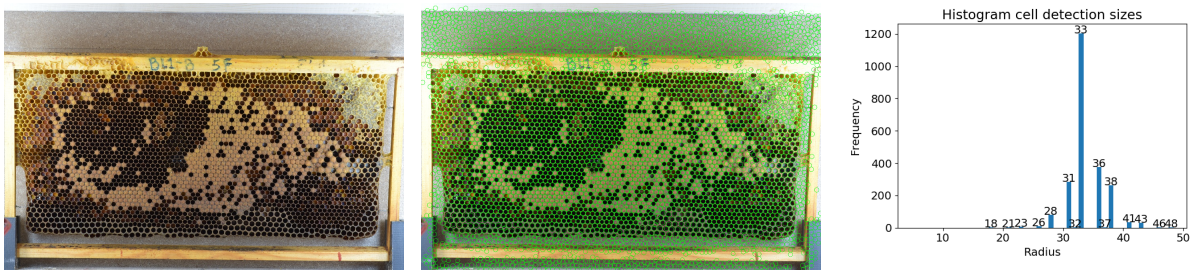
(a) (b) (c)

Figure D.2: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-CREA



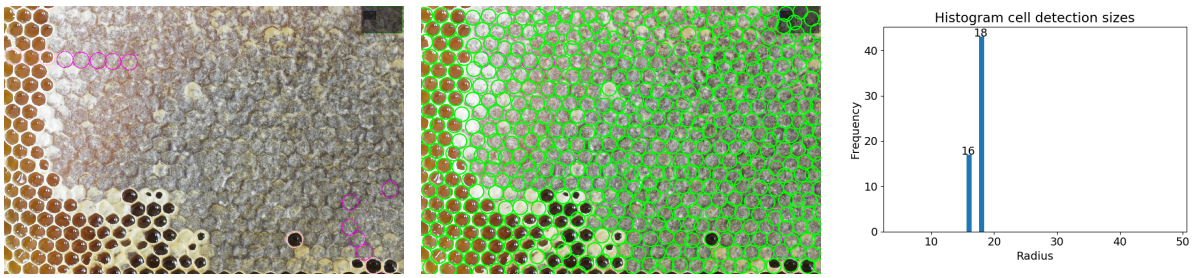
(a) (b) (c)

Figure D.3: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-PT



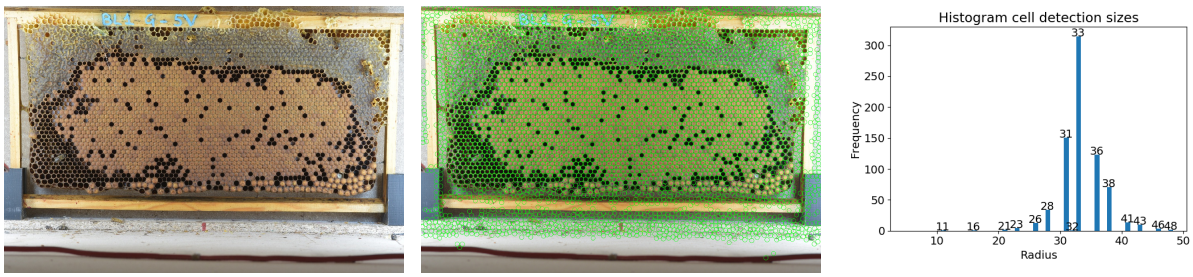
(a) (b) (c)

Figure D.4: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-PT



(a) (b) (c)

Figure D.5: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-PT

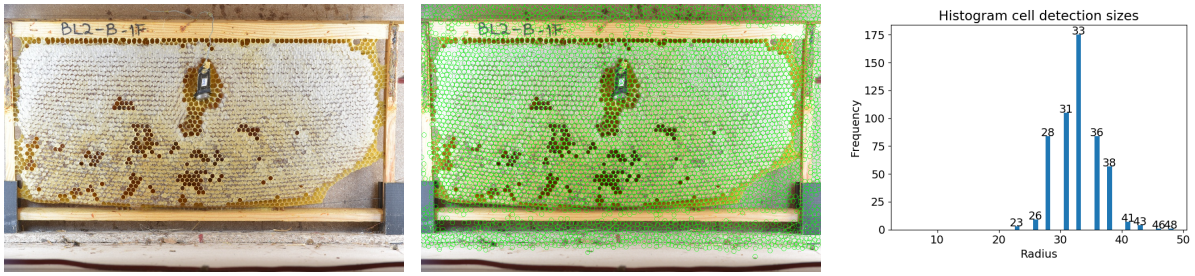


(a) (b) (c)

Figure D.6: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-PT

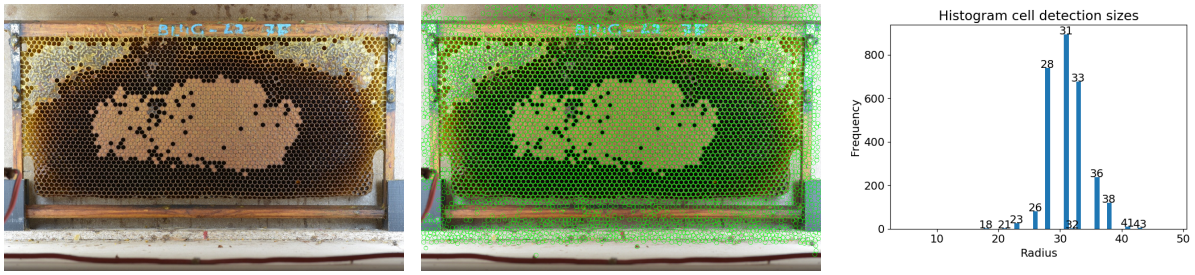
(a) (b) (c)

Figure D.7: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-PT



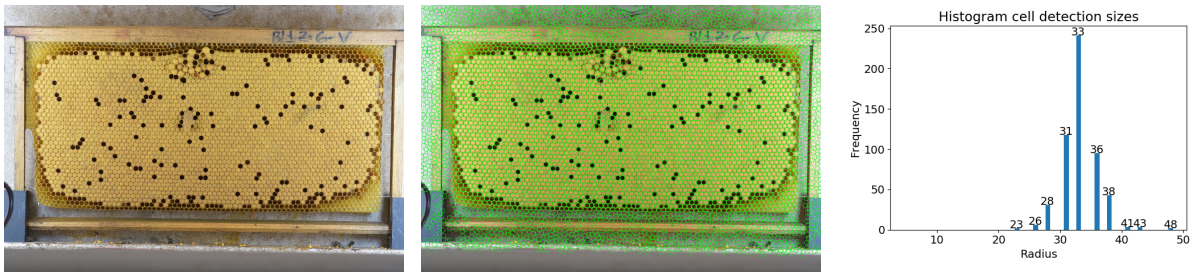
(a) (b) (c)

Figure D.8: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-PT



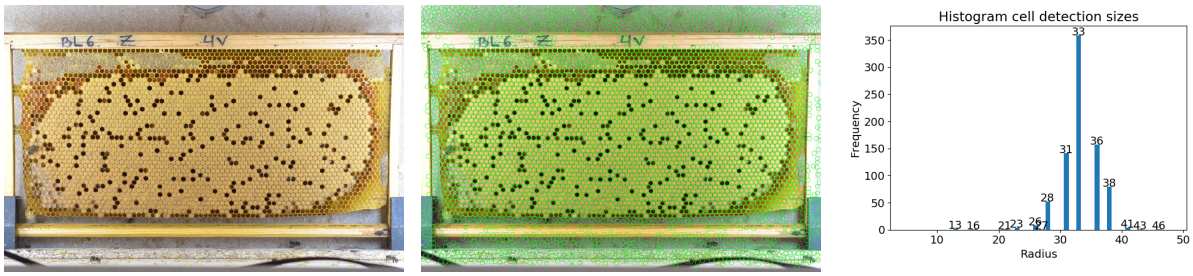
(a) (b) (c)

Figure D.9: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-PT



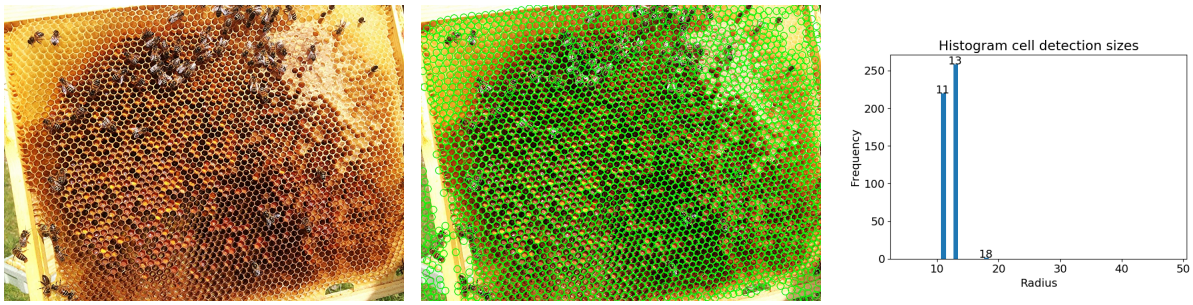
(a) (b) (c)

Figure D.10: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-PT



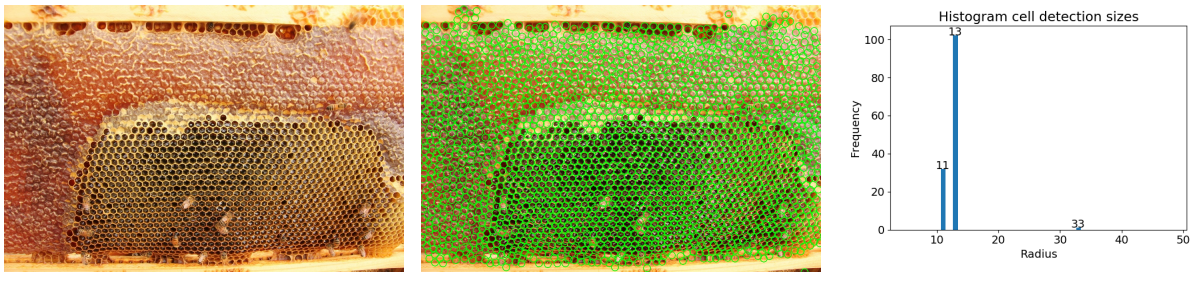
(a) (b) (c)

Figure D.11: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from the dataset DS-COMB-PT



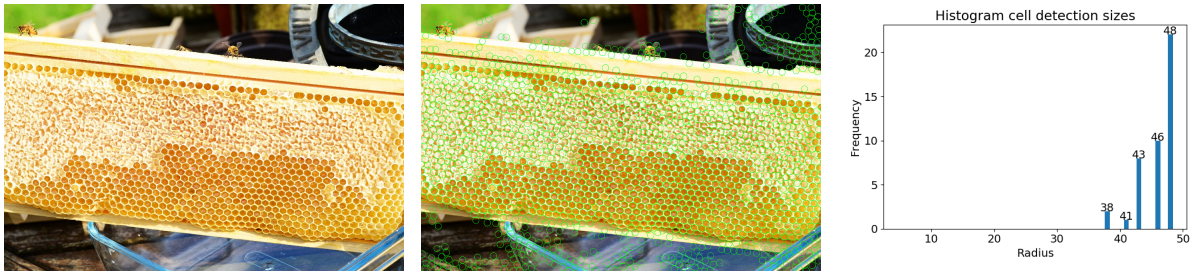
(a) (b) (c)

Figure D.12: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from internet: https://goo.gl/a1cRbK



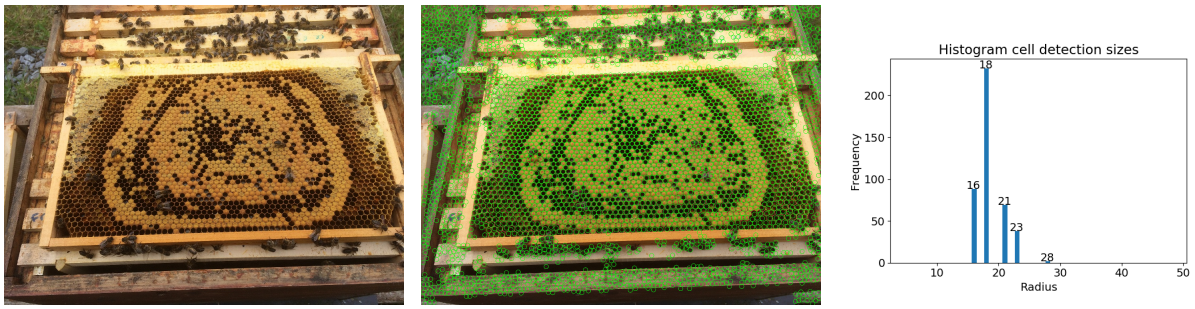
(a) (b) (c)

Figure D.13: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from internet: https://goo.gl/y7k9AM



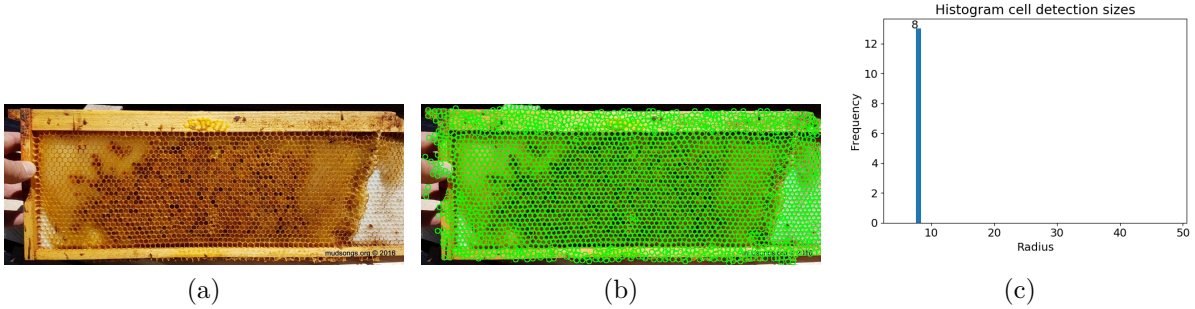
(a) (b) (c)

Figure D.14: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from internet: https://goo.gl/b8ozeF



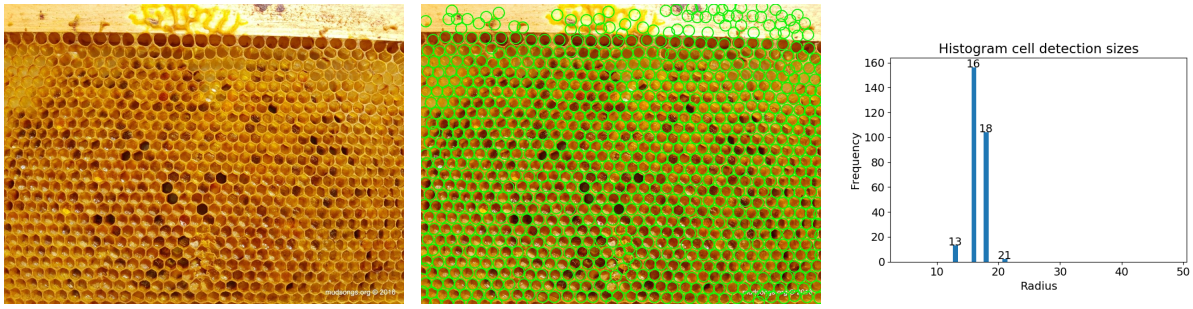
(a) (b) (c)

Figure D.15: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from internet: https://goo.gl/ydMyiT



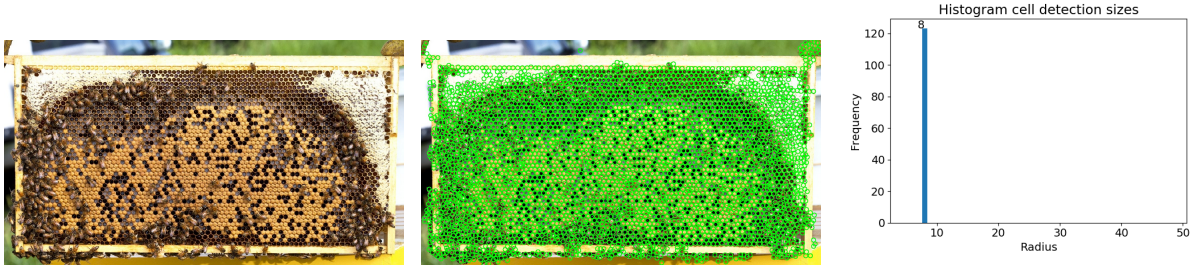
(a) (b) (c)

Figure D.16: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from internet: https://goo.gl/pCAUEh



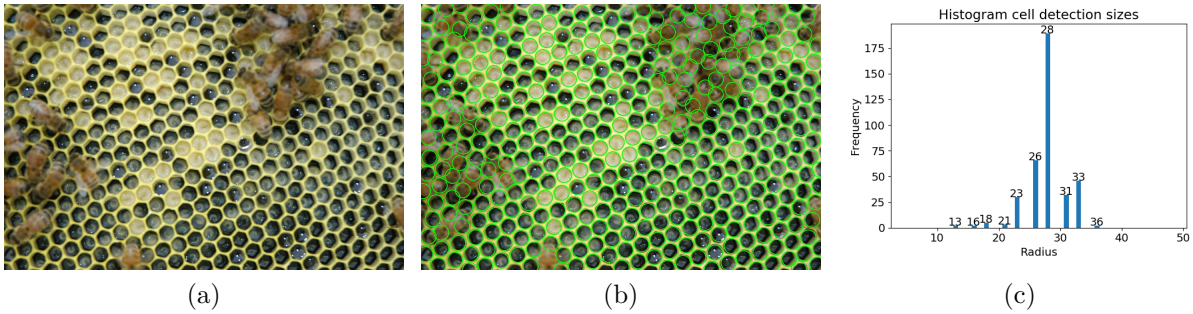
(a) (b) (c)

Figure D.17: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from internet: https://goo.gl/4p2jjs



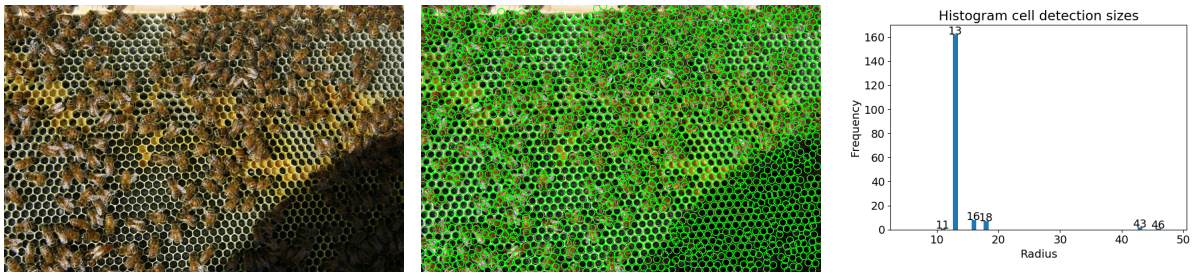
(a) (b) (c)

Figure D.18: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from internet: https://goo.gl/eJPe2w



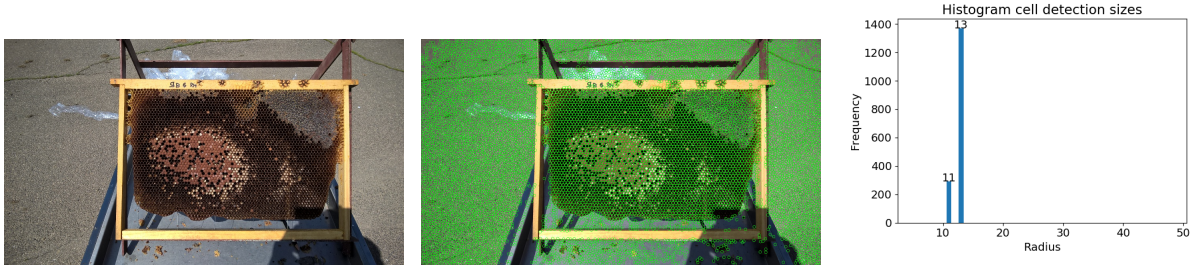
(a) (b) (c)

Figure D.19: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from internet: https://goo.gl/tzDKtJ



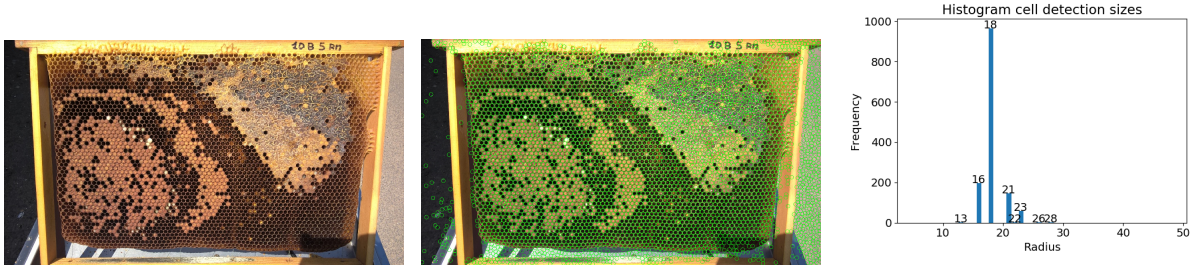
(a) (b) (c)

Figure D.20: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from internet: https://goo.gl/tzDKtJ



(a) (b) (c)

Figure D.21: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from dataset DS-COMB-CREA



(a) (b) (c)

Figure D.22: (a) Original Image; (b) Cells detected and (c) Most frequent cell radius. Image from dataset DS-COMB-CREA

Appendix E

CSS Image Results

This appendix presents comparisons made between manually segmented regions and regions segmented using the CSS method. This method was presented in section 3.5.3 and discussed in section 5.3.3.

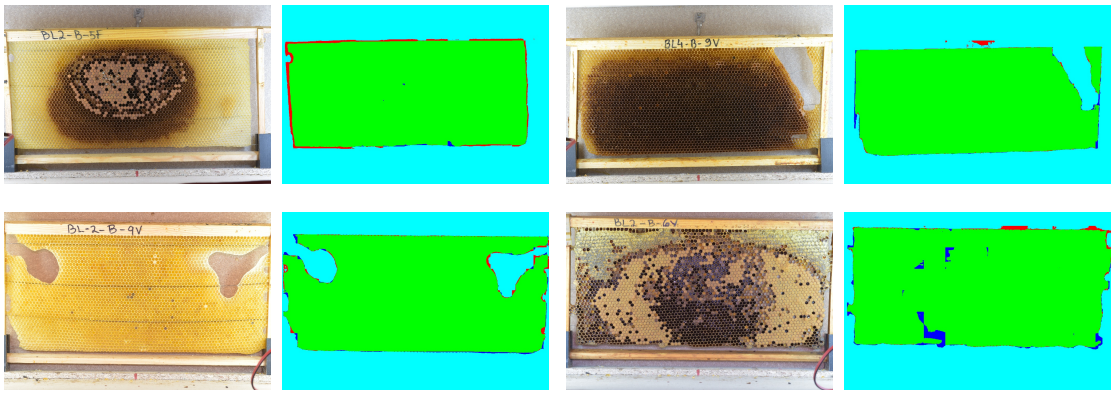


Figure E.1: Comparison between areas annotated and predicted by the CSS method on DS-COMB-PT images .

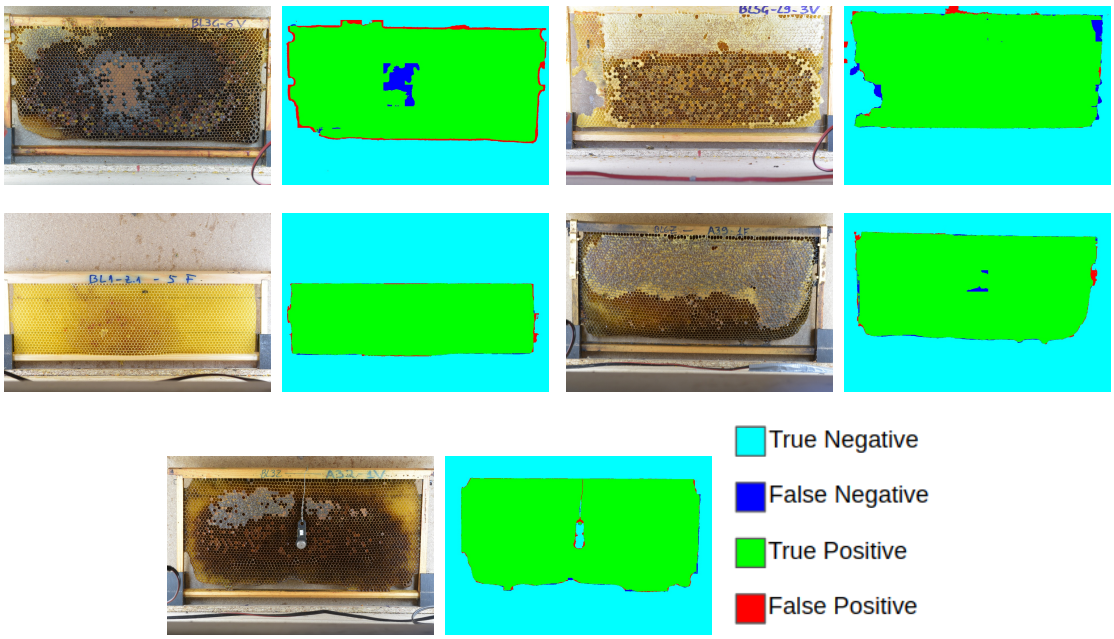


Figure E.2: Comparison between areas annotated and predicted by the CSS method on DS-COMB-PT images .

Appendix F

Comparison of Cell Detections Performed by Humans and Automatically

Results of the comparison between cells detected by our cell detection algorithm (Section 3.3) with false detection removal using the CSS-LC method and by humans.

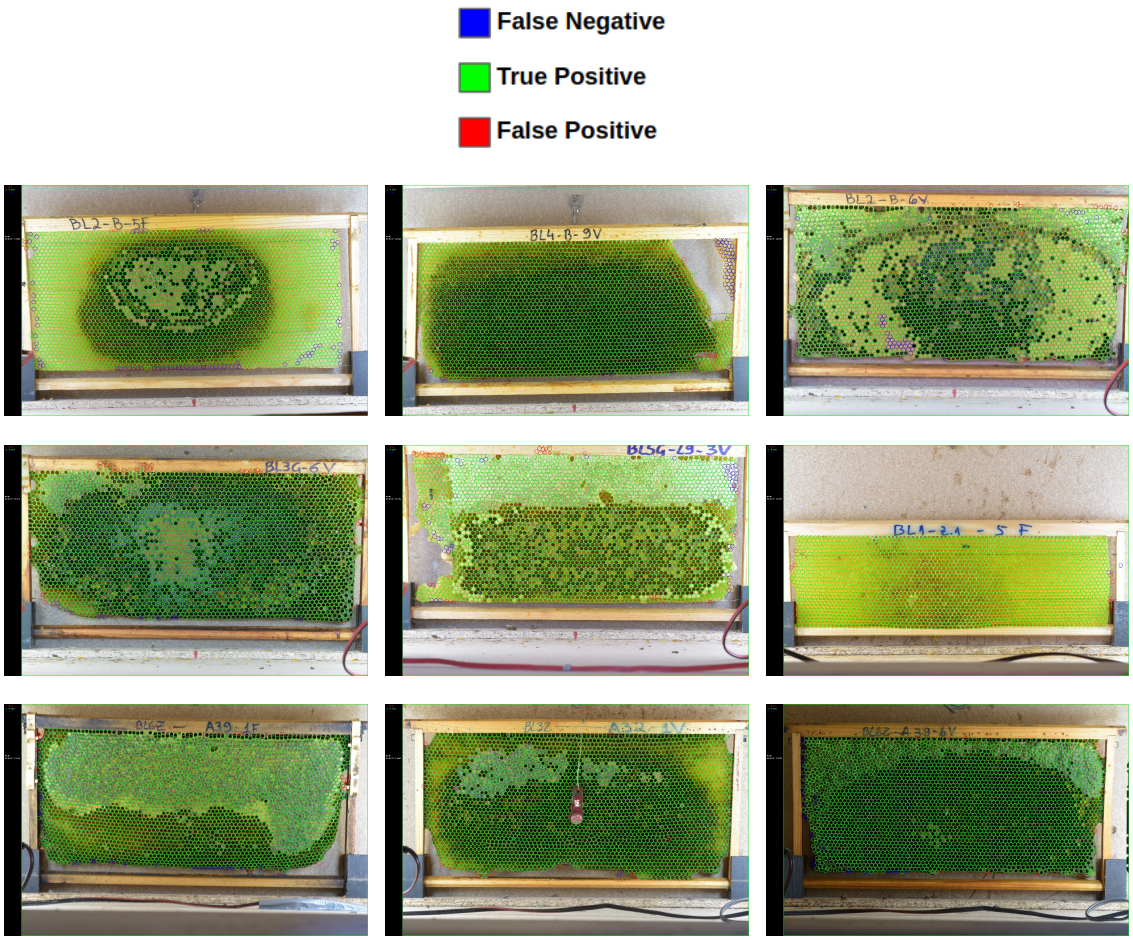


Figure F.1: Results of the comparison between cells detected by our cells detection algorithm and by humans.

Appendix G

Metrics Calculated Over Different CNN Architectures Results

Results of metrics comparison over different models trained with and without Data Augmentation. There results were calculated over the DS-COMB-CELL-TEST dataset.

MODEL	PRECISION	RECALL	F1-SCORE
MobileNet DA	0,943	0,943	0,943
InceptionResNetV2 DA	0,941	0,941	0,941
NASNetMobile DA	0,941	0,939	0,939
DenseNet201	0,940	0,939	0,939
NASNetMobile	0,938	0,938	0,938
DenseNet121	0,940	0,937	0,938
InceptionV3	0,937	0,935	0,935
InceptionResNetV2	0,938	0,935	0,935
MobileNet	0,934	0,933	0,933
DenseNet169	0,933	0,931	0,931
MobileNetV2	0,931	0,930	0,930
Resnet50	0,930	0,930	0,930
Xception	0,929	0,927	0,926
DenseNet201 DA	0,929	0,922	0,923

Table G.1: Precision Recall and F1-Score calculated over different models using the DS-COMB-CELL-TEST dataset.

Appendix H

Confusion Matrices for Cell Classification by Class

This appendix presents all confusions matrices calculated using the different architectures trained using or not Data Augmentation.

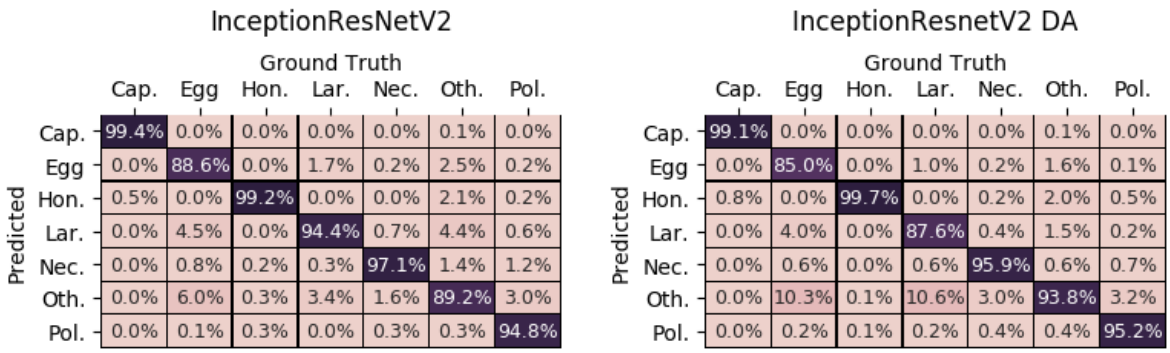


Figure H.1: Confusion Matrices by Class

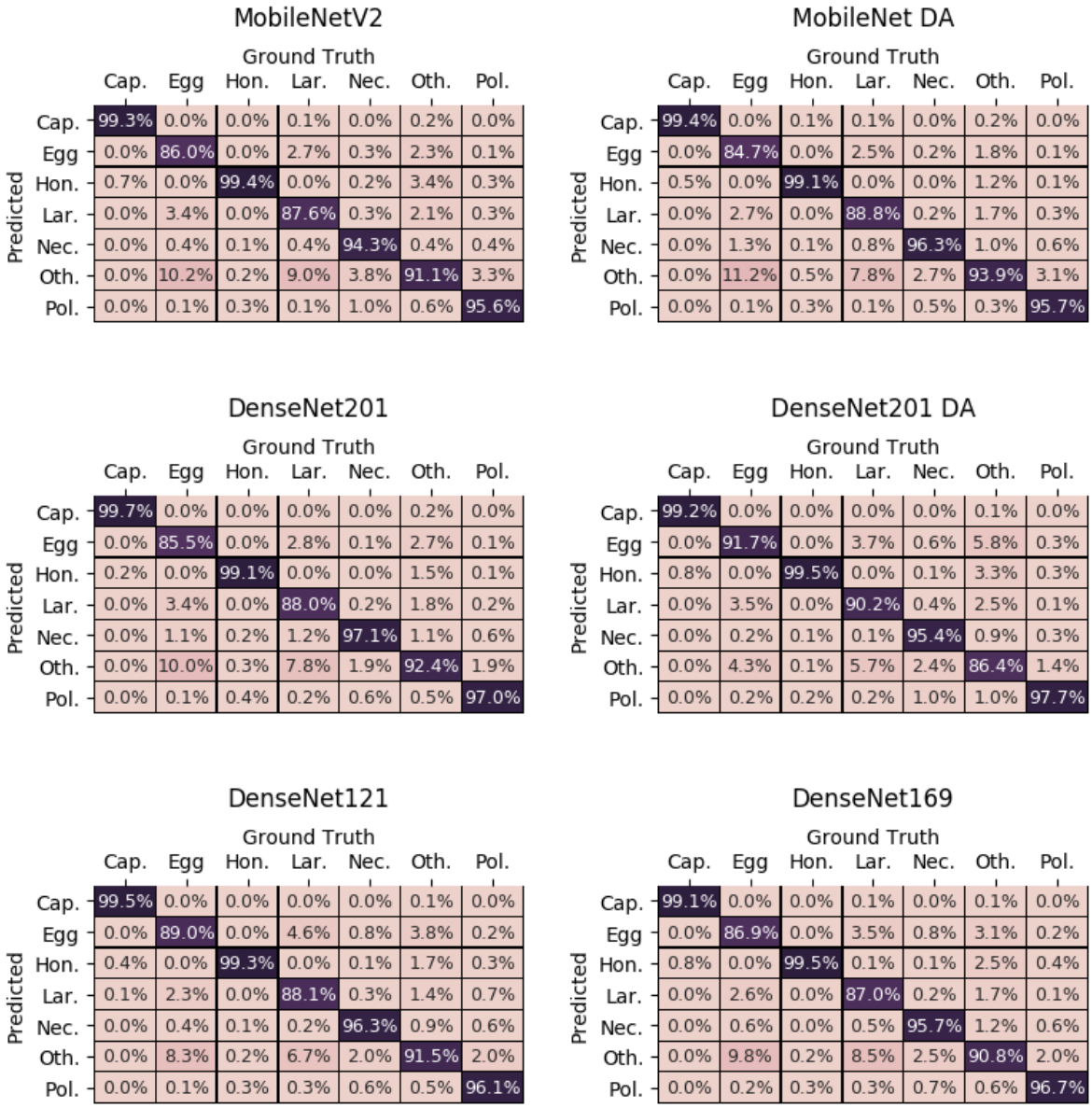


Figure H.2: Confusion Matrices by Class

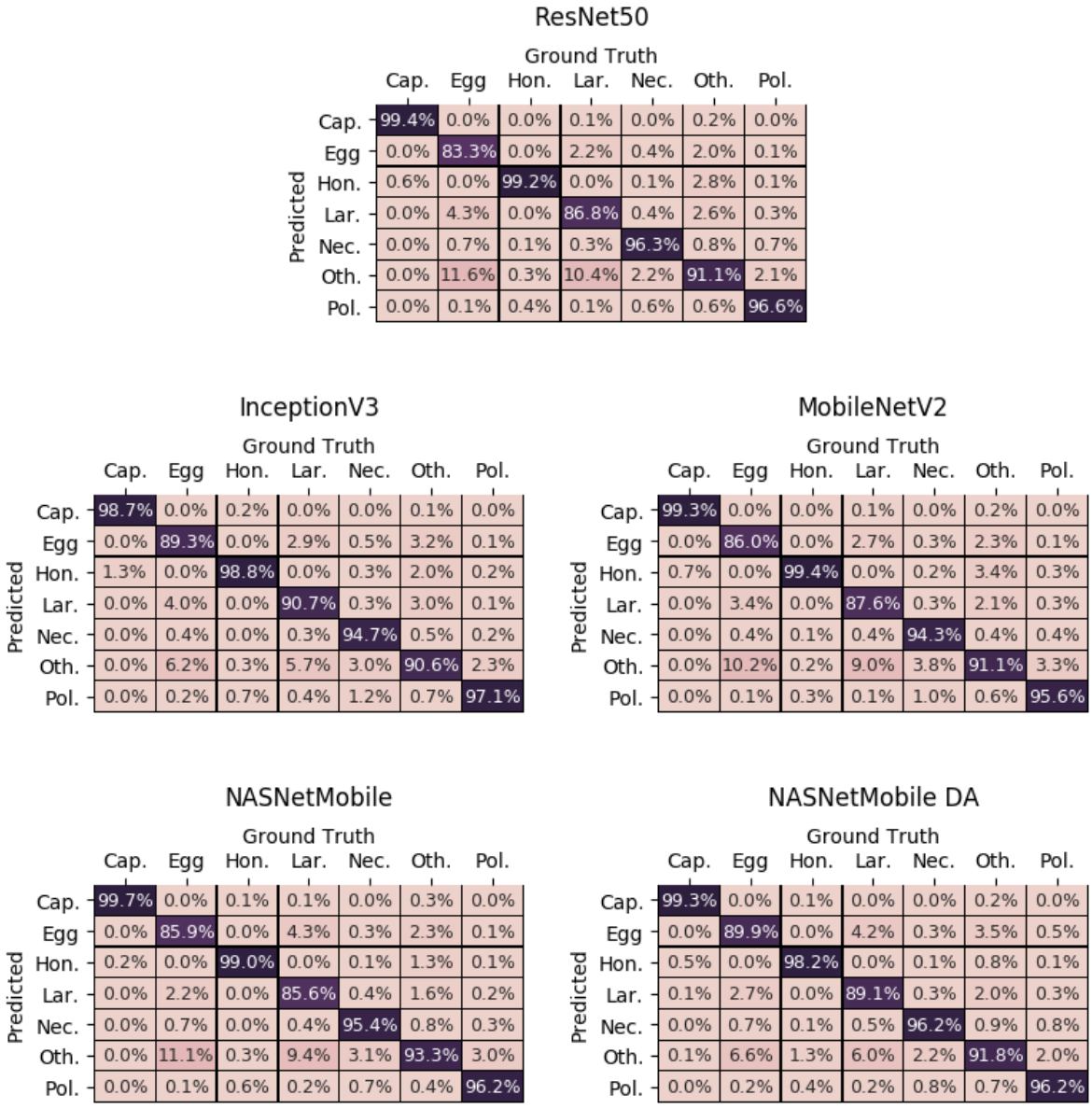


Figure H.3: Confusion Matrices by Class

Appendix I

Metrics Calculated Over Different CNN Architectures by Class

Results of metrics by class with different models. Results calculated using the DS-COMB-CELL-TEST dataset.

MODEL	CLASS	PRECISION	RECALL	F1-SCORE	SUPPORT
DenseNet121	capped	0,997	0,995	0,996	8361
DenseNet121	egg	0,756	0,890	0,817	3962
DenseNet121	honey	0,921	0,993	0,956	4678
DenseNet121	larva	0,925	0,881	0,903	6232
DenseNet121	nectar	0,952	0,963	0,957	4996
DenseNet121	other	0,952	0,915	0,933	20748
DenseNet121	pollen	0,963	0,961	0,962	4937
DenseNet169	capped	0,997	0,991	0,994	8361
DenseNet169	egg	0,789	0,869	0,827	3962
DenseNet169	honey	0,883	0,995	0,935	4678
DenseNet169	larva	0,919	0,870	0,894	6232
DenseNet169	nectar	0,936	0,957	0,946	4996
DenseNet169	other	0,943	0,908	0,925	20748

DenseNet169	pollen	0,961	0,967	0,964	4937
DenseNet201	capped	0,996	0,997	0,996	8361
DenseNet201	egg	0,819	0,855	0,837	3962
DenseNet201	honey	0,933	0,991	0,961	4678
DenseNet201	larva	0,913	0,880	0,896	6232
DenseNet201	nectar	0,927	0,971	0,949	4996
DenseNet201	other	0,946	0,924	0,935	20748
DenseNet201	pollen	0,966	0,970	0,968	4937
DenseNet201 DA	capped	0,996	0,992	0,994	8361
DenseNet201 DA	egg	0,712	0,917	0,802	3962
DenseNet201 DA	honey	0,857	0,995	0,921	4678
DenseNet201 DA	larva	0,891	0,902	0,897	6232
DenseNet201 DA	nectar	0,954	0,954	0,954	4996
DenseNet201 DA	other	0,961	0,864	0,910	20748
DenseNet201 DA	pollen	0,944	0,977	0,960	4937
InceptionResNetV1	pollen	0,980	0,948	0,964	4937
InceptionResNetV2	capped	0,996	0,994	0,995	8361
InceptionResNetV2	egg	0,844	0,886	0,865	3962
InceptionResNetV2	honey	0,906	0,992	0,947	4678
InceptionResNetV2	larva	0,834	0,944	0,886	6232
InceptionResNetV2	nectar	0,923	0,971	0,946	4996
InceptionResNetV2	other	0,964	0,892	0,926	20748
InceptionResNetV2 DA	capped	0,998	0,991	0,995	8361
InceptionResNetV2 DA	egg	0,890	0,850	0,869	3962
InceptionResNetV2 DA	honey	0,899	0,997	0,946	4678
InceptionResNetV2 DA	larva	0,917	0,876	0,896	6232
InceptionResNetV2 DA	nectar	0,954	0,959	0,956	4996
InceptionResNetV2 DA	other	0,934	0,938	0,936	20748
InceptionResNetV2 DA	pollen	0,973	0,952	0,963	4937

InceptionV3	capped	0,996	0,987	0,992	8361
InceptionV3	egg	0,804	0,893	0,846	3962
InceptionV3	honey	0,894	0,988	0,939	4678
InceptionV3	larva	0,877	0,907	0,892	6232
InceptionV3	nectar	0,972	0,947	0,959	4996
InceptionV3	other	0,955	0,906	0,930	20748
InceptionV3	pollen	0,947	0,971	0,959	4937
MobileNet	capped	0,997	0,994	0,996	8361
MobileNet	egg	0,822	0,860	0,841	3962
MobileNet	honey	0,894	0,995	0,942	4678
MobileNet	larva	0,893	0,879	0,886	6232
MobileNet	nectar	0,927	0,966	0,946	4996
MobileNet	other	0,943	0,911	0,927	20748
MobileNet	pollen	0,976	0,958	0,967	4937
MobileNet DA	capped	0,995	0,994	0,994	8361
MobileNet DA	egg	0,860	0,847	0,854	3962
MobileNet DA	honey	0,940	0,991	0,965	4678
MobileNet DA	larva	0,919	0,888	0,903	6232
MobileNet DA	nectar	0,935	0,963	0,949	4996
MobileNet DA	other	0,940	0,939	0,939	20748
MobileNet DA	pollen	0,977	0,957	0,967	4937
MobileNetV2	capped	0,995	0,993	0,994	8361
MobileNetV2	egg	0,837	0,860	0,848	3962
MobileNetV2	honey	0,855	0,994	0,919	4678
MobileNetV2	larva	0,902	0,876	0,889	6232
MobileNetV2	nectar	0,969	0,943	0,956	4996
MobileNetV2	other	0,934	0,911	0,922	20748
MobileNetV2	pollen	0,960	0,956	0,958	4937
NASNetMobile	capped	0,993	0,997	0,995	8361

NASNetMobile	egg	0,814	0,859	0,836	3962
NASNetMobile	honey	0,940	0,990	0,964	4678
NASNetMobile	larva	0,922	0,856	0,887	6232
NASNetMobile	nectar	0,951	0,954	0,952	4996
NASNetMobile	other	0,935	0,933	0,934	20748
NASNetMobile	pollen	0,965	0,962	0,963	4937
NASNetMobile DA	capped	0,994	0,993	0,994	8361
NASNetMobile DA	egg	0,776	0,899	0,833	3962
NASNetMobile DA	honey	0,953	0,982	0,967	4678
NASNetMobile DA	larva	0,908	0,891	0,899	6232
NASNetMobile DA	nectar	0,942	0,962	0,952	4996
NASNetMobile DA	other	0,954	0,918	0,936	20748
NASNetMobile DA	pollen	0,956	0,962	0,959	4937
Resnet50	capped	0,994	0,994	0,994	8361
Resnet50	egg	0,853	0,833	0,843	3962
Resnet50	honey	0,878	0,992	0,931	4678
Resnet50	larva	0,878	0,868	0,873	6232
Resnet50	nectar	0,952	0,963	0,957	4996
Resnet50	other	0,934	0,911	0,922	20748
Resnet50	pollen	0,963	0,966	0,965	4937
Xception	capped	0,998	0,990	0,994	8361
Xception	egg	0,826	0,883	0,853	3962
Xception	honey	0,898	0,992	0,943	4678
Xception	larva	0,956	0,744	0,837	6232
Xception	nectar	0,934	0,959	0,946	4996
Xception	other	0,914	0,931	0,922	20748
Xception	pollen	0,945	0,974	0,959	4937

Table I.1: Precision Recall and F1-Score calculated by class over different models using the DS-COMB-CELL-TEST

Appendix J

Comparison Between Regions Most Annotated and Regions with More Wrong Predictions

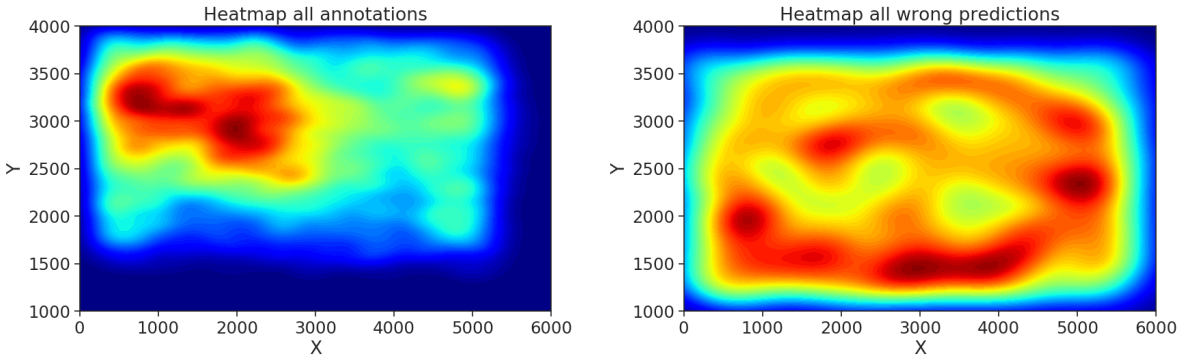


Figure J.1: Comparison between most annotated areas and with more errors.

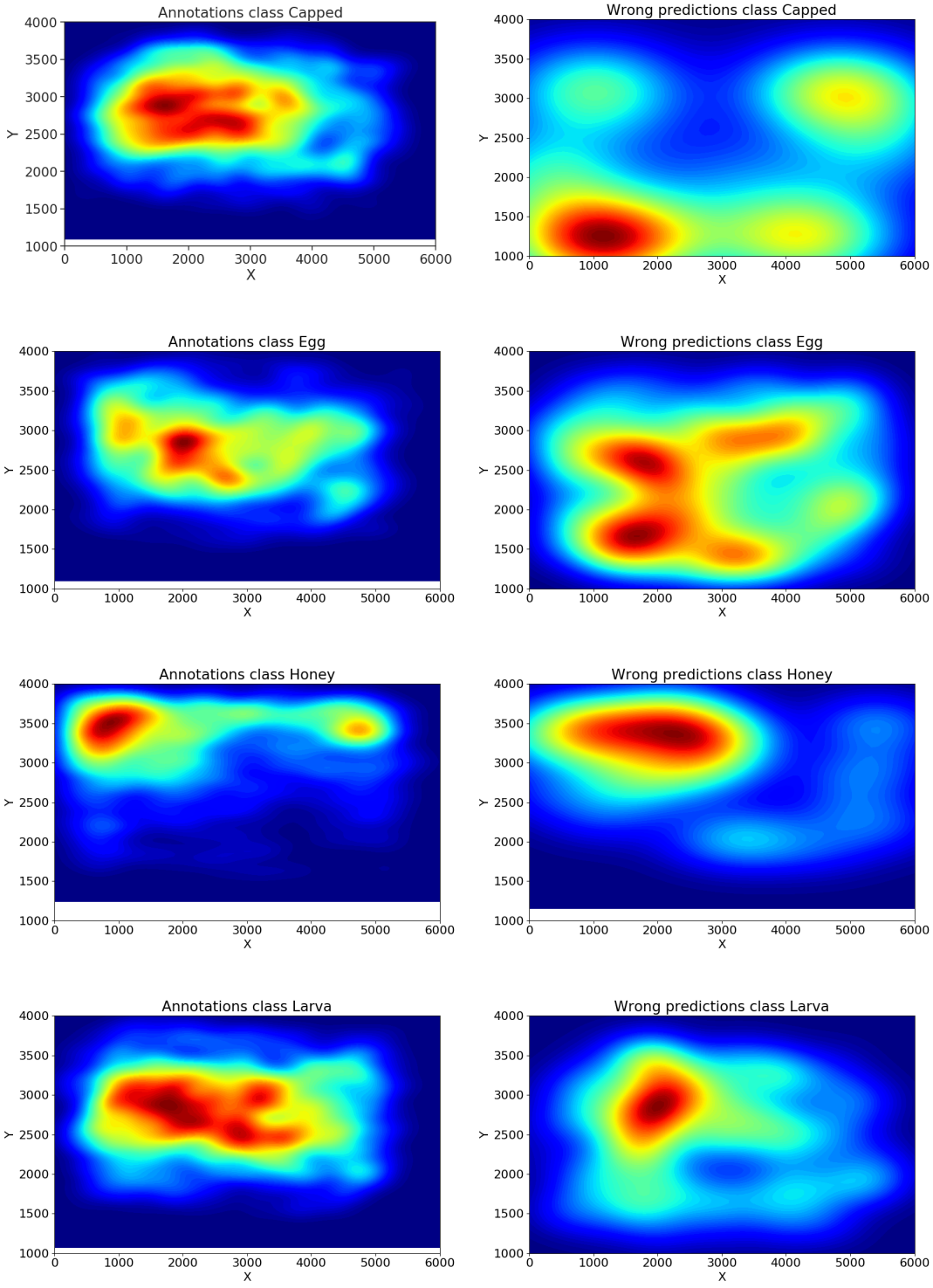


Figure J.2: Comparison between most annotated areas and with more errors.

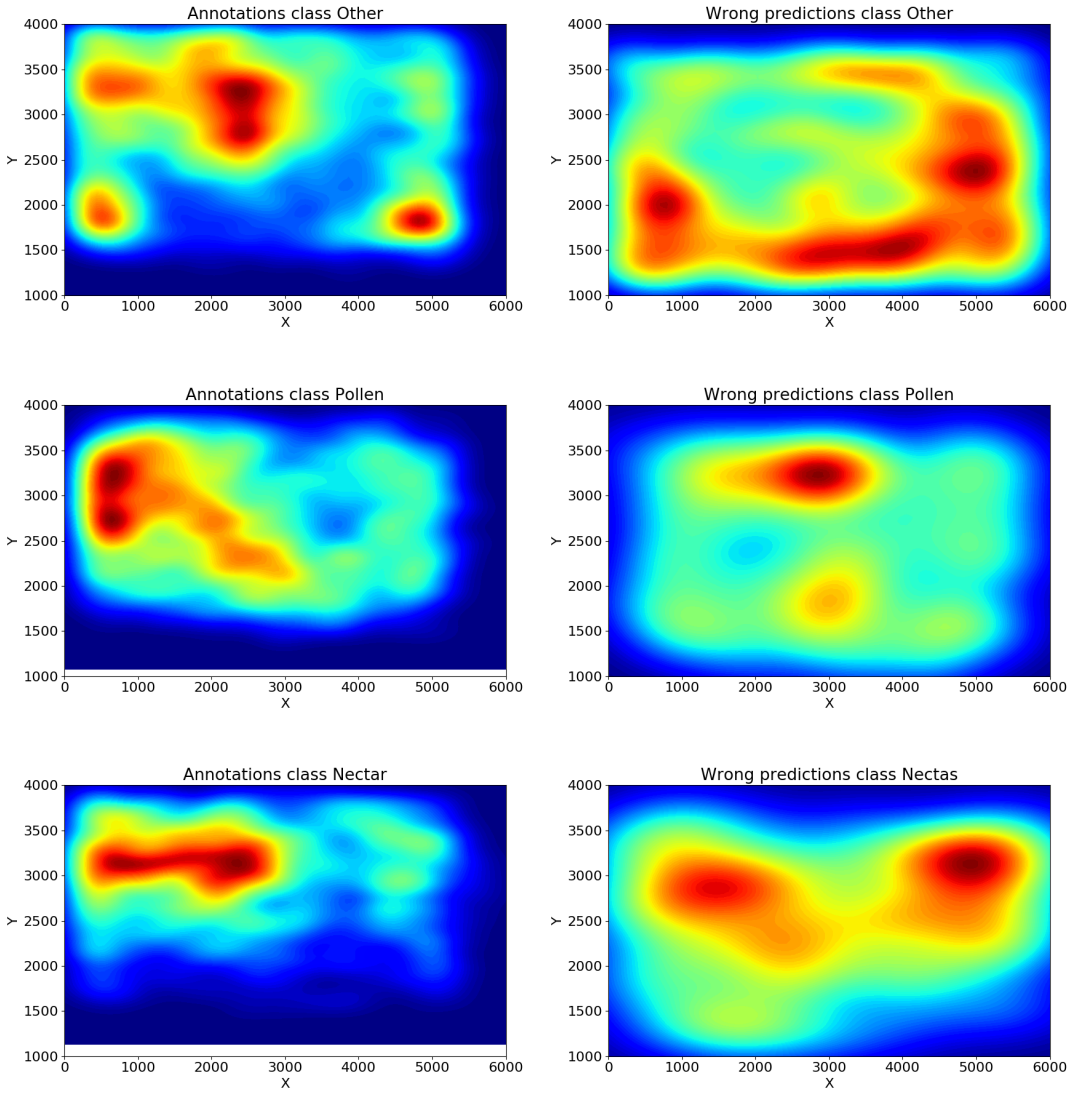


Figure J.3: Comparison between most annotated areas and with more errors.