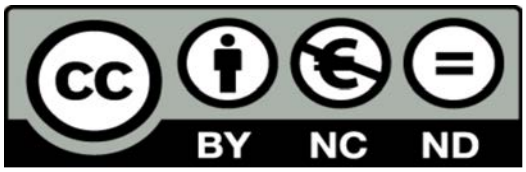
UNIVERSIDAD DE LA RIOJA

TESIS DOCTORAL

Título
Deep Detection and Segmentation Models for Plant Physiology and Precision Agriculture
Autor/es
Ángela Casado García
Director/es
Jonathan Heras Vicente y María Vico Pascual Martínez-Losa
Facultad
Facultad de Ciencia y Tecnología
Titulación
Departamento
Matemáticas y Computación
Curso Académico



publicaciones.unirioja.es

E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

PhD. Thesis

Deep Detection and Segmentation Models for Plant Physiology and Precision Agriculture

Ángela Casado García

Written under the supervision of
Dr. Jónathan Heras Vicente
Dra. Vico Pascual Martínez-Losa

Dissertation submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Departamento de Matemáticas y Computación

Logroño, September 2023

UNIVERSIDAD DE LA RIOJA

PhD. Thesis

Modelos de Detección y Segmentación para la Fisiología de Plantas y Agricultura de Precisión

Ángela Casado García

Escrita bajo la supervisión de
Dr. Don Jónathan Heras Vicente
Dra. Doña Vico Pascual Martínez-Losa

Memoria presentada para la obtención del grado de Doctor

Departamento de Matemáticas y Computación

Logroño, septiembre de 2023

This work was partially supported by Ministerio de Economía y Competitividad [MTM2017-88804-P], Ministerio de Ciencia e Innovación [PID2020-115225RB-I00, RTC2017-6640-7], a FPI grant from Comunidad de La Rioja 2020 and grants ATUR20/04, ATUR21/05, ATUR 22/06 from Universidad de La Rioja.

Agradecimientos

Para los que están,
para los que ya no están,
para los que estarán.

Para mi Padre

Ahora que esta etapa llega a su fin, sólo puedo recordar cómo empezo hace cuatro años. Fue en un aeropuerto en el que entré sin un futuro muy claro, y después de dos horas de espera y conversaciones, salí de él con la que es la mejor decisión de mi vida. Pero no vamos a engañarnos no todo ha sido de color de rosa, ha habido blancos, negros y muchísimos grises; he reido, he llorado, he disfrutado, he aprendido mucho y este es el momento de daros las gracias a todos aquellos que habéis formado parte de esta etapa.

En primer lugar, gracias a mis directores Jónathan y Vico, sin vosotros esta tesis no hubiera salido adelante, gracias por los consejos, recomendaciones y por estar siempre ahí. En especial, gracias a Jónathan por darme esta grandísima oportunidad cuando nadie creía en mi, ni siquiera yo. Y por aguantarme, que para qué vamos a negarlo, soy muy petarda y probablemente sea quien más guerra te haya dado y te dará. ¡Gracias por tu enorme paciencia y sobre todo por corregir la memoria en azul!

Gracias al Departamento de Matemáticas y Computación, a muchos de vosotros os tuve como profesores en la carrera y ahora he podido trabajar codo con codo con vosotros. Me habéis enseñado que no todo es trabajar; siempre hay tiempo para un café, una cena, un chiste o una excursión. En especial gracias al grupo de informática Psycotrip que siempre habéis estado ahí cuando ha sido necesario. Y no me puedo olvidar de las nuevas incorporaciones por hacer los cafés tan locos.

He tenido la gran suerte de compartir este camino con grandes amigos, Adrián, Dani, Jorge, Manu y Patri. Con vosotros he compartido de todo, desde apuestas de comida, hasta el estrés del papeleo. En especial, me gustaría agradecer a Adrián y Manu con los que he ido de congreso en múltiples ocasiones, de las que han surgido

muchas anécdotas; como aquel congreso donde me caí por las escaleras y en vez de ayudarme os quedastéis mirandome, o cuando teníamos que llevar unos posters y nos los dejamos en el apartamento, o la rabia que os daba cuando me libraba de las preguntas del chair. Sin todos vosotros todo esto no hubiera sido tan divertido, ni tendríamos tanto que recordar.

El día que Guti me dijo cuales eran "mis dominios" temblé. Me había tocado con Alberto y Edgar. Jamás había hablado con ellos pero me daban mucho respeto. Ahora puedo decir que sin ellos junto con Emilio (el único sensato de este despacho) hubiera tenido más espantadas y menos ideas absurdas. Gracias a los tres por nuestros más y nuestros menos.

No puedo olvidarme de Esther y Tesi; sin Esther la mitad del papeleo hubiera estado mal hecho y sin Tesi los días no hubieran empezado con tanta alegría.

También quería agradecer al equipo de Stiima, desde el primer momento me sentí arropada por vosotros y me hicisteis ser parte de la familia que formáis. Grazie per tutto.

Y por último y no por ello menos importante, quería agradecer a mi familia. No es necesario ser de sangre para ser familia: Joaquín y Bego gracias por vuestro apoyo, como me dijistéis en una ocasión *"Esto es como las carreras de fondo, lo importante no es ser el primero, si no llegar a la meta"*. Tati, Tite y María, me habéis acompañado, me habéis hecho desconectar, me habéis hecho levantarme, me habéis apoyado incondicionalmente celebrando las alegrías y las penas, haciéndome rabiar hasta el infinito, gracias por este "cúmulo" de cosas. Mama, eres la persona con la que más discuto, con la que más me enfado y la que siempre está detrás mío, has sido mi conejillo de indias para probar los programas, has sido mi mejor público, me has escuchado ensayar en sitios inimaginables. Sé que siempre estaréis ahí, así que de mayor espero ser un poquito como todos vosotros. Os quiero.

Abstract

Computer Vision is a multidisciplinary field that combines concepts from Artificial Intelligence, image processing, visual perception, and data science to enable computers to understand and analyse visual content in a similar way to humans. In the last years, significant advancements have been made in Computer Vision thanks to the development of algorithms and techniques based on Deep Learning methods. Two areas of Computer Vision that have numerous applications in various fields such as biology, agriculture, and medicine are Object detection and semantic segmentation. Currently, the most successful techniques to tackle these two tasks are also based on Deep Learning methods. However, although these methods have achieved excellent results, using such techniques in contexts outside machine learning can be complex. This is due to the large number of images that are required to train Deep Learning models (which can be difficult to obtain in context like biomedicine or precision agriculture), the process of annotating images (a time-consuming and expertise-demanding problem), and the technical difficulties for training and using Deep Learning models by domain experts. The aim of this thesis is to address these limitations through different theoretical developments, and evaluate the proposed solutions in actual contexts.

First of all, we have focused on the development of methods that allow us to improve the performance of object detection models. For this purpose, we have developed an algorithm that improves the accuracy and robustness of object detection models by means of an ensemble method. This algorithm is also the basis to define semi-supervised learning methods that reduce the number of annotated images that are needed to train object detection models. Moreover, to facilitate the create and usage of object detection models, we have developed an open source tool that simplifies the process of creating and using object detection models, thanks to a simple to use graphical interface. Furthermore, we have generalise our work to facilitate the creation and usage of models for any computer vision task. Finally, the developed techniques and tools have served as the foundation for addressing real-world problems in plant physiology, and in precision agriculture. As a summary, this work is a step towards the democratisation of Deep Learning models for users outside the machine learning community.

Resumen

La Visión por Computador es un campo multidisciplinar que combina conceptos de Inteligencia Artificial, procesamiento de imágenes, percepción visual y ciencia de datos para permitir que los ordenadores comprendan y analicen contenido visual de manera similar a los humanos. En los últimos años, se han logrado avances significativos en la Visión por Computador gracias al desarrollo de algoritmos y técnicas basadas en métodos de aprendizaje profundo o Deep Learning. La detección de objetos y la segmentación semántica son dos áreas de la Visión por Computador que tienen numerosas aplicaciones en diversos campos como la biología, la agricultura y la medicina. Actualmente, las técnicas más exitosas para abordar estas dos tareas también se basan en métodos de Deep Learning. Sin embargo, aunque estos métodos han logrado excelentes resultados, utilizar dichas técnicas en contextos fuera del aprendizaje automático puede ser complejo. Esto se debe al gran número de imágenes requeridas para entrenar modelos de Deep Learning (que pueden ser difíciles de obtener en contextos como la biomedicina o la agricultura de precisión), el proceso de anotación de imágenes (un problema que consume mucho tiempo y requiere de experiencia) y las dificultades técnicas para entrenar y utilizar modelos de Deep Learning por parte de usuarios no expertos. El objetivo de esta tesis es abordar estas limitaciones a través de diferentes desarrollos teóricos y evaluar las soluciones propuestas en contextos reales.

En primer lugar, nos hemos centrado en el desarrollo de métodos que nos permiten mejorar el rendimiento de los modelos de detección de objetos. Para este propósito, hemos desarrollado un algoritmo que mejora la precisión y la robustez de los modelos de detección de objetos mediante métodos de ensemble. Este algoritmo también es la base para definir métodos de aprendizaje semisupervisado que reducen el número de imágenes anotadas necesarias para entrenar los modelos de detección. Además, para facilitar la creación y uso de modelos de detección, hemos desarrollado una herramienta que simplifica el proceso de creación y uso de modelos de detección gracias a una interfaz gráfica fácil de usar. Además, hemos generalizado nuestro trabajo para facilitar la creación y uso de modelos para cualquier tarea de Visión por Computador. Por último, las técnicas y herramientas desarrolladas anteriormente han servido como base para abordar problemas en fisiología

de plantas, y en agricultura de precisión. En resumen, este trabajo es un paso hacia la democratización de los modelos de Aprendizaje Profundo para usuarios fuera de la comunidad de aprendizaje automático.

Contents

Chapter 1

Introduction

Computer Vision is a multidisciplinary field that combines concepts from Artificial Intelligence, image processing, visual perception, and data science to enable computers to understand and analyse visual content in a similar way to humans [1]. Roughly speaking, Computer Vision aims to develop algorithms and techniques that allow machines to "see" and comprehend images and videos. This involves extracting useful information from images by means of tasks, such as object detection and recognition [2, 3], motion tracking [4], image segmentation [5], 3D reconstruction [6], or complex scene analysis [7]. Computer Vision has applications in a wide range of fields. In medicine, for example, it is used for diagnosis and early detection of diseases through medical imaging [8]. In the automotive industry, it is employed in driver assistance systems and autonomous vehicles to detect and recognize traffic signs, pedestrians, and real-time obstacles [9]. It is also applied in security and surveillance, satellite image processing, augmented reality, robotics, industrial automation, and many other domains where visual analysis is crucial [10].

The term Computer Vision began to be used in the mid-1960s when researchers started exploring image processing and the ability of computers to interpret visual information [11]. However, the field itself and its recognition as a specific discipline developed gradually in the following decades. In its early stages, researchers primarily focused on the recognition of simple patterns in images, such as printed characters or basic geometric shapes [12]. One of the early significant milestones was the development of the Canny edge detection algorithm in 1986, which allowed for accurate identification of object contours in images [13]. As the 1980s progressed, more sophisticated techniques were introduced, such as the use of geometric models and texture analysis to recognise objects in images [14]. However, Computer Vision remained a significant challenge due to the complexity and variability of real-world images. Since then, the field has experienced significant growth, especially in the last decade with the rise of Machine Learning and Deep

Learning methods [15].

The traditional approach to apply Machine Learning methods to Computer Vision problems can be summarised as follows. First of all, a person has to design an algorithm that extracts some descriptors from a given image (for example, descriptors such as LBP [16] or Haar [17] have been used to determine the relevant characteristics of an image and applied to face detection). Specifically, the goal is to extract descriptors that are discriminative enough to describe the content of images. The second step consists in training a Machine Learning algorithm (such as k-NN [18], Random Forest [19] or Neural Networks [20]) to perform a given task. The main problem is that we have to consider that there can be a huge variability of conditions in images; for instance lighting, occlusions, changes in angles, or atmospheric phenomena. Therefore, it might be difficult to choose the correct set of descriptors, and this might lead to unexpected behaviours of the models. This problem has been recently tackled by means of Deep Learning methods.

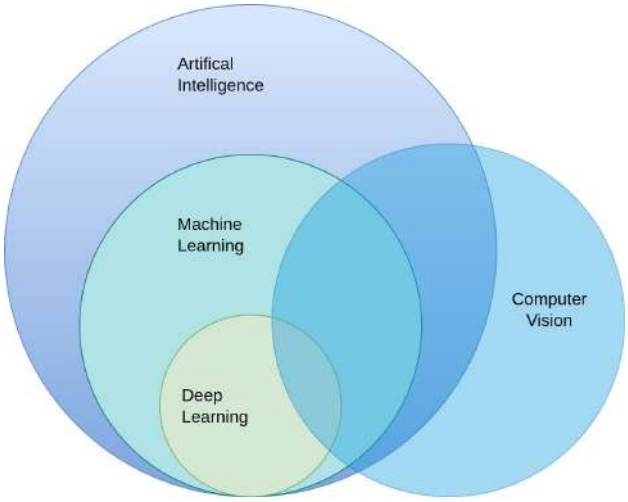


Figure 1.1: Relations among Artificial Intelligence, Machine Learning, Deep Learning and Computer Vision.

Deep Learning is a subset of techniques from Machine Learning (see Figure 1.1) based on Neuronal Networks. This set of techniques tries to learn the representation of a given image in terms of other simpler representations through a hierarchy of increasing complexity and abstraction (see Figure 1.2) [21]. This process of learning a useful representation is carried out during the training process of a network, where the lower layers of the network encode a basic representation of the problem, and the higher layers use the lower layers to build abstractions. For example, in Figure 1.2, the first layer of the network focuses on basic patterns, which are then used to construct abstractions like ears and eyes in deeper layers, which

in turn are used to build complete faces. Therefore, the main difference between traditional Machine Learning and Deep Learning is that in traditional Machine Learning, image descriptors are manually selected, whereas in Deep Learning, descriptors are learned at the same time than the algorithms are trained.

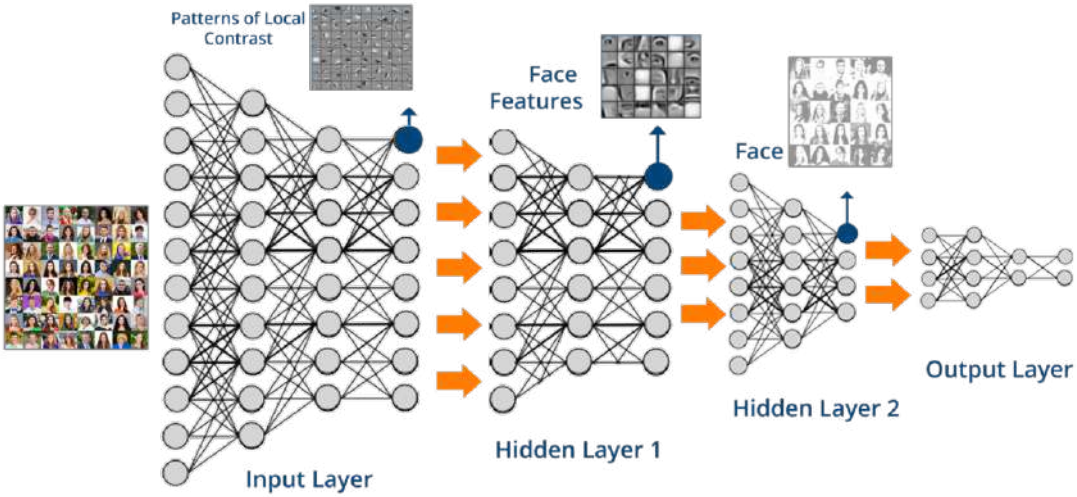


Figure 1.2: Extraction of representation from a Deep Learning model.

Deep Learning methods are currently the state-of-the-art approach to deal with most Computer Vision tasks. Among those tasks, three of them are worth mentioning due to their numerous applications. Those tasks are image classification, object detection and semantic segmentation. Image classification is probably the most well-known problem in Computer Vision, and it involves categorising an image into one of many possible classes, see Figure 1.3(a). When iterating on the classification problem, we end up with the need to detect and classify multiple objects simultaneously in a given image. Object detection is the problem of finding and classifying a variable number of objects in an image. The key difference here is the "variable" part. Unlike classification problems, the output of object detection models is of variable length because the number of detected objects can change from one image to another, as shown in Figure 1.3(b). Finally, semantic segmentation goes beyond object detection by providing a precise pixel-by-pixel classification for each detected object. This task allows for the separation and distinction of different objects within an image at the pixel level, see Figure 1.3(c). For these three tasks, given a dataset of images and their corresponding annotations, Deep Learning algorithms are trained to produce a model. Lastly, given a new image as input such trained model will output the corresponding label. In

this work, we will mainly focus on the tasks of object detection and semantic segmentation.

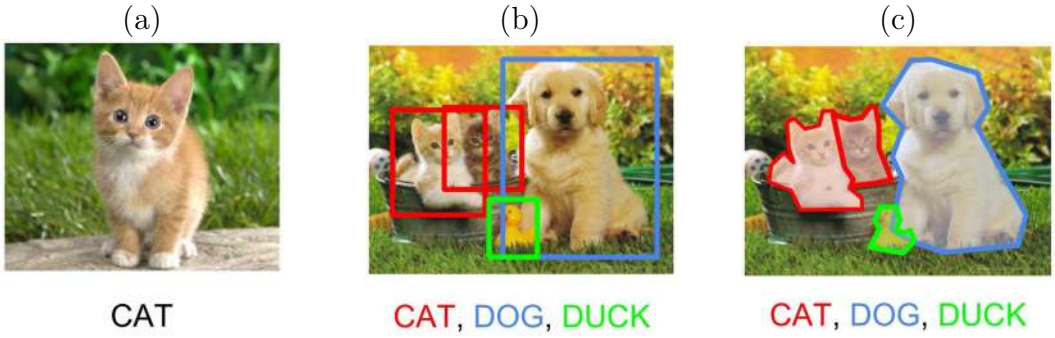


Figure 1.3: Problems of Computer Vision: (a) Image Classification. (b) Object Detection. (c) Semantic Segmentation.

Deep Learning methods for object detection are mainly based on Convolutional Neuronal Network (CNNs), a special kind of neuronal network that uses specialised layers that performs convolutions to extract meaningful features from images [22]. The CNNs models for object detection can be divided into two categories: two-phase and one-phase algorithms. In two-phase algorithms; the first step involves generating a group of potential bounding rectangles or region proposals that are considered "interesting". In the second step, these proposals are classified using convolutional neural networks. Examples of two-phase algorithms include R-CNN [23], Fast R-CNN [24], and Faster R-CNN [25]. In the case of, one-phase algorithms, they divide the image into regions, which are then processed by a CNN. Finally, these regions are modified and grouped based on the obtained prediction. Examples of one-phase algorithms are SSD [26] and YOLO [27].

In general, the first algorithms are more accurate but slower (due to the two-phase approach), which poses challenges for real-time image processing. On the other hand, the second algorithms are faster, although slightly less accurate because everything is done simultaneously. More recently, the transformer architecture has also been used to define new Deep Learning models for object detection. Example of these transformers models are DETR [28] or YOLOS [29]. The concrete details of these for object detection architectures are not necessary for the work presented in this memoir, and, therefore, we refer the interested reader to [30] — where a detailed introduction to the state-of-the-art Deep Learning methods for object detection can be found.

In the case of semantic segmentation, the Deep Learning architectures are based on either fully convolutional networks (FCN) [31] or encoder-decoder networks [32]. FCN architectures extract features from a given image using a backbone of convo-

lutional layers and generate an initial coarse classification map. The classification map is a spatially reduced version of the original image. Then, deconvolutional layers restore the original resolution of the classification map to output the final segmentation mask. In the encoder-decoder architectures, the encoder is usually made of several convolutional and pooling layers responsible for extracting the features and generating an initial coarse prediction map. In these architectures, the encoder is known as the backbone. The decoder, commonly composed of convolution, deconvolution and/or unpooling layers, is responsible for further processing the initial prediction map, increasing its spatial resolution gradually and generating the final prediction. The Unet architecture [32] was the first network to propose an encoder-decoder architecture to perform semantic segmentation in medical contexts. From that seminal work, several variants have been proposed [33]. Likewise, the object semantic segmentation case, details of these architectures are not necessary for the work presented in this memoir, hence, we refer the interested reader to [34] — where a detailed introduction to the state-of-the-art Deep Learning methods for semnatic segmentation can be found.

Once that we have introduced the context of this work, in the next section we present the limits of Deep Learning methods and our goals.

Challenges and objectives

In spite of the success of Deep Learning methods to tackle Computer Vision tasks, there are several challenges that have not been addressed yet. First of all, Deep Learning methods require a large amount of annotated images to achieve accurate results [35]. Although massive datasets such as Imagenet [36], COCO [37], and Pascal VOC [38] exist, acquiring images can be challenging in contexts like biomedicine or precision agriculture due to budget constraints, data privacy, or the need of invasive procedures [39]. Additionally, the images need to be annotated, which involves assigning labels to each object in the image and providing their positions. Therefore, image annotation is a time-consuming and expertise-demanding problem, particularly in object detection and segmentation tasks.

In the literature, we can find several techniques that deal with such limitations. Among those techniques, we can highlight methods such as transfer learning [40], data augmentation [41], or semi-supervised learning methods [42]. Some of those techniques are based on the notion of ensembles [43]. Ensemble methods combine the output of multiple models to produce more robust results. However, as many other techniques in the Deep Learning context, ensemble methods are mainly applied in image classification problems, and some of the methods developed for image classification tasks cannot be directly applied to other Computer Vision problems.

There are also technical limitations when using Deep Learning techniques. First of all, there are multiple ways of approaching the same problem, and those approaches are usually implemented in different libraries. This may require different sets of files, dependencies, and structures, making it difficult to transfer knowledge and solutions between different problems. Moreover, users of Deep Learning methods need a considerable experience to train and use Deep Learning models and this might be a challenge for many non-expert users. Finally, Deep Learning methods are often computationally intensive and require specialised hardware, such as graphics processing units (GPUs) or tensor processing units (TPUs), to achieve efficient training and inference. This can be a barrier for those who do not have access to these resources or lack the knowledge to use them effectively.

The last problem of Deep Learning methods is the use of synthetic or artificially generated datasets for testing such methods. These datasets can be useful when there is a lack of sufficient real-world data or when specific aspects of the data need to be controlled and manipulated for a more detailed analysis. Synthetic datasets can also be helpful for training and fine-tuning models in initial stages. However, if those methods are not tested against real datasets, it is unknown whether the generated models are capable of solving real-world problems. Hence, it is necessary to evaluate the performance and generalisation of Deep Learning models in real-world scenarios.

Once that we have explained some of the limitations of Deep Learning methods, we outline the objectives proposed in this work to address the aforementioned problems in the context of object detection and semantic segmentation. Namely, the main objectives of this work are:

O1. Develop new algorithms to improve the effectiveness of object detection models.

O2. Reduce the amount of resources necessary to train object detection models.

O3. Facilitate all the stages involved in the process of training and using object detection models.

O4. Generalise the methods developed in the previous objectives to other Computer Vision tasks such as sementic segmentation.

O5. Evaluate the proposed solutions in real scenarios.

The rest of this memoir is organized as follows. In Chapter 2, we will develop new methods to improve the performance of object detection models based on ensemble methods (Objective O1). Moreover, those new methods will allow us to reduce the number of images used to create new models (Objective O2). In Chapter 3, we will present an infrastructure that facilitates the construction and

usage of object detection models (Objective O3), and we will also introduce how our methods can be generalised (Objective O4). Finally, in Chapters 4 and 5, we will evaluate the methods and algorithms developed in the previous chapters using datasets from real-world problems in two domains: plant physiology and precision agriculture (Objective O5). The memoir ends with a chapter which includes some conclusions and further work, and a chapter that showcases the publications associated with this work.

Chapter 2

Ensemble Methods for Object Detection

In this chapter, we focus on providing a general approach, based on ensemble methods, to improve the accuracy and robustness of object detection models. In addition, this approach allows us to reduce the number of images that are needed to train object detection models by means of two semi-supervised learning methods.

Ensemble methods have been successfully employed in the literature to improve object detection models. For instance, the mmAP in the COCO dataset was improved from 50.6 to 52.5 in [44], or the mAP in the Pascal VOC dataset increased by 3.2% in [45]. In fact, the leading methods on datasets like Pascal VOC or MS COCO are based on the usage of ensembles [46, 47]. However, the process of ensembling object detectors poses several challenges. First of all, some ensemble approaches for object detection depend on the nature of the detection models — for example, the procedure to ensemble models explained in [48] can only be applied to models based on the FasterRCNN algorithm — therefore, these methods cannot be generalised and lack the diversity provided by the ensemble of different algorithms. Related to the previous point, those ensemble methods require the modification of the underlying algorithms employed to construct the models, and this might be challenging for many users. In order to deal with this problem, there are ensemble methods that work with the output of the models [49, 50]; but, again, they are focused on concrete models, and only work if the models are constructed using the same framework. Finally, it does not exist an open-source library that provides general ensemble methods for object detection, and this hinders their use.

We have tackled the aforementioned challenges by designing a generic method that serves to ensemble the output produced by detection algorithms; that is, *bounding boxes* which indicate the position and category of the objects contained in an image. The method can be employed with any detection model independently of its underlying algorithm and the framework employed to construct it. In particular,

the contributions of this part of the memoir are the following ones:

- We present a general method for ensembling object detectors independently of the underlying algorithm; and, in addition, we devised several voting strategies to carry out the ensembling process.

- As a by-product of our ensemble method, we defined a test-time augmentation procedure that can be applied to boost the accuracy of object detection models. Moreover, we explain how to reduce the burden of annotating images in the context of object detection using two semi-supervised learning techniques based on ensemble methods.

- We conducted a comprehensive study of the impact of our ensemble method and the devised voting strategies, and show the benefits of our method as well as the advantages of using test-time augmentation and semi-supervised learning methods.

- We implemented our methods in the `EnsembleObjectDetection` library[1]. This open-source library can be extended to work with any object detection model independently of the algorithm and framework employed to construct it.

The rest of the chapter is organised as follows. In the next section, we provide the necessary background to understand the rest of the chapter. Subsequently, our approach to ensemble object detection models, and the extension of such an approach for test-time augmentation, and semi-supervised learning is presented in Section 2.2. In Section 2.3, we present the main highlights of the library that implements our methods. After that, an analysis of the impact of our methods on different datasets is provided in Section 2.4. The chapter ends with the main conclusions of this part of the memoir.

2.1 Background

In this section, we briefly provide the necessary background and definitions needed to understand the rest of the chapter. We start by providing the main definitions used in the context of object detection.

[1]The library is available at `https://github.com/ancasag/ensembleObjectDetection`.

2.1.1 Definitions

Object detection is the task of determining the position and category of multiple objects in an image. Formally, an object detection model can be seen as a function that given an image I returns a list of detections $D = [d_1, \ldots, d_N]$ where each d_i is given by a triple $[b_i, c_i, s_i]$ that consists of a bounding box, b_i, the corresponding category, c_i, and the corresponding confidence score, s_i.

In order to evaluate the performance of detection models, we can use several metrics, but all of them need to determine the overlap of bounding boxes. To that aim, the IoU metric [51] is employed. Considering two bounding boxes b_1 and b_2, the IoU formula for finding the overlapped region between them is given by:

$$IoU(b_1, b_2) = \frac{area(b_1 \cap b_2)}{area(b_1 \cup b_2)}.$$

Now, we can use the IoU to define the mAP metric [51]. Suppose we want to evaluate our object detector on a set of test images T and that objects of a set of classes ζ, with $\#|\zeta| = n$, can appear in these images. We start by defining what a correct detection is. Given the *ground truth* of an object of a class $C \in \zeta$ (that is, a bounding box provided by an annotator), we will say that such an object is correctly detected, if our model produces a prediction where the IoU of the *ground truth* with the prediction is greater than a given threshold, and C is equal to the predicted class. The threshold value is usually set to 0.5; although this value may vary.

From the detected objects, we compute their IoU with respect to each bounding box of the ground truth. Using these values and the IoU threshold, we can determine the number of correct detections for each class in an image. Next, for each one of the images, we obtain the number of real objects of a given class $C \in \zeta$ in that image I; and we define the precision of class C in an image I as follows:

$$Precision_{C,I} = \frac{\#\text{ of correct detections of class } C \text{ in } I}{\#\text{ of objects of class } C \text{ in } I}.$$

Since we are interested in evaluating the model on the set of images T, we define the Average Precision for class C as:

$$Average\ Precision_C = \frac{\sum_{I \in T} Precision_{C,I}}{\text{total number of images } T \text{ with objects of class } C}.$$

To represent the overall performance of our model, we will take the mean of all the Average Precision defining the *mean Average Precision* (mAP) as:

$$mAP = \frac{\sum_{C \in \zeta} Average\ Precision_C}{\#\text{ of classes}}.$$

Sometimes, we use the notation mAP@threshold to indicate the used IoU threshold; for instance, if 0.5 is used as threshold, we write mAP@0.5.

Other metrics have been used in the literature to evaluate the performace of object detection models. Let us define these metrics in a formal way, using the following concepts:

- *True Positive* (TP) are the correctly detected objects; that is, the model produces a prediction of the same class of the object, and the IoU between the ground truth of the object and the predicted bounding box is over a fixed threshold.
- *False Positive* (FP) are the objects predicted by the model but that do not appear in the ground truth.
- *False Negative* (FN) are the objects that are in the ground truth but that have been not predicted by the model.

Then, we can define the precision, the recall and the F1-score as follows:

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

$$\text{Recall} = \frac{TP}{FN + TP}$$

$$\text{F1-score} = \frac{2 * TP}{2 * TP + FP + FN}$$

Roughly speaking, the precision (not to be confused with the precision of the class C in an image I) is the relationship between the number of correctly predicted detections and total predicted detections; the recall, is the ratio of the number of correctly predicted detections to the total detections; and the F1-score takes into account the precision and the recall. As in the case of mAP, we can indicate a threshold value for these metrics using precision@threshold, recall@threshold and F1@threshold. Once the basic notions about object detection have been presented, we explain how ensemble methods have been applied for object detection models in the literature.

2.1.2 Ensemble methods for object detection

Ensemble methods combine multiple models to obtain a final output [43]. These methods have been successfully employed for improving accuracy in several machine learning tasks, and object detection is not an exception. We can distinguish two kinds of ensembling techniques for object detection: those that are based on

the nature of the algorithms employed to construct the detection models, and those that work with the output of the models.

In the case of ensemble methods based on the nature of the algorithms, different strategies have been mainly applied to two-stage detectors. Some works have been focused on ensembling features from different sources before feeding them to the region proposal algorithm [52, 44], others apply an ensemble in the classification stage [53, 54], and others employ ensembles in both stages of the algorithm [48, 55, 56]. In the case of ensemble methods based on the output of the models, the common approach consists in using a primary model which predictions are adjusted with a secondary model. This procedure has been applied in [50] by combining Fast-RCNN and Faster-RCNN models, in [45] by combining Fast-RCNN and YOLO models, and in [49] by using RetinaNet and Mask R-CNN models. Another approach to combine the output of detection models is the application of techniques to eliminate redundant bounding boxes like Non-Maximum Suppression [57], Soft-NMS [58], NMW [59], fusion [60] or WBF [61]. However, these techniques do not take into account the classes of the detected objects, or the number of models that detected a particular object; and, therefore, if they are blindly applied, they tend to produce lots of false positives.

In our work, we propose a general method for ensembling the output of detection models using different voting strategies. The method is independent of the underlying algorithm and framework of the models, and allows us to easily combine a variety of multiple detection models. In addition, our method opens the door to apply techniques such as test-time augmentation.

2.1.3 Test-time augmentation

Data augmentation [62, 63] is a technique widely employed to train deep learning models that consists in generating new training samples from the original training dataset by applying transformations. There is a variant of data augmentation for the test dataset known as *test-time augmentation* [64]. This technique randomly applies a set of transformations to the test images to create new images, performs predictions on them, and, finally, returns an ensemble of those predictions.

Due to the cost of collecting data in the context of object detection, data augmentation strategies such as random scaling [45] or cropping [26] are widely employed [65]. On the contrary, and due to the lack of a general method to combine predictions of object detectors, test-time augmentation has been mainly applied in the context of image classification [64]. As far as we are aware, test-time augmentation has only been applied for object detectors in [60], and only using colour transformations. This limitation is due to the fact that some transformations, like flips or rotations, change the position of the objects in the image and this issue must be taken into account when combining the predictions. The method pre-

sented in the next section deals with this problem and allows us to apply test-time augmentation with any object detection model.

2.2 Methods

In this section, we explain our ensemble algorithm for combining the output of object detection models. Such an algorithm can be particularised with different strategies that are also explained in this section. Moreover, we explain how our algorithm can be applied for test-time augmentation, and for semi-supervised learning.

2.2.1 Ensemble of object detectors

We start by explaining the procedure that we have designed to combine object detections obtained from several sources. The input of our ensemble algorithm is a list $LD = [D_1, \ldots, D_m]$ where each D_i, with $i \in \{1 \ldots m\}$ where m is the number of sources, is a list of detections for a given image I as explained in Section 2.1. Usually, each D_i comes from the predictions of a detection model; but, as we will see in Section 2.2.2, this is not always the case. In general, each list D_i is a list of detections produced using a particular method M_i for a given image.

Given the list LD, our ensemble algorithm consists of four steps. First of all, the list LD is flattened in a list $F = [d_1, \ldots, d_k]$, since the provenance of each detection d_i is not relevant for the ensembling algorithm. Subsequently, the elements d_i of F are grouped together based on the overlapping of their bounding boxes and their classes. This step is employed to group the elements of F producing as a result a list $G = [D_1^G, \ldots, D_n^G]$ where each D_i^G is a list of detections such that for all $\bar{d}(= [\bar{b}, \bar{c}, \bar{s}]), \hat{d}(= [\hat{b}, \hat{c}, \hat{s}]) \in D_i^G$, $IoU(\bar{b}, \hat{b}) > 0.5$, and $\bar{c} = \hat{c}$; that is, those detections from the same class and that overlap at least a 0.5 IoU. At this point, each list $D_i^G \in G$ is focused on a candidate object of the image, and the size of D_i^G will determine whether our algorithm considers whether such a region actually contains an object. Namely, this decision can be taken using three different voting strategies:

- *Affirmative.* In this strategy, all the lists D_i^G are kept. This means that whenever one of the methods that produce the initial predictions says that a region contains an object, such a detection is considered as valid.

- *Majority.* In this case, only the lists D_i^G which length is greater than $m/2$ (where m is the size of the initial list LD) are kept. This means that the majority of the initial methods must agree to consider that a region contains an

object. This strategy is analogous to the majority voting strategy commonly applied in ensemble methods for image classification [66].

- *Unanimous*. In the last strategy, only the lists D_i^G which length is equal to m are kept (where m is the size of the initial list LD). This means that all the methods must agree to consider that a region contains an object.

After applying one of the aforementioned strategies, we end up with a list $G' \subseteq G$. Since each list $D_k^{G'} \in G'$ might contain several detections for the same region, the last step of our algorithm is the application of the non-maximum suppression (NMs) algorithm to each $D_k^{G'}$ [57]. The final result is a list $D = [d_1, \ldots, d_n]$ with the ensemble detections. Our ensemble algorithm is summarised graphically in Figure 2.1.

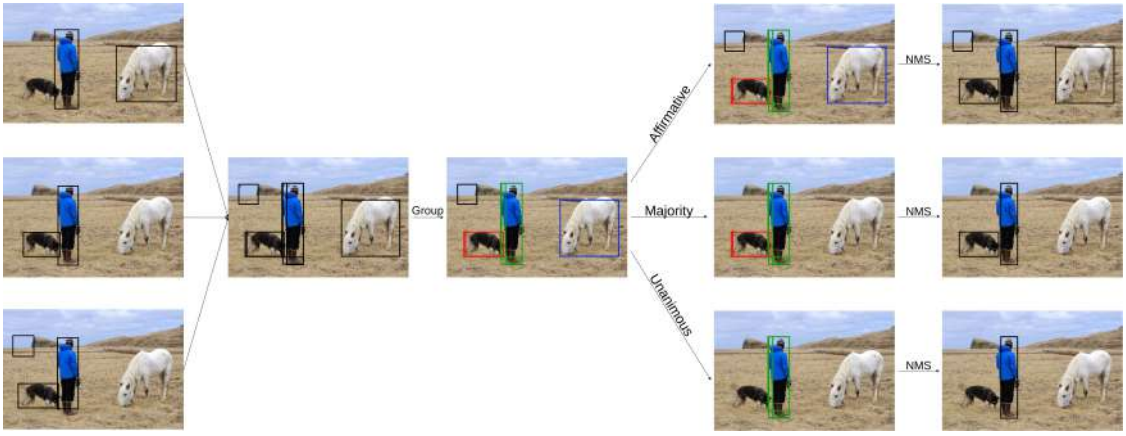


Figure 2.1: Example of the workflow of our ensemble algorithm. Three methods have been applied to detect the objects in the original image: the first method has detected the person and the horse; the second, the person and the dog; and, the third, the person, the dog, and an undefined region. The first step of our ensemble method groups the overlapping regions. Subsequently, a voting strategy is applied to discard some of those groups. The final predictions are obtained using the NMs algorithm.

From a theoretical point of view, the affirmative strategy reduces the number of objects that are not detected (false negatives) — since some objects that are not detected with a concrete approach might be detected by the others — but increases the number of incorrect detections (false positives) — this is due to the fact that the false positives obtained with each approach are accumulated. The unanimous strategy has the opposite effect, it reduces the number of false positives but increases the number of false negatives — since all the approaches

that generated the initial detections must agree to detect an object. In general, the majority strategy provides a better trade-off, and, therefore, at first glance, the affirmative and unanimous strategies might seem too lose and too restrictive respectively to be useful. However, as we will show in Section 2.4, they can produce better results than the majority approach depending on the performance of the detection models (for instance, if the detection models produce few false positives, but lots of false negatives, the affirmative strategy might be more useful than the other two strategies).

As we explained at the beginning of this section, the most natural way of producing the input of our ensemble algorithm is by using the predictions that are outputted by several object detection models. However, there are other ways, for instance, applying test-time augmentation.

2.2.2 Test-time augmentation for object detectors

Test-time augmentation (from now on, TTA) in the context of image classification is as simple as applying multiple transformations to an image (for example, flips, rotations, colour transformations, and so on), making predictions for each of the transformed images using a particular model, and finally returning the ensemble of those predictions [64]. On the contrary, in the context of object detection, TTA is not as straightforward due to the fact that there are some transformations, like flips or crops, that alter the position of the objects in the image. This explain why the works that apply TTA for object detection only apply colour operations [60] — since those transformations do not alter the position of the objects in the image. We have faced this limitation of the TTA method taking as a basis the ensemble algorithm presented previously.

First of all, we define the notion of *detection transformation*. Given an image I and a list of detections for I, D, a detection transformation, denoted by DT_t, is an operation that returns a transformed image I^t and a list of detections D^t such that the cardinal of D^t is the same of D, and all the objects detected by D in I are also detected by D^t in I^t. Roughly speaking, a detection transformation has an inverse, denoted by DT_t^{-1}, to go from the detections in I^t to the detections in I.

Example 2.2.1. Given an image I of size (W_I, H_I) where W_I and H_I are respectively the width and height of I, and a list of detections $D = [d_1, \ldots, d_n]$ such that for each $d_i = [b_i, c_i, s_i]$ and b_i is given by (x_i, y_i, w_i, h_i) where (x_i, y_i) is the position of the top-left corner of b_i, and w_i and h_i are respectively the width and height of b_i; then, the horizontal flip detection transformation denoted by DT_{hflip}, applies an horizontal flip to the image I, and returns it together with the list $D^{hflip} = [d_1^{hflip}, \ldots, d_n^{hflip}]$ where $d_i^{hflip} = [(W_I - x_i, y_i, w_i, h_i), c_i, s_i]$ — in this case,

the inverse DT_{hflip}^{-1} is itself. Another example is the equalisation transformation, denoted by $DT_{equalized}$, that applies the histogram equalisation to the image I and returns it together with $D^{equalized} = D$. Finally, the identity detected transformation, denoted by DT_I, does not modify neither I or D. These examples are depicted in Figure 2.2.

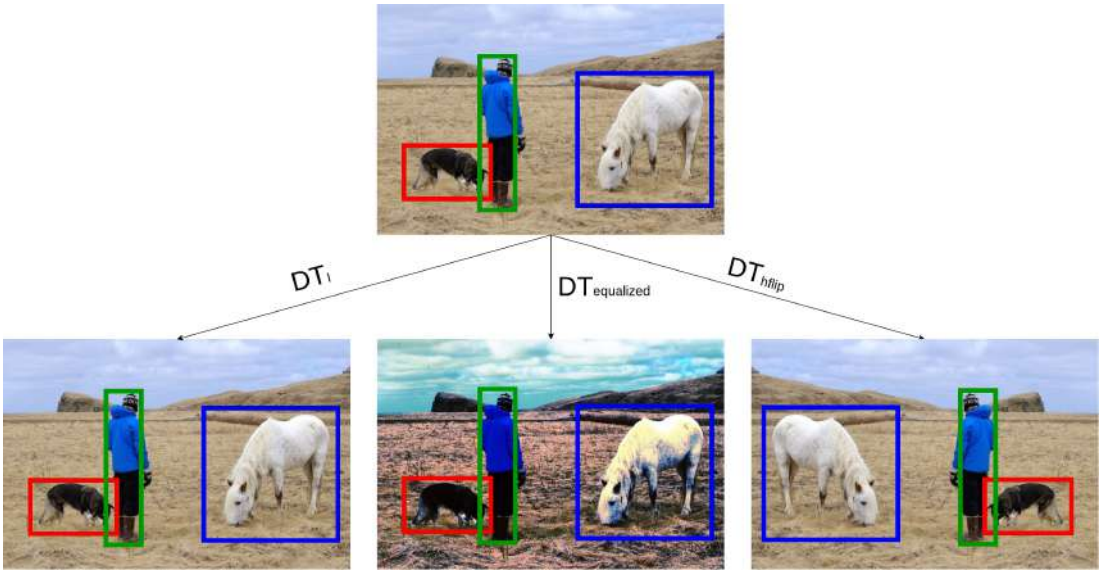


Figure 2.2: Example of none, equalisation transformation and horizontal flip detection transformation.

Now, we can define the following procedure to apply TTA for object detection. Given an image I, an object detection model M, and a list of image transformations $T_1, \ldots, T_n$, we proceed as follows. First of all, we apply each image transformation T_i to I, obtaining as a result new images $I_1, \ldots, I_n$. Subsequently, we detect the objects in each I_i using the model M, and produce the lists of detections $D_1, \ldots, D_n$. For each, (I_i, D_i), we apply a detection transformation that returns a list of detections D_i^t in the correct position for the original image I — such a detection transformation has to take into account whether the image transformation changed the position of the objects in the image. Finally, we ensemble the predictions using the procedure presented in the previous section using one of the three voting strategies. An example of this procedure is detailled in Figure 2.3.

The ensemble of models and TTA are two techniques that can be employed to improve the accuracy of object detectors, see Section 2.4. In addition, they are the basis of two semi-supervised learning techniques that tackle one of the main problems faced when training object detection models: the annotation of images.

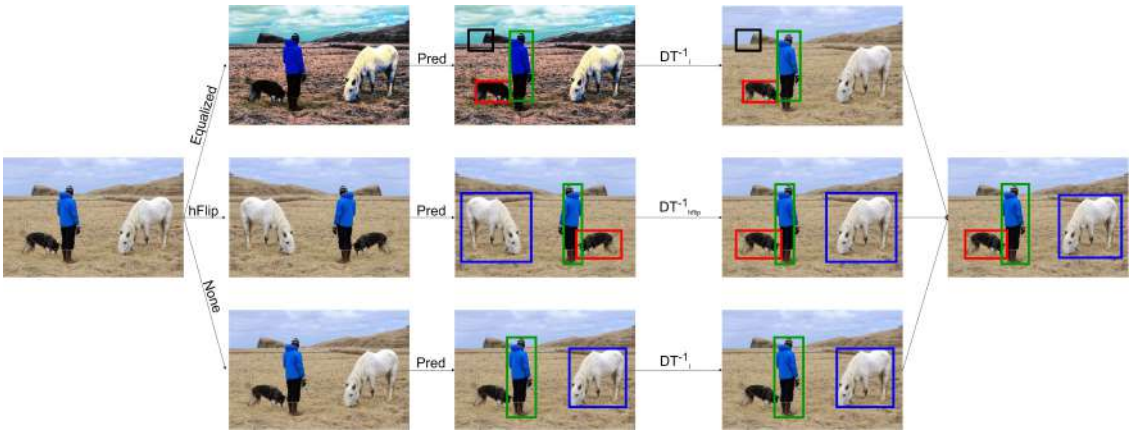


Figure 2.3: Example of the workflow of TTA for object detectors. First, we apply three transformations to the original image: a histogram equalisation, a horizontal flip, and a none transformation (that does not modify the image). Subsequently, we detect the objects in the new images, and apply the corresponding detection transformation to locate the objects in the correct position for the original image for the equalisation and none transformation, DT_I is applied, where as for the horizontal flip transformation, DT_{hflip} is applied. Finally, the detections are ensembled using the majority strategy.

2.2.3 Data and model distillation

Deep learning methods are data demanding, and acquiring and annotating the necessary amount of images for constructing object detection models is a tedious and time-consuming process that might require specialised knowledge [39]. This has lead to the development of semi-supervised learning techniques [42], a suite of methods that use unlabelled data to improve the performance of models trained with small dataset of annotated images. Self-training [42] is a particular case of semi-supervised learning where the model predictions are employed as ground truth to train a new model. However, training a model on its own predictions does not usually provide any benefit; and this has lead to the development of techniques like *data distillation* and *model distillation*.

Data distillation [67] applies a model trained on manually labelled data to multiple transformations of unlabelled data, ensembles the multiple predictions, and, finally, trains a model on the union of manually and automatically labelled data (see Figure 2.4). Similarly, model distillation [68] obtains multiple predictions of unlabelled data by using several models, ensembles the result, and trains a model with the combination of manually and automatically annotated data (see Figure 2.5). Both techniques can also be combined as shown in [69].

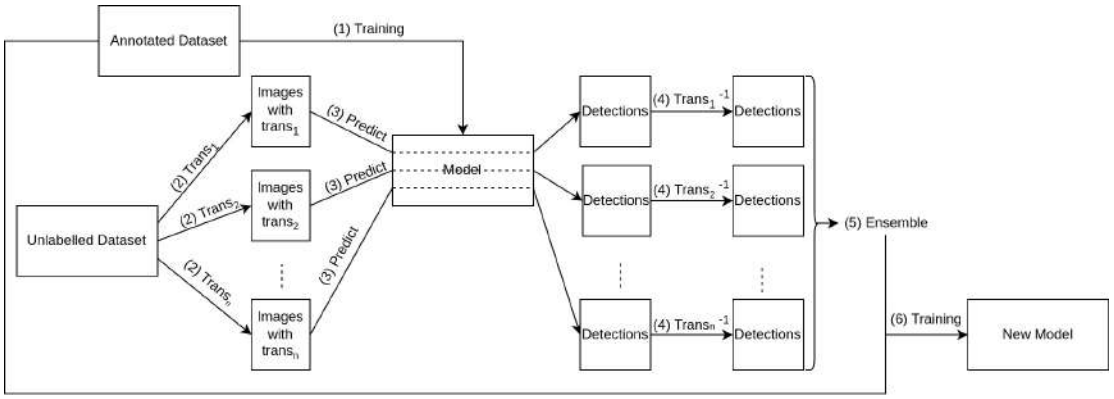


Figure 2.4: Data distillation process. (1) A model is trained with manually labelled data, (2) multiple transformations are applied to unlabelled data, (3) the model trained in (1) is used for making predictions over the images with transformations, (4) the inverses of the transformations are used to recover the position of the detections in the original image, (5) the multiple predictions are ensembled, (6) and, finally, a model is trained on the union of manually and automatically labelled data.

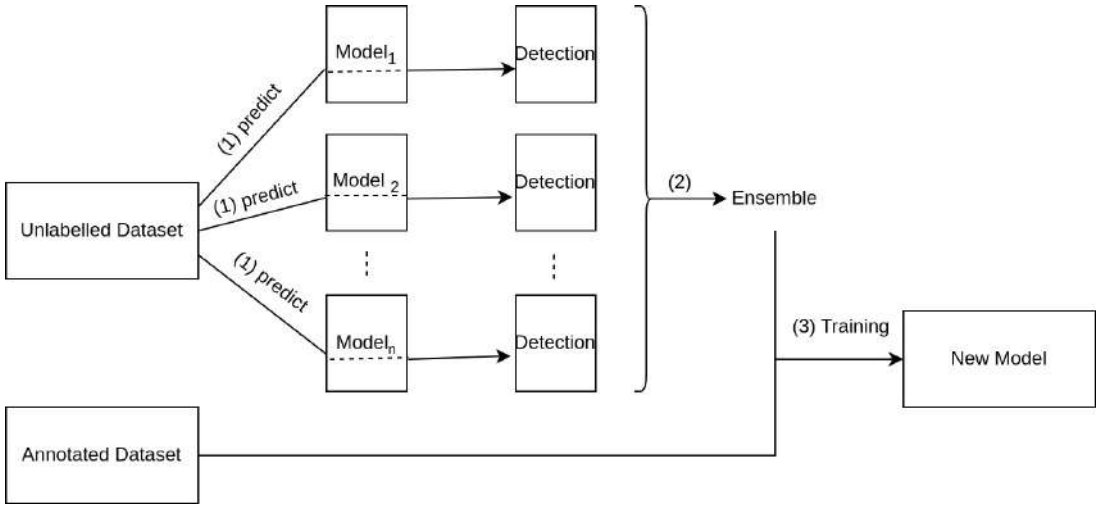


Figure 2.5: Model distillation process. After training several models with manually labelled data: (1) Those models are used to obtain multiple predictions of unlabelled data, (2) the multiple predictions are ensembled, (3) and, finally, a model is trained with the combination of manually and automatically annotated data.

Even if data distillation was applied to object detection in [67], these method have not been widely employed in this context due to the lack of a library for

ensembling predictions of detection models. This problem has been overcome thanks to the techniques and the library developed in our work, and, as we show in Section 2.4, different ensembling schemes to the one proposed in [67] will might have a better impact on the distillation methods.

2.3 The EnsembleObjectDetection library

The techniques presented in the previous section have been implemented as an open-source library called `EnsembleObjectDetection`. This library has been implemented in Python and relies on several third-party libraries like Numpy [70], OpenCV [71], or CLoDSA [72] (the last one provides the functionality to implement the image and detection transformations of the TTA procedure).

As we have previously mentioned, the `EnsembleObjectDetection` library has been designed to be applicable to models constructed with any framework and underlying algorithm. In order to provide such a functionality, we have used the *Abstract Factory Pattern* [73], a design pattern that provides interfaces for creating families of related or dependent objects without specifying their concrete classes. In our case, we have defined an abstract class, called `IPredictor` (however, this cannot be implemented directly in Python, so we have used the architecture of classes presented in Figure 2.6), with a `predict` method that takes as input a folder of images, `imgFolder`, and produces as a result a folder, `outputFolder`, that contains XML files in the Pascal VOC format with the predictions of the model for each image of the input folder — the PascalVOC format was choosen to provide a standard output format since each object detection model produces its output in a particular way. Hence, for each detection framework that we want to include in the `EnsembleObjectDetection` library, we have to provide a class that extends the `IPredictor` class and implements the `predict` method. Currently, the `EnsembleObjectDetection` library supports models trained using the Darknet [74] and MxNet [75] frameworks, and several Keras libraries [76, 77]. For example, to implement models for the Darknet library, we provide a class, `DarknetPredictor`, where the user has to provide the necessary configuration files to create a new model: the path to the weights of the model, the path to the file with the names of the classes that can be detected with the model, a threshold for the confidence level and the configuration file with the architecture of the model.

In addition, the `EnsembleObjectDetection` library provides another abstract class, called `IEnsembleStrategy` (see Figure 2.7), that provides the functionality to apply the ensemble of models and the test-time augmentation method. In particular, this class provides two abstract methods called `ensemble` and `tta`. The `ensemble` method receives as input a list of models (as `IPredictor` objects), the path to the image folder and the path to the output folder. This method will serve

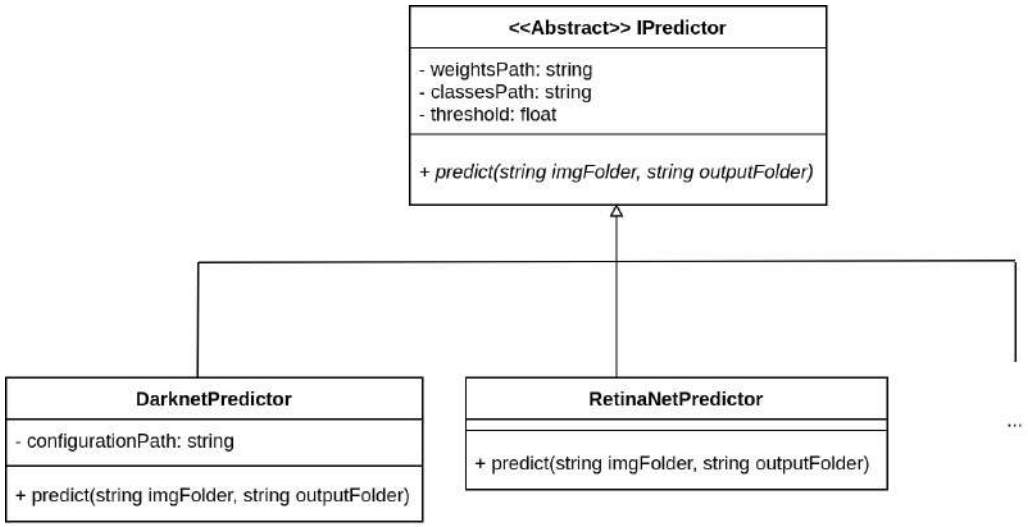


Figure 2.6: Class diagram of the **EnsembleObjectDetection library**

to use each `IPredictor` object to obtain the detections on the images, ensemble them, and store the result using the Pascal VOC format in the output folder. Similary, for the `tta` method, that receives as input a model (an `IPredictor` object), a list of image transformations provided by the CLODSA library or any other library that associates for each image transformation a detection transformation with inverse, the path to the image folder, and the path to the output folder. This method will serve to use the `IPredictor` object to obtain the detections on the transformed images ensemble the detections of the multiple transformations of an image using the given strategy after applying the corresponding detection transformation, and store the result using the Pascal VOC format in the output folder.

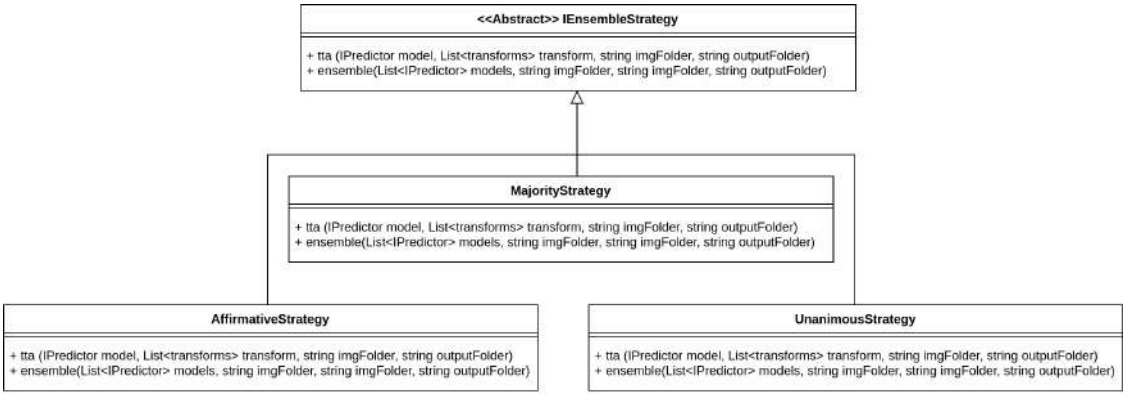


Figure 2.7: Class diagram of the **IEnsembleStrategy**.

Note that the `IEnsemblesStrategy` class does not implement neither the `tta` or `ensemble` method. Such a functionality is provided by the classes that inherit from `IEnsemblesStrategy` and that implement the different strategies to ensemble the detections. Currently, we provide three classes that inherit from the `IEnsemblesStrategy`, and that implement respectively the affirmative, majority, and unanimous strategies presented previously. This design allows us to easily include new ensemble strategies without modifying the already implement code.

Finally, the functionality to apply the data and model distillation techniques presented in Section 2.2.3 is not provided by the `EnsembleObjectDetector` library. This is due to the fact that in order to apply those semi-supervised learning methods, we need to train object detection models; and the `EnsembleObjectDetector` library is focused on using already trained models. However, providing those semi-supervised learning methods in a simple way can facilitate the construction of object detection models to many non-expert users; and, therefore, we have developed an end-to-end application to build object detection models that incorporates those semi-supervised learning methods and that will be presented in the next chapter. We focus now on the results that can be obtained thanks to our methods.

2.4 Results

In this section, we conduct a thorough study of the ensemble methods presented previously by using three different datasets.

2.4.1 Pascal VOC

In the first case study, we used the Pascal VOC dataset [38], a popular project designed to create and evaluate algorithms for image classification, object detection and segmentation. This dataset consists of natural images containing objects of 20 categories; and, the metric to evaluate the performance of detection models in this dataset is the mAP@0.5 (since we use the same IoU threshold troughout this section, we just write mAP).

For our experiments with the Pascal VOC dataset, we have employed 5 models pre-trained for this dataset using the MxNet library [75]; namely, a Faster R-CNN model, two YOLO models (one using the Darknet backbone and another one using the MobileNet backbone) and two SSD models (one using the ResNet backbone and the other using the MobileNet backbone). The performance of these models on the Pascal VOC test set is given in the first five rows of Table 2.1. As can be seen in such a table, the best model is the YOLO model using the Darknet backbone with a mAP of 69.78%. Such a mAP can be greatly improved by using model ensembling and TTA.

Datasets	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Faster R-CNN	0.69	0.70	0.77	0.71	0.64	0.50	0.78	0.78	0.80	0.45	0.74	0.58	0.79	0.80	0.79	0.70	0.42	0.75	0.67	0.77	0.67
SSD MobileNet	0.62	0.59	0.70	0.61	0.51	0.33	0.68	0.71	0.78	0.43	0.57	0.61	0.69	0.79	0.71	0.61	0.36	0.59	0.60	0.77	0.67
SSD ResNet	0.64	0.62	0.80	0.70	0.57	0.42	0.78	0.79	0.88	0.50	0.73	0.63	0.78	0.80	0.80	0.70	0.39	0.69	0.65	0.79	0.69
YOLO Darknet	0.69	**0.80**	0.72	0.70	0.57	0.60	0.80	0.80	0.81	0.43	0.75	0.63	0.78	0.81	0.71	0.70	0.39	0.71	0.65	0.79	0.70
YOLO MobileNet	0.59	0.62	0.71	0.52	0.49	0.43	0.70	0.71	0.70	0.36	0.66	0.47	0.68	0.71	0.62	0.61	0.24	0.60	0.55	0.70	0.61
All affirmative	0.77	0.79	0.80	0.79	**0.72**	0.64	**0.87**	0.86	0.88	**0.55**	0.83	0.68	**0.87**	0.87	0.80	0.78	0.51	0.79	**0.75**	0.78	0.76
All majority	0.68	0.71	0.72	0.71	0.60	0.44	0.79	0.80	0.80	0.45	0.76	0.63	0.80	0.81	0.71	0.70	0.39	0.71	0.65	0.78	0.62
All unanimous	0.51	0.54	0.63	0.45	0.35	0.27	0.62	0.63	0.63	0.32	0.53	0.42	0.63	0.63	0.63	0.53	0.17	0.54	0.51	0.62	0.54
Three best affirmative	**0.77**	0.79	**0.80**	**0.79**	0.71	**0.64**	0.86	**0.87**	**0.89**	0.55	**0.83**	**0.69**	**0.87**	**0.88**	**0.80**	**0.78**	**0.51**	**0.79**	0.75	**0.84**	**0.76**
Three best majority	0.71	0.71	0.80	0.71	0.60	0.52	0.80	0.80	0.81	0.51	0.76	0.64	0.80	0.81	0.80	0.70	0.47	0.71	0.71	0.78	0.70
Three best unanimous	0.61	0.63	0.72	0.63	0.52	0.35	0.71	0.72	0.81	0.39	0.61	0.57	0.71	0.72	0.72	0.62	0.33	0.62	0.58	0.71	0.62
Three worst affirmative	0.73	0.77	0.79	0.77	0.66	0.56	0.78	0.79	0.80	0.52	0.80	0.62	0.79	0.80	0.80	0.77	0.48	0.76	0.73	0.79	0.74
Three worst majority	0.66	0.71	0.71	0.62	0.60	0.43	0.70	0.71	0.80	0.44	0.68	0.56	0.71	0.80	0.72	0.70	0.39	0.69	0.64	0.79	0.69
Three worst unanimous	0.52	0.54	0.63	0.54	0.44	0.27	0.62	0.63	0.63	0.32	0.53	0.49	0.63	0.63	0.63	0.53	0.24	0.53	0.50	0.62	0.53

Table 2.1: Results for the Pascal VOC dataset applying our model ensemble algorithm. The first five rows provide the result for the base models. The next three rows correspond with the ensemble for the base models. Rows 9 to 11 contain the results of applying the ensemble techniques to the three best base models (SSD ResNet, YOLO Darknet and Faster R-CNN); and the last three rows contain the results of ensembling the worst three best models (YOLO MobileNet, SSD MobileNet and Faster R-CNN). The best results are in bold face.

For model ensembling, we conducted an ablation study by considering the ensemble of the five models, the ensemble of the three models with the best mAP (that are Faster R-CNN, YOLO with the Darknet backbone, and SSD with the ResNet backbone), and the three models with the worst mAP (that are YOLO with the Mobilenet backbone and the two SSD models). The results for such an ablation study are provided in the last 9 rows of Table 2.1. In those results, we can notice that all the ensembles conducted with the affirmative strategy obtained better results than the individual models — the best result were obtained by ensembling the three best models (mAP of 77.50%, almost an 8% better than the best individual model). On the contrary, the unanimous strategy produced worse results than the individual models; and the majority strategy only achieved a better mAP when the three best models were combined. These results are due to the fact that the individual models produce few false positives, and some objects that are detected by one of the models are missed by the others. Therefore, the affirmative strategy helps to greatly reduce the number of false negatives but without considerably increasing the number of false positives; on the contrary, the unanimous strategy is too restrictive and increases the number of false negatives. Something similar happens with the majority strategy. If we focus on the results for each particular category, we can notice an improvement of up to a 10% with respect to the results obtained by the best individual model. In addition, we applied TTA to all the base models to improve their accuracy. Namely, we have applied three kinds of data augmentations: colour transformations (applying gamma and histogram normalitation, and keeping the original detection), position transformations (applying a horizontal flip, a rotation of 10^{o}, and keeping the original prediction) and the

	No TTA	TTA Colour			TTA Position			TTA All		
		Aff.	Maj.	Una.	Aff.	Maj.	Una.	Aff.	Maj.	Una.
Faster R-CNN	0.69	0.69	0.69	0.08	0.53	0.53	0.22	0.63	0.61	0.21
SSD MobileNet	0.62	0.63	0.63	0.09	0.58	0.58	0.52	0.61	0.58	0.47
SSD ResNet	0.64	0.70	0.70	0.08	0.65	0.65	0.60	0.68	0.63	0.09
YOLO Darknet	0.69	**0.71**	**0.71**	0.09	0.68	0.68	0.63	0.70	0.68	0.57
YOLO MobileNet	0.59	0.61	0.61	0.10	0.57	0.57	0.50	0.61	0.58	0.44

Table 2.2: Results for the Pascal VOC dataset applying TTA. In the first column, we provide the result for the models without applying TTA. The rest of the table is divided into three blocks of three columns (one per each voting strategy): the first block provides the results with colour transformations, the second contains the results for position transformations, and, the last block presents the results combining all the transformations. The best results are in bold face.

combination of both. Moreover, for each augmentation scheme, we have applied the three voting strategies, see Table 2.2. As in the case of model ensembling, all the models were improved (the improvement ranges from 0.08% to 5.54%) thanks to TTA when using the affirmative strategy; but the most beneficial augmentation scheme varies from model to model. For instance, the SSD model with the ResNet backbone as improved with the three augmentation schemes; but, the Faster R-CNN model only improved with the colour scheme. Regarding the other voting strategies, the unanimous strategy obtained worst results than the original models; and the majority strategy only got better results in some cases. The explanation for these results is the same provided previously for model ensembling. It is also worth noting that adding more augmentation techniques does not always improve the ensembling results.

As a conclusion for this study, we can say that model ensembling is more beneficial than TTA; and, this is due to the fact that the former introduces a higher variability (thanks to the heterogeneity of models) in the predictions than the latter.

2.4.2 Stomata detection

In the second example, we applied TTA and data distillation to two proprietary datasets of stomata images. Stomata (singular "stoma") are pores on a plant leaf that allow the exchange of gases, mainly CO_2 and water vapor, between the atmosphere and the plant — this problem will be addressed more thoroughly in Chapter 4.

In order to analyse stomata of plant leaves, plant biologists take microscopic

images of leaves, and manually measure the stomata density in those images. This is a tedious, error-prone, time-consuming and subjective task due to the large number of stomata in each image but, it can be automatised by means of detection algorithms. In particular, we have constructed a stomata detection model using the YOLOv3 algorithm implemented in the Darknet framework [78]. The YOLO model was trained by using 4,050 stomata images, and it was evaluated on a test set of 450 images using the F1-score and the mAP (we used 0.5 as threshold), see the first row of Table 2.3. As can be seen from that table, the number of false positives is considerably higher than the number of false negatives, and this will have an impact in the voting strategy to apply. In this section, we show how such a model can be improved thanks to TTA.

Datasets	F1-score	TP	FP	FN	mAP
Original	0.90	16600	2341	1239	0.84
Affirmative	0.88	**17003**	3600	**836**	0.80
Majority	0.92	16509	1551	1330	0.84
Unanimous	0.80	12272	**589**	5567	0.61
Colour	**0.93**	16502	1324	1337	**0.85**
Flips	0.92	16480	1448	1359	0.85
Rotations	0.91	16463	1999	1376	0.82
Flips & colour	0.92	16572	1529	1267	0.84
Flips & rotations	0.92	16580	1659	1259	0.84
Rotations & colour	0.92	16556	1633	1283	0.84

Table 2.3: Results of our YOLO model for stomata detection. In the first row, we provide the results for the original model. In the next three rows, we have applied TTA with 9 transformations using the three voting strategies; and, in the next six rows, we have applied the TTA method for different kinds of transformations and using the majority strategy. The best results are in bold face.

First of all, we applied TTA by using 9 transformations: three colour transformations (histogram normalisation, gamma correction and Gaussian blurring), three flips (vertical, horizontal and both) and three rotations (90^{o}, 180^{o} and 270^{o}). Moreover, we applied the three voting schemes — the results for these experiments are given in rows 2 to 4 in Table 2.3. As can be seen in that table, the only strategy that improved the results is the majority approach, that improved a 2% the F1-score. Note that each strategy had the expected effect, the affirmative scheme increased the number of FP and decreased the number of FN; on the contrary, the unanimous strategy had the opposite effect. Then, the majority strategy provided

the best trade-off by considerably reducing the number of FP but only slightly increasing the number of FN.

In addition to the above results, we have also inspected the impact of each kind of transformation for this dataset. In particular, we applied TTA using colour transformations, rotation transformations, flip transformations, and their combinations — see the last 6 rows of Table 2.3. In this case, we only included the majority strategy in Table 2.3 since the other strategies produced the same effect previously explained. As can be seen from those results, the same improvement obtained using all the transformation can be achieved by using only some of them — this considerably reduces the time needed to apply TTA. In fact, we achieved better results by applying TTA using only colour transformations. This indicates that it is necessary to study different combinations of transformations to find the one that produces the best results; and this shows the benefits of having a library like the one presented here.

Finally, we also studied the benefits of data distillation in the context of stomata detection. It is worth noting that each stomata image contains approximately 45 stomata, and, hence, annotating those images is a time-consuming task. Therefore, the application of semi-supervised learning techniques, like data distillation, can reduce the burden of annotating those images. In our data distillation experiments, see Table 2.4, we have employed a dataset of stomata images from a different variety than the original dataset employed for the results of Table 2.3.

Such a dataset contains 450 annotated images for training, 150 annotated images for testing, and 1,620 unlabelled images. Using the 450 annotated images, we constructed a YOLO model that achieved a F1-score of 0.85 and a mAP of 0.8 when using a confidence threshold of 0.25, and a F1-score of 0.83 and a mAP of 0.79 when using a confidence threshold of 0.5. Using such models, we have applied data distillation using three schemes: applying colour transformations (gamma correction, histogram, normalisation and Gaussian blurring), applying flips (vertical, horizontal and both), and combining colour and flip transformations. Moreover, we used the three voting schemes, see Table 2.4. In this case, the best strategy consists in applying all the transformations together with the unanimous strategy. This improved a 3% the F1-score value, and an 8% the mAP. Note that using the unanimous strategy, we can increase the confidence threshold to consider a detection as correct since using such a strategy the new model is trained with images where the detections have been agreed by the predictions of the 9 transformations. Based on the results obtained for this example, we have shown the benefits of trying different alternatives for TTA; and, in addition, that techniques like data distillation can produce accurate models starting from small datasets of images.

	Confidence 0.25		Confidence 0.5	
Datasets	F1-score	mAP	F1-score	mAP
Original	0.85	0.80	0.83	0.79
All Affirmative	0.84	0.82	0.80	0.82
All Majority	0.86	0.85	0.87	0.85
All Unanimous	0.86	**0.88**	**0.88**	**0.88**
Colour Affirmative	0.86	0.86	0.83	0.86
Colour Majority	0.82	0.78	0.77	0.78
Colour Unanimous	0.74	0.77	0.62	0.77
Flips Affirmative	0.83	0.80	0.76	0.78
Flips Majority	0.04	0.61	0	0.55
Flips Unanimous	0.01	0.30	0	0.30

Table 2.4: Results of data distillation for the stomata dataset. This table is divided into two blocks: in the first block (columns 2 and 3), we use a confidence threshold of 0.25; and, in the second block (the last two columns), we use a confidence threshold of 0.5. In the first row of the table, we provide the result for the original model. In the next three rows, we present the results of applying data distillation with colour and flip transformations. In rows 5 to 7, we include the results of applying data distillation only using colour transformations, and the last three rows present the results of applying data distillation using flip transformations. The best results are in bold face.

2.4.3 Table detection

In the last case study, we analysed the effects of model ensembling and model distillation for table detection — an important problem since it is a key step to extract the semantics from tabular data [79]. To this aim, we employed the ICDAR2013 dataset [80], and the Word part of the TableBank dataset [81]. Both datasets have been designed to test table detection algorithms; however, the ICDAR2013 dataset is too small to directly apply deep learning algorithms (it only contains 238 images), and the TableBank dataset is a big enough dataset (160k images) but it was semi-automatically annotated and contains several incorrect annotations. Therefore, these two datasets provide the perfect scenario for applying model ensembling and model distillation. In particular, we have trained different models for the ICDAR2013 dataset, and have improved them by using our ensemble algorithm, and by applying model distillation using the TableBank dataset.

In our experiments, we have split the ICDAR2013 dataset into a training set of 178 images and a testing set of 60 images. Using the training set, we have constructed three models using the YOLO algorithm, implemented in the Darknet library; the SSD algorithm, implemented in MxNet; and the Mask RCNN algorithm, implemented in the Keras library [82] — note that we have employed different libraries and algorithms, but our ensemble library can deal with all of them. These three models have been evaluated (see the three first rows of Table 2.5) in the testing set using the W1Avg F1-score [83], a metric that computes the weighted sum of the F1-score using different IoU thresholds ranging from 0.6 to 0.9 — the F1-score at 0.6 is employed to measure the number of tables that are detected, even if the detection bounding boxes are not perfectly adjusted; and, on the contrary, the F1-score at 0.9 measures the tables that are detected with a bounding box perfectly adjusted to them. As can be seen in Table 2.5, the best model is obtained using the YOLO algorithm (WAvgF1-score of 0.63).

The three models can be improved thanks to model ensembling, and model distillation. First of all, we have ensembled the three models using the three voting strategies (see rows 3 to 5 of Table 2.5), and we have obtained an improvement of 2% using the affirmative strategy, and a 6% using the majority approach; as we have seen previously, the unanimous approach is too restrictive and obtains the worst results.

Moreover, we have applied model distillation using the TableBank dataset; applying the three voting strategies, and retraining the three models, see the last 9 rows of Table 2.5. Using this approach we have improved the SSD model a 4%, the Mask R-CNN model a 6%; but, the YOLO model did not improve at all. However, if we inspect the F1-score value at 0.6, the improvement is more evident, SSD improved a 2%, Mask R-CNN a 10%, and YOLO a 5%. This is due to the fact that the ensemble of models usually produces bounding boxes that are not perfectly adjusted to the objects; the issue of improving those adjustments remain as further work.

As a conclusion of this example, we can again notice the benefits of applying our ensemble algorithm, and the improvements that can be achieved applying model distillation when a large dataset of images is available, even if it is not annotated. Seeing the benefits of this study of applying our ensemble algorithm, we believe that it could be extended to other computer vision tasks, for example, semantic segmentation, but that remains as further work.

2.5 Conclusions

In this part of the memoir, we have presented an ensemble algorithm that works with the bounding boxes produced by object detection models, and, hence, it is

	F1@0.6	F1@0.7	F1@0.8	F1@0.9	WAvgF1
Mask R-CNN	0.58	0.52	0.39	0.19	0.39
SSD	0.85	0.79	0.62	0.26	0.59
YOLO	0.83	0.8	0.69	0.33	0.63
Affirmative	0.86	0.81	0.69	0.37	0.65
Majority	**0.88**	**0.84**	**0.75**	**0.42**	**0.69**
Unanimous	0.4	0.36	0.28	0.08	0.26
Mask R-CNN Affirmative	0.58	0.54	0.39	0.11	0.37
Mask R-CNN Majority	0.68	0.63	0.49	0.13	0.45
Mask R-CNN Unanimous	0.57	0.48	0.29	0.06	0.32
SSD Affirmative	0.77	0.71	0.58	0.24	0.54
SSD Majority	0.87	0.8	0.67	0.32	0.63
SSD Unanimous	0.82	0.74	0.56	0.26	0.56
YOLO Affirmative	0.75	0.71	0.57	0.18	0.52
YOLO Majority	0.88	0.8	0.67	0.32	0.63
YOLO Unanimous	0.77	0.69	0.52	0.16	0.50

Table 2.5: Results for the ICDAR2013 dataset applying model ensembling and model distillation. The first three rows are the results for the base models. The next three rows include the results of applying our ensemble method with the three different voting strategies; and, the next three blocks provide the results of applying model distillation to the three base algorithms. The best results are in bold face.

independent of the underlying algorithm employed to construct those models. Our ensemble algorithm can be particularised with three voting strategies (affirmative, majority, and unanimous) that have different effects depending on the performance of the base models. Namely, the affirmative strategy works better when the detections of the base models are mostly correct (that is, there are few false positives) but several objects are left undetected (that is, there are lots of false negatives); the unanimous strategy obtains better results in the opposite case; and, the majority strategy provides a better trade-off when there is not a significant difference between false negatives and false positives in the base models.

In addition, the ensemble method presented here has been employed to define a test-time augmentation procedure for object detection that improves the accuracy of object detection models. Moreover, the ensemble of models and the test-time augmentation procedure are the basis for data and model distillation, two semi-

supervised learning techniques that can considerably reduce the number of images that must be manually annotated to train an object detection model; but that, up to now, had not been broadly adopted in the context of object detection due to the lack of a simple to use ensemble method.

As a by-product of this work, we have developed `EnsembleObjectDetection`, an open-source library that implements all the methods presented troughout this chapter. This library provides support for models constructed with several algorithms and deep learning frameworks, and can be easily extended with others models. Our methods and library have been tested with several datasets, and we have improved some models up to a 10%.

In this chapter, we have seen how we can improve the performance of object detection models, but such models must be built first, and this can be a challenge that we address in the next chapter.

Chapter 3

Simplifying the construction and usage of object detection models

In the previous chapter, we have focused on improving the accuracy of object detection models. However, for many users, building and using those models is not a straightforward task. In this chapter, we explain our work to simplify the construction and usage of object detection models.

The process of training and using object detection models poses several challenges. First of all, for both the creation and usage of those models, it is necessary to have some programming skills, and know how to use the detection algorithms and the variety of libraries that implement them. Another problem that arises when building detection models is that they need a large number of annotated images, and this is a tedious, error-prone, and time-consuming task that may require expert knowledge [39]. Furthermore, annotation tools for object detection might produce the annotation files in a format that is not suitable for all object detection frameworks. In addition, once object detection models are trained, using them is not trivial since most of them can only be used within the framework where they were built; and, therefore, it is necessary to have the necessary dependencies installed in the users' computers, and know how the specific framework works. Finally, it is usually not possible to interact with the models' predictions due to the lack of a simple and intuitive graphical interface designed for that purpose.

Taking into account the aforementioned problems, we have developed an open-source end-to-end application that allows users to conduct three tasks: annotate images for object detection, build object detection models, and use them. This tool is called LabelDetection[1] and its main features are described as follows.

- LabelDetection is a graphical tool that helps the user in all the steps required to train a variety of object detection algorithms.

[1] `https://github.com/ancasag/LabelDetection`

- LabelDetection can be employed to train object detection models not only from fully annotated datasets but also from partially annotated datasets thanks to the semi-supervised techniques presented in the previous chapter.
- LabelDetection can be used as the graphical interface for a wide variety of object detection models trained used different libraries. In addition, LabelDetection allows users to interact with the predictions generated by those models.
- LabelDetection can improve the accuracy of object detection models thanks to test-time augmentation, and without writing a single line of code.

In the rest of the chapter, we explain how LabelDetection has been designed and implemented. Namely, we first analyse the existing libraries and annotation tools for object detection. Subsequently, in Section 3.2, we introduce the architecture and design of LabelDetection. We use this tool for the detection of wheat heads to show the feasibility and benefits of using LabelDetection in Section 3.3. In Section 3.4, we generalise our work to other Computer Vision tasks. Finally, we end the chapter with some conclusions.

3.1 Background

In this section, we briefly introduce the existing libraries and annotation tools for object detection.

3.1.1 Object detection libraries

Deep learning for object detection is a growing field where new architectures and algorithms are publicly released in a monthly basis [84]. However, in their initial form, most new architectures are released as research artefacts that are not prepared for using them in custom datasets. This problem has been solved with the development of several object detection libraries, see Table 3.1, that provide a common pipeline to train different detection models. The outstanding work conducted by the developers of object detection libraries might be enhanced by including two features that are missed in all those libraries.

The first feature that is missing in these libraries is a simple way of producing the input dataset that will be used for training the algorithms. Each library requires images annotated in a particular format (being Pascal VOC and COCO the most common formats), and structured in a particular way (split of training and test sets, and configuration files). This hinders the usage of the libraries since the task of annotating the images must be conducted in an external tool, and the

Library	Language	Year	Underlying library	Annotation format	# models
Darknet detection [85]	C++	2018	Darknet	YOLO	7
YOLOv7 [86]	Python	2022	PyTorch	YOLO	6
Ultralytics [27]	Python	2022	PyTorch	COCO, YOLO, VOC, CSV	2
Detectron [87]	Python	2019	PyTorch	COCO, VOC, KITTI	14
IceVision [88]	Python	2020	PyTorch	COCO, VOC, YOLO, KITTI	4
MaskRCNN-benchmark [89]	Python	2018	PyTorch	COCO, VOC	2
MXNet Detection [90]	Python	2019	MXNet	COCO, VOC, YOLO, MxNet RecordIO	11
MMDetection [91]	Python	2019	PyTorch	COCO, VOC, YOLO, TXT	35
SimpleDet [92]	Python	2019	MXNet	COCO, VOC, KITTI	9
Tensorflow Detection API [55]	Python	2019	Tensorflow	COCO, VOC, TFRRecord	4
Tensorpack [93]	Python	2019	Tensorflow	COCO, VOC, YOLO, TFRecord	6

Table 3.1: General features of libraries for object detection.

output produced by those tools must be manually organised, and in some cases transformed to the correct format.

The second issue with object detection libraries is the lack of an interface to interact with the predictions produced by the trained models. Object detection models can be mainly employed within the library that was used for producing them and, hence, some programming skills are required. Moreover, the predictions produced by the models in a given image are usually drawn on the image; and, hence, it is not possible to interact with them (that is, add, remove or edit the predicted bounding boxes), a task that is necessary when using the detection models for analysing images. These two missing features could be provided by annotation tools for object detection.

3.1.2 Annotation tools for object detection

Annotation tools are a fundamental component for training computer vision algorithms. These tools allow developers to easily label objects in images or videos, and save the annotations to later train the machine learning models. These tools improve annotation accuracy, since they can automate much of the annotation process, and this allows a faster and more efficient annotation, enabling the annotation of large amounts of data in a short amount of time.

There are various object detection annotation tools, each with their own features, see Table 3.2. Many of these tools can be used locally so it is not necessary to upload the dataset to the cloud avoiding in this way privacy issues. There are also online tools that avoid the installation of different libraries in the local computer and allow collaborative annotation.

When working with object detection annotation tools, one of the main challenges is that each tool may have its own format for outputting annotated images. For example, the output of LabelImg are XML files in the Pascal VOC format, whereas YOLOMark stores labels in the YOLO format using txt files. Hence, the

Tool	Language	Year of creation	Output format	Local/Online
LabelImg [94]	Python	2015	VOC, YOLO	Local
CVAT [95]	Python	2016	VOC, COCO, YOLO, KITTI, JSON	Local
VGG Image Annotator (VIA) [96]	JavaScript	2016	JSON, CSV, COCO, VOC, Avoc	Online
RectLabel [97]	Mac App	2017	VOC, COCO, YOLO, CSV, Custom	Local
YOLOMark [98]	Python	2018	YOLO	Local
VoTT [99]	TypeScript	2018	COCO, VOC, YOLO, TFRecord, CSV, Custom	Online or Local
Labelbox [100]	TypeScript	2018	COCO, VOC, YOLO, TFRecord, CSV, Custom	Online
Supervisely [101]	Python	2018	COCO, VOC, YOLO, TFRecord, KITTI, Custom	Local
Anno-Mage [102]	TypeScript	2019	JSON, CSV, YOLO, VOC, Custom	Online
CV-Toolkit [103]	Python	2020	COCO, VOC, YOLO, TFRecord, Custom	Local
Labelbox Training Data Platform [100]	TypeScript	2020	COCO, VOC, YOLO, TFRecord, Custom	Online

Table 3.2: Annotation tools for object detection.

output of LabelImg cannot be directly used to train a model with the Darknet library, or the output of YOLOMark with IceVision. It is worth noting that converting annotated data from one format to another can be an error-prone process, which may lead to a loss of information. Therefore, it is important to have a tool that, in addition to annotate images, allows us to train different object detection models with different libraries, thus avoiding annotation format problems; and this is what we have tried to achieve with LabelDetection.

3.2 LabelDetection

LabelDetection is a graphical tool implemented in Python, and developed using LabelImg as a basis. LabelDetection solves the problems of both object detection libraries and annotation tools. First, it provides the necessary features to annotate a dataset of images and convert it to multiple formats suitable for most object detection libraries. Moreover, it generates all the necessary files to train an object detection model using different libraries from an annotated dataset. Finally, it allows users to employ object detection models trained with different libraries and interact with the predictions. It is worth noting that users of LabelDetection can use any of the aforementioned three features (annotation, training and prediction) independently of the others. The rest of this section is devoted to explain how we have developed such a functionality.

3.2.1 LabelDetection for image annotation

LabelDetection uses the features provided by LabelImg to allow efficient and precise annotation of objects in images. The use of LabelImg as a basis of LabelDetection was due to the fact that LabelImg is a widely used tool for annotating object detection datasets, and has been tested by many users. To use LabelDetection for annotation, the first step consist in loading the images into the user interface. The next step is to delimit each object of the image and provide its associated label,

see Figure 3.1. Additionally, LabelDetection allows adjusting the position and size of the annotation box to ensure that it fits perfectly around the object.

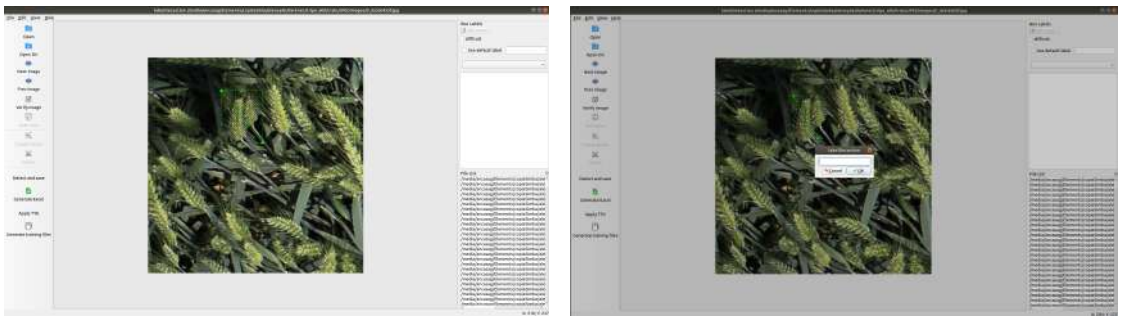


Figure 3.1: Create a bounding box and provide a label to the object.

Once all objects in the image have been annotated, the annotations are saved in an XML file, using the Pascal VOC format. Such an annotation can be used to train object detection models in different deep learning frameworks as we will see in the next section.

3.2.2 LabelDetection for training models

LabelDetection helps users in the process of training object detection models with different libraries. Such a functionality has been implemented in Python and relies on several third-party libraries like Numpy [70], OpenCV [71], CLoDSA [72] and the `EnsembleObjectDetection` library presented in the previous chapter. We start by explaining the common issues faced when training an object detection model and how they have been solved in LabelDetection.

To be able to use an annotated dataset to train an object detection algorithm with a given library, it is necessary that the dataset must be stored in a concrete format. In addition, the dataset must be split into a training and testing set. However, the algorithms are sensitive to the folder structure containing the training and testing files, and such an organisation depends on the concrete algorithm and library. Moreover, object detection algorithms require several configuration files, that also depend on the particular algorithm and library implementing them.

LabelDetection solves these issues by generating a zip file containing the dataset annotated with the structure and format required by different algorithms and also the necessary configuration files. Moreover, LabelDetection generates a Jupyter notebook [104] that configures the environment, installs the necessary libraries, trains a model with the training set, and finally evaluates the model against the testing set — the Jupyter notebooks generated by LabelDetection can be run

either locally, provided the users have a GPU; or using cloud services like Google Colaboratory [105].

LabelDetection can currently generate Jupyter notebooks for the following algorithms and libraries: YOLO [85], based on the Darknet library [74]; SSD [26], using the MxNet framework [75]; and several algorithms implemented in Keras, namely, Mask R-CNN [76], RetinaNet [77], and some of the latest detection algorithms: EfficientDet [106], FSAF [107] and FCOS [108]. Moreover, we provide the functionality to apply data augmentation for training the models by just selecting the augmentation techniques that will be applied. The interface used to provide this functionality is shown in Figure 3.2. In addition, LabelDetection has been designed to easily provide this functionality for other algorithms as we explain as follows.

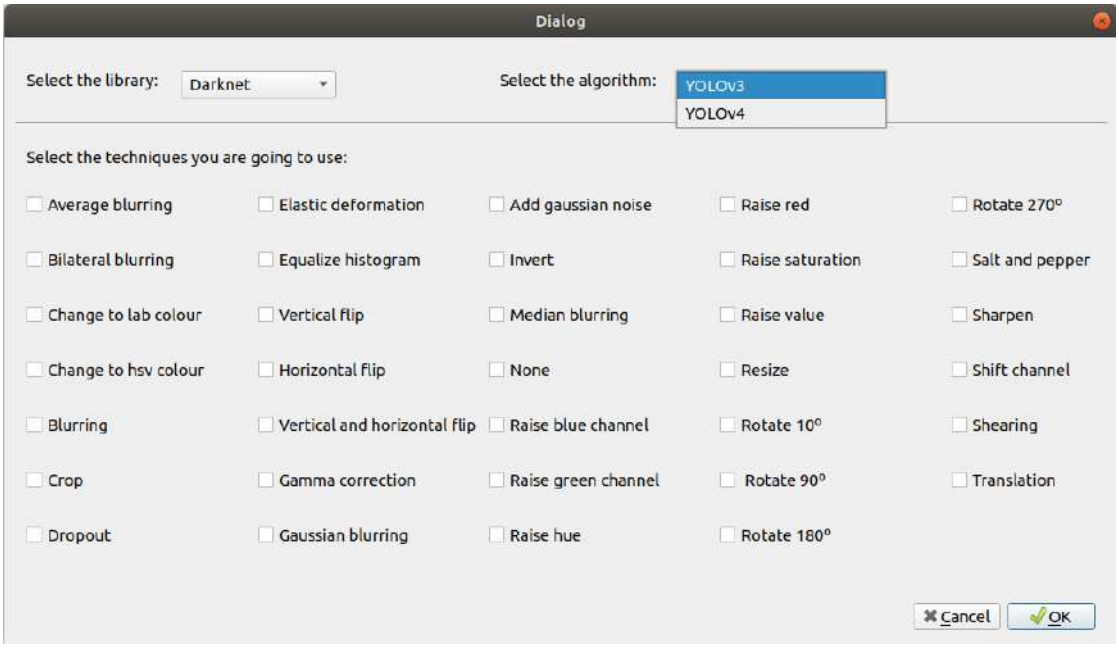


Figure 3.2: LabelDetection interface for generating the necessary files to train object detection models. The users choose which library they are going to use, the algorithm and finally the techniques they are going to apply.

We have used the *Abstract Factory Pattern* to provide the functionality to generate the necessary training files in LabelDetection. In particular, we have defined an abstract class, called `ITrainer` (see Figure 3.3), with a `generateTrainFiles` abstract method that will be in charge of generating all the files that are necessary to train a model with a given library. The `ITrainer` class has four arguments: `initialWeights`, `pathAnnotatedImages`, `tranforms` and `outputPath`.

The `pathAnnotatedImages` is the path to the image folder together with the annotations in the Pascal VOC format, and the `outputPath` is the path to the output folder. The `transforms` argument is a list of transformations to apply data augmentation. Finally, the `initialWeights` argument is used to apply transfer learning [40], a technique that reuses a model trained with a large amount of data to create another model that has less data and that will be applied in our library — if the argument is empty the model is trained from scratch but this is not usually recommended since it takes a lot of time and worse results are obtained. For each library that is included in LabelDetection, we provide a class that inherits from `ITrainer`, and generates the training files taking into account the characteristics of each library. For example to train models for the Darknet library, we provide a class called `DarknetTrainer`, that receives an additional argument, `model`, that indicates the concrete YOLO version model to be built.

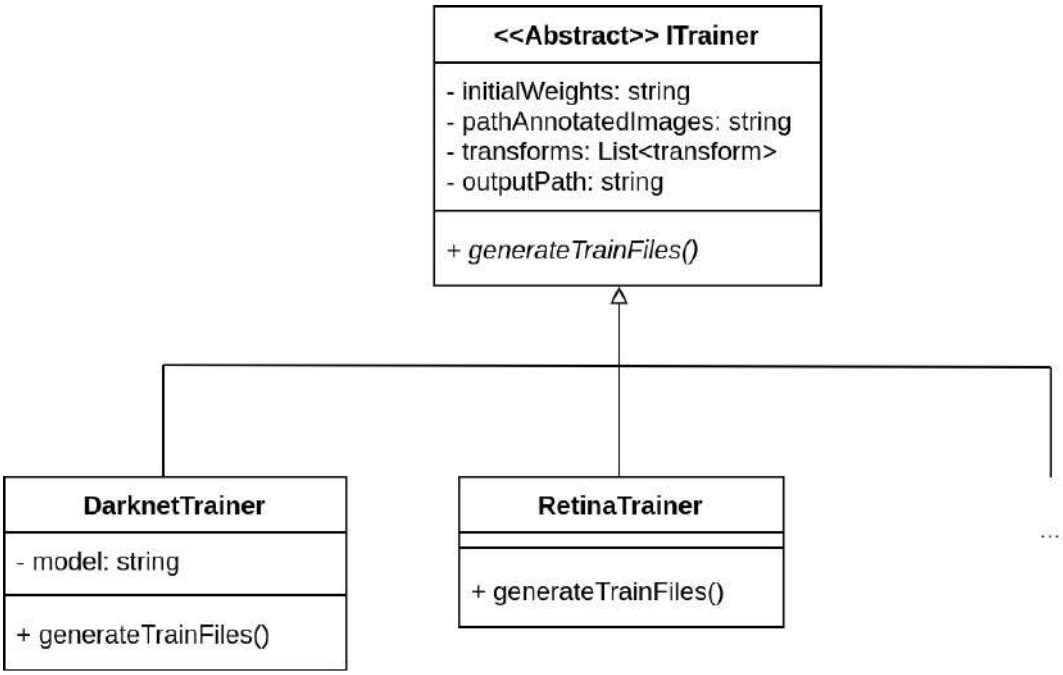


Figure 3.3: Class diagram of **ITrainer** classes.

In general, a problem that users may encounter when training object detection models is that it takes too much time to annotate the necessary images to achieve good results. This problem has been faced in LabelDetection by providing a simple to use approach to apply the semi-supervised learning methods presented in the previous chapter.

3.2.3 Semi-supervised learning in LabelDetection

LabelDetection offers the possibility of applying semi-supervised learning methods as we explain as follows. Given a partially annotated dataset and an object detection algorithm, LabelDetection will generate the necessary files to automatically annotate the unlabelled images and later train the algorithm combining the manually and automatically annotated images. This functionality is provided to the users by means of the graphical interface of Figure 3.4.

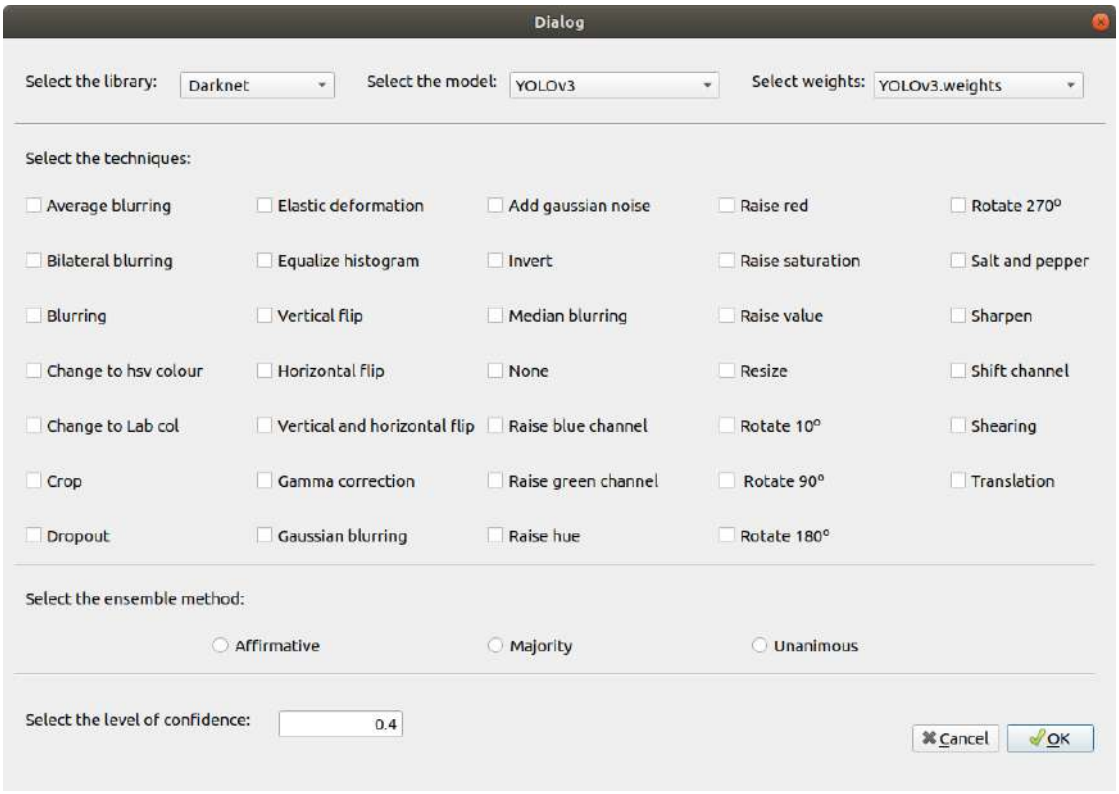


Figure 3.4: Interface for apliying Data Distillation and TTA. The users choose which library they are going to use, the model, the weights to apply data distillation and finally the techniques they are going to apply.

This functionality is provided in LabelDetection with an abstract class called `ISemiSupervisedTrainer` (see Figure 3.5). The `ISemiSupervisedTrainer` class has an abstract method called `generateSemiSupervisedFiles` that will generate the necessary files to annotate the unlabelled images stored in a path given by the attribute `pathUnlabelledImages`, and to train the object detection algorithm. The later functionality is provided by means of an `ITrainer` object, called `trainer`, that is an attribute of the `ISemiSupervisedTrainer` class. In this way,

the semi-supervised learning algorithm can be applied to all the object detection algorithms available in LabelDetection.

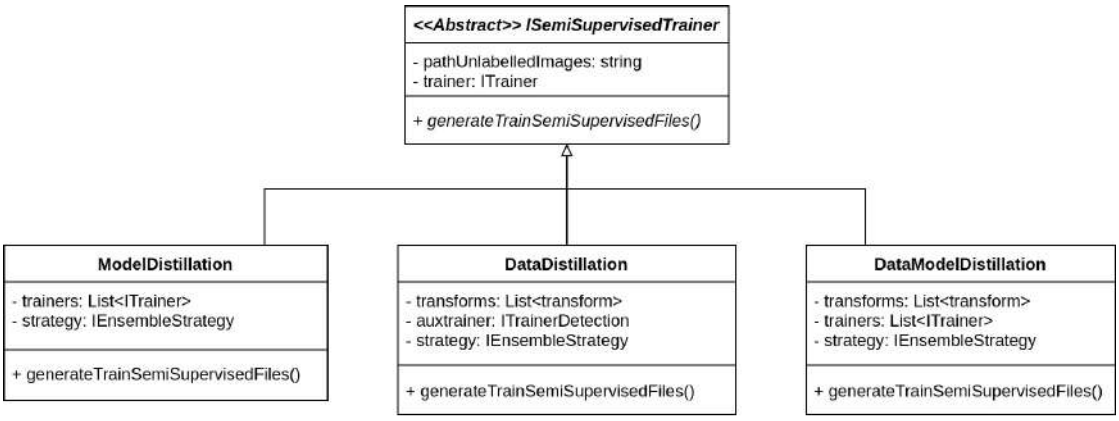


Figure 3.5: Class diagram of **ISemiSupervisedTrainer** classes.

As we have seen in the previous chapter, there are several semi-supervised methods, and we have implemented them as classes that inherit from the class `ISemiSupervisedTrainer`. For instance, the data distillation method (implemented by the `DataDistillation` class) requires an additional `ITrainer` object, called `auxtrainer`, a list of transformations and an `IEnsembleStrategy` object (defined in the `EnsembleObjectDetection library`). The method that generates the training files of this class proceeds as follows. It first uses the `generateTrainFiles` of the `auxTrainer` object to generate the training files to create a model only with a set of manually labelled images. The result produced by those files will be an `IPredictor` object that using the TTA functionality provided by the `IEnsembleStrategy` object will annotate the images from `pathUnlabelledImages`. Finally, `trainer` will be used to generate the training files to build a model combining both the labelled and unlabelled images. Similarly, we have implemented the functionality to apply model distillation and to combine model and data distillation. Note that this design allows us to implement other automatic annotation strategies that do not require ensemble methods; for instance PseudoLabeling [109]. Such a functionality could be implemented by extending the `ISemiSupervisedTrainer` class.

Once object detection models are trained, it remains the task of using them in a simple way.

3.2.4 LabelDetection for object detection

LabelDetection can be employed not only to train object detection models, but also to use them — this facilitates the dissemination of object detection models

and avoids the development of new graphical interfaces for individual models, a task that is almost an art [110].

LabelDetection can use models trained with any of the algorithms indicated in Section 3.2.2 (as in the case of model creation, this functionality can be easily extended to other algorithms), and it only requires the weights of those models (see Figure 3.6). This functionality is provided thanks to the `IPredictor` class from the `EnsembleObjectDetection` library. That is, users do not need to install any additional library or write a single line of code. Moreover, the detections obtained by the models can be visualised and modified using the LabelDetection interface; and, a summary of them can be exported to an Excel file.

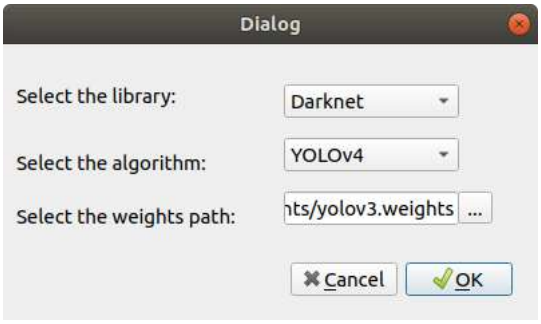


Figure 3.6: Load a model in LabelDetection to predict.

In addition, LabelDetection allows users to apply TTA to all the supported models in order to improve their accuracy. This technique can be employed by LabelDetection users thanks to the `EnsembleObjectDetection` library and using a simple to use interface to select the model and the transformations to apply, see Figure 3.4.

3.3 Application

In this section, we employ the different algorithms and methods included in LabelDetection to illustrate the advantages of using this tool. Towards this aim, we have used the Global WHEAT Dataset [111], a dataset for wheat head detection from field optical images, see Figure 3.7. All images from this dataset share a common format of 1024×1024 pixels with a resolution of 0.1-0.3mm per pixel. The dataset contains 6278 high-resolution RGB images (4578 were used for training, 422 for testing, and there were 1578 unlabelled images) and 190000 labelled wheat heads collected from several countries around the world at different growth stages with a wide range of genotypes.

Using LabelDetection, we built detection models for this dataset in the Google Colaboratory environment. Namely, we trained the models using only the labelled

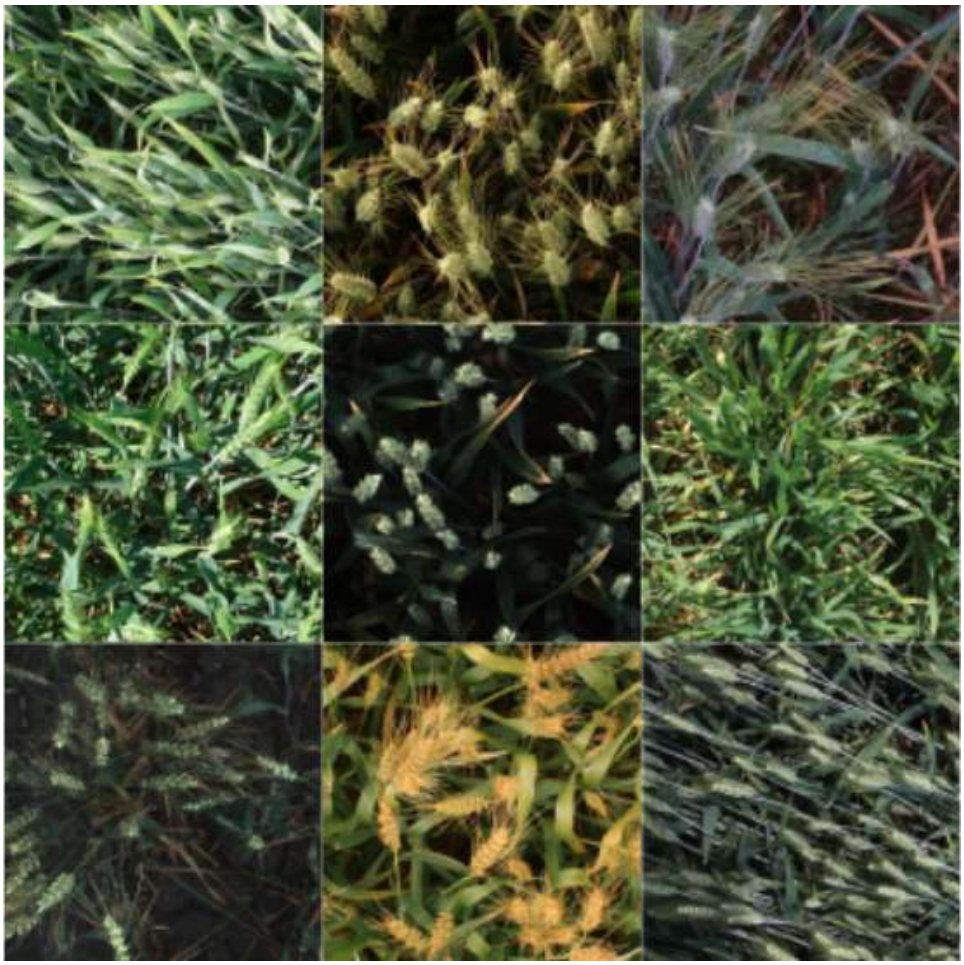


Figure 3.7: Different range of cultivars.

data with the v3, v4, and CSResnet versions of the YOLO algorithm; and the FSAF, FCOS, and EfficientDet algorithms. All the models were trained for 50 epochs and using the default parameters of each algorithm. Moreover, we studied the impact of applying TTA and data distillation in all the models.

The results, in terms of precision, recall and F1-score, of our study are presented in Table 3.3. The first block of Table 3.3 contains the results obtained from the models trained using only the labelled data, the second block contains the results obtained by applying TTA, and in the last block we include the results of the models trained using data distillation.

The model that offers the best trade-off between precision and recall was the version 4 of the YOLO algorithm, and the other 2 versions of the YOLO algorithm also achieved an F1-score over 0.80. This was also the case for the FSAF algorithm, whereas FCOS achieved a F1-score of 0.6 and EfficientDet was the worse model.

	Normal			TTA			Data distillation		
Algorithm	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Csresnet	**0.94**	0.69	0.80	**0.93**	0.73	0.82	**0.93**	0.76	0.84
EfficientDet	0.40	0.31	0.35	0.93	0.79	0.85	**0.93**	0.80	0.86
FCOS	0.67	0.54	0.60	0.89	0.80	0.84	0.92	0.85	0.89
FSAF	0.93	0.85	0.89	0.81	0.81	0.81	0.83	0.87	0.85
YOLO v3	0.95	0.79	0.86	0.91	**0.83**	0.87	0.92	0.87	0.89
YOLO v4	0.89	**0.92**	**0.91**	0.90	0.81	**0.90**	0.90	**0.91**	**0.91**

Table 3.3: Results for the Global Wheat dataset using the different networks and methods available in LabelDetection.

The performance of all the aforementioned models, but FSAF and the v4 version of YOLO, were improved thanks to the application of TTA. As transformations, we applied both vertical and horizontal flips, a rotation of 90^{o}, gamma correction, and histogram normalisation. Moreover, we considered the predictions on the images without transforming them. Using this approach, we achieved an improvement ranging from 1% (in the v3 version of YOLO) to 50% (in the EfficientDet model).

Finally, we applied data distillation by using the unlabelled images that were pseudo labelled by transforming them (using vertical and horizontal flips) and ensembling the predictions obtained by each model. All architectures but the v4 version of YOLO an FSAF improved their performance between a 3% and a 51%.

This entire process could be carried out from the functionality presented in the previous chapter, but it would require a considerable effort, and not all users might be able to do it because programming knowledge would be necessary. This is because it involves writing a line of code. This case study allows us to illustrate that LabelDetection can be employed to construct a variety of detection models and easily compare them without having experience working on deep learning or too much experience working with the libraries. Moreover, those models can be considerably improved thanks to data distillation and TTA. The advantage of the former technique is that it is faster in inference time, but it takes longer to train the models.

3.4 Generalising to other Computer Vision tasks

Up to this point, we have focused on solving different drawbacks of object detection models. However, we have noticed that the techniques presented in these two chapters of the memoir can be generalised to other Computer Vision problems such as semantic segmentation or image classification. In this section, we will explore a proof of concept approach for extending our methods to multiple Computer Vision

tasks. Our approach consists of four modules that can be particularised for different Computer Vision tasks: one for prediction, one for training, one for applying ensemble methods and one for applying semi supervised learning techniques.

The first module is focused on simplifying the usage of Computer Vision models for making predictions. In order to implement such a functionality, we have used the *Abstract Factory Pattern* to enhance the functionality of the `IPredictor` class presented in Chapter 2. In particular, we have redefined the `IPredictor` abstract class with a `predict` method that takes as input a folder of images, `imgFolder`, and outputs a folder, `outputFolder`, that contains files in the corresponding format for the problem being addressed. Moreover, the `IPredictor` class needs some configuration files to load a model that are independent of the Computer Vision task: the path to the weights of the model and a threshold for the confidence level, see Figure 3.8.

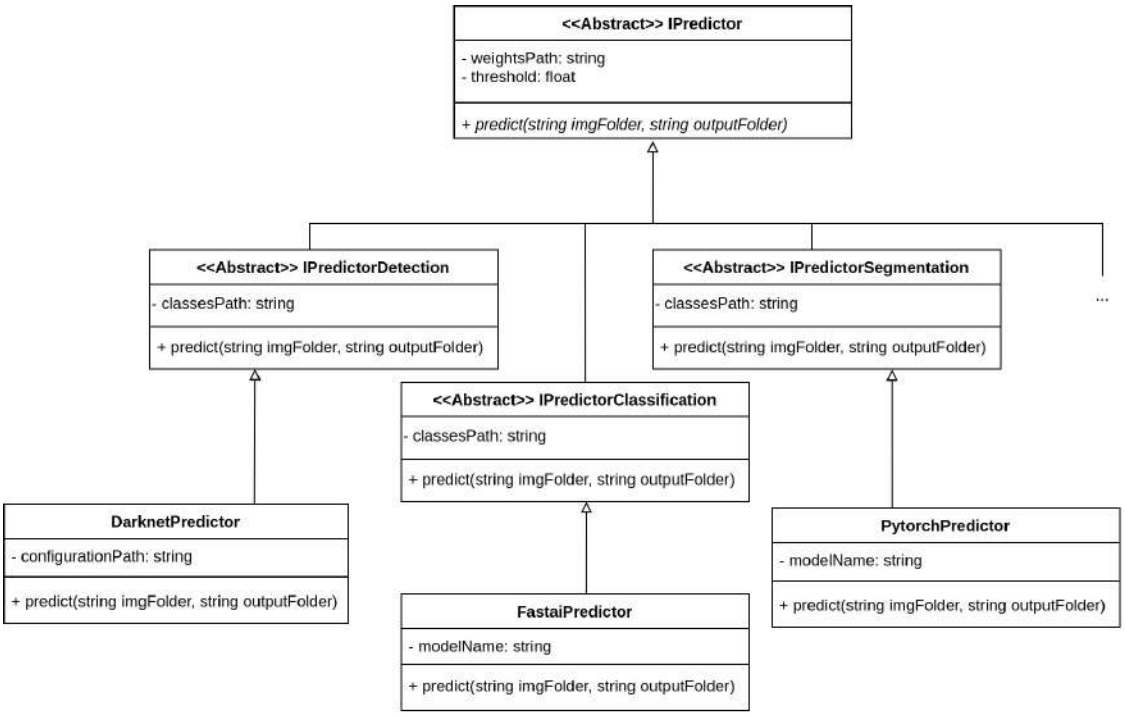


Figure 3.8: Class diagram of **IPredictor** classes.

From a conceptual point of view, we could use the notion of generics [112] to parametrise the Computer Vision task in `IPredictor` using `IPredictor<T>` where `T` would be the Computer Vision problem. However, this cannot be directly implemented in Python, so we have used the architecture of classes presented in Figure 3.8, where we have defined an abstract class that extends `IPredictor` for each Computer Vision task. In particular, we have defined several abstract

classes that extends the class `IPredictor` for particular problems; for instance `IPredictorSegmentation` or `IPredictorClassification` — other classes can be easily included. From those classes, we can implement particular classes for concrete algorithms and libraries. For instance, in the semantic segmentation case, we have implemented a class for the Pytorch library, by providing a `PytorchPredictor` class, where the user has to provide the necessary configuration files to create a new model and also a `modelName` to identify the concrete architecture that is loaded. Similarly, in the case of image classification, we provide a class called `FastAIPredictor` that allows users to make predictions with any FastAI model by providing the name of the model and the other parameters of the abstract class called `IPredictorClassification`. To sum up, this module provides a common interface to use Computer Vision models trained with different algorithms and libraries; and, hence, facilitates their use.

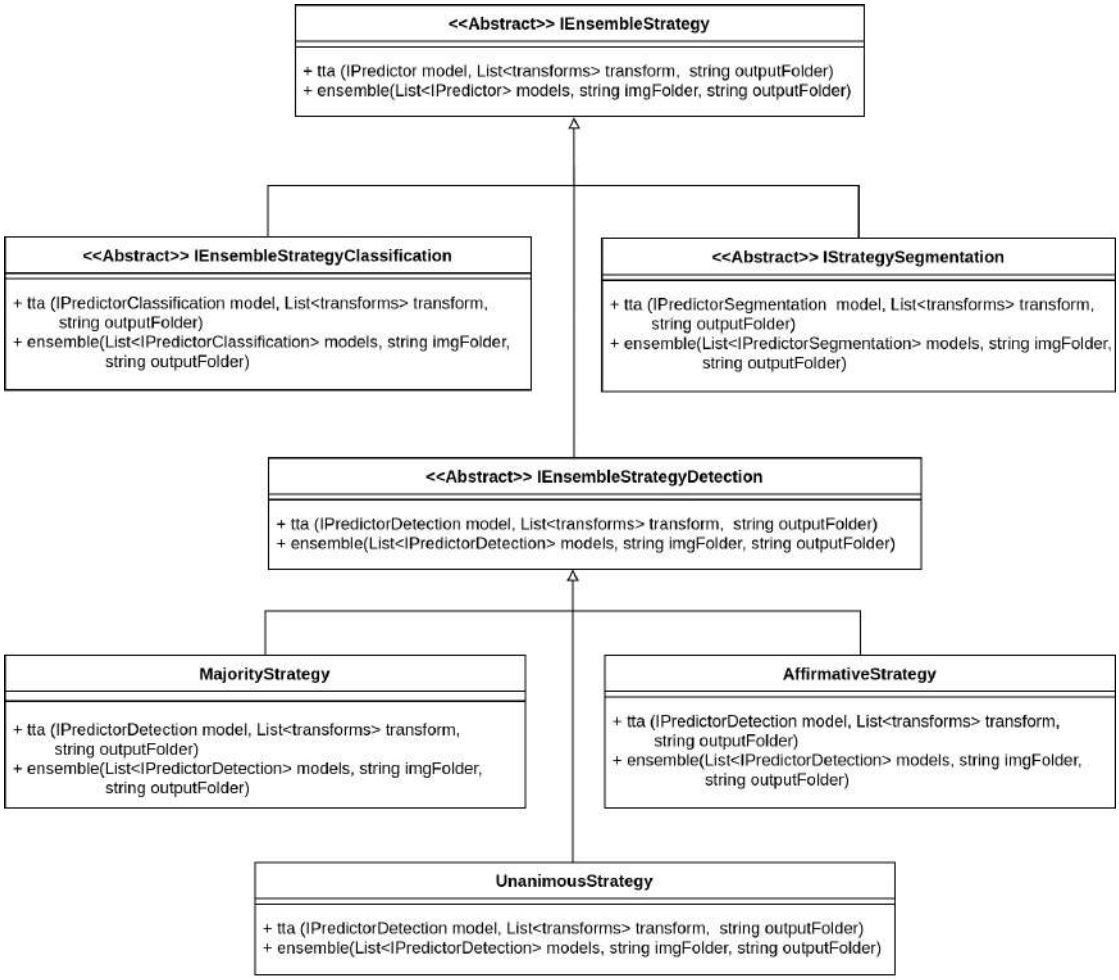


Figure 3.9: Class diagram of **IEnsembleStrategy** classes.

The second module helps users to apply ensemble methods to improve the performance of Computer Vision models. As in the previous module, we have used the *Abstract Factory Pattern* to define an `IEnsembleStrategy` abstract class. This class has two abstract methods named `tta` and `ensemble` with the same signature presented in Chapter 2, see Figure 3.9. These methods are not implemented in the `IEnsembleStrategy` class, but this functionality is provided by the classes that inherit from it and depend on the concrete Computer Vision task. Namely, for each Computer Vision task, we have defined an `IEnsembleStrategyTask` abstract class where the `ensemble` method receives as input a list of `IPredictorTask` models, the path to the image folder and the path to the output folder; and similarly for the `tta` method — a similar signature was already presented in Chapter 2 for the object detection case.

It is worth noting that the `IEnsembleStrategyTask` classes are abstract since several strategies for ensembling predictions can be applied in the Computer Vision tasks. Therefore, those strategies are implemented by classes that inherit from the corresponding `IEnsembleStrategyTask` and provide the actual functionality for the `tta` and `ensemble` methods. Thanks to this approach is straightforward to apply ensemble methods to Computer Vision models coming from different libraries.

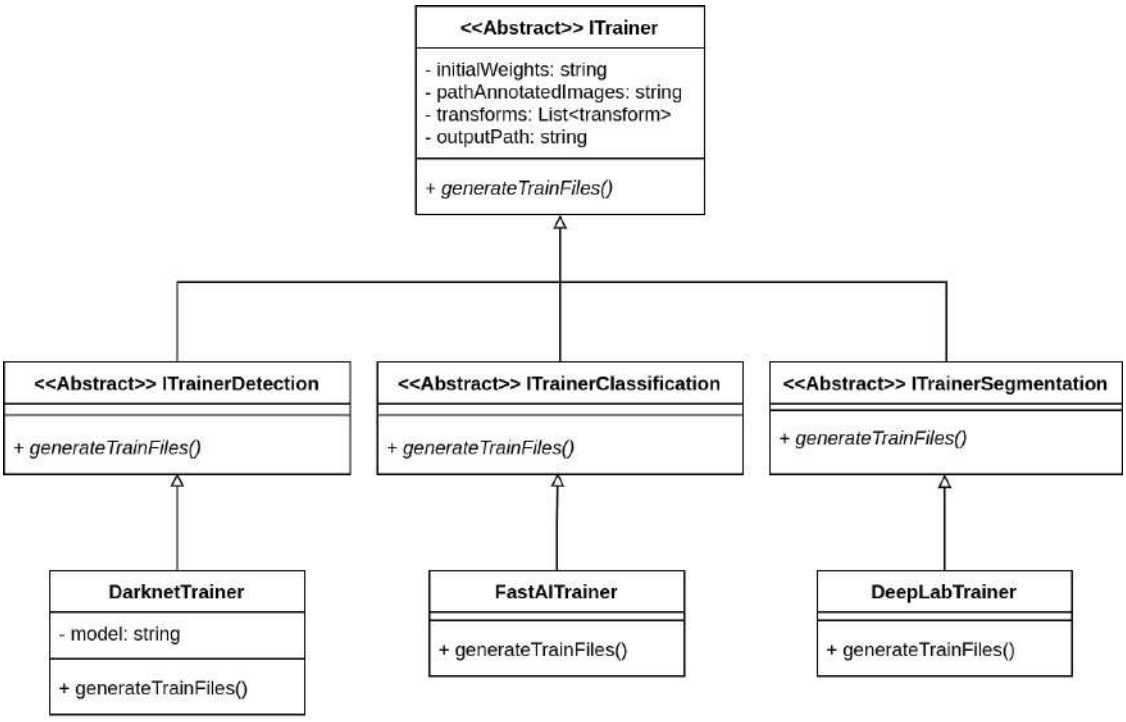


Figure 3.10: Class diagram of **ITrainer** classes.

The third proposed module is focused on simplifying the process to train Computer Vision models. This is achieved with an abstract `ITrainer` class with the same arguments and methods presented for the `ITrainer` class of LabelDetection, see Figure 3.10. Given a path of images, some initial weights to apply transfer learning, a list of transformations to apply data augmentation, and an output path; the `generateTrainFiles` method of this class will be in charge of generating all the necessary files to train a Computer Vision model. The particularities of each Computer Vision task are implemented by means of abstract `ITrainerTask` classes, and the actual implementation is given by the classes that extend those `ITrainerTask` classes.

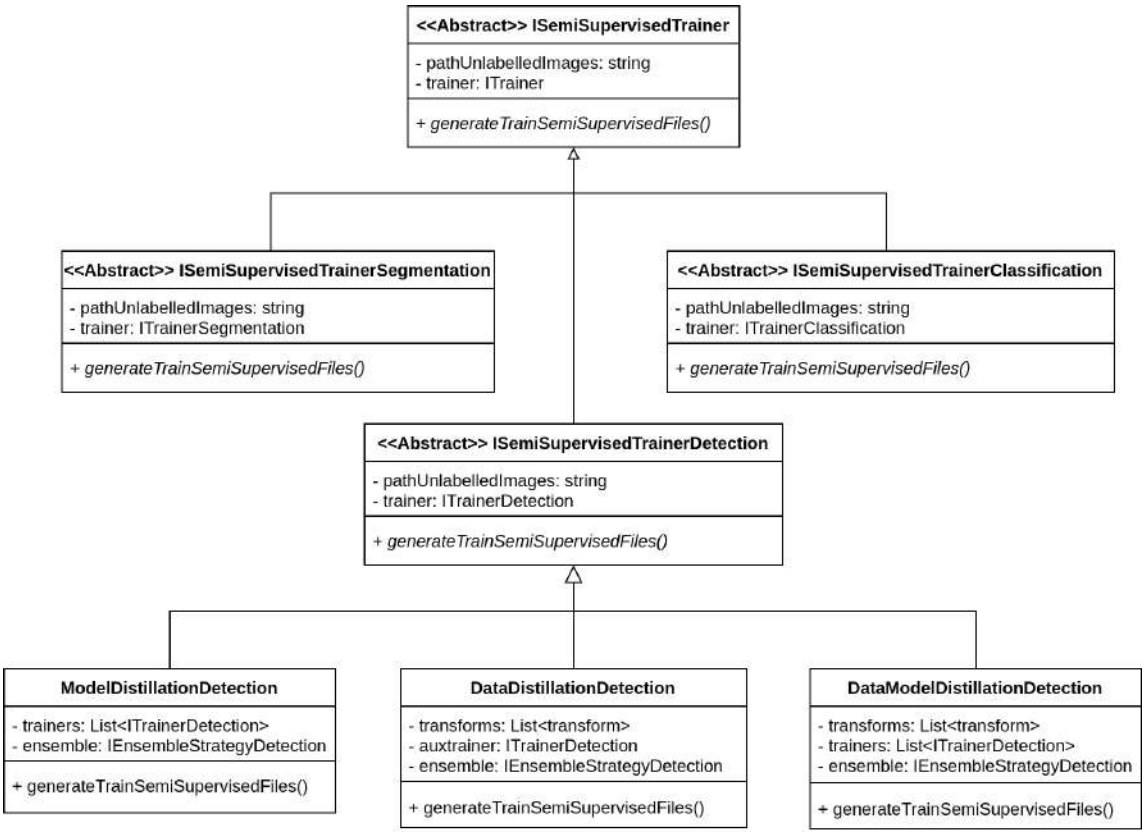


Figure 3.11: Class diagram of **ISemiSupervisedTrainer** classes.

Finally, the last module serves to reduce the burden of annotating a great amount of images to train Computer Vision models by means of semi-supervised learning methods. To implement such a functionality, we have defined an abstract class called `ISemiSupervisedTrainer`, see Figure 3.11. This class provides a method `generateTrainSemiSupervisedFiles` that will generate the necessary files to train a Computer Vision model with a semi-supervised technique pro-

vided a path of unlabelled images and a `ITrainer` object — this is the same approach presented previously for LabelDetection. It is worth noting that the `ISemiSupervisedTrainerTask` classes are abstract since several techniques for semi-supervised learning can be applied in Computer Vision tasks as we showed previously. Moreover, for each of these classes, the `ITrainer` object comes from the corresponding `ITrainerTask` class.

Similarly to the work presented for LabelDetection the four models presented in this section can be used independently of each other for using Computer Vision models and training them. The approach presented in this section has been implemented in a library available at `https://github.com/ancasag/GeneralisingCV`. This library is a work in progress and several tasks remain as further work. In particular, we want to populate our library with multiple Computer Vision tasks and integrate such a library with graphical interfaces like we did with LabelDetection to create end-to-end applications for multiple Computer Vision tasks. This is a step towards the democratisation of deep learning models.

3.5 Conclusions

In this section, we have presented LabelDetection, an end-to-end graphical application that aims to facilitate the construction and usage of robust object detection models by providing access to state-of-the-art detection algorithms, and also simplifying the application of advance techniques like semi-supervised learning methods or test-time augmentation. In the future, we plan to extend LabelDetection with new object detection models and libraries, a straightforward task thanks to the design of this tool.

After analysing the functionality of LabelDetection, we realised that it could be interesting to extend it to also work with other Computer Vision problems, and also to particularise it to concrete object detection tasks. We showed how Label Detection can be generalised to deal with any Computer Vision task thanks to the development of a library that makes easier to create and use Computer Vision models, and the application of ensemble methods and semi-supervised learning techniques. In addition, our approach is independent of the underlying library and algorithm that we use to train the models, and it can be easily extended.

In the rest of the memoir, we change our aim and focus on concrete problems in plant physiology and precision agriculture that can be solved thanks to our methods and tools.

Chapter 4

Applications to the Study of Plant Physiology

In the previous chapters, we have explained how Deep Learning models for object detection can be improved, and how we have built an end-to-end application to train and use those models. In this chapter, we focus on the use of those methods to construct tools that deal with actual problems in the study of plant physiology. In particular, we address the problems of detecting stomata and bladder cells. The results presented in this chapter come from a collaboration with the Department of Plant Biology and Ecology from the University of the Basque Country, and with the Department of Crop, Soil, and Environmental Science from the Auburn University.

4.1 Measuring Stomatal Density

Stomata (singular "stoma") are pores on a plant leaf that allow the exchange of gases, mainly CO_2 and water vapor, between the atmosphere and the plant. Stomata respond to changes in the environment and regulate the photosynthesis and the transpiration of plants, and thus their productivity and water use efficiency. Because of their critical nature, scientists have studied the number, density, size, and behaviour of stomata to understand plant physiology against stress [113, 114], to improve crop breeding and crop management programs [113, 115], to model CO_2 dynamics in the atmosphere, and to predict future carbon and water cycles [114].

In order to analyse stomata, plant biologists take microscopic images of leaves, and manually measure characteristics such as stomata density, individual stomata opening, and morphological traits like the size and shape of the stomata guard cells (a pair of cells that regulate the opening and closing of the stomatal pore). Some computer programs, like ImageJ [116], are used for those tasks, but they have

little to no automatisation [117, 118, 119]. Note that those measurements are repeated over hundreds of images from different plant species and growth conditions; hence, this is a tedious, error-prone, time-consuming and, in some cases, subjective task due to the large number of stomata in each image. Hence, researchers have tried to automatise this task since the 1980s [120]. Initially, the proposed detection methods were based on the application of image processing techniques and were focused on specific plant species. Omasa and Onoe [120] applied the Fourier transform and a thesholding mechanism to measure stomatal aperture from *Sunflower* leaves. In [121], stomata from *Olea europaea* leaves were detected using several filters and the background subtraction technique from plant leaves with fluorescence. Image processing techniques, namely preprocessing techniques and the watershed algorithm, were applied in [122] to segment stomata from *tomato* leaves. Several mathematical morphology techniques were employed in [123] to automatically detect stomata from 5 different species; and, the MSER algorithm was applied to measure stomatal aperture in *grapevines* [124]. Finally, the most recent work based on image processing techniques for stomata detection employed Wavelet spot detection and the Watershed transform on images from *Ugni Molinae* species [125].

In spite of the success of some image processing approaches — for instance, Duarte et al. [125] reported a precision over 98% — those methods require users to fix some parameters and do not usually generalise to different species and conditions. This has led to the adoption of more advanced computer vision and machine learning techniques. Namely, a genetic algorithm and self-organizing map clustering method was proposed in [126] to detect stomata from *Arabidopsis* leaf surface; and, the template matching algorithm was employed in [127] to detect stomata in *wheat*. In addition, up to 2013, object detection using machine learning was conducted by performing object classification using different feature extractors via a sliding window on multiscale images [128]. This approach has been applied for stomata detection with the Haar feature extractor for *Quercus afares* Pomel and *Quercus suber* L. leaves [129], and the HOG feature extractor for grapevines [130] and dayflowers [131].

Nowadays, Deep Learning techniques are the state of the art approach to deal with stomata detection. The first works using deep learning employed convolutional models as feature extractors and combined them with the sliding window technique. For instance, in [132], 5 classical feature extractors were compared with six deep learning descriptors for detecting stomata in *maize* leaves; and, in [133] a deep neural network was employed for stomata detection in 38 species. Moreover, in [134], deep convolutional networks were applied to generate heatmaps that indicate the position of stomata in a wide variety of species. Finally, general deep object detection algorithms like SSD [26] or Faster R-CNN [25] have also been

trained to construct accurate detection models for species like rice [135], Chinese Necklace poplar and black poplar [136], or soybean [137].

There are two main drawbacks in the methods proposed in the literature for stomata detection. First of all, most of those methods can only be applied to particular species and do not generalise properly since images from plant leaves greatly vary from species to species, and also depend on the conditions and equipment employed to capture them. And secondly, the necessary code to run these methods on the users' images is not generally available. Currently, there are only three publicly available tools for detecting stomata: DeepStoma [131], StomataCounter [134] and Stomata Detector [138]. Unfortunately, those tools do not provide the necessary features to interact with the predictions, and the underlying models cannot be easily retrained to adapt them for new species — StomataCounter provides an on-demand mechanism for re-training the underlying algorithm, but this requires sending the images to the developers of StomataCounter and waiting until they produce a new model. These drawbacks are tackled in the current work with the development of a stomata detection model, and an a simple-to-use application based on LabelDetection, called LabelStoma[1], that employs such a model. The main features of LabelStoma are described as follows.

- LabelStoma is a open-source graphical tool that helps user in detecting stomata in leaves of plants of several species.
- LabelStoma can be employed to measure stomatal density.
- LabelStoma allows non-expert users to use and train models for stomata detection. Moreover, the predictions of LabelStoma can be easily modified.
- LabelStoma provides a simple method to adapt the model to other species of plants using a small number of annotated images.

In the rest of the section, we explain how LabelStoma has been designed and implemented. We first explain the methods used to capture images for stomata detection, and to train the detection models. Subsequently, we show the results obtained by our models under different conditions. Finally, we explain how we have developed LabelStoma.

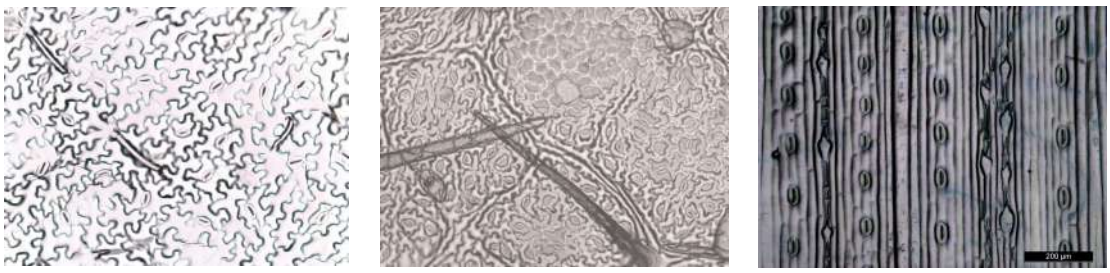
4.1.1 Materials and methods

In this section, we present both the biological and computational materials and methods employed in our work for stomata detection.

[1] `https://github.com/ancasag/labelStoma`

4.1.1.1 Biological material

In our experiments, we have employed images from three species that were acquired by three laboratories. These laboratories work with several plant species and varieties, both dicotyledons (common bean, *Phaseolus vulgaris*; and soybean, *Glycine Max*) and monocotyledons (barley, *Hordeum vulgare*). Besides, the plants were grown under different environmental conditions such as drought, and elevated CO_2 which could enhance even more the variability in the size, density and distribution of stomata [139, 140]. Images were captured using different equipment (microscopes and digital cameras) and conditions (focus and magnification) forming three datasets. Examples of the images from the three datasets are provided in Figure 4.1.



(a) (b) (c)

Figure 4.1: Examples of images from 3 species: (a) Common bean. (b) Barley. (c) SoyBean.

Common bean dataset. Leaf imprints were taken from recently cut leaves of twelve different varieties of common beans (*Phaselous vulgaris L.*): Arrocina de Álava, Amarilla de Kuartango, Borlotto, Coco Blanco, Canela de León, Lingot, Morada de Gandarias, Negrita, Negra de Basaburua, Pinta Alavesa, Riñón de León and Verde de Orbiso; grown in a greenhouse under well-watered and drought conditions (100% and 40% field capacity respectively) during the summer of 2017. Samples were collected in the morning, when the leaves are more turgid and the possibility of tearing them was reduced.

Imprints of epidermal cells were taken according to the technique described by [141], from the youngest full expanded leaf from the adaxial (upper) and abaxial (lower) side of the blade, because common bean is an amphiestomatic species. Briefly, a drop of cyanoacrylate adhesive (Superglue-3, Loctite Corporation, Henkel Ibérika, S.A., Barcelona) was spread on the surface of the leaf epidermis. To remove excess of adhesive, a strip of acetate paper was attached and gently pressed down during three seconds. Once the acetate paper has been carefully removed, the leaf surface with adhesive is gently pressed against a microscope slide for 10 seconds.

At that point, the slide is carefully removed from the leaf to avoid that any leaf traces remain attached to the adhesive.

Once in the laboratory, five digital photographs of 1280×960 px were taken of each imprint, using a digital camera (Digital Sight DS-Fi1, Nikon) attached to a light microscope (Eclipse 80i, Nikon). The pictures were taken without coverslip, with the $20\times$ objective, without filters and medium light intensity. A total of 1050 images were captured employing this procedure.

Barley dataset. Images from barley stomatal prints of barley plants grown under different environmental conditions (drought, elevated CO_2 and/or elevated temperature applied alone or in combination) were used. Barley plants (*Hordeum vulgare* cv. Henley) were grown in a Conviron PGR15 controlled-environment growth chamber (Conviron, Manitoba, Canada). The light regimen was 14h of light and 10h of darkness. During the light period a photosynthetic photon flux density (PPFD) of 400 μmol m^{-2} s^{-1} was applied to plants. Light was provided by a combination of incandescent bulbs and warm-white fluorescent lamps. For each environmental condition, the youngest fully expanded leaf on the main tiller per plant was used for leaf imprinting. To measure the stomatal traits in steady-state conditions, imprints were 3h after lights turned on.

Leaf imprints were obtained from both adaxial and abaxial leaf surfaces from the center of the leaf along its length and from the entire leaf width. Immediately after application of cyanoacrylate glue (Loctite Superglue-3, Loctite Corporation, Henkel Ibérica, S.A., Barcelona.) on the leaf, it was gently pushed onto a glass microscope slide for a few seconds. After that, the leaf was removed obtaining the final impression on the glass slide. The prints were stored in darkness to prevent degradation. Epidermal prints were observed and images acquired using a Nikon ECLIPSE 50i fluorescence microscope (Nikon corporation, Japan) with a Leica DFC 420C camera (Leica Microsystems, Germany).

Four images were taken at $4\times$ and $10\times$ magnifications from each epidermal print, using the LAS V3.7 program (Leica Microsystems, Germany). For that, each epidermal print was divided into four squares. For each square, one image was taken at $4\times$ and one at $10\times$ magnification. The objective of taking a photo for each square was to analyse the variability that could be in the sample itself. After that, to delimit the area in the image for assessment of stomatal traits, the epidermal idioblast cells (siliceous and suberose cells) were identified on the print. The images and analyses that were taken at the $10\times$ magnification comprised the stomata between two rows of epidermal idioblast cells. A total of 1050 images were captured employing this procedure.

Soybean dataset. Two soybean (*Glycine max.* L.) cultivars (PI 398223 and PI 567201) were grown under field conditions at the Bradford Research Center (University of Missouri, Columbia, Missouri, USA) during the 2017 growing season. Plants were grown on a field modified to restrict rooting to depths of 0.3 0.6 and 0.9 m [142, 143] with 4 replications per depth and genotype. The site was modified in 1978 by excavating a series of 61-m long and 7.5-m wide channels that were lines with plastic to limit rooting depth, and drain tiles to eliminate chances for water logged conditions. Following installation of the plastic liner and the drain tiles, the channels were filled with top soil. The range in top soil depths creates environments with different soil water holding capacity and thus differences in water stress producing differences in soybean yield [143, 144].

Leaf samples for epidermal prints were taken at four different growth stages (V5, R2, R4, R6) [145]. Leaf imprinting was carried out for both the adaxial and the abaxial side of the leaf. The leaf surfaces were prepared by applying cyanoacrylate glue (Gorilla Super-glue, USA) near the longitudinal center of the uppermost, fully expanded leaf on one side of the midrib using a brush to homogeneously distribute the glue. To remove excess of adhesive, a strip of acetate paper was attached and gently pressed down for one second. Immediately after careful removal of the acetate paper, the leaf was gently pressed against a glass slide for 5 seconds. After that, the leaf was removed to leave a final impression on the glass slide.

The prints were stored in darkness to prevent their degradation. The prints were observed using a Leica DM 5500B Compound Microscope outfitted with a Leica DFC290 Color Digital Camera (Leica Microsystems, Germany). Each epidermal print was examined at 20× magnification and four composed images, consisting of at least 20 images captured using the z-stacking capability of the microscope, were generated. A total of 1050 images were captured employing this procedure.

A summary of the features of the three datasets employed in this work is provided in Table 4.1.

Dataset	♯ Images	Image size	Mean(std) stomata per image	Magnification
Common bean	1050	1280 × 960	31.92(25.12)	20×
Barley	1050	1728 × 1296	26.03(24.30)	4× and 10×
Soybean	1050	1600 × 1200	55.12(25.29)	20×

Table 4.1: Features of the three datasets employed in this work.

4.1.1.2 Computational methods

In this work, we have employed the YOLOv3 algorithm [85], and its implementation in the Darknet library [146], to create several stomata detection models from the aforementioned datasets. YOLO is one of the multiple algorithms based on Deep Learning [147], and has been employed in several contexts such as agriculture [148], medicine [149] or security [150]. The YOLO algorithm frames object detection as a regression problem where a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation.

Using the datasets presented in the previous section we have trained several YOLO models. Namely, the datasets were annotated using LabelDetection and split using 85% for training and 15% for testing (imprints of the same plant only appeared either on the training set or in the test set). If not stated otherwise, all the models built in this section were trained using the default parameters in YOLO (namely, we set the base learning rate to 0.001, the momentum to 0.9, the weight decay to 0.0005, the batch size to 64, the subdivisions to 16, and we trained the models for 125000 steps), and using a GPU GeForce RTX 2080 Ti. During the training process, we faced three main challenges that commonly arise when working with microscopic images, namely the size of the training images, the number of training images, and the time required for the manual annotation of the training images.

The first challenge occurs due to the fact that YOLOv3 is constrained to work with images of limited resolution; namely, images of size 416×416 or 600×600. However, standard microphotographs are taken on a higher resolution, and resizing them was not a feasible option since the details of the stomata are lost when the images are scaled. In order to deal with this problem, the images of the dataset were sliced in images of size 416×416 (since this size allows us to make more experiments in a reasonable time), and used for training. It is worth mentioning that the size problem only occurs during the training stage, since YOLO models can be employed to detect objects in images of bigger size once the model is trained — a YOLO model trained with patches of images can be employed for inference on full size images by just changing the input size parameter.

The second challenge is the considerable amount of images required to train Deep Learning models. This problem was faced using *data augmentation* [62, 63], a technique already explained in Chapter 2. In our experiments, we have tried different augmentation schemes (namely, colour and geometric transformations) using the CLODSA library [72]. Moreover, we have also applied transfer learning in the experiments starting from a model pre-trained in the Pascal VOC dataset.

The final challenge is the time needed to manually annotate images for object detection. This is a common problem when working with detection problems, as we have seen in Chapter 2 and thus has lead to the use of semi-supervised

learning techniques. In our study, we employed data distillation [67] using the `EnsembleObjectDetection` library presented in Chapter 2 and using as a basis the files generated by LabelDetection.

In the next sections, we first address the construction and evaluation of individual models for each of the three datasets presented in Section 4.1.1.1. Subsequently, we combine the three datasets and construct several models from such a combined dataset. Finally, we show that our model generalises properly to stomata images of plant species, by evaluating the performance of our model using the images from the dataset presented in [134] and the grapevines dataset from [130].

4.1.2 Individual models

First of all, for each dataset presented in the previous section, we trained and evaluated four models using different data augmentation regimes. Namely, using the training dataset without augmentations; increasing the training dataset with geometric transformations (by applying horizontal flips, vertical flips, horizontal and vertical flips, a rotation of 90^{o} to each image of the original training dataset, and keeping the original image, we obtained a training dataset of 4500 images); increasing the training dataset with colour transformations (by applying gamma and histogram normalisation, and Gaussian and average blurring to each image of the training dataset, and keeping the original image, we obtained a training dataset of 4500 images); and, increasing the training dataset by combining both geometric and colour transformations (this dataset contains 8100 images). The constructed models were evaluated using the corresponding test set (without augmentations) and using different metrics such as precision, recall, F1-score, and IoU (Intersection over Union); and using two threshold values for positive predictions (0.25 and 0.5) — experiments with other thresholds were conducted but worse results were obtained.

The results of this study are presented in Table 4.2. All models achieved values greater than 90% for precision, recall and F1-score, but none of them excelled over the others. The metrics employed in this study measure different aspects of the model. A high precision indicates that the detections produced by the model were correct, even if this means that some stomata were not detected. In our experiments, to achieve the highest precision, a confidence level of 0.5 had to be used for all datasets. The best augmentation procedure varied from dataset to dataset. Namely, geometric augmentations produced better models for the soybean and common bean dataset, whereas colour transformations were more beneficial for the barley dataset. In order to obtain a higher recall (that is, detect as many stomata as possible, even if some of the detections are incorrect), a confidence level of 0.25 had to be used for all datasets. In this case, none of the data augmentation procedures benefit either the soybean or the barley dataset; in

Dataset	Augmentation	threshold = 0.25							threshold = 0.50						
	(Train size/Test size)	Prec.	Rec.	F1	TP	FP	FN	IoU	Prec	Rec.	F1	TP	FP	FN	IoU
Soybean	None (900/150)	0.90	**0.95**	0.92	7695	897	424	0.69	0.93	0.91	0.92	7352	592	767	0.72
	Geometric (4500/150)	0.91	0.94	0.92	7616	756	503	0.71	0.95	0.79	0.86	6417	316	1702	0.76
	Colour (4500/150)	0.92	0.89	0.90	7258	671	861	0.70	0.93	0.87	0.90	7059	527	1060	0.72
	Both (8100/150)	0.90	**0.95**	**0.93**	7720	835	399	0.71	**0.95**	0.83	0.89	6765	371	1354	**0.75**
Common Bean	None (900/150)	0.96	0.96	**0.96**	4721	186	180	0.80	0.97	0.93	0.95	4554	130	347	0.81
	Geometric (4500/150)	0.94	0.97	**0.96**	4616	269	156	0.77	0.97	0.93	0.95	4449	142	323	0.79
	Colour (4500/150)	0.97	0.93	0.95	4572	128	329	0.81	**0.98**	0.80	0.88	3924	73	977	**0.82**
	Both (8100/150)	0.93	**0.98**	**0.96**	4685	352	87	0.77	0.95	0.95	0.95	4555	219	217	0.79
Bearly	None (900/150)	0.90	**0.92**	**0.91**	4096	470	356	0.66	0.94	0.86	0.90	3834	240	618	0.70
	Geometric (4500/150)	0.89	0.91	0.90	4063	482	389	0.65	**0.96**	0.76	0.85	3396	153	1056	0.71
	Colour (4500/150)	0.89	0.91	0.90	4057	490	395	0.65	0.94	0.84	0.89	3746	248	706	0.69
	Both (8100/150)	0.88	**0.92**	0.90	4105	544	347	0.67	0.95	0.75	0.84	3347	176	1105	**0.73**

Table 4.2: Results of the different models for the three datasets. Prec., Rec., F1, TP, FP, FN, IoU stand respectively for Precision, Recall, F1-score, True Positive, False Positive, False Negative and Intersection over Union. In bold face the best result for each metric and dataset.

contrast, the highest recall in the common beans dataset is obtained when both colour and geometric transformations were applied. A trade-off between precision and recall metrics is provided by measuring the F1-score. As in the case of the recall metric, a confidence level of 0.25 had to be used to obtain the best F1-score for all datasets (note that this value decays when increasing the confidence level since the precision increases but the recall decreases), and, only the soybean dataset improved with one of the augmentation procedures (namely, when both geometric and colour transformations are applied). Precision, recall, and F1-score are metrics that assess how well a model is able to detect stomata, but they do not indicate how the detections are adjusted to the stomata, but this can be assessed with the IoU. For this metric, the best results were obtained using 0.5 as confidence level and using different augmentation schemes; namely, the colour scheme for the common beans dataset and the combination of colour and geometric transformations for the other two datasets.

These results show that it is possible to achieve accurate results with the YOLO model for several datasets of stomata images. However, the main drawback of these models is that they have been trained to work with specific species, and, therefore do not generalise to other species. We explain how we have faced this problem in the following section.

4.1.3 A combined model

The models described in the previous section were specialised for stomatal images of particular species, and were not designed to be applied to other species. Thus, we combined the three datasets to generate one training dataset and one test dataset

Augmentation	thresh = 0.25							thresh = 0.50						
(Train size/Test size)	Prec.	Rec.	F1	TP	FP	FN	iOu	Prec	Rec.	F1	TP	FP	FN	iOu
None (2700/450)	0.91	**0.93**	0.92	16482	1655	1309	0.70	0.95	0.85	0.90	15201	806	2590	0.73
Geometric (13500/450)	0.92	0.92	0.92	16340	1515	1451	0.70	**0.96**	0.81	0.88	14468	654	3323	0.74
Colour (13500/450)	0.91	**0.93**	0.92	16484	1647	1307	0.70	0.95	0.82	0.88	14658	717	3133	0.74
Both (24300/450)	0.92	**0.93**	**0.93**	16590	1447	1201	0.72	**0.96**	0.76	0.85	13569	532	4222	**0.76**

Table 4.3: Results for the model trained with the combined dataset. Best results are in bold face.

from the images of all three species, and used the combined training dataset to train four YOLO models using the augmentation procedures described in the previous section — as in the case of individual models, we build these new models.

The performance of these models is reported in Table 4.3. Similar to the results achieved for the models based on single datasets (Section 4.1.2), all models achieved values higher than 90% for precision, recall and F1-score. In addition, although differences between the models were limited, the results for the model constructed using the dataset augmented with colour and geometric transformations achieved slightly better results than those of the other models. In addition, as show in Table 4.4 this model performs properly well when evaluated for each individual dataset.

Precision

Dataset	Soybean	Common bean	Barley	Combined
Soybean	**0.91**	0.88	**0.94**	0.90
Common bean	0.82	0.93	0.34	0.80
Barley	0.69	0.64	0.90	0.84
Combined	**0.91**	**0.95**	0.92	**0.92**

Recall

Dataset	Soybean	Common bean	Barley	Combined
Soybean	**0.95**	0.84	0.22	0.74
Common bean	0.74	**0.98**	0.13	0.65
Barley	0.03	0.08	0.92	0.27
Combined	0.92	0.96	**0.93**	**0.93**

F1-score

Dataset	Soybean	Common bean	Barley	Combined
Soybean	**0.93**	0.86	0.36	0.81
Common bean	0.78	**0.96**	0.19	0.72
Barley	0.05	0.14	0.91	0.41
Combined	0.92	**0.96**	**0.93**	**0.93**

Table 4.4: Results for the best models trained with each dataset evaluated on the other datasets. The brighter the red color in the heatmap, the higher the associated metric.

The three tables included in Table 4.4 indicate how well the best model trained for each dataset generalises to the other datasets. As expected, the model trained using the combined dataset obtains good results for the three metrics in all datasets. Whereas, in general, the other models only obtain good results when evaluated against their own datasets (the exception is the precision achieved by the soybean model for the Barley dataset, but its recall is really low). This occurs due to, not only the difference of species, but also the different magnification level employed to acquire the images. Indeed, the combined model produced results that were similar to those results of individual models when evaluated against individual datasets. Namely, the combined model achieved equal or higher precision than the individual models, but equal or slightly worse recall. The achieved F1-score was similar for the specific models and the combined model for each dataset.

Figure 4.2: Results of the combined model for different degraded versions of the test set by applying blurring, adding salt-and-pepper noise, and applying gamma correction to obtain darker and brighter images. The kernels values for blurring the images are given by $2 \times i + 1$ for i from 1 to 9; the probability for adding the salt-and-pepper noise is given by $i/100$ for i from 1 to 9; the gamma values to obtain darker images are given by $1 - i/10$ for i from 1 to 9; and the gamma values to obtain brighter images are given by $i + 1$ for i from 1 to 9.

Finally, we studied how the model performed when the quality of the images was degraded. To this aim, we altered the images of our testing set by applying 4 transformations; namely, by blurring the images with increasing kernel sizes, adding salt and pepper noise with increasing probability, applying gamma correction with gamma values lower than 1 (the smaller the gamma values the darker the images), and applying gamma correction with gamma values higher than 1 (the higher the gamma values the brighter the images). The precision, recall, and F1-score achieved by the combined model for the degraded version of the test set is presented in Figure 4.2. As expected, the performance of the model decays when the quality of the images is considerably reduced by applying either blurring with big kernels or when excessive noise is added; however, it is robust to a reasonable amount of noise and changes in brightness and contrast.

Hence, the model trained with the combined dataset works well for images of stomata acquired using different magnification levels (namely, 4×, 10× and 20×) and from the three species employed for training the model — an example of the output images produced by the model is provided in Figure 4.3. However, the question remains whether the model can be used to analyse images of epidermal prints from other species that were not employed to train the model. This question is studied in the following section.

4.1.4 Generalising to multiple species

To test whether our YOLO model can be applied to analyse epidermal images of families that were not used for training purposes, we employed images from 64 families of plants obtained from the dataset presented in [134]. This is a challenging

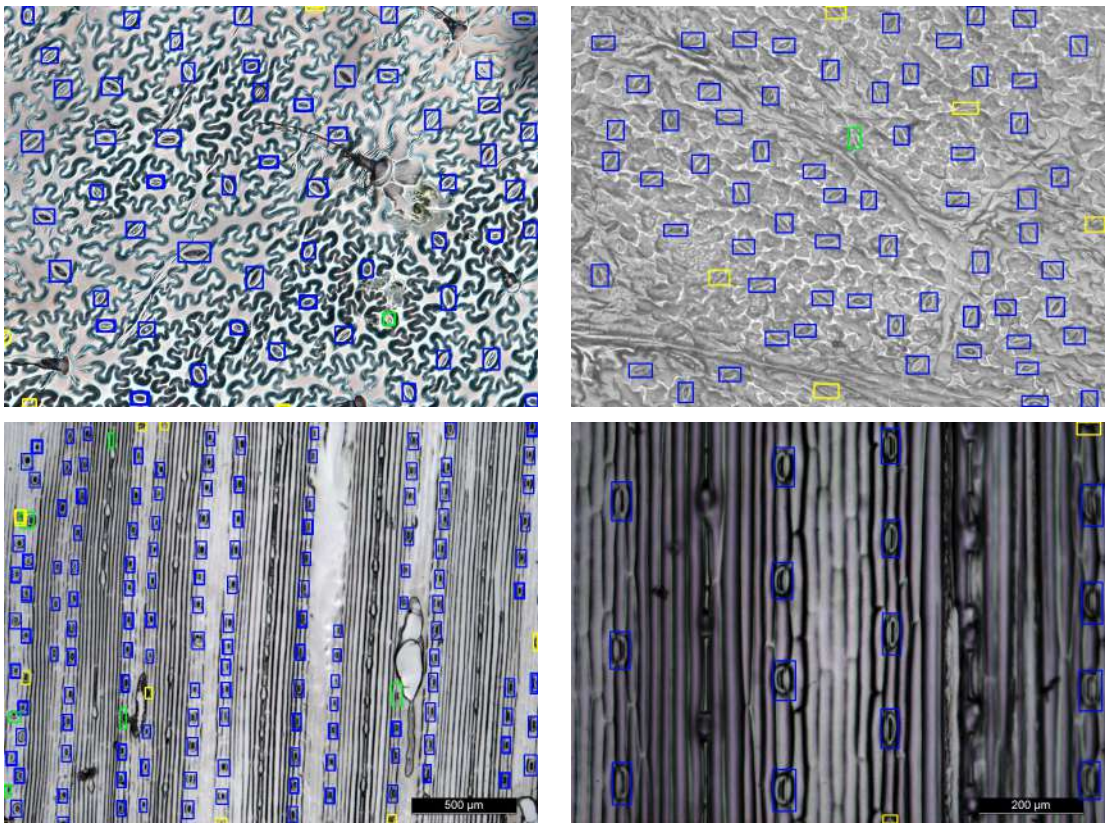


Figure 4.3: Output images produced by the combined model. *Top-Left.* Image from the Common bean dataset. *Top-Right.* Image from the Soybean dataset. *Bottom-Left.* Image from the Barley dataset acquired at 4× magnification. *Bottom-Right.* Image from the Barley dataset acquired at 10× magnification. The detected stomata are enclosed with blue boxes, whereas the non-detected and mis-detected stomata are enclosed with green and yellow boxes, respectively

dataset due to the variety of stomata sizes and shapes; and also the diversity of texture and quality of the images. Due to this considerable variability that was not present in the training set, our YOLO model only achieved a mean F1-score of 0.47, and for none of the families the F1-score was over 0.9, see Figure 4.4, this is a common problem in Deep Learning known as *domain shift* [151]. In this context, we approach this problem by applying two techniques based on ensembles: test-time augmentation (TTA) [152] and progressive resizing [153].

TTA is a technique previously explained in Chapter 2. In this context, we applied TTA using vertical flips, horizontal flips, and colour transformations to deal with the variability of stomata shapes, and the different textures and qualities of the images. In addition, in order to handle the variability of stomata sizes,

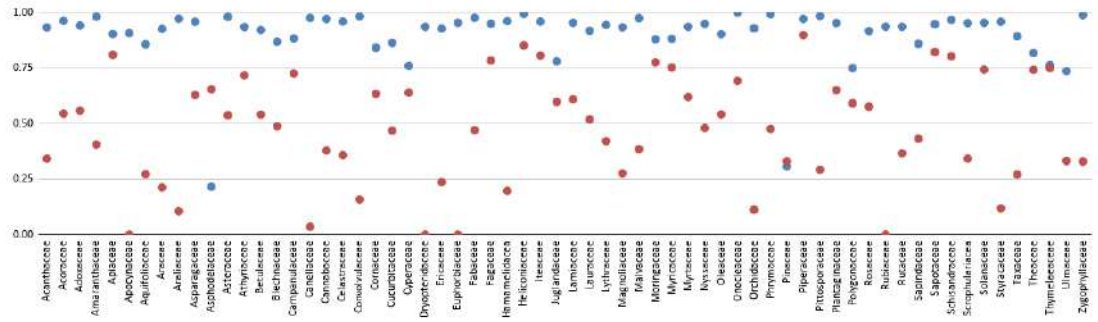


Figure 4.4: F1-score obtained by our YOLO model for 64 different families using the plain model (red circles), and combining the model with test-time augmentation and progressive resizing (blue circles).

we employed progressive resizing (PR); a technique usually applied at training time that can also be used at test time as follows. Namely, in order to obtain the predictions for an image, we resized such an image to three different sizes (416×416, 832×832, and 1248×1248), detected stomata in the three images using our YOLO model, and combined the results. This procedure allowed us to detect stomata of different sizes. The combination of TTA and progressive resizing with our YOLO model achieved a mean F1-score of 0.9 for the whole dataset, and a mean improvement of 43% for each family of the dataset, see Figure 4.4. It is worth underlying that using this method, the model is not retrained, and the achieved F1-score for the 64 families is similar to the one obtained for the species of our training set.

Finally, we conducted an ablation study to determine the importance of each technique in the final result, see Table 4.5. The first point to notice is the benefit of applying TTA, independently of the image size — an improvement of, up to a, 18% can be achieved with this method. However, the main boost for our model (more than a 20% improvement) was obtained thanks to the progressive resizing method. Finally, the combination of test time augmentation and progressive resizing improved the F1-score another 7%.

The results presented in this section show how our YOLO model can be employed to deal with a dataset of images with a huge variability among the stomata. However, in most scenarios, researchers work with a particular type of images, and it is simpler to adjust the model to work with those images as we show in the next section.

Technique	Precision	Recall	F1-score
416	0.96 (0.18)	0.47 (0.22)	0.60 (0.23)
832	0,91 (0.20)	0.50 (0.24)	0.62 (0.23)
1248	0.86 (0.24)	0.36 (0.22)	0.47 (0.24)
416 TTA	0.98 (0.07)	0.62 (0.24)	0.73 (0.22)
832 TTA	0.90 (0.17)	0.69 (0.25)	0.76 (0.22)
1248 TTA	0.88 (0.16)	0.56 (0.25)	0.65 (0.22)
PR (416–832)	0.96 (0.12)	0.72 (0.21)	0.80 (0.18)
PR (416–832–1248)	0.94 (0.13)	0.77 (0.22)	0.83 (0.19)
PR (416–832) + TTA	0.96 (0.04)	0.86 (0.17)	0.89 (0.14)
PR (416–832–1248) + TTA	0.93 (0.06)	0.90 (0.16)	0.90 (0.13)

Table 4.5: Mean (standard deviation) precision, recall and F1-score for the families of the dataset from [134]. The three first rows provide the results using different image sizes (416×416, 832×832, and 1248×1248); the following three rows show the results applying test-time augmentation (TTA); the next two, the results of applying progressive resizing (PR); and the last two, the results for the combination of PR and TTA.

4.1.5 Retraining the model for a particular species

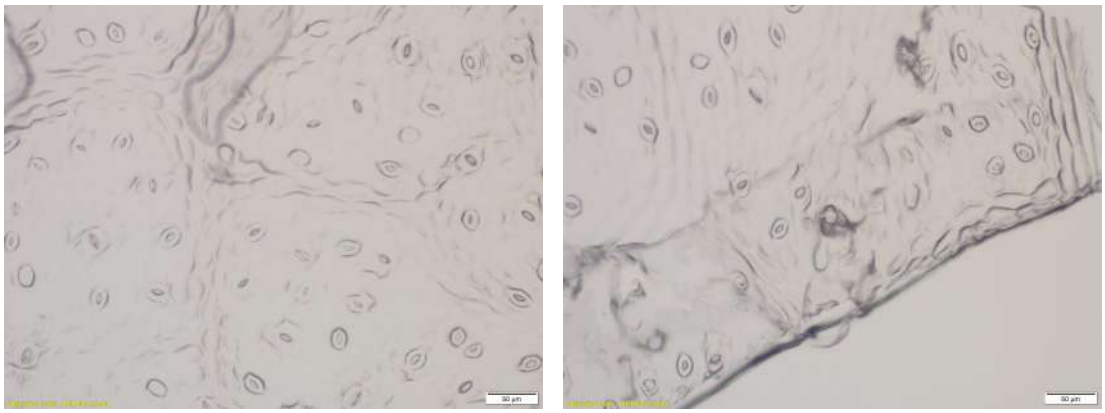


Figure 4.5: Examples from the grapevines dataset.

In order to illustrate how our YOLO model can be adjusted to work with a species that was not used during training time, we employed images of stomata from *grapevines* which are contained in the dataset presented in [130]. Specifically, the

dataset consists of 866 images for training and 57 for testing, two images from this dataset are show in Figure 4.5.

Three experiments were conducted with these images. First, the YOLO model created in Section 4.1.3 was run with the *grapevines* dataset. Second, we used 5 images from the training set of the grapevines dataset to train a new model pretrained with our model. Finally, we applied the data distillation technique presented in the previous chapter to the grapevines dataset. The results of these three experiments are presented in Table 4.6.

Dataset	Precision	Recall	Accuracy	F1-score
Template matching [130]	0.56	0.65	0.44	0.60
MSER [130]	0.53	0.37	0.28	0.44
COD [130]	**0.91**	0.79	0.74	0.85
Without retraining	0.88	0.81	0.73	0.85
Retrain with 5 images	0.90	0.91	0.83	0.91
Data distillation	0.90	**0.92**	**0.84**	**0.92**

Table 4.6: Results for the grapevines dataset presented in [130]. Best results are shown in bold face. This table is divided into two blocks, the former contains the results presented in [130], and the latter corresponds to the results obtained using our model.

Using our model trained on the combined dataset, we obtained similar results (lower precision and accuracy but equal F1-score and higher recall) to those presented in [130] as shown in Table 4.6. This demonstrates the robustness of our approach. Not surprisingly, the results were worse than those obtained for the images of our dataset. This issue was solved by retraining our model with just 5 annotated images from the grapevines dataset — we chose 5 images to ilustrate that our approach works with a small number of images. The model was retrained for 6000 steps and using the default parameters in YOLO. This yielded a 6% improvement in terms of F1-score, 9% in terms of accuracy, and 12% in terms or recall over the original model. Moreover, the F1-score was similar to the one obtained for our datasets. This shows that, with a small annotation effort, good models can be obtained using the model developed in this work as a basis. Finally, we applied the data distillation technique to automatically annotate 177 images from the training set of the grapevines dataset. Using those automatically images and retraining the model for 6000 steps, we obtained an improvement of 7% in terms of F1-score, 10% in terms of accuracy, and 13% in terms of recall with respect to the original model. Data distillation avoids the task of annotating images since

those annotations are automatically generated; but, it is more computationally consuming than the process of annotating a few images.

As a conclusion of these 4 sections, the model trained with the 3 original datasets not only performed well for images of the species employed for training it, but it also succeeded with images from other species. Furthermore, the model can be retrained using a few annotated images, or even using images without annotation, to improve a model for a particular species. However, the process of using the model to detect stomata, and retraining this model might be difficult for users without a machine learning background. Therefore, we have developed LabelStoma a graphical tool that simplifies these tasks.

4.1.6 LabelStoma

LabelStoma is a graphical tool implemented in Python, and developed using LabelDetection as a basis. LabelStoma aims to facilitate the use and creation of stomata detection models. Towards this aim, the functionality of LabelDetection has been restricted to the model presented in the previous sections. In this section, we present the main features of LabelStoma and how it has been designed.

The first set of features included in LabelStoma, see Figure 4.6, are devoted to measure stomatal density in images. To this aim, LabelStoma employs the YOLO model presented in Section 4.1.3, and, subsequently measures the stomatal density by computing the mean number of stomata per image. The model has been integrated in LabelStoma as an `IPredictor` object — this allows us to easily replace the model with new versions.

Moreover, LabelStoma provides the necessary functionality to modify those detections by adding and/or removing stomata to/from those detected by the model. Additionally, the box associated with each stoma can be adjusted. Neither DeepStoma [131], StomataCounter [134] or Stomata Detector [138], the other publicly available tools for stomata detection, offer this functionality. Moreover, LabelStoma not only provides the detections obtained for each image, but it also can generate a summary of the results in the form of an Excel file. The Excel file includes the stomatal density per image, and, other statistics like the mean width and height of the detected stomata, see Figure 4.7. It is worth noting that neither test-time augmentation or progressive resizing methods, presented in Section 4.1.4, are applied for detecting stomata — the interested reader can find how to apply these techniques in the project webpage.

The second set of features provided by LabelStoma are devoted to train new stomata detection models using transfer learning. As we have shown in the previous section, the model underpinning LabelStoma generalises quite well to images from species of plants that were not used to train the model; but, the accuracy from new species can be improved by retraining the base model with minimal ef-

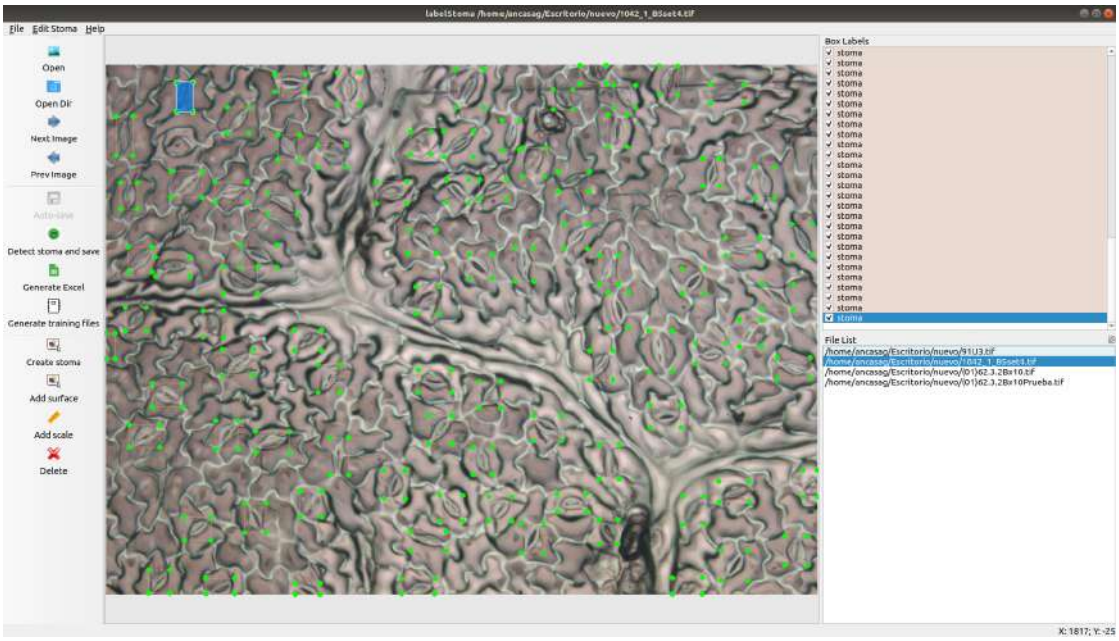


Figure 4.6: LabelStoma interface.

fort. LabelStoma simplifies that process thanks to an `ITrainer` object that allows users to apply transfer learning using our model as a basis.

Just to recall the process to fine-tune a stomata detection model, it consists of six steps: annotate a dataset of images, split the dataset into a training and test sets, apply data augmentation, slice the images, train the model, and evaluate the trained model. In addition training a model requires the user to configure the training environment, generate the configuration files for training the algorithm, and launch the training process. Thanks to LabelStoma, the user only has to manually annotate a few images (as shown in Section 4.1.5, annotation of five images is sufficient to retrain the base model to improve the model performance for images from a new species), and the other steps to train the model are conducted automatically by LabelStoma. LabelStoma automatises the last 5 steps and the user is only in charge of annotating a few images (we have shown in the previous section that it is enough with 5 images). To this aim, and using the functionality provided by LabelDetection, LabelStoma generates a zip file (that contains the dataset annotated with the structure required by the YOLO algorithm and also the necessary configuration files) and a Jupyter notebook that configures the environment, installs the necessary libraries, trains a YOLO model with the training set, and finally evaluates the model against the testing set. The Jupyter notebooks generated by LabelStoma can be run either locally, provided the user has a GPU, or using cloud services like Google Colaboratory — note that this equipment is

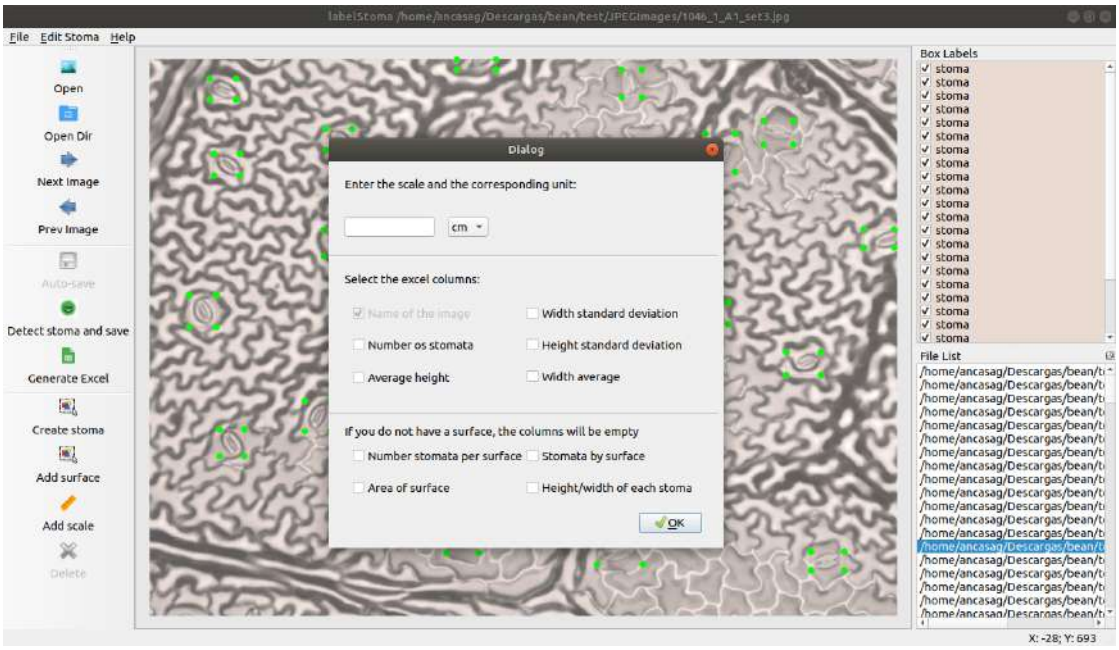


Figure 4.7: Statistics provided by LabelStoma to summarise stomatal density.

only required if the users want to adjust the underlying model of LabelStoma; otherwise, there are not any special hardware requirements to run LabelStoma for inference. Moreover, LabelStoma can also apply the data distillation procedure presented previously to improve the accuracy of the generated model by using an `ITrainer` object. Finally, new models trained with the explained functionality can be included in LabelStoma and used for detection in new images.

As a conclusion, LabelStoma is a graphical tool that aims to facilitate the detection of stomata, and the creation and use of stomata detection models. Due to the results obtained with LabelStoma for stomatal measurement, we applied the same approach to the problem of measuring bladder cells.

4.2 Measuring Epidermal Bladder Cells

Salinity presents one of the major problems for agricultural production, and tremendous efforts are made to provide salt tolerance to crops [154, 155]. One of the mechanisms involved in the response of halophyte plants to high saline soils are the epidermal bladder cells (EBC), which are specialized structures that accumulate salt and other metabolites [156]. EBC are present only in Aizoaceae and Amaranthaceae [157] and are thought to be involved in salinity tolerance [158, 159], as well as in UV-B protection, drought and other stresses tolerance [160, 161].

Recent interest has grown to understand the functioning of EBC and the molecular mechanisms of EBC formation and salt accumulation. However, the role of the EBC size, density or volume remains to be confirmed [162], as few studies have addressed the relevance of these traits, since these measurements are mostly manual using computer programs like ImageJ that have little to no automatisation [160, 162, 163, 161]. Those measurements are repeated over multiple images from different plant species, organs and growth conditions. Therefore, this is a tedious, error-prone, time-consuming and, in some cases, subjective task due to the large number of EBC in each image. We have tackled this problem by employing the object detection techniques presented in the previous chapters and following the same approach presented for LabelStoma. Namely, the main contributions of this part of the memoir are:

- We compare different models for the detection of EBC, and check their robustness using different parameters.
- We train a YOLO model to detect EBC in plant leaves using the techniques previously explained to train models with microscope images.
- We develop LabelGlandula[2], an open-source and easy-to-use graphical tool for measuring EBC density.

In the rest of the section, we explain how LabelGlandula has been designed and implemented. We first explain the methods used to capture EBC images, and to train the models. Subsequently, we analyse and compare different Deep Learning models for EBC detection. Finally, we introduce the architecture and design of LabelGlandula, and apply it to a case study.

4.2.1 Materials and methods

In this section, we present both the biological and computational materials and methods employed for building EBC detection models.

4.2.1.1 Biological methods

Images of leaf EBC from one quinoa variety (Marisma) grown under different salinity, elevated CO_2 and elevated temperature conditions were used. Besides, the density of EBC decreased as leaves become older, therefore, leaves of different age were used.

EBC can be easily removed when the plants are manipulated so the leaves were cut carefully from the plant and were hold with a tweezer. In order to obtain EBC

[2] `https://github.com/ancasag/labelGlandula`

images from both adaxial and abaxial surfaces, a cross cut was made, obtaining two leaf pieces, one for the analysis of each leaf side. Carefully and maintaining the EBC intact, from the central part of each leaf side, two leaf discs of 0.6 cm in diameter were cut in order to have a flat surface and, therefore, an image focused on the EBC.

Leaf EBC were observed using a Nikon SM645 Stereo Microscope 0.8× - 5× (Nikon Corporation, Japan) with a zoom range of $2X$. Images were captured with an Optikam ®Microscopy Digital USB Camera 4083.B3 (OPTIKA Microscopes, Italy) and Optika View 7.1 program (OPTIKA Microscopes, Italy). A total of 178 images of size 2048×1536 were acquired using this procedure and manually annotated using the LabelDetection, see Figure 4.8 for some samples of the captured images.

Figure 4.8: Images of leaf EBC.

4.2.1.2 Computational methods

Using the EBC dataset, we trained several models using different Deep Learning architectures for object detection. Namely, we used the two-phase algorithms Faster R-CNN [25] and EfficientDet [106], and the one-phase algorithms FCOS [108], CSResnet [164], FSAF [107], YOLOv3 [85] and YOLOv4 [164]. All the models were trained using the by-default parameters, and using a GPU GeForce RTX 2080 Ti. The necessary files to train those models were generated using LabelDetection (that is, LabelDetection was used to generate the configuration files for each library and the notebooks with the instructions for creating the models).

As in the case of the construction of stomata models, during the training process, we faced the images size challenge that arises when working with microscopic images. In order to deal with this problem, the images of the dataset were split into patches of size 600×600, and used for training (in this case, a bigger image size than in the stomata case was used due to the smaller number of experiments that were conducted).

4.2.1.3 Experimental study

In order to validate the aforementioned detection models, a 5-fold cross validation approach was employed. To evaluate the performance of the detection models, we measured their precision, recall, F1-score and mAP. In our statistical study the results were taken as the mean and standard desviation of the different metrics for the 5 folds.

In order to determine whether the results obtained were statistically significant, several null hypothesis tests were performed using the methodology presented in [165, 166]. We summarise the steps of this methodology as follows. In order to choose between a parametric or a non-parametric test to compare the models, we check three conditions: independence, normality and heteroscedasticity — the use of a parametric test is only appropriate when the three conditions are satisfied [165].

The independence condition is fulfilled in our study since we performed 5 different runs with independent folds. We use the Shapiro-Wilk test [167] to check normality — with the null hypothesis being that the data followed a normal distribution — and, a Levene test [168] to check heteroscedasticity — with the null hypothesis being that the results were heteroscedastic.

Since we are going to compare more than two models, an ANOVA test [166] is employed if the parametric conditions are fulfilled, and a Friedman test [166] otherwise. In both cases, the null hypothesis is that all the models haved the same performance. Once the test for checking whether a model is statistically better than the others is conducted, a post-hoc procedure is employed to address the multiple hypothesis testing among the different models. A Holm post-hoc procedure [169], in the non-parametric case, or a Bonferroni-Dunn post-hoc procedure [166], in the parametric case, is used for detecting significance of the multiple comparisons [165, 166] and the p values should be corrected and adjusted. We perform our experimental analysis with a level of confidence equal to 0.05. In addition, the size effect is measured using Cohen's d [170] and Eta Squared [171].

4.2.2 Results

In this section, we present the results of our study of object detection algorithms for EBC. In Table 4.7, we have summarised the mean and standard deviation of the 5 folds for each detection algorithm. The model trained with the YOLOv4 algorithm achieved the best results in terms of recall, F1-score and mAP. The YOLOv3 and FasterRCNN algorithms produced models with a better precision than YOLOv4, but their performance in the other metrics was considerably worse than YOLOv4.

For our statistical study, we focused on the F1-score metric, that provides a

	Precision	Recall	F1-score	mAP
YOLOv3	0.94(1.4)	0.75(10.2)	0.83(6.8)	0.84(1.5)
YOLOv4	0.92(0.4)	**0.90(0.4)**	**0.91(0.4)**	**0.89(0.1)**
FCOS	0.86(1.1)	0.80(2.4)	0.83(1.3)	0.70(3.3)
CSResnet	0.92(0.4)	0.83(2.0)	0.87(1.0)	0.84(0.8)
FSAF	0.86(5.0)	0.79(1.9)	0.83(0.8)	0.70(3.1)
EfficientDet	0.92(1.2)	0.78(2.4)	0.84(1.1)	0.72(3.2)
FasterRCNN	**0.94(0.7)**	0.81(1.4)	0.87(0.8)	0.78(0.4)

Table 4.7: Mean (and standard deviation) of detection models for EBC. In bold face the best result for each metric and dataset.

trade-off between precision and recall — the F1-score of the models is summarised in Figure 4.9. In order to compare the trained detectors, the non-parametric Friedman's test was employed since the normality condition was not fulfilled (Shapiro-Wilk's test $W = 0.855043$; $p = 0.000301$). The Friedman's test performed a ranking of the models compared (see Table 4.8), assuming as null hypothesis that all the models had the same performance. We obtained significant differences ($F = 14.85; p < 4.89 \times 10^{-7}$), with a large size effect eta squared = 0.53.

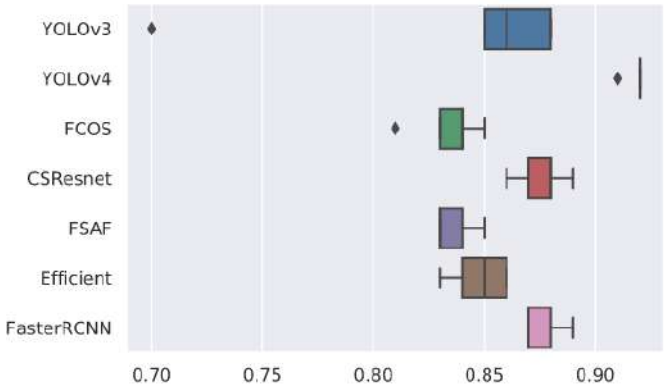


Figure 4.9: Results from 5 independent runs in F1-score for selected object detection algorithms.

Technique	F1-score	Friedman's test average ranking
YOLOv4	0.91 (0.40)	7
CSResnet	0.87 (1.01)	5.4
FasterRCNN	0.87 (0.80)	5.2
YOLOv3	0.83 (6.8)	3.5
EfficientDet	0.84 (1.16)	3
FSAF	0.83 (0.80)	2
FCOS	0.83 (1.35)	1.9

Table 4.8: Friedman's test for the F1-score of the object detection models.

The Holm algorithm was employed to compare the control model (winner) with all the other models adjusting the p value, results are shown in Table 4.9. As it can be observed in Table 4.9, there are three object detection algorithms with no significant differences as we failed to reject the null hypothesis. The size effect is also taken into account using Cohen's d, and as it is shown in Table 4.9, it is medium or large when we compare the winning model with the rest of the models.

Technique	Z value	p value	adjusted p value	Cohen's d
CSResnet	1.17108	0.241567	0.375366	4.84
FasterRCNN	1.31747	0.187683	0.375366	5.93
YOLOv3	2.56174	0.010415	0.031245	1.55
EfficientDet	2.9277	0.00341479	0.0136592	7.18
FSAF	3.65963	0.000252584	0.00126292	11.59
FCOS	3.73282	0.00018935	0.0011361	7.51

Table 4.9: Adjusted p-values with Holm, and Cohen's d. Control technique: YOLOv4.

As it can be seen in the results presented in this section, the algorithm that produces the best model is obtained with YOLOv4. However, using this model might be challenging for non-expert users since it requires experience working with deep learning libraries. We have addressed this issue by developing a user-friendly application, called LabelGlandula, that particularises LabelDetection to work only in the context of EBC.

4.2.3 LabelGlandula

LabelGlandula, is a graphical tool implemented in Python to detect and measure EBC based on LabelDetection. Like the functionality provided by LabelStoma, the first set of features included in LabelGlandula, see Figure 4.10, are devoted to measure the amount of EBC in a set of images. Given a folder with images, LabelGlandula detects the EBC in each image using the YOLOv4 model. In addition, LabelGlandula provides the necessary features to modify the detections by adding and/or removing EBC from those detected by the model. Moreover, the box associated with each EBC can be adjusted. Finally, LabelGlandula not only provides the detections obtained for each image, but can also generate a summary of the results in the form of an Excel file. Such an Excel file includes the number of EBC in each image and other statistics such as average area of EBC, folial area or covered folial area.

As in the case of LabelStoma, other features provided by LabelGlandula are devoted to retrain new EBC detection models with minimal effort. LabelGlandula simplifies that process thanks to an `ITrainer` object that allows users to apply transfer learning using our model as a basis.

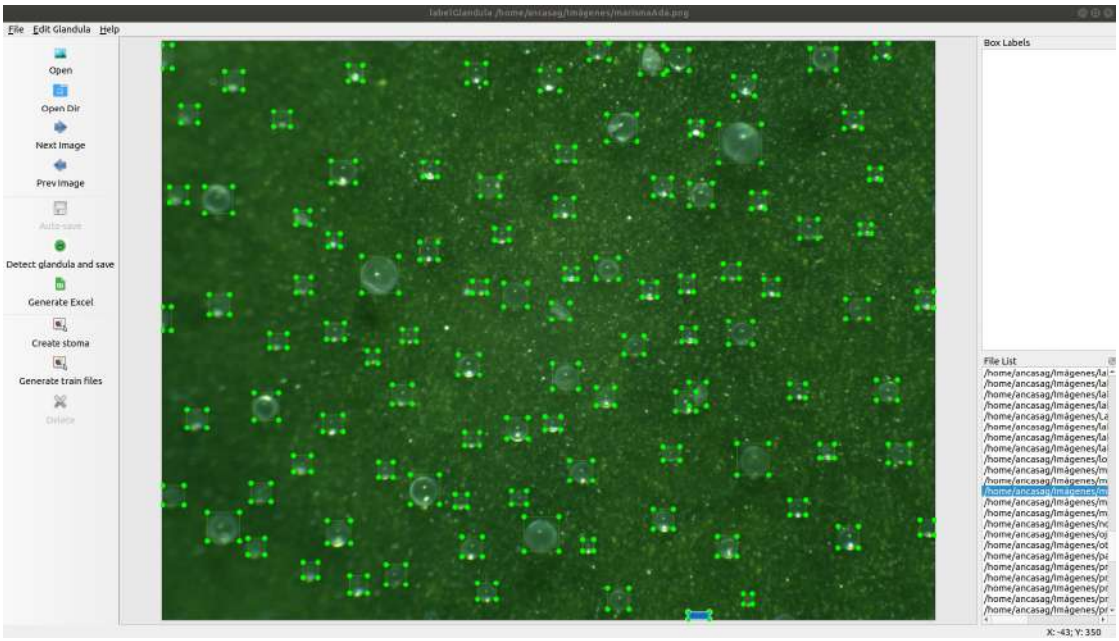


Figure 4.10: LabelGlandula interface.

In order to illustrate the feasibility of using LabelGlandula for counting and measuring EBC from different images to those used for training the underlying detection model, a case study is presented in the following section.

4.2.4 Case study

In this case study, we analyse how LabelGlandula behaves with images from three experiments that were grown under different conditions and whose images were not used for training the original model. To this aim, we used 11 datasets that are summarised in Table 4.10 some samples from those datasets are provided in Figure 4.11.

From the same experiment used to train the original model, we selected EBC images not used for training, or evaluating the precision of the model. Besides, in order to validate the model for conditions not explored in the training set, we used images of the same variety; but taken from the stems. For stem analysis, in the main stem section where the petiole of the youngest fully expanded leaf joins the stem, one thin slice of the epidermis containing EBC was cut longitudinally with a scalpel. The slice was carefully put on a slide avoiding the removal of the EBC. The stem slices were observed in the same way as the leaf pieces. In addition, we used images from another quinoa variety, concretely, Pasancalla. Pasancalla was characterized by having white and purple coloured EBC while Marisma showed only

white coloured EBC. Pasancalla images were taken in the same way as explained in Section 4.2.1. All these factors affect EBC features, so the images between datasets are different among them.

Name	Variety	Conditions	Leaf/Stem	Leaf side	Concentration	♯ images
MALB0	Marisma	Actual	Leaf	Abaxial	0 mM	8
MALD0	Marisma	Actual	Leaf	Adaxial	0 mM	8
MALB125	Marisma	Actual	Leaf	Abaxial	125 mM	8
MALD125	Marisma	Actual	Leaf	Adaxial	125 mM	8
MAS0	Marisma	Actual	Stem	-	0 mM	8
MAS500	Marisma	Actual	Stem	-	500 mM	8
MCLB0	Marisma	Climate Change	Leaf	Abaxial	0 mM	8
MCLD0	Marisma	Climate Change	Leaf	Adaxial	0 mM	8
MCLD500	Marisma	Climate Change	Leaf	Adaxial	500 mM	8
PALB0	Pasancalla	Actual	Leaf	Abaxial	0 mM	8
PALD0	Pasancalla	Actual	Leaf	Adaxial	0 mM	6

Table 4.10: Features of the testing datasets.

In order to evaluate the usage of LabelGlandula in those datasets, we investigated a pipeline where the YOLO model was used for detecting the EBC, and a user edits, by means of LabelGlandula, those detections to add missing EBC and remove those that were not correct. For our experiments, we evaluated, see Table 4.11, the YOLOv4 model using the same metrics presented in Section 4.2.2, and we have also included the number of True Positive (TP), False Positive (FP), and False Negative (FN) EBC.

Dataset	Precision	Recall	F1-score	mAP	TP	FP	FN
MALB0	0.99	0.97	0.98	0.90	942	8	24
MALD0	0.98	0.93	0.95	0.90	603	9	42
MALB125	0.99	0.92	0.95	0.90	3656	17	316
MALD125	0.99	0.96	0.97	0.90	1803	5	72
MAS0	0.99	0.86	0.92	0.81	780	6	118
MAS500	0.97	0.80	0.88	0.79	2301	62	544
MCLB0	0.99	0.96	0.98	0.90	2178	4	72
MCLD0	0.99	0.93	0.96	0.90	1187	3	89
MCLD500	0.99	0.95	0.97	0.90	2327	10	100
PALB0	0.99	0.90	0.95	0.90	4676	14	465
PALD0	0.97	0.80	0.88	0.80	2793	76	659

Table 4.11: Results for the case study.

From those results, we can draw two main conclusions. First of all, the precision of the model is over 97% for all datasets, this indicates that most detections produced by the model are correct, and the user only needs to remove a few of them (in the worst case, a 2.64% of the detected EBC). The second conclusion is related to the number of EBC that must be added by the users; that is the recall.

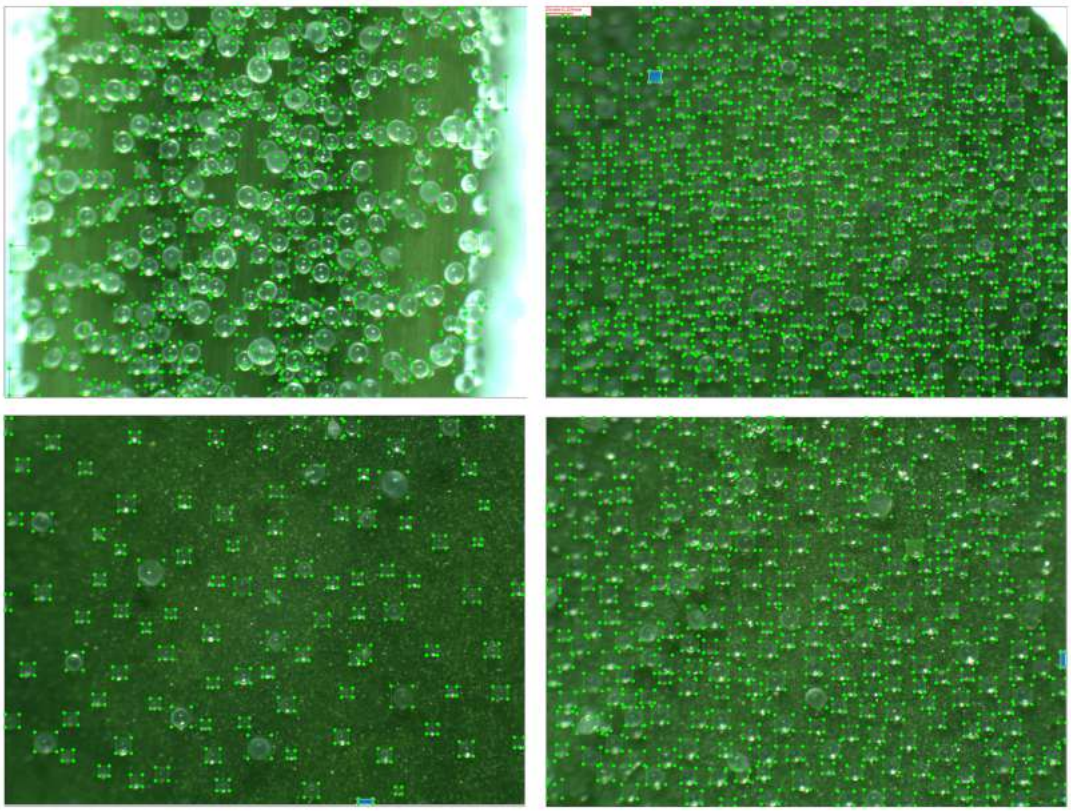


Figure 4.11: Samples from the testing datasets and predictions produced by LabelGlandula. *Top-Left.* Image from the MAS500 dataset. *Top-Right.* Image from the PALB0 dataset. *Bottom-Left.* Image from the MALD0 dataset. *Bottom-Right.* Image from the MCLD500 dataset.

For this metric, the model achieved a value over 90% for all the datasets but 3 of them. The value for those datasets is over 80%; however, this means that almost a 20% of the EBC must be manually added. This happens because images taken with the conditions of those 3 datasets where not included in the training dataset. This issue is known as domain shift, and it is an open problem for Deep Learning models [172]. In our context, the problem is mitigated thanks to LabelGlandula that allows the user to easily add EBC; however, further research is needed in this context, for instance by retraining the model or applying semi-supervised learning techniques. In contrast with LabelStoma, and in spite of the fact that it would be straightforward to apply those techniques thanks to the functionality of LabelGlandula, they have not been studied yet because biologist considered these results obtained by the model acceptable, and the principal investigator from the biological side got sick and the collaboration is currently on pause.

4.3 Conclusions

In this part of the memoir, we have seen how the techniques and tools presented in the previous chapters can be applied to deal with two actual problems in plant physiology: stomata detection and EBC measurement. As a result, we have developed LabelStoma and LabelGlandula, two open-source tools that are not only used by the researchers from Auburn University and University of the Basque Country, but also by other teams in, for instance, French Guyane and Italy.

Thanks to the development of LabelStoma, the analysis of plant stomata among species will be more reliable. Furthermore, it will help to the better understanding of CO_2 and H_2O dynamics in plants related processes, such as photosynthesis and transpiration, and in the atmosphere, to predict future carbon and water cycles. Likewise, LabelGlandula will make the analysis of EBC more reliable and will allow plant biologists to advance their understanding of the role of the EBC size, density and volume in salinity, drought and other stress tolerance. Namely, LabelGlandula will help to understand the functioning of EBC and the molecular mechanisms of EBC formation and salt accumulation, and to transfer this knowledge to crops one day [157, 160].

Chapter 5

Applications to Precision Agriculture

In the previous chapter, we have seen how the methods and tools developed for object detection in the first chapters of this memoir can be applied to concrete problems in plant physiology. As we showed in Chapter 3, those methods and tools can be generalised to other Computer Vision tasks such as semantic segmentation. In this chapter, we focus on such a generalisation for a particular task: precision agriculture. Namely, we address the problem of segmenting in-field raw images data (natural images) in viticulture. The results presented in this chapter come from a collaboration with the group of the Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing, from the National Research Council of Italy.

In order to present how we have addressed the segmentation of viticulture images, we divide this chapter into three sections. First, we explain how we have trained and compared various deep architectures for segmenting different elements in a vineyard, and approached the problem of having a limited number of annotated images by applying semi-supervised learning methods. Subsequently, we show that our segmentation models generalise properly on a highly variable dataset. Finally, we present how depth information can be used to improve the results of the semantic segmentation models. The three sections are organised following the same scheme: first, we provide an introduction where we put the problem in context; subsequently, we present the dataset, architectures and metrics that have been used; finally, we end each section with the results, a discussion and some conclusions.

All datasets and models developed in this part of the memoir are available in the project webpage[1].

[1] `https://github.com/ancasag/PrecisionAgricultureApp`

5.1 Semi-Supervised Learning for Semantic Segmentation in Viticulture

Sustainability is a crucial goal that involves ecological, economic and social concerns to impact the health of present and future societies. Scientific progress has developed new automatic tools to assist the human workforce by integrating artificial intelligence and robotics to meet such high-level needs. These efforts affect all production fields but, significantly, agriculture, whose improvement needs to face sustainability-related topics, such as finite resource management, yield optimization and pest control. In general, every sustainable goal can require actual crop monitoring by implementing low-cost technologies (cameras) and reliable methodologies (machine and deep learning techniques) in engineered solutions [173]. These requirements translate into the need for developing image acquisition and processing systems for extracting helpful information for the farmer. At a low level, systems must identify specific targets by applying semantic inference mechanisms, including image classification or segmentation.

In general, crop monitoring without physical contact of the targets can be clustered in remote and proximal sensing, depending on the sensor-plant distance and, thus, the level of details of the achievable information. Remote sensing typically refers to aerial imaging from satellites, unmanned aerial vehicles (UAVs) or airplanes. UAVs are equipped with imaging sensors, such as hyperspectral, LIDAR and RGB cameras [174, 175], to compute vegetation indicators, e.g. the normalized difference vegetation index (NDVI) or canopy size and volume [176] or to create semantic maps of the fields [177, 178, 179, 180, 181]. In proximal sensing, acquisitions are taken from the ground, close to the target, and with more details. Typical sensors include color, hyperspectral and infrared (IR) thermal cameras and LIDAR [182, 183], targeted to object segmentation, fruit counting, phenotype analysis, plant classification and disease monitoring [184, 185, 186]. In proximal sensing, data can be collected in structured and well-controlled environmental contexts, such as greenhouses [187, 188], or under excellent acquisition conditions, typically manual, with high-resolution sensors [189]. Referring to extensive crops, the practical implementation of proximal sensing is achievable through agricultural robots working in-field. However, any approach to extensive monitoring must face the actual problems of natural images, such as low resolution, motion blurring, occlusions and uncontrolled lighting conditions.

The processing of natural images captured from ground robotic platforms, and more specifically the semantic segmentation of images, has been proposed mainly for weed detection [190, 191, 192, 193], even sharing the same input dataset [194] and a common processing background, centered on Deep Learning [21]. More specifically, input images, which are often reported in terms of NDVI, are pro-

cessed by convolutional neural networks (CNNs) for pixel or area classification, trained from scratch or by applying transfer learning [40]. Deep learning is often used to segment objects of interest, such as fruits, leaves, infrastructures (wires and poles) and single branches [195, 196]. In horticulture, several methodologies have been presented for monitoring fruit orchards through flower classification for thinning [197], fruit classification for automatic harvesting [198] and segmentation of supporting infrastructures, such as wires [199].

Automatic procedures for object segmentation are even more attractive in those areas of horticulture of high added value, such as viticulture [200]. Here, monitoring at the plant scale allows vine-growers to understand possible spatial variabilities and find fine-tuned solutions. For instance, a ResNet deep residual network and a region-based convolutional neural network to detect green shoots in grapevine canopies and precisely segment the trajectories of cordons for thinning purposes was presented in [201]. Grape cluster and canopy segmentation using an artificial neural network and a genetic algorithm on images of a publicly available dataset [202] were proposed by [203], whereas a comparison of three neural networks for segmentation of grape clusters tested on the Embrapa Wine Grape Instance Segmentation Dataset was presented in [204].

In any of the above cases, all sensors were standard RGB cameras, which provide a flat 2D representation of the targets. In contrast, RGB-D cameras, able to produce three-dimensional (3D) colored models of the crops, can give more information, helpful for fruit monitoring and counting [205]. Several technologies, including complex setups of dedicated 3D cameras [206, 207] or integrated low-cost consumer-grade cameras, such as the Microsoft Kinect v1 and v2 cameras (Redmond, WA, USA) [208, 209, 210, 211, 212], have been used for plant phenotyping, fruit counting and automatic robotic harvesting. Even low-cost stereo cameras, such as those of the Intel Realsense family (R200 and D4xx, Santa Clara, CA, USA), have gained attention in fruit detection and plant phenotyping [213] since they can effectively model the outdoors without suffering from illumination variability due to sunlight [214]. Several works processing color images acquired by the Intel RealSense R200 and D435 for object segmentation have been presented [215, 216, 217]. Although RGB-D cameras help yield monitoring, output color images are often of low quality and resolution due to the actual scope of such low-cost cameras, which are mainly designed for robot navigation, mapping and manipulation. Natural image segmentation from color data is still an open problem since its effective solution enables the effective use of the depth channel.

In this scenario, this work extends previous work by [218] for the exact segmentation of plant leaves and wooden structures (trunks, branches, canes, etc.), artificial infrastructures (poles, ropes, cables, etc.) and fruits. Here, multiple network architectures have been compared to find the best solution for natural image

segmentation. Even a refined ground truth was considered to further improve the quality of segmentation. Moreover, we have studied three semi-supervised learning approaches to deal with the small size of an annotated dataset by taking advantage of unlabeled images.

5.1.1 Materials and methods

In this section, we present both the viticulture and computational materials and methods employed for building the semantic segmentation models.

5.1.1.1 Input datasets

In our experiments, we tackled the problem of segmenting natural images captured in-field by the low-cost consumer-grade Intel Realsense R200 camera (Santa Clara, CA, USA).

Our datasets consist of 405 color images acquired by the Intel Realsense R200 in a vineyard in Switzerland (Räuschling, (N47º 14' 27.6", E8º 48' 25.2")). The camera was mounted on a moving agricultural tractor (Niko Caterpillar, Bühl/Baden, Germany) and acquired lateral views of the line of the grape plants at a distance between 0.8 and 1 m. Under those conditions, every image covered a horizontal field of view between 0.9 and 1.2 m to completely frame every plant in a single image. The tractor moved within lines at an average speed of 1.5 m/s. Image frame rate was then tuned according to the robot speed and the horizontal field of view of the camera to frame the same plant in at least three consecutive captures. A camera frame rate of 5 Hz was enough to produce image overlaps, corresponding to about 0.3 m. The image resolution was limited to 640×480 pixels to match the maximum resolution of the depth data stream. It is worth noticing that, although video sequences were produced to create overlaps, the proposed implementations did not take advantage of object tracking strategies, like the one of [204]. All methodological approaches considered images individually, without managing multiple detections of the same elements. A sample image of the dataset is shown in Figure 5.1.

As shown in Figure 5.1, the resulting images are poor in detail and clearness. Due to the effect of the movement of the tractor, and also the low quality of both camera sensor and optics, images suffer from blurring, soft hue and weak contrast. Moreover, JPEG compression applied to images further decreased the quality of the acquisition. For example, the inset of Figure 5.1 shows how similar the appearance of the foreground grape bunches and the background small leaves are.

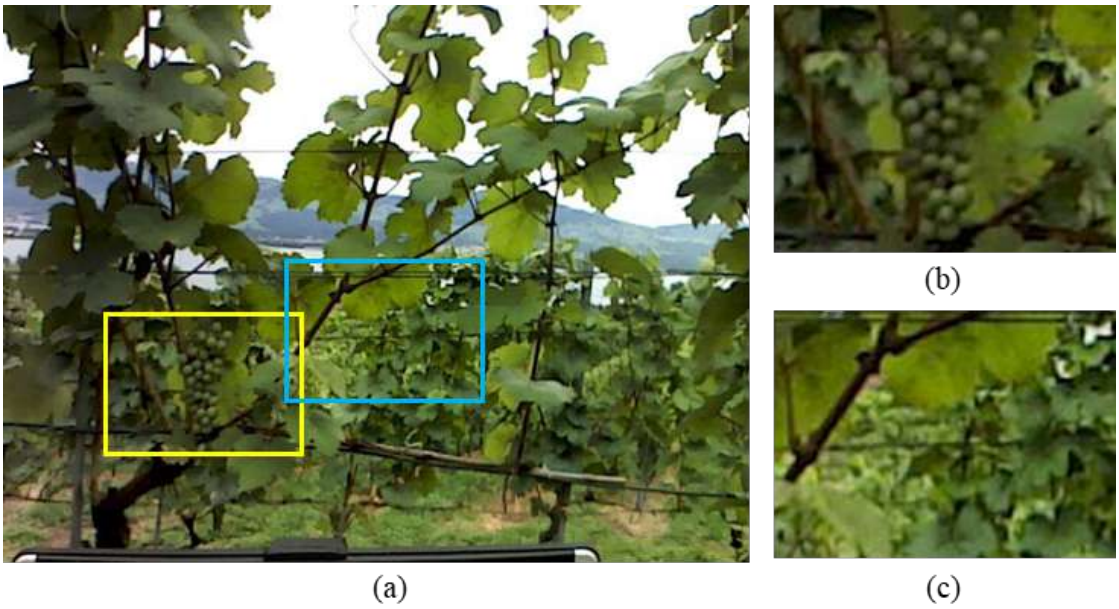


Figure 5.1: (a) A sample color image acquired by the Intel Realsense R200. (b) and (c) are magnifications of the area enclosed by the yellow and cyan boxes, respectively.

The automatic segmentation of natural images is achieved by representing them in more descriptive and discriminative feature spaces, learned from actual images, where pixels having similar semantic attributes can be grouped and labeled in different classes. A set of annotated images is thus required to train the model and then evaluate the segmentation results on ground truth.

Manual annotation is a complex, time-demanding and tedious task. For this reason, annotation is typically limited to a small subset of all the images acquired in-field. However, unlabeled images were captured under the same experimental conditions and could give further information to tune the training of the networks using semi-supervised approaches.

The whole dataset of 405 natural color images from the Intel Realsense R200 camera was thus split into two sets of 85 manually annotated images and 320 unlabelled images. The 20–80 proportion was chosen to give more evidence to the improvement of results due to the semi-supervised approaches. Within these lines, images were processed to segment five classes of interest:

- Bunch: bunches of white grapes;
- Pole: supporting infrastructure made of concrete or metal poles;
- Wood: canes, cordons and trunks of the plant;

- Leaves: canopy leaves of the grape; and,
- Background: the remaining objects framed by the camera, such as the ground, the sky and far grape lines.

Manual annotation was performed twice on the same images to produce two sets of labels:

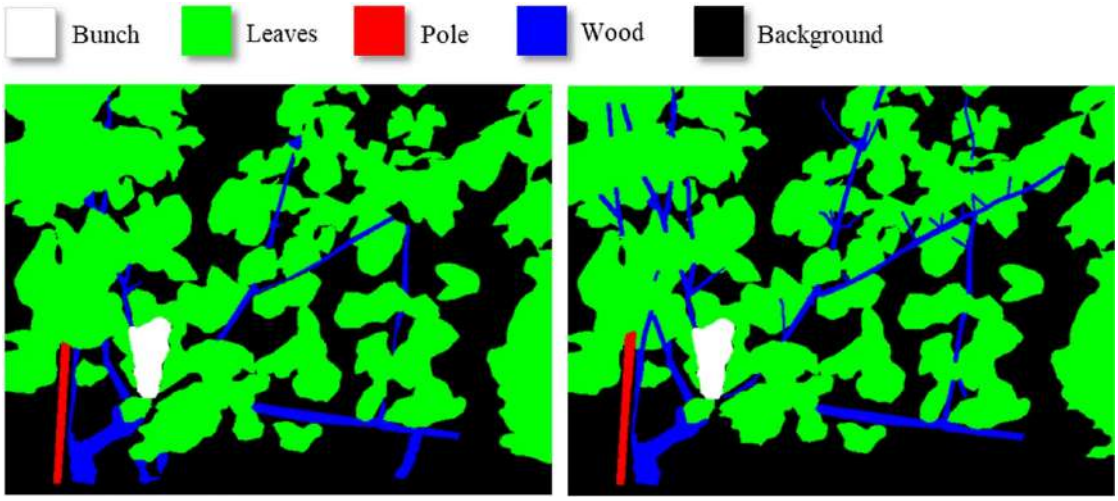
- Bunch/leaves-detection-oriented (BLDO) labels: BLDO labels were the same as in [216, 219, 213] and were mainly focused on the bunch and leaves segmentation. The corresponding ground truth was obtained for each image, giving different priority levels to each class. First, bunches were annotated as closed objects, even if their appearance slightly differed from what was expected as the effect of a crossing object or image artifacts. Then, plant leaves, poles and wooden structures were annotated with the same strategy but with decreasing priority levels. The background was the last labeled class, enclosing the remaining pixels; and,
- Object-segmentation-oriented (OSO) labels: OSO labels were created for an object segmentation task, as typically referred to in the corresponding literature. Annotation gave equal priority to every class to label objects as they appeared in the image.

An insight into the difference between the two kinds of labels is shown in Figure 5.2. Specifically, all wooden structures or supporting infrastructures, i.e. poles, have more weight and are better detailed since they are no longer included in the leaves class. In the following lines, all analyses were run on BLDO labels for enabling the comparison with previous results in [216, 219]. Then, further experiments on the best models, trained by OSO labels, are presented to discuss the importance of manual labeling for accurate segmentation results.

Once the datasets and their annotations have been presented, the following subsections detail the network architectures and the semi-supervised algorithms employed in this work.

5.1.1.2 Semantic segmentation models

As stated in the previous section, the 85 labeled images, split into training and test sets, were used to set up and evaluate several deep segmentation architectures. The training set was used to fine-tune several deep-learning segmentation architectures [220] to produce inference on natural images in the form of output masks. With more details, 13 architectures, summarised in Table 5.1, were trained. All the selected architectures were based on either fully convolutional networks (FCN) [31] or encoder-decoder networks [32].



(a) (b)

Figure 5.2: Annotations of the image in Figure 5.1: (a) bunch/leaves-detection-oriented labels (BLDO); (b) object-segmentation-oriented labels, resulting in a fine refinement of the BLDO labels in (a).

FCN architectures extract features from a given image using a backbone of convolutional layers and generate an initial coarse classification map. The classification map is a spatially reduced version of the original image. Then, deconvolutional layers restore the original resolution of the classification map to output the final segmentation mask. The main two drawbacks of this architecture are the loss of information when working with high-resolution images and its speed. For tackling the high-resolution problem, in the HRNet architecture [221], high-resolution representations were maintained by connecting high-to-low resolution convolutions in parallel and repeatedly conducting multi-scale fusions across parallel convolutions. Atrous convolutions were instead used in the DenseApp architecture [222] to face the same resolution issue. For tackling the problem of the high time requirements, the ContextNet architecture [223] used factorized convolution, network compression and pyramid representation, while the CGNet architecture [224] employed a context-guided block.

In the encoder-decoder architectures, the encoder is usually made of several convolutional and pooling layers responsible for extracting the features and generating an initial coarse prediction map. In these architectures, the encoder is known as the backbone. The decoder, commonly composed of convolution, deconvolution and/or unpooling layers, is responsible for further processing the initial prediction map, increasing its spatial resolution gradually and generating the final prediction. The Unet architecture [32] was the first network to propose an

Type	Segmentation architecture	Backbones
FCN	CGNet	CGNet
FCN	ContextNet	ContextNet
FCN	DenseApp	DenseApp
FCN	HRNet	W30
Encoder-decoder	Bisenet	Resnet18, Resnet34
Encoder-decoder	DeepLabV3+	EfficientNet-B3, Resnet50, Resnext50
Encoder-decoder	FPENet	FPENet
Encoder-decoder	LedNet	Resnet50
Encoder-decoder	PAN	EfficientNet-B3, Resnet50
Encoder-decoder	Unet	EfficientNet-B3, Resnest50, Resnet50
Encoder-decoder	Unet++	EfficientNet-B3, Resnest50, Resnet50
Attention	Manet	EfficientNet-B3, Resnest50, Resnet50
Attention	OCNet	Resnet50

Table 5.1: Segmentation architectures and the backbones employed in this work.

encoder-decoder architecture to perform semantic segmentation in medical contexts. From that seminal work, several variants have been proposed to address the two main limitations of the Unet architecture that are the same as previously mentioned for the FPN architecture: the loss of information when working with high-resolution images and its speed. Regarding the issues related to the use of images of high-resolution:

- The DeepLabV3+ [225] architecture introduces the notion of atrous convolutions to extract features at an arbitrary resolution.
- The PAN architecture [226] adopted a global attention upsample module to squeeze high-level context and embedded it into low-level features as guidance.
- The FPENet architecture [227] defines a MEU module that used attention maps to embed semantic concepts and spatial details to low-level and high-level features.
- The Unet++ architecture [228] redesigns the connection between the encoder and the decoder components of the architecture.

Referring to the speed issue:

- The Bisenet architecture [229] proposed a fast downsampling strategy to obtain a sufficient receptive field.
- The LedNet architecture [230] employed an attention pyramid network in the decoder.

All the aforementioned architectures are based on convolutional operations. In addition, two architectures based on the attention mechanism, namely OCNet [231] and Manet [232], were considered.

All the architectures with their respective backbones presented in Table 5.1 were trained using the PyTorch [233] and FastAI [153] libraries on an Nvidia RTX 2080 Ti GPU. The procedure presented in [153] was employed to set the learning rate for the different architectures (this procedure trains the model for a few epochs while gradually increasing the learning rate and monitoring the loss at each step. After training, it is possible to determine the learning rate where the loss is decreasing rapidly and exhibits stability). Also, early stopping was applied in all the architectures to avoid overfitting. The code employed to train these models can be generated with the functionality of the framework presented at the end of Chapter 3.

After training, all the models were then evaluated on the test set of 25 annotated images using the mean segmentation accuracy of the c-th class (MSA_c):

$$MSA_c = mean\left(\frac{TP_c}{n_{obs,c}}, \forall \text{ images} \in \text{dataset}\right).$$

where TP_c is the number of true positives, i.e. correct pixel labels over the entire population of the c-th class ($n_{obs,c}$) [216].

5.1.1.3 Semi-supervised learning methods

As stated previously, the dataset contains 320 additional unlabeled images. In this case, semi-supervised learning approaches can help the training phase by adding more information from unlabeled images as we have seen in Chapter 2. For this reason, three semi-supervised learning approaches were employed: PseudoLabeling [109], Distillation [234] and Model Distillation [68] — all these methods are available in the framework developed in Chapter 3. Figure 5.3 presents a sketch of each of these semi-supervised learning methods. These methods are generalisation of those presented for object detection in Chapter 2; so, we just provide a brief overview of them here.

The PseudoLabeling approach consists of two steps: given a deep learning architecture, a first model is trained using that architecture on a manually labeled dataset to make predictions in an unlabeled dataset; secondly, the manually and automatically-labeled datasets are combined to train a new model using the same previous architecture. This PseudoLabeling approach was applied to all the architectures presented in the last section (Table 5.1).

The Distillation approach is similar to PseudoLabeling, but in the second step, the trained model might have a different underlying architecture than the model trained on the first step. In this case, all the models of Table 5.1 were trained

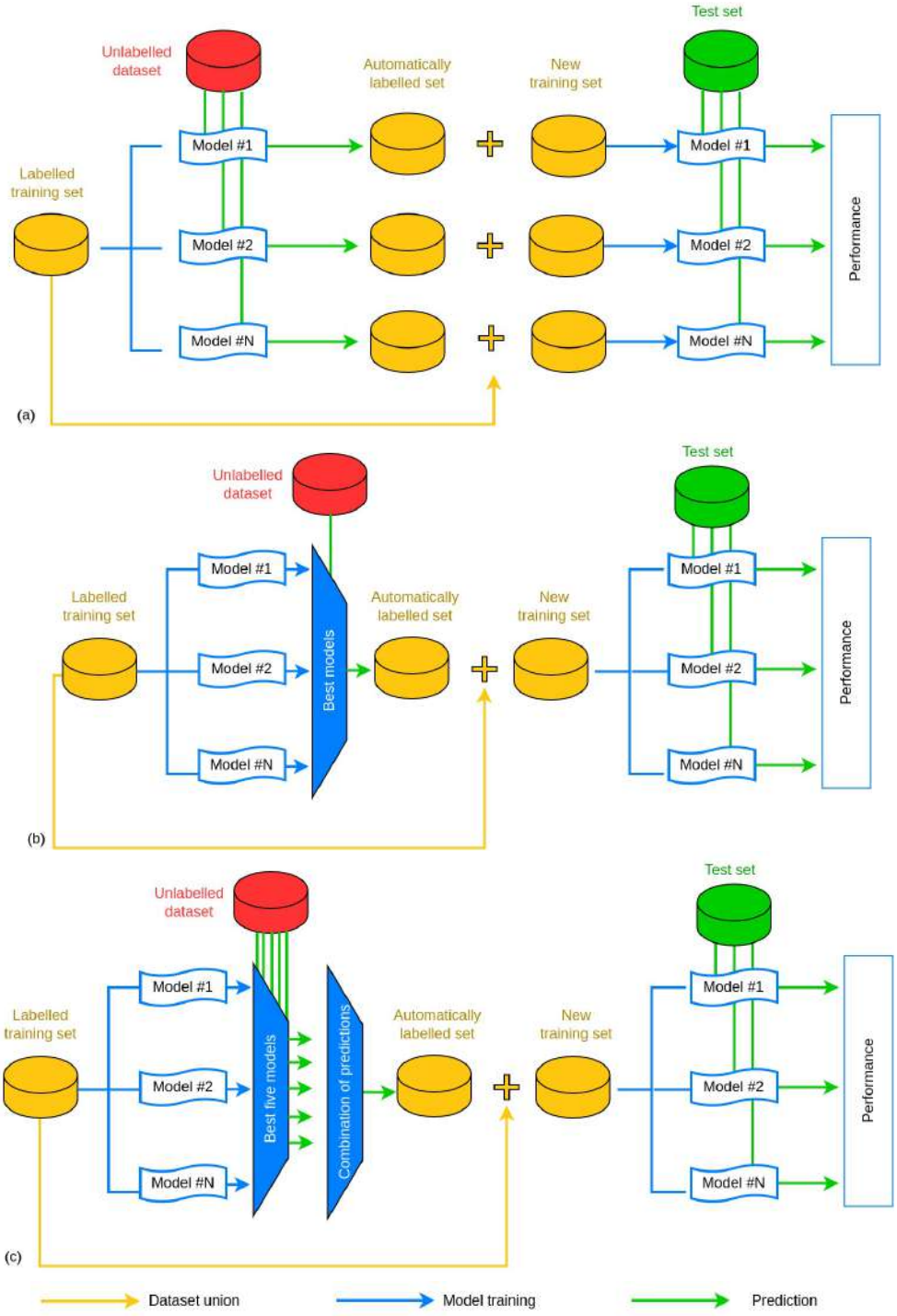


Figure 5.3: Schemes of the semi-supervised approaches presented in this analysis: (a) PseudoLabeling, (b) Distillation and (c) Model Distillation. Yellow, blue and green arrows refer to the processes of dataset union between manual and automatically labeled datasets, model training on the corresponding training set, and prediction of the input images with the model crossed by the arrow, respectively. The models enumerated from 1 to N represent the architectures and backbones of Table 5.1.

using the training procedure presented in the previous section, but only the best model was used for generating the automatically labeled dataset. Then, both sets (manually and automatically labeled) were combined to re-train all the architectures from Table 5.1.

Finally, Model Distillation differs from the Distillation approach in the production of the automatically labeled dataset. Instead of using a single model for making predictions in an unlabeled dataset, predictions are generated from an ensemble of models. In this approach, the five models with the best total MSA produced the predictions on the unlabeled dataset, which were then combined to create single images. Finally, as in the previous approaches, the manually and automatically-labeled datasets were used to train all the architectures presented in the last section.

In addition to searching for the best-performing model, a statistical study was conducted to determine whether the results obtained with the different semi-supervised learning approaches were statistically significant. To this aim, several null hypothesis tests were performed using the methodology presented by [165, 166], and already explained in the previous chapter.

5.1.2 Results and discussion

The performance of the trained networks (both by applying and without applying the semi-supervised learning methods) was evaluated considering an independent test set of 25 images. Performance was first assessed using the BLDO labels to compare the results with those in [216], where several classification networks (namely, AlexNet, GoogleNet, VGG16 and VGG19) were implemented to construct probability maps from image patches generated using a sliding window. Then, the best models were trained and tested using the OSO labels to show the influence of manual annotation on the segmentation results. Finally, the inference time of the different architectures was analysed.

5.1.2.1 Evaluation of the semi-supervised learning methods

All but two deep segmentation networks trained without semi-supervised learning methods, see Table 5.2, outperformed the approach presented in [216].

Namely, the total MSA (average of the five MSA_c values) of the best segmentation model improved by more than 15%, and the bunch MSA more than 5%. It is worth mentioning that the approach presented in [216] was aimed to help only the segmentation of the bunch class. For this reason, the improvement in the segmentation of the bunch class was lower than the one of the other classes, which was much more considerable.

Network	Background	Leaves	Pole	Bunch	Wood	Total
AlexNet	0.7691	0.7432	0.5486	0.7480	0.6677	0.6952
GoogleNet	0.6961	0.7280	0.5155	0.7441	0.5909	0.6549
VGG16	0.7671	0.7482	0.7646	0.7373	0.4713	0.6977
VGG19	0.8053	0.6899	0.7605	0.8058	0.3697	0.6863
Bisenet-ResNet18	0.8626	0.7327	0.8243	0.8199	0.8275	0.8160
Bisenet-ResNet34	0.8346	0.7755	0.8380	0.8354	0.8401	0.8293
CgNet	0.8480	0.7346	0.8203	0.8210	0.8234	0.8133
ContextNet	0.8118	0.7585	0.8248	0.8221	0.8260	0.8142
DeepLabV3+-EfficientnetB3	0.8292	0.7904	0.8415	0.8375	0.8446	0.8331
DeepLabV3+-ResNext50	0.8523	0.8067	0.8531	0.8509	0.8585	**0.8478**
DeepLabV3+-ResNet50	0.7959	0.6830	0.7825	0.7802	0.7854	0.7713
DenseApp	0.8721	0.6729	0.7919	0.7884	0.7972	0.7859
FPENet	0.1505	0.3910	0.2887	0.2828	0.2863	0.2849
HRNet	**0.9042**	0.7393	0.8337	0.8324	0.8393	0.8308
LedNet	0.8289	0.6993	0.8035	0.8013	0.8028	0.7920
Manet-EfficientnetB3	0.8463	0.8019	0.8536	**0.8569**	0.8545	0.8469
Manet-Resnest50	0.8448	0.7974	0.8501	0.8467	**0.8602**	0.8442
Manet-Resnet50	0.8107	0.7898	0.8485	0.8451	0.8517	0.8363
OCNet	0.8267	0.7669	0.8265	0.8259	0.8345	0.8206
Pan-EfficientnetB3	0.8133	0.7645	0.8228	0.8211	0.8251	0.8144
Pan-Resnet50	0.7995	**0.8209**	0.8445	0.8404	0.8472	0.8358
Unet-EfficientnetB3	0.8494	0.7093	0.8227	0.8204	0.8262	0.8109
Unet-Resnest50	0.8693	0.6602	0.8018	0.8004	0.8064	0.7915
Unet-Resnet50	0.8791	0.7320	0.8268	0.8229	0.8376	0.8220
Unet++-EfficientnetB3	0.8718	0.7672	0.8383	0.8385	0.8459	0.8349
Unet++-ResNest50	0.2076	0.7158	0.5068	0.5061	0.5085	0.5003
Unet++-ResNet50	0.8540	0.7892	**0.8557**	0.8460	0.8522	0.8435

Table 5.2: Mean segmentation accuracy (percentage) computed on test images of the deep learning models trained by the manually labeled dataset. In bold face the best result for each class and dataset.

If we compare the segmentation networks, there were four networks (Deep LabV3+-ResNext50, Manet-EfficientnetB3, Manet-Resnest50 and Unet++-Res Net50) with a total MSA of over 84%. Among them, the DeepLabV3+-ResNext50 showed better segmentation accuracy than the other networks. With the focus on the bunch class, the DeepLabV3+-ResNext50 and the Manet-EfficientnetB3 networks shined before the others achieving an MSA over 85% for that class. The Pan-Resnet50 model produced the best segmentation of the leaves, while the Unet++-ResNet50 model outperformed the others for the pole class and the Manet-Resnest50 model for the wood class. This illustrates the importance of testing different architectures since they focus on various aspects of the images.

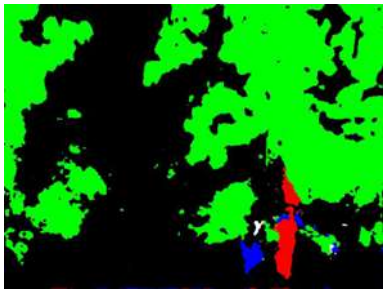
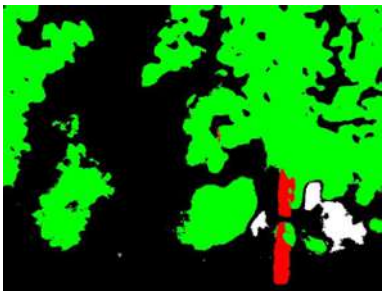
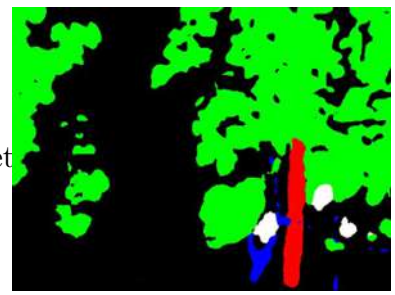
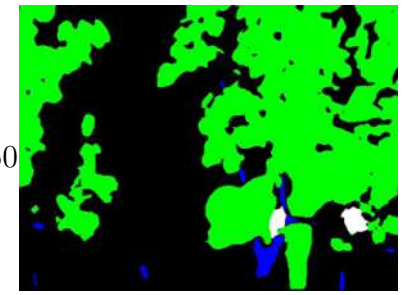
Input Image		Ground Truth	
Pan-Resnet50 (Leaves)		Unet++-ResNet50 (Pole)	
Manet-Efficientnet (Bunch)		Manet-Resnest50 (Wood)	

Table 5.3: Example of the segmentation results using the best model for each class.

Therefore, different models can be employed with different aims. For instance, if the final objective is measuring the production of grape bunches, DeepLabV3+-ResNext50 or Manet-EfficientnetB3 models should be used since they provided the best accuracy for the bunch class. In contrast, if this segmentation aims at trimming, Manet-Resnest50 model should be used since it offered the best accuracy for the wood class.

In addition to the raw numbers, several conclusions can be drawn by observing the segmentations of the best model for each class in Table 5.3. For the same image, even if all the models achieved a mean bunch segmentation accuracy of over 80%, only the Manet-EfficientnetB3 model could detect three of the four grape bunches. In addition, some leaves partially occluded the last bunch, making segmentation difficult since that region was segmented as either background or leaves by all the models.

The impact of the different semi-supervised learning methods for the studied

networks is provided in Table 5.4. At the same time, Table 5.5 shows the effects of applying these approaches on the segmentation mask output of the DeepLabV3+-ResNext50 model, which produced the best total MSA with plain training. From Table 5.5, it can be noticed that the segmentations made by using the semi-supervised learning methods were more precise than those produced by the original models. This happens because the semi-supervised methods helped to smooth the predictions. It is also worth mentioning that training using semi-supervised learning methods could help detect objects, like grape bunches in the pseudolabeling approach of Table 5.5, that were not previously seen by the models trained only with the manually annotated data.

Model	Plain training	PseudoLabeling	Distillation	Model Distillation
Bisnet-ResNet18	0.8160	0.8391	0.8400	0.8160
Bisnet-ResNet34	0.8293	0.8410	0.8378	0.8363
CgNet	0.8133	0.8290	0.8354	0.8314
ContextNet	0.8142	0.7859	0.8344	0.8324
DeepLabV3+-EfficientnetB3	0.8331	0.8482	0.8496	0.8482
DeepLabV3+-ResNext50	**0.8478**	0.8545	0.8545	**0.8586**
DeepLabV3+-ResNet50	0.7713	0.8554	0.8549	0.8549
DenseApp	0.7859	0.8364	0.8347	0.8289
FPENet	0.2849	0.8352	0.8368	0.8328
HRNet	0.8308	0.8489	0.8507	0.8519
LedNet	0.7920	0.8370	0.8472	0.8465
Manet-EfficientnetB3	0.8469	0.8554	**0.8554**	0.8554
Manet-Resnest50	0.8442	0.8375	0.8457	0.8375
Manet-Resnet50	0.8363	0.8343	0.8285	0.8343
OCNet	0.8206	0.8305	0.8246	0.8243
Pan-EfficientnetB3	0.8358	0.8539	0.8339	0.8342
Pan-Resnet50	0.8144	0.8197	0.8357	0.8362
UNet-EfficientnetB3	0.8109	0.8369	0.8487	0.8369
UNet-Resnest50	0.7915	0.8537	0.8537	0.8537
UNet-Resnet50	0.8220	0.8506	0.8506	0.8506
Unet++-EfficientnetB3	0.8349	0.8212	0.8445	0.8445
Unet++-ResNest50	0.5003	0.8526	0.8526	0.8526
Unet++-ResNet50	0.8435	**0.8566**	0.8537	0.8566

Table 5.4: Total mean segmentation accuracy (percentage) from applying the different semi-supervised learning procedures to label the testing images. In bold face the best result for each metric and dataset.

With more details, the PseudoLabeling approach produced a mean improvement of 5.62% (with a standard deviation of 13.04%). Only four networks got worse results using this training approach while, in some cases, namely for the FPENet model in Table 5.6, the improvement was over 55%. In Table 5.6, grape bunches and other objects that were not segmented with the initial FPENet model

were correctly detected using the FPENet version trained with the PseudoLabeling approach. Similarly, the distillation method produced a mean improvement of 6.01% (with a standard deviation of 12.91%), with only two networks having worse results. Finally, the model distillation method also considerably improved the performance of the models (a mean of 5.80% with a standard deviation of 12.90%). However, this improvement was slightly lower than the distillation approach.

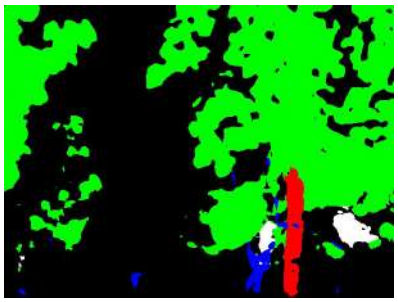
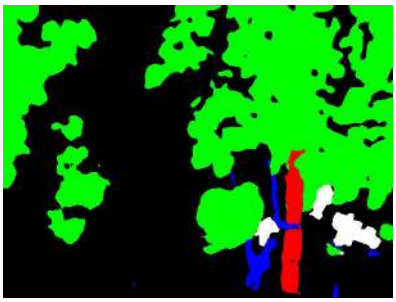
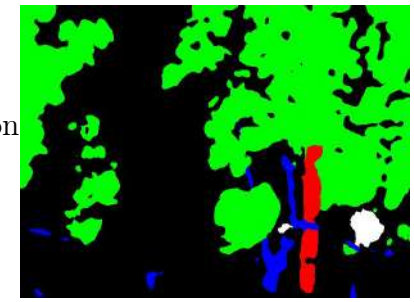
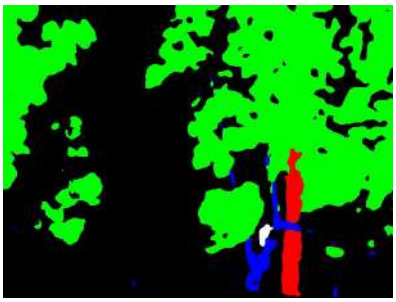
Input Image		Ground Truth	
Plain		Pseudo Labeling	
Distillation		Model Distillation	

Table 5.5: Example of the segmentation results using DeepLabV3+-ResNext50 with the four training strategies.

As stated before, a statistical analysis was performed to determine significant differences among the training procedures. Since the normality condition was not fulfilled (Shapiro–Wilk's test $W = 0.313172$; $p = 0.000000$), Friedman's nonparametric test was employed to compare the training procedures. Friedman's test performed a ranking of the training procedures under comparison (see Table 5.7), assuming as null hypothesis that all the models have the same performance. In this

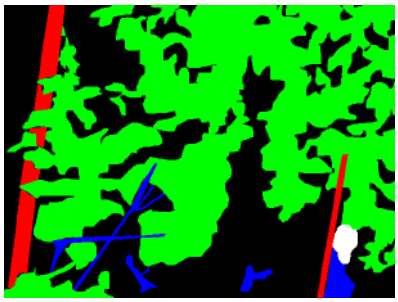
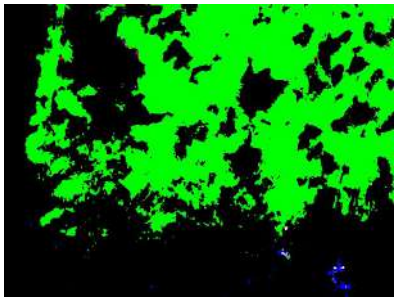
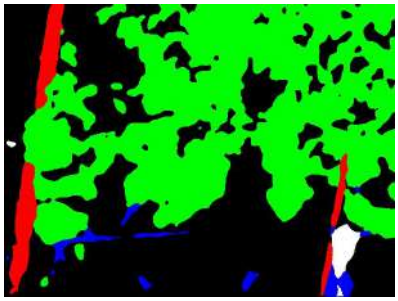
Input Image		Ground Truth	
FPENet Plain		FPENet Pseudo Label-ing	

Table 5.6: Example of the segmentation results using DeepLabV3+-ResNext50 with the four training strategies.

case, significant differences arised ($F = 15.66$; $p < 8.4\text{-}8e$) with a large size effect eta squared 0.13. The distillation method produced the best models. Moreover, looking at the standard deviation values of Table 5.7, the performance variability produced by the distillation approach is considerably reduced compared with plain training. Consequently, models can be trained more efficiently but can lead to poor results if only manually annotated data is used.

Training Technique	Mean Total MSA (std)	Friedman's Test Avg. Ranking
Plain training	0.7837 (0.1263)	1.2246
PseudoLabelling	0.8397 (0.0158)	2.7518
Distillation	0.8436 (0.0091)	3.2808
Model Distillation	0.8415 (0.0114)	2.7427

Table 5.7: Friedman's test for the mean Total MSA of the training methods.

Table 5.8 shows the results of the application of the Holm algorithm to compare the control training procedure (winner, based on distillation) with all the other training approaches, adjusting the p-value. Results proved significant differences between the semi-supervised learning procedures and the plain training approach, while all the semi-supervised learning methods produced the same outcomes. The

size effect was also taken into account using Cohen's d, and, as shown in Table 5.8, it is medium or large when the winning approach was compared with the rest of the models.

Training Technique	Z Value	p-value	Adjusted p-value	Cohen's d
PseudoLabelling	0.992945	0.320737	0.625041	0.2966
Model Distillation	1.00995	0.312521	0.625041	0.2011
Plain training	3.85956	0.000113	0.000340	0.6570

Table 5.8: Adjusted p-values with Holm and Cohen's d. Control technique: Distillation.

In summary, semi-supervised learning methods provided a considerable boost to all segmentation models without requiring the annotation of additional images. Providing precise annotations was a time-consuming task, and, therefore, reducing the annotation load could help the adoption of deep learning methods. However, deep learning models can only learn what is provided in the annotations. For segmentation tasks in agriculture, several small objects that are far from the objective are annotated as background, making unfeasible their automatic segmentation, even applying semi-supervised learning methods. This could be solved by a more fine-grained annotation, implementing object-segmentation-oriented (OSO) labels, as show in the next section.

5.1.2.2 Evaluations with OSO labels

As described in Section 5.1.1.1, a different annotation scheme was followed to produce more refined labels suitable for object segmentation models (OSO labels). These labels were used to train the same segmentation models of Table 5.1, following the plain training approach. The new results of the different architectures trained with the OSO labels are shown in Table 5.9.

Several models achieved a total MSA of over 85%, including DeepLabV3+-ResNet50, Pan-Resnet50, HRNet and all the versions of the Unet and Unet++ architectures. The best overall model was HRNet, with a total MSA of 85.91%. This model also obtained the best accuracy for the leaves, pole and bunch classes. In contrast, the best models for segmenting wood and the background were based on the Unet++ architecture. The outstanding results of the HRNet model were due to the design of its architecture, which aggregated the output representations at four different resolutions, thus allowing models to provide a precise segmentation of objects with different scales.

Segmentation maps in Table 5.10 help draw additional conclusions about the models trained with the BLDO and OSO labels. For the same image, the best over-

all model trained with the BLDO labels (DeepLabV3+-ResNext50 using Model Distillation) and the best model trained with the OSO labels (HRNet using plain training) could both segment grape bunches and leaves. However, the segmentation of smaller objects, such as small wooden fragments, was much better when models were trained with OSO labels. In contrast, BLDO labels were not accurate enough to train a model able to recognise of such small objects, even using any semi-supervised learning approach.

Network	Background	Leaves	Pole	Bunch	Wood	Total
Bisenet-ResNet18	0.7445	0.7124	0.7937	0.7947	0.8192	0.7832
Bisenet-ResNet34	0.8201	0.7827	0.8372	0.8366	0.8611	0.8333
CgNet	0.8301	0.7746	0.8376	0.8387	0.8547	0.8326
ContextNet	0.7805	0.7855	0.8297	0.8320	0.8521	0.8238
DeepLabV3+-EfficientnetB3	0.8735	0.7743	0.8452	0.8448	0.8585	0.8421
DeepLabV3+-ResNext50	0.8411	0.8110	0.8525	0.8513	0.8665	0.8485
DeepLabV3+-ResNet50	0.8666	0.7947	0.8531	0.8531	0.8665	0.8502
DenseApp	0.8153	0.7547	0.8205	0.8223	0.8439	0.8170
FPENet	0.6656	0.6972	0.7427	0.7434	0.7655	0.7329
HRNet	0.8541	**0.8256**	**0.8621**	**0.8631**	0.8725	**0.8591**
LedNet	0.7847	0.7684	0.8289	0.8255	0.8502	0.8197
Manet-EfficientnetB3	0.8463	0.8019	0.8536	0.8569	0.8545	0.8469
Manet-Resnest50	0.8033	0.5519	0.7154	0.7107	0.7288	0.7059
Manet-Resnet50	0.8294	0.7684	0.8403	0.8391	0.8595	0.8335
OCNet	0.8296	0.7312	0.8149	0.8155	0.8402	0.8115
Pan-EfficientnetB3	0.8221	0.8140	0.8503	0.8512	0.8630	0.8453
Pan-Resnet50	0.8602	0.7973	0.8539	0.8556	0.8682	0.8509
Unet-EfficientnetB3	0.8689	0.7938	0.8539	0.8551	0.8678	0.8512
Unet-Resnest50	0.8806	0.7972	0.8575	0.8582	0.8714	0.8556
Unet-Resnet50	0.8690	0.8031	0.8557	0.8567	0.8678	0.8535
Unet++-EfficientnetB3	0.8729	0.7960	0.8571	0.8581	**0.8730**	0.8549
Unet++-ResNest50	**0.8828**	0.7999	0.8603	0.8603	0.8725	0.8578
Unet++-ResNet50	0.8727	0.8088	0.8584	0.8581	0.8673	0.8556

Table 5.9: Mean segmentation accuracy (percentage) computed on test images of the deep learning models trained on the improved version of the OSO dataset. In bold face the best result for each metric and dataset.

Therefore, whether it is better to produce a dataset with a coarse annotation that is later combined with semi-supervised learning methods, or a dataset with a fine-grained annotation depend on the final aim of the trained models. Production monitoring or vegetation indices estimation require the segmentation of the main objects of the images (bunches and leaves), achievable even with coarse datasets carrying information about their appearance. However, tasks like trimming or robot harvesting require more precise segmentation to interact with the

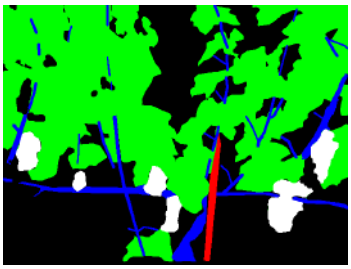
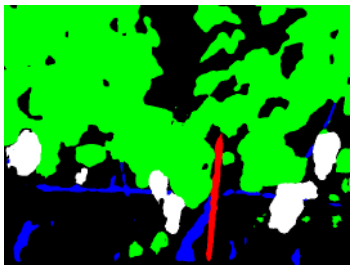
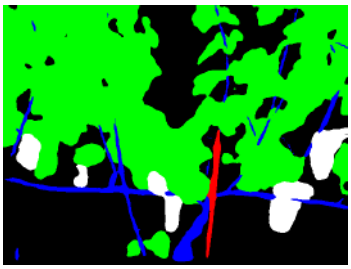
Original Image		OSO labels	
Prediction DeepLabV3+-ResNext50 using Model Distillation with BLDO labels		Prediction HRNet using plain training with OSO labels	

Table 5.10: Comparison of the results obtained with the best model trained with the BLDO labels (DeepLabV3+-ResNext50 using Model Distillation) and the best model (HRNet) trained with refined OSO labels.

environment appropriately. Here, it was mandatory to invest more time and effort in producing a fine-grained annotation of the images.

5.1.2.3 Time inference performance

This comparative study ends with the analysis of the inference time of the models since producing segmentation in a reasonable time is as crucial as obtaining precise results. This will enable their actual implementation for accurate yield monitoring and robot harvesting in almost real-time. The inference times of each model using an Nvidia RTX 2080 Ti GPU and an Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz are shown in Figure 5.4. It is worth noticing that the inference time was independent of the training method or the dataset used to construct the models as it only depends on the selected architecture. The DeepLabV3+-ResNext50 model, which obtained the best accuracy with BLDO labels, could process 100 images in 26.1 ms using a GPU and 315 with a CPU; whereas the HRNet model, which obtained the best accuracy with the OSO labels, processed 100 images in 26.3 ms using a GPU and 118 ms with a CPU. The best model at inference time was the ContextNet model, which segmented 100 images in 11.6 ms using a GPU and 68.9 ms using a CPU. This model also provided the best trade-off between accuracy and inference time. Therefore, ContextNet would be the preferred model to be implemented in-field for real-time processing.

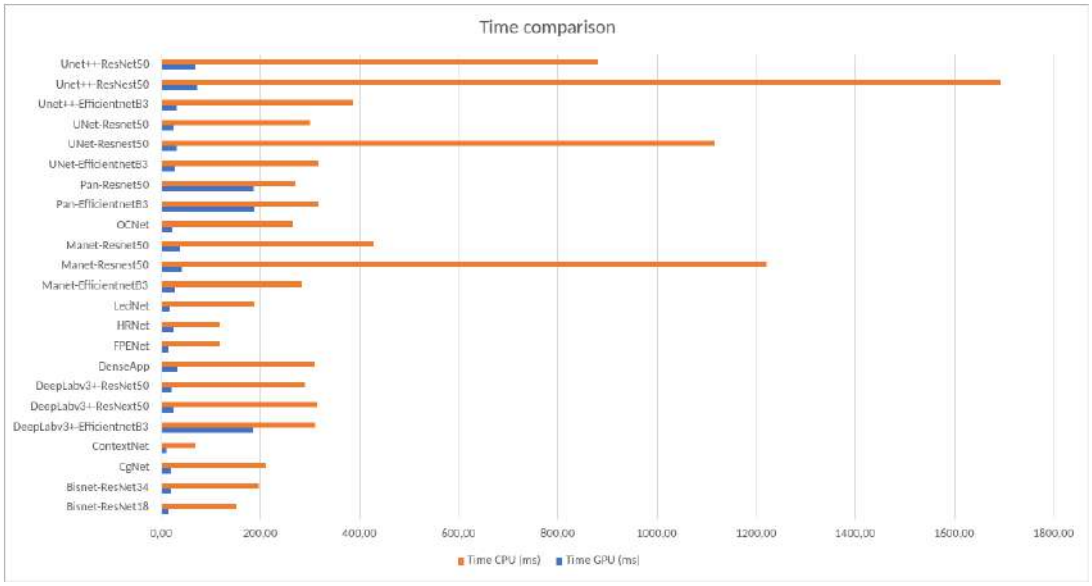


Figure 5.4: Inference time (in milliseconds) for 100 images of each segmentation model using both a CPU and a GPU.

As a conclusion, we have shown that the application of semi-supervised learning methods produce more accurate models than models only trained with manually labelled images. Moreover, we have proven that coarse labels can be efficiently used to model objects of large sizes and more detailed labels can be used to create fine-grained segmentation models. In the next section, we will prove that our approach generalises when applied to different datasets.

5.2 Generalization of the deep learning models

As we have seen in the previous section, the exact inference of productivity traits needs proper software solutions falling in semantic segmentation. One of the main problems in the related literature is the lack of generalisation of the segmentation models to new scenarios. Typical approaches consider single datasets for both training and testing the methodologies. This clearly induces a positive bias to the performance. Moreover, datasets often focus on specific maturation degrees, for instance, collected before pruning or harvesting [235, 236].

This part of the chapter tackles these two problems to demonstrate the that generalises to new scenarios the approach proposed in the previous section with the best performing models.

5.2.1 Materials and methods

In this section, we present both the viticulture and computational materials and methods employed for building semantic segmentation models and test their generalisation.

5.2.1.1 Dataset

The proposed experiments considered three datasets of natural images captured in different vineyards.

(a) (b) (c)

Figure 5.5: Sample images and corresponding ground truth. Images were taken in (a) July 2021, (b) September 2021, and (c) October 2021. In the ground truths, black, green and white segments refer to background, leaves and grape bunches, respectively.

A low-cost RGB-D sensor, the Intel RealSense D435 (Santa Clara, CA, USA), mounted onboard a moving vehicle, acquired the first D_{RS} dataset. The experimental field was in San Donaci (Italy), and the grape variety was Vitis vinifera, cultivar "Negroamaro" (red grape variety). The camera was tilted by 90° to have data in portrait mode. The plants were acquired frontally, at a distance between 0.8 and 1 m, as the vehicle flowed through the rows of vines. The D_{RS} dataset consisted of three sets of images, captured at three stages of the seasonal grapevine phenological development, i.e., at fruit set (July 16th, 2021), before harvesting (September 10th, 2021), and at the beginning of leaf fall (October 22nd, 2021). This choice guarantees good variability among the images of the dataset of both grape color and shape (from July to September) and leaves color (from September to October). The total amount of color images (1280×720 pixel resolution) equals to 29464. Since manual annotation is time demanding and labor intensive, only 265 images were manually labeled (98 in July, 98 in September, and 69 in

October), specifically, the datasets consists of 212 images for training and 54 for testing.

Figure 5.6: Color images of the D_{AK} dataset. Acquisitions were taken on *Top-Left.* May 16th, *Top-Right.* June 23rd, *Bottom-Left.* August 2nd, and *Bottom-Right.* October 3rd, 2022.

Manual annotation was performed to create a segmentation ground truth of three classes: the canopy (high vegetation other than the trunk), the grapes (grape bunches), and background (the remaining pixels). Three image samples taken at different times are shown in Figure 5.5. It is worth noting that the RGB-D camera returned more data, including infrared images and depth maps. However, only color information was used at this moment.

The Microsoft Azure Kinect (Redmond, WA, USA) camera captured the images of the second dataset considered (D_{AK}). In this case, images were acquired in a commercial field in Andria (Italy), devoted to a red wine grape variety (Vitis

vinifera, cultivar "Nero di Troia"). The camera was manually handled and acquisitions were captured holding still the setup on a tripod. In this case, the images were taken frontally to the plants between 0.8 and 1.2 m, keeping the camera horizontal. Six experimental campaigns were repeated between May 2022 and October 2022, producing 92 color images of 2048 × 1536 pixel resolution. These images captured 210 grape bunches (an average of 2.28 grape bunches per image).

(a) (b)

Figure 5.7: Color images from the dataset D_{SPh}. (a) Bombino red grave variety and (b) Nero di Troia red grape variety.

Four sample images taken in May, June, August, and October 2022 are shown in Figure 5.6.

The third dataset, D_{SPh}, is made of images captured by a Xiaomi Redmi Note 9 Pro smartphone (Beijing, China) in two commercial fields sited in Ruvo di Puglia (Italy) with vine plantations of two red grape varieties (Vitis vinifera, cultivars "Bombino" and "Nero di Troia"). The smartphone was handheld to capture images in portrait mode at a distance between 0.8 and 1.2 m from the plants, with a resolution of 3472 × 4640 pixels. The dataset includes 32 photos of the Bombino red grape variety and 15 photos of the Nero di Troia red grape variety. Both sets of acquisitions were performed in September 2021, before harvesting. 114 grape

bunches were captured in the D_{SPh} dataset, corresponding to 2.43 bunches per image on average. Figure 5.7 shows two sample images of the dataset D_{SPh} of the two varieties of red wine grapes.

5.2.1.2 Computational methods

A comparison of models trained on a subset of 212 images of D_{RS} was proposed to determine which one outperforms the others, and to generalise its application to D_{AK} and D_{SPh}. The following encoder-decoder networks were considered (these are the four networks that got the best results in the previous section): UNet++ [228] with ResNet50 backbone, DeepLabv3+ [225] with ResNext50 backbone and two Multi-Attention Networks (MANet) with EfficientNetB3 and ResNest50 backbones. The architectures with their respective backbones were trained using the same approach presented in the previous section.

The performance of the four presented models was compared in a test phase. After training, the developed networks were used to make predictions on the 54 test images, which were not used during training, even if acquired under the same experimental conditions. For each $c-$th class of the three target classes, segmentation accuracy (SA_c) was computed by exploiting the annotated ground truths:

$$SA_c = \frac{TP_c}{n_{obs,c}}, \forall \text{ images} \in D_{RS}$$

where TP_c is the number of correct pixel labels of the $c-$th class and $n_{obs,c}$ is its population. The arithmetical mean was then used to obtain the mean segmentation accuracy (MSA) [216]. The results of the application of the trained model to the other two datasets (D_{AK} and D_{SPh}) were not discussed in terms of MSA since images were not annotated. On the contrary, the correct detection of grape bunches was assessed through recall, precision and F1-score.

5.2.2 Results and Discussion

As discussed above, the four considered models have been trained and tested on the color images of the D_{RS} annotated dataset. The results on the test are provided in Table 5.11, which shows the SA of the three target classes and the global MSA of the four trained models.

Network	$SA_{Background}$	SA_{Canopy}	SA_{Grapes}	MSA
Unet++ - ResNet50	0.9548	**0.7998**	0.9446	0.9364
DeepLabv3+ - ResNext50	0.9766	0.7857	**0.9510**	**0.9472**
Manet - EfficientNetB3	**0.9772**	0.7645	0.9501	0.9450
Manet - ResNest50	0.9358	0.7846	0.9315	0.9205

Table 5.11: SA of the three classes of the dataset D_{RS} and global MSA computed on test set. In bold face the best result for each metric and dataset.

The performance analysis in Table 5.11 revealed that all four network architectures could accurately segment the three classes of interest. In general, the canopy class showed lower values of SA, since the high temporal variability within the dataset D_{RS} induced significant differences in the canopy appearance. However, this problem did not affect the grape class, which was always detected with accuracy values higher than 93%, even if the grape appearance varied in both sizes and colors.

The comparison of the four trained models also showed that the DeepLabv3+ architecture with ResNext50 backbone showed the best MSA and SA_{Grapes}, with 94.72% and 95.10%, respectively. Since model generalization is aimed at grape bunch segmentation, the tests on the other two datasets (D_{AK} and D_{SPh}) were made with the best model working on the labeled dataset D_{RS}, i.e. with the DeepLabv3+–ResNext50 model.

The DeepLabv3+ architecture with the ResNext50 backbone trained with the D_{RS} dataset was then applied to the 92 color images of the D_{AK} dataset. Before processing, the 92 images were cropped to match the portrait proportion (9:16) of the images in D_{RS}. Starting from the predicted bunch segments in the images of the D_{AK} dataset, the model generalisation capability was assessed in terms of grape bunch detection. Within these lines, a detection is correct (true positive, TP) if the intersection of the predicted and expected bunch segment is not void. Misdetections (false negatives, FN) and wrong detections (false positives, FP) were then used to compute test metrics. As a result, the DeepLabv3+-ResNext50 scored 72.86%, 66.52% and 69.55% in recall, precision and F1 scores, experts consider these results to be either sufficiently good for the information they need.

Four examples of prediction on images acquired at different ripeness levels and light conditions are shown in Figure 5.8. Specifically, Figure 5.8(a) and (b) prove an excellent recall in segmenting white and red grapes, i.e. regardless of their ripeness level. On the other hand, Figure 5.8(c) and (d) show the source of errors that down the final scores. Concerning Figure 5.8(c), leaves turning brown became false positive predictions of grape bunches, which decreased the precision value. Moreover, challenging light conditions, such as the one in Figure 5.8(d), did not

allow the detection of dark grapes, with a consequent decrease in the recall.

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

Figure 5.8: Predictions on images taken at (a) grapevine development, (b) before harvesting, (c) at leaf fall and (d) with opposite light. Images have a superimposed semi-transparent mask: light green segments refer to the leaves and light red segments correspond to detected grape bunches.

The same DeepLabv3+-ResNext50 model was finally tested on the 47 images of the D_{SPh} dataset for grape bunch detection. In this case, the network scored 66.67%, 89.41% and 94.41% in recall, precision and F1-scores, respectively. Three sample predictions made on images of the two different grape varieties (Bombino and Nero di Troia) are shown in Figure 5.9. The recall in grape bunch detection was lower than that achieved in the D_{AK} dataset since the camera was handheld and its point-of-view had a large variability. This effect is evident in Figure 5.9(c), where grape bunches were not detected and included in the leaf class. On the

contrary, the precision in grape detection was higher than its counterpart of the D_{AK} dataset since images of D_{SPh} were taken at the same seasonal phenological development, before harvesting. For this reason, false positives were limited to 9 over 76 true positive predictions of grape bunches. It is worth underlining that the different grape varieties of D_{SPh} did not affect the final scores.

(a) (b) (c)

Figure 5.9: Sample predictions on images of the D_{SPh} dataset framing (a) the Bombino and (b) the Nero di Troia red grape variety. Subplot (c) shows an example of misdetection due to the altered camera point-of-view. Images have a semi-transparent mask where light green and light red refer to leaf and grape segments.

We have shown in this section that the approach followed to create segmen-

tation models generalises when applied to unseen conditions during training. In the next section, we will take advantage of additional information provided by the RGB-D cameras to further improve the performance of the models.

5.3 Taking Advantage of Depth Information

As we have explained previously, our datasets were captured with a RGB-D camera, but depth information was not initially considered.

In general, convolutional neural models can automatically segment crop elements based on their colour and texture attributes from RGB images [237], and depth information can reduce the uncertainty of the segmentation of objects having similar appearance information [238]. However, it is not clear what is the optimal way of fusing RGB and depth information. Several works suggest [239] that depth information can help the segmentation of classes of close depth, appearance and location. On the contrary, it is better to use only RGB information to recognize object classes containing high variability of their depth values [240].

In this part of the work, depth information has been incorporated into deep learning models to obtain more accurate segmentations in viticulture. In particular, using a dataset of images taken with an Intel Realsense D435 stereo camera, RGB-D images have been used to train four segmentation architectures (Unet++-ResNet50, DeepLabV3-ResNext, Manet-Efficient, and Manet-ResNeSt) and compared against models trained with only RGB images. The main drawback of RGB-D models is the necessity of using images captured with cameras able to acquire depth information, and such cameras are usually more expensive than RGB cameras. In order to deal with this issue, we have employed a Dense Prediction Transformer model [241] to generate the depth channel from RGB images. This allows the usage of RGB-D models with RGB images captured, for instance, with a mobile phone.

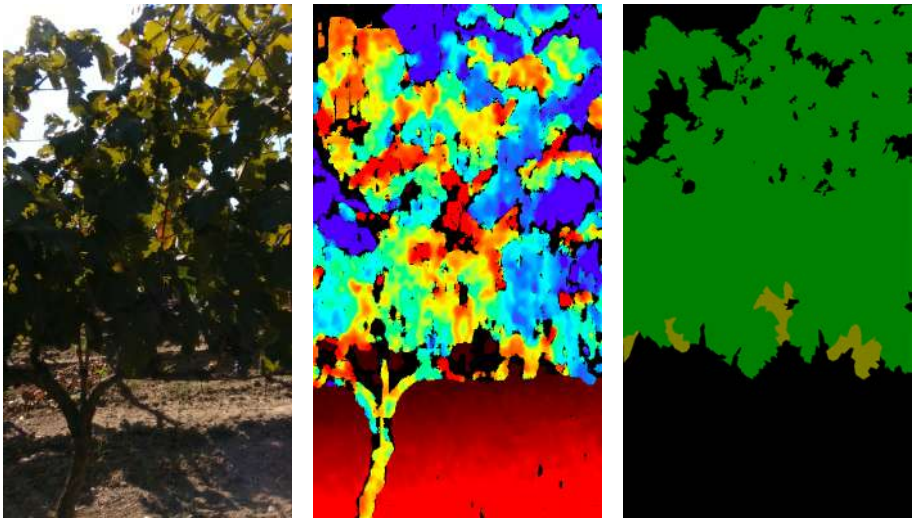
5.3.1 Materials and methods

In this section, we present both the dataset and computational materials and methods employed to take advantage of depth information.

5.3.1.1 Dataset

For this work, the employed dataset is the D_{RS} dataset that we have introduced in the previous section. The main difference is that we have also used the depth information of each image provided by the Intel Realsense D435 camera.

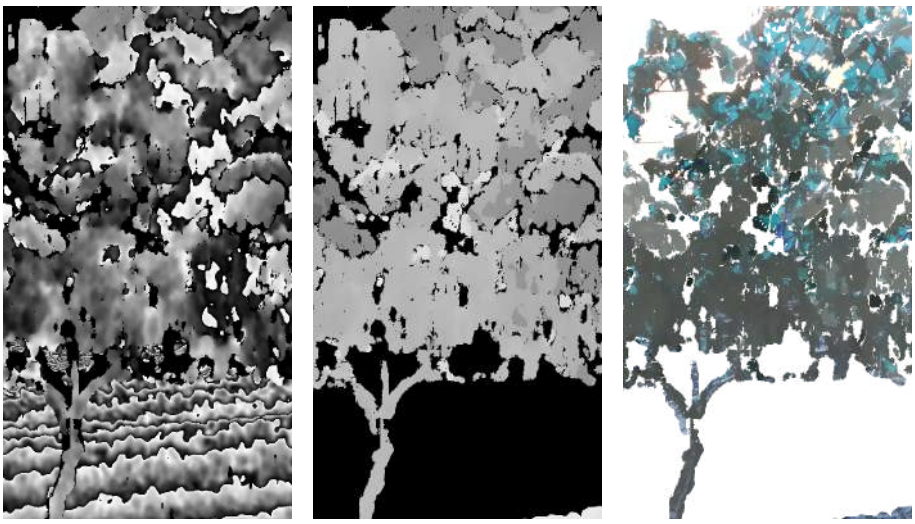
The dataset consists of 265 colour images in PNG format, see Figure 5.10(a). In addition, the Intel Realsense D435 camera provides the depth of each image in the RAW format, see Figure 5.10(b). Finally, the images were manually annotated to produce the masks with the regions corresponding to the grape bunches and canopy, see Figure 5.10(c). The dataset was divided into two subsets: the training set (212 images) and the test set (54 images).



(a) (b) (c)

Figure 5.10: (a) A sample colour image acquired by the Intel Realsense D435 camera; (b) The corresponding depth image; (c) The annotation of the image. Black pixels correspond with the background, yellow pixels with the regions of the grape bunches, and green pixels with the canopy regions.

There are three versions of the dataset: RGB, RGB-D, and RGB-D-generated. In the RGB version of the dataset, the images of both the training and test set are RGB images – that is, the depth information was discharged. In the RGB-D version, the information from RGB channels and depth channel of the images of both the training and test set was combined as follows. The RAW images captured with the Intel Realsense D435 camera provide information about the depth of objects that are located up to 65 metres away (see Figure 5.11(a)); however, plants are located less than 3 metres away; hence, the depth information related to objects farther than 3 metres away is removed from the image (see Figure 5.11(b)). Finally, such an image is combined with the RGB image obtaining an RGB-A image with four channels where one of them is the alpha channel, see Figure 5.11(c).



(a) (b) (c)

Figure 5.11: RGB-D transformation process of the image from Figure 2(a). (a) Depth image with information up to 65 metres; (b) Depth image with information up to 3 metres; (c) RGB-A image produced by combining the image from Figure 2(a) and the image from Figure 3(b).

Finally, in the third version of the dataset (called RGB-D-generated), the images of the training set were generated by using the aforementioned procedure. However, the images of the test set were generated as follows. From the RGB images of the test set, their depth information was computed by means of the Dense Prediction Transformer presented in [241]. Then, the RGB images and the automatically generated depth images were combined to obtain RGB-A images — as far we are a ware, this is the first time that this procedure has been applied in the agricultural settings. In Figure 5.12 an example of the images generated following this process is presented.

5.3.1.2 Computational methods

As in the previous section the Unet++ architecture with a ResNext50 backbone [228], the DeepLabV3 architecture with a ResNext50 backbone [225], and the Manet architecture with an EfficientNetB3 and a ResNest50 backbone have been employed to construct segmentation models with the 3 versions of the dataset (note that the models built with the RGB-D version of the dataset are the same than the models built with the RGB-D-generated version of the dataset). These architectures were chosen since they obtained the best results in the study presented in Section 6.1. The architectures with their respective backbones were

(a) (b) (c)

Figure 5.12: Generation of images of the test set from the RGB-D-generated dataset. (a) original image, (b) generated depth image obtained by using the transformer model, (c) RGB-A image obtained by combining images (a) and (b).

implemented and trained using the same procedure presented previously. In order to feed RGB-A images to these architectures, they were converted to RGB images using the Pillow library [242].

After training, all the models were then evaluated using the mean segmentation accuracy that we explained in the previous section.

5.3.2 Results and Discussion

The performance of the trained networks was first evaluated using the RGB version of the dataset, these results were presented in Table 5.12. If the segmentation networks are compared, DeepLabV3+-ResNext50 showed better overall segmentation accuracy than the other networks. The Unet+-ResNet50 model produced the best results for canopy segmentation with an accuracy of 79%, whereas the DeepLabV3+-ResNext50 model, with an accuracy of 94%, outperformed the others for segmenting objects of the grape class.

The results for the RGB-D version of the dataset are presented in Table 5.13. The RBG-D models improved between 2% and 4% the overall mean segmentation accuracy of their RGB counterparts. For this version of the dataset, the best model was built using the architecture Unet++ with a ResNet50 backbone; this model achieved a segmentation accuracy for the canopy of 82%, for grape bunches of 95%,

Network	$SA_{Background}$	SA_{Canopy}	SA_{Grapes}	MSA
Unet++-ResNet50	0.9548	**0.7998**	0.9446	0.9364
DeepLabV3+-ResNext50	0.9766	0.7857	**0.9510**	**0.9472**
Manet-EfficientnetB3	**0.9772**	0.7645	0.9501	0.9450
Manet-ResNest50	0.9358	0.7846	0.9315	0.9205

Table 5.12: Mean segmentation accuracy (percentage) computed on test images of the RGB dataset. In bold the best results.

and an overall mean segmentation accuracy of 95%. This shows the positive effect of adding the depth information to the RGB image, since adding such information allows the models to focus on the objects of interest, and also discard elements of the background that can be wrongly classified as either leaves or grape bunches.

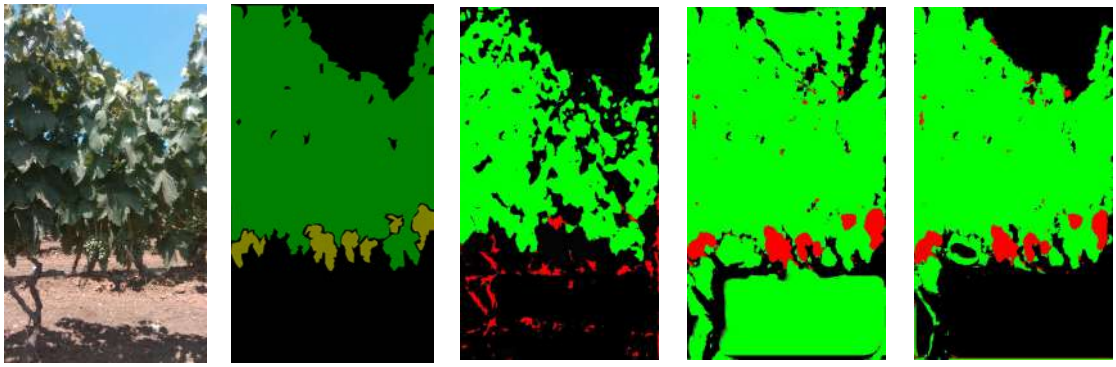
Network	$SA_{Background}$	SA_{Canopy}	SA_{Grapes}	MSA
Unet++-ResNet50	**0.9791**	**0.8191**	**0.9583**	**0.9547**
Deeplab-ResNext	0.9704	0.8115	0.9533	0.9482
Manet-Efficientnet	0.9553	0.6770	0.9550	0.9296
Manet-ResNest	0.9752	0.8151	0.9552	0.9512

Table 5.13: Mean segmentation accuracy (percentage) computed on test images of the RGB-D dataset. In bold the best results.

Finally, the results obtained with the models for the RGB-D-generated dataset are presented in Table 5.14. As previously mentioned, these results are obtained with the models trained on the RGB-D dataset but evaluated with images where depth information was automatically generated. Using this approach, the overall mean segmentation of all the models improved up to 0.47%. Again, the best results were achieved with the model built using the architecture Unet++ with a ResNet50 backbone. Such a model obtained a segmentation accuracy of 96% for grape bunches (an improvement of 0.26% regarding the best previous model), and 86% for canopy (an improvement of 4.6% regarding the best previous model). This improvement is due to the fact that depth images automatically generated provide a higher level of detail at close distances than depth images captured with the camera, see Figure 5.11(b) and Figure 5.12(b). Hence, in addition to removing regions that are not relevant to the models (as in the case of the images from the RGB-D dataset), images from the RGB-D-generated dataset preserve some information that was discarded in the RGB-D images.

Network	$SA_{Background}$	SA_{Canopy}	SA_{Grapes}	MSA
Unet++-ResNet50	**0.9754**	**0.8654**	**0.9609**	**0.9580**
Deeplab-ResNext	0.9681	0.8612	0.9574	0.9529
Manet-Efficientnet	0.9513	0.7234	0.9592	0.9331
Manet-ResNest	0.9729	0.8541	0.9585	0.9546

Table 5.14: Mean segmentation accuracy (percentage) computed on test images of the RGB-D-generated dataset. In bold the best results.



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

Figure 5.13: (a) Original Image; (b) Mask; (c) Prediction Unet++-ResNet50; (d) Prediction U++ ResNet50 with RGB-D; (e) Prediction U++ ResNet50 with RGB-D Generate.

In addition to the raw numbers, several conclusions can be draw from the segmentation of the best models for each class in Figure 5.13. As we can be seen in Figure 5.13(c), the best RGB segmentation model finds where the leaves are but misses many of them. For the grapes, such a model is not able to find them and gets confused with the pole. On the contrary, the RGB-D model, see Figure 5.13(d), knows where the grapes are and can differentiate the pole, but mixes the leaves with the background. Finally, when applied the RGB-D model to an image where depth is automatically generated, see Figure 5.13(e), the model is perfectly capable of detecting where are the leaves and grapes. As we have explained before, this happens because the generated depth image allows us to preserve some information that is removed when the depth from the camera is used.

As a conclusion of this section, we have shown the benefit of working with RGB-D images instead of only using RGB images. Moreover, we have seen that once models are trained, we can apply the models to RGB images by automatically

generating depth information using a deep learning model.

5.4 Conclusions

In this part of the memoir, we have developed models for the segmentation of different elements in a vineyard. Moreover, we have seen how the performance of those models can be improved thanks to the application of semi-supervised learning methods, and also using depth information both captured with a RGB-D camera and automatically generated using deep learning models. Furthermore, we have proven that those models generalise to unseen conditions.

The work presented in this chapter is a first step towards improving automatic monitoring of performance and production in vineyards. Thanks to our contributions it it possible to create accurate segmentation models that work with low-cost cameras and can be applied in-field to help farmers.

Conclusions and Further work

In this work, we have developed several methods and tools that facilitate the construction and usage of object detection models and improve their performance. Moreover, we have given the first steps towards generalising our methods to other Computer Vision tasks such as image classification or semantic segmentation. Finally, we have proven the benefits provided by our methods in actual problems coming from plant physiology and precision agriculture. From our point of view, the main contributions accomplished in this work are the following ones.

In Chapter 2, we presented a generic ensemble algorithm for object detection models that can be applied independently of the algorithm and library used for training those models. Our ensemble algorithm can be used not only to combine the output of several object detection models but also to apply test-time augmentation. Thanks to these methods it is possible to considerably improve the performance of object detection models. Moreover, our ensemble algorithm can be employed to reduce the burden of annotating datasets since it was the basis to define two new semi-supervised learning algorithms (called data-distillation and model distillation). Finally, all the methods presented in the Chapter 2 have been implemented in an open-source library called `EnsembleObjectDetection`.

The methods presented in Chapter 2 require object detection models that have been previously trained; however, this task is not straightforward for many users. Therefore, in Chapter 3, we designed and implemented a graphical application, called LabelDetection, that aims to facilitate the construction and use of object detection models, and also simplifies the application of advanced techniques such as data distillation or test time augmentation to users without programming experience. LabelDetection is designed to be used with multiple libraries and frameworks, and it can be easily extended to others. Specifically, LabelDetection allows for annotations to be made and modified. It also generates the necessary files for training models and enables the application of semi-supervised learning techniques, eliminating the need for a large annotated dataset. Finally, LabelDetection can use different pre-trained object detection models for making predictions, and user can modify those predictions.

The methods presented in the first two chapters of the memoir were focused on

object detection tasks; however, they can be generalised to tackle other Computer Vision problems such as image classification or semantic segmentation. A step towards such a generalisation was presented at the end of Chapter 3 where we designed four modules. The first module was devoted to simplify the use of deep learning models independently of the underlying library using models, the second module facilitates the training process, the third module serves to improve models by applying ensemble methods, and finally, the last modules reduce the burden of annotating images by applying semi-supervised learning techniques.

Our techniques and methods have been not only tested with standard academic datasets but also they have been applied to solve actual problems. Namely, in Chapter 4, the techniques and tools discussed in previous chapters have been employed to address two specific problems in plant physiology: stomata detection and Epidermal Bladder Cells (EBC) measurement. As a result, we have developed two open-source tools called LabelStoma and LabelGlandula. These tools have been utilized by researchers from Auburn University and the University of the Basque Country (coauthors of the tools) but also by teams in other regions such as French Guyane and Italy. LabelStoma has significantly improved the reliability of plant stomata analysis across different species and serves biologist to understand CO_2 and H_2O dynamics in plant-related processes like photosynthesis and transpiration. In the case of LabelGlandula, it enhances the accuracy of EBC analysis and enables plant biologists to advance their knowledge of the role played by EBC size, density, and volume in stress tolerance, such as salinity and drought. Moreover, it facilitates the exploration of EBC functionality and the molecular mechanisms behind EBC formation and salt accumulation.

Finally, in Chapter 5, we have developed Deep Learning models specifically tailored for the segmentation of different elements in a vineyard to this aim, the generalisation of the developed methods for object detection has played a key role. Namely, we have demonstrated how the performance of these models can be enhanced through the usage of semi-supervised learning methods and the incorporation of depth information. Furthermore, we have shown that these models can successfully generalise to previously unseen conditions. This study is a step towards improving the management of finite resources, performance optimization, and pest control; and, in general, sustainable agriculture.

As further work, we envision three main research lines. First of all, there are several technical improvements that can be incorporated into our libraries and tools. In the case of the `EnsembleObjectDetection` library, we would like to test whether techniques like Soft-NMS [58], NMW [59], fusion [60] or WBF [61] produce better results than the NMS algorithm currently employed by our ensemble algorithm. For LabelDetection, we plan to expand it with new algorithms and object detection libraries. Moreover, we want to fully develop the general library

presented at the end of Chapter 3, and thoroughly test it.

The second research line is devoted to facilitate the construction and use of Deep Learning models for different Computer Vision task. It is especially relevant to reduce the resources, both in terms of data and computation, that are required to train Deep Learning models. Among the existing approaches, techniques such as continual or active learning are promising pathways but they are usually focused on image classification tasks, and therefore, it is necessary to generalise those methods to other Computer Vision tasks. Moreover, foundational models such as Segment Anything [243] can be instrumental for our aim but they have to be easily adaptable to be usable in actual problems. Lastly, new techniques must be developed to deal with the domain shift problem.

Finally, the third research line that we envision is focused on tackling actual problems that can have a real impact on people's lives. Deep Learning methods have the potential to improve processes that require tedious and time consuming tasks in areas such as biology, agriculture or medicine. So, guided by specialists of those fields, we will tackle problems by means of the methods that we will develop and, in this way, we will help specialists in their daily lives.

Conclusiones y Trabajo Futuro

En este trabajo, hemos desarrollado varios métodos y herramientas que facilitan la construcción y el uso de modelos de detección de objetos y mejoran su rendimiento. Además, hemos dado los primeros pasos hacia la generalización de nuestros métodos a otras tareas de visión por computador, como la clasificación de imágenes o la segmentación semántica. Por último, hemos demostrado los beneficios proporcionados por nuestros métodos en problemas reales relacionados con la fisiología de las plantas y la agricultura de precisión. Desde nuestro punto de vista, las principales contribuciones logradas en este trabajo son las siguientes.

En el Capítulo 2, presentamos un algoritmo de ensemble genérico para modelos de detección de objetos que se puede aplicar independientemente del algoritmo y la biblioteca utilizados para entrenar esos modelos. Nuestro algoritmo de ensemble se puede utilizar no solo para combinar la salida de varios modelos de detección de objetos, sino también para aplicar aumento en tiempo de test. Gracias a estos métodos, es posible mejorar el rendimiento de los modelos de detección de objetos. Además, nuestro algoritmo de conjunto se puede utilizar para reducir la carga de anotar conjuntos de datos, ya que fue la base para definir dos nuevos algoritmos de aprendizaje semisupervisado (llamados destilación de datos y destilación de modelos). Por último, todos los métodos presentados en el Capítulo 2 se han implementado en una biblioteca de código abierto llamada `EnsembleObjectDetection`.

Los métodos presentados en el Capítulo 2 requieren modelos de detección de objetos que hayan sido previamente entrenados; sin embargo, esta tarea no es sencilla para muchos usuarios. Por lo tanto, en el Capítulo 3, diseñamos e implementamos una aplicación gráfica llamada `LabelDetection`, que tiene como objetivo facilitar la construcción y el uso de modelos de detección de objetos, y también simplificar la aplicación de técnicas avanzadas como la destilación de datos o el aumento en tiempo de test a usuarios sin experiencia en programación. LabelDetection está diseñado para ser utilizado con múltiples bibliotecas y marcos de trabajo, y se puede ampliar fácilmente a otros. Específicamente, LabelDetection permite realizar y modificar anotaciones. También genera los archivos necesarios para entrenar modelos y permite la aplicación de técnicas de aprendizaje semisupervisado, eliminando la necesidad de un gran conjunto de datos anotados. Por

último, LabelDetection puede utilizar diferentes modelos de detección de objetos preentrenados para hacer predicciones, y el usuario puede modificarlos.

Los métodos presentados en los dos primeros capítulos de la memoria se centraron en tareas de detección de objetos; sin embargo, se pueden generalizar para abordar otros problemas de visión por computador, como la clasificación de imágenes o la segmentación semántica. Se dio un paso hacia esa generalización al final del Capítulo 3, donde diseñamos cuatro módulos. El primer módulo se dedicó a simplificar el uso de modelos de aprendizaje profundo independientemente de la biblioteca subyacente que se utilice para los modelos, el segundo módulo facilita el proceso de entrenamiento, el tercer módulo sirve para mejorar los modelos mediante la aplicación de métodos de conjunto y, finalmente, los últimos módulos reducen la carga de anotar imágenes mediante la aplicación de técnicas de aprendizaje semisupervisado.

Nuestras técnicas y métodos no solo se han probado con conjuntos de datos académicos estándar, sino que también se han aplicado para resolver problemas reales. En el Capítulo 4, se emplearon las técnicas y herramientas discutidas en los capítulos anteriores para abordar dos problemas específicos en fisiología de las plantas: la detección de estomas y la medición de las Células de Vesícula Epidérmica (EBC). Como resultado, hemos desarrollado dos herramientas de código abierto llamadas LabelStoma y LabelGlandula. Estas herramientas han sido utilizadas por investigadores de la Universidad de Auburn y la Universidad del País Vasco (coautores de las herramientas), pero también por equipos en otras regiones como la Guayana Francesa e Italia. LabelStoma ha mejorado significativamente la confiabilidad del análisis de estomas en diferentes especies vegetales y ayuda a los biólogos a comprender la dinámica del CO_2 y H_2O en procesos relacionados con las plantas, como la fotosíntesis y la transpiración. En el caso de LabelGlandula, mejora la precisión del análisis de EBC y permite a los biólogos vegetales avanzar en su conocimiento del papel que desempeñan el tamaño, la densidad y el volumen de las EVC en la tolerancia al estrés, como la salinidad y la sequía. Además, facilita la exploración de la funcionalidad de las EBC y los mecanismos moleculares detrás de la formación de EBC y la acumulación de sal.

Finalmente, en el Capítulo 5, hemos desarrollado modelos de Aprendizaje Profundo específicamente diseñados para la segmentación de diferentes elementos en un viñedo. En este sentido, la generalización de los métodos desarrollados para la detección de objetos ha desempeñado un papel clave. Específicamente, hemos demostrado cómo se puede mejorar el rendimiento de estos modelos mediante el uso de métodos de aprendizaje semisupervisado y la incorporación de información de profundidad. Además, hemos demostrado con éxito que estos modelos pueden generalizarse bien a condiciones previamente no vistas. Este estudio ha dado lugar a una mejora en la gestión de recursos limitados, la optimización del rendimiento

y el control de plagas.

Como trabajo futuro, tenemos tres líneas principales de investigación en mente. En primer lugar, hay varias técnicas que se pueden incorporar a nuestras bibliotecas y herramientas. En el caso de la biblioteca `EnsembleObjectDetection`, nos gustaría probar si técnicas como Soft-NMS [58], NMW [59], fusion [60] o WBF [61] producen mejores resultados que el algoritmo NMS usado por nuestro método de ensemble. Para LabelDetection, planeamos ampliarlo con nuevos algoritmos y bibliotecas de detección de objetos. Además, queremos desarrollar completamente la biblioteca general presentada al final del Capítulo 3 y probarla a fondo.

La segunda línea de investigación está dedicada a facilitar la construcción y el uso de modelos de Aprendizaje Profundo para diferentes tareas de Visión por Computadora. Es especialmente relevante reducir los recursos, tanto en términos de datos como de cálculo, que se requieren para entrenar modelos de Aprendizaje Profundo. Entre los enfoques existentes, técnicas como el aprendizaje continuo o activo son caminos prometedores, pero generalmente se centran en tareas de clasificación de imágenes, por lo que es necesario generalizar esos métodos a otras tareas de Visión por Computador. Además, modelos fundamentales como Segment Anything [243] pueden ser necesarios para nuestro objetivo, pero deben ser fácilmente adaptables para ser utilizados en problemas reales. Por último, deben desarrollarse nuevas técnicas para hacer frente al problema del cambio de dominio.

Finalmente, la tercera línea de investigación que tenemos en mente se centra en abordar problemas reales que puedan tener un impacto real en la vida de las personas. Los métodos de Aprendizaje Profundo tienen el potencial de mejorar los procesos que requieren tareas tediosas y que consumen mucho tiempo en áreas como la biología, la agricultura o la medicina. Por lo tanto, guiados por especialistas en esos campos, abordaremos problemas mediante los métodos que desarrollaremos y de esta manera, ayudaremos a los expertos en su vida diaria.

Publications

In this chapter, the main contributions that make up the memoir are presented. In particular, for each of these contributions a small table with its most important data is presented.

Ensemble Methods for Object Detection	
Authors	Angela Casado-García and Jónathan Heras
Journal	Frontiers in Artificial Intelligence and Applications
Impact factor	0.25 (2022)
Rank	Q4 (Scimago)
Publisher	IOS Press
Volumen	325
Issue	-
Pages	2688-2695
Year	2020
Month	February
ISSN	0922-6389
DOI	10.3233/FAIA200407
State	Published
Cites	56 (Google Scholar)
Project webpage	`https://github.com/ancasag/ensembleObjectDetection`
Author's contribution	The PhD student was in charge of designing, developing and testing the generic method that serves to ensemble the output produced by detection algorithms. Also, she has read and approved the final manuscript.
Results presented in Chapter	Chapter 2

LabelDetection: simplifying the use and construction of deep detection models	
Authors	Angela Casado-García and Jónathan Heras
Journal	Lecture Notes in Artificial Intelligence
Impact factor	0.302 (2005)
Rank	Q4 (JCR)
Publisher	Springer
Volumen	12882
Issue	12
Pages	14-22
Year	2021
Month	September
ISSN	0302-9743
DOI	10.1007/978-3-030-85713-4_2
State	Published
Cites	47 (Google Scholar)
Project webpage	`https://github.com/ancasag/LabelDetection`
Project stats	60(`https://pypistats.org/packages/labeldetection`)
Author's contribution	The PhD student was in charge of designing, developing and testing the App. Also, she has read and approved the final manuscript
Results presented in Chapter	Chapter 3

LabelStoma: A Tool for Stomata Detection based on the YOLO algorithm	
Authors	Angela Casado-García, Arantza del Canto Romero, Alvaro Sanz-Saez, Usue Pérez-López, Amaia Bilbao-Kareaga, Felix B. Fritschi, Jon Miranda-Apodaca, Alberto Muñoz-Rueda, Anna Sillero-Martínez, Ander Yoldi-Achalandabaso, Maite Lacuesta, and Jónathan Heras
Journal	Computers and Electronics in Agriculture
Impact factor	6.757 (2021)
Rank	Q1 (JCR)
Publisher	Elsevier
Volumen	178
Issue	10
Pages	105751
Year	2020
Month	November
ISSN	0168-1699
DOI	10.1016/j.compag.2020.105751
Cites	19 (Google Scholar)
Project webpage	`https://github.com/ancasag/labelStoma`
Project stats	186(`https://pypistats.org/packages/labelstoma`)
Author's contribution	The PhD student was in charge of designing, developing and testing the App. Also, she has read and approved the final manuscript
Results presented in Chapter	Chapter 4

Deep Detection Models for Measuring Epidermal Bladder Cells	
Authors	Angela Casado-García, Aitor Agirresarobe, J. Miranda-Apodaca, Jónathan Heras and Usue Pérez-López
Journal	Lecture Notes in Computer Science
Impact factor	0.32 (2022)
Rank	Q2 (Scimago)
Publisher	Springer
Volumen	13256
Issue	44
Pages	131-142
Year	2022
Month	April
ISSN	0302-9743
DOI	10.1007/978-3-031-04881-4_11
State	Published
Cites	25 (Google Scholar)
Project webpage	`https://github.com/ancasag/LabelDetection`
Project stats	186(`https://pypistats.org/packages/labeldetection`)
Author's contribution	The PhD student was in charge of designing, developing and testing the App. Also, she has read and approved the final manuscript
Results presented in Chapter	Chapter 4

Semi-Supervised Deep Learning and Low-Cost Cameras for the Semantic Segmentation of Natural Images in Viticulture	
Authors	Angela Casado-García, Jónathan Heras, Annalisa Milella, and Roberto Marani
Journal	Precision Agriculture
Impact factor	5.767 (2021)
Rank	Q1 (JCR)
Publisher	Springer
Volumen	23
Issue	6
Pages	2001-2026
Year	2022
Month	June
ISSN	1385-2256
DOI	10.1007/s11119-022-09929-9
State	Published
Cites	4(Google Scholar)
Project webpage	`https://github.com/ispstiima/S3CavVineyardDataset`
Project stats	
Author's contribution	The PhD student was in charge of designing, developing and testing the generic method that serves to apply semi supervised techniques in sementic segmentation. Also, she has read and approved the final manuscript.
Results presented in Chapter	Chapter 5

Bibliography

[1] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005)*, volume 1, pages 886–893 vol. 1, 2005.

[3] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[4] J. C. Arellano-González, H. I. Medellín-Castillo, J. J. Cervantes-Sánchez, and M. A. García-Murillo. Assessment of computer vision methods for motion tracking of planar mechanisms. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 236(8):4093–4104, 2022.

[5] Y. Yu, C. Wang, Q. Fu, et al. Techniques and challenges of image segmentation: A review. *Electronics*, 12(5), 2023.

[6] M. Poggi and T. B. Moeslund. Computer vision for 3d perception and applications. *Sensors*, 21(12), 2021.

[7] E. Saraee, M. Jalal, and M. Betke. Visual complexity analysis using deep intermediate-layer features. *Computer Vision and Image Understanding*, 195:102949, 2020.

[8] J. Gao, Y. Yang, P. Lin, and D. S. Park. Computer vision in healthcare applications. *Journal of Healthcare Engineering*, 2018:5157020, 2018.

[9] E. Dilek and M. Dener. Computer vision applications in intelligent transportation systems: A survey. *Sensors*, 23(6), 2023.

[10] K. Okarma. Applications of computer vision in automation and robotics. *Applied Sciences*, 10(19), 2020.

[11] J. G. Shanahan and L. Dai. Introduction to computer vision and real time deep learning-based object detection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3523–3524, New York, NY, USA, 2020. Association for Computing Machinery.

[12] D. Bhatt, C. Patel, H. Talsania, et al. Cnn variants for computer vision: History, architecture, application, challenges and future scope. *Electronics*, 10(20), 2021.

[13] S. E. Corchs, G. Ciocca, E. Bricolo, and F. Gasparini. Predicting complexity perception of real world images. *PLOS ONE*, 11(6):1–22, 2016.

[14] A. Rosenfeld. Computer vision: basic principles. *Proceedings of the IEEE*, 76(8):863–868, 1988.

[15] K. Aggarwal, M. M. Mijwil, A.-H. Al-Mistarehi, et al. Has the future started? the current growth of artificial intelligence, machine learning, and deep learning. *Iraqi Journal For Computer Science and Mathematics*, 3(1):115–123, 2022.

[16] M. Pietikäinen. Image analysis with local binary patterns, 2005.

[17] T. Mita, T. Kaneko, and O. Hori. Joint haar-like features for face detection, 2005.

[18] O. Kramer. K-nearest neighbors, 2013.

[19] S. J. Rigatti. Random forest, 01 2017.

[20] F. Izaurieta and C. Saavedra. Redes neuronales artificiales. Available: http://www.uta.cl/charlas/volumen16/Indice/Ch-csaavedra.pdf.

[21] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[22] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET'2017)*, pages 1–6, 2017.

[23] R. Girshick, J. Donahue, T. Darrell, et al. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'14, pages 580–587. IEEE, 2014.

[24] R. Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[25] S. Ren, K. He, R. Girshick, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in Neural Information Processing Systems*, 28:91–99, 2015.

[26] W. Liu, D. Anguelov, D. Erhan, et al. Ssd: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision*, volume 9905 of *ECCV 2016*, pages 21–37, 2016.

[27] Ultralytics. Yolov8 - state-of-the-art object detection. `https://github.com/ultralytics/yolov5`, 2022. Accessed: June 20, 2023.

[28] N. Carion, F. Massa, G. Synnaeve, et al. End-to-end object detection with transformers. In *Computer Vision (ECCV'2020)*, pages 213–229, 2020.

[29] Y. Fang, B. Liao, X. Wang, et al. You only look at one sequence: Rethinking transformer in vision through object detection. In *Advances in Neural Information Processing Systems*, volume 34, pages 26183–26197, 2021.

[30] S. K. Pal, A. Pramanik, J. Maiti, and P. Mitra. Deep learning in multi-object detection and tracking: State of the art. *Applied Intelligence*, 51(9):6400–6429, 2021.

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

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. Wells, and A. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015*, Cham, Switzerland, 2015. Springer.

[33] N. Ibtehaz and M. S. Rahman. Multiresunet: Rethinking the u-net architecture for multimodal biomedical image segmentation. *Neural Networks*, 121:74–87, 2020.

[34] Y. Xing, L. Zhong, X. Zhong, and W. Wang. An encoder-decoder network based fcn architecture for semantic segmentation. *Wirel. Commun. Mob. Comput.*, 2020, 2020.

[35] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 843–852, 2017.

[36] Imagenet. http://www.image-net.org/. Accessed: June 20, 2023.

[37] T-Y. Lin, M. Maire, S. Belongie, et al. Microsoft COCO: Common Objects in Context. In *Proceedings of the European Conference on Computer Vision*, volume 8693 of *ECCV'14*, pages 740–755, 2014.

[38] M. Everingham, S. Eslami, L. Van Gool, et al. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015.

[39] J. Irvin, P. Rajpurkar, M. Ko, et al. Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 590–597, 2019.

[40] C. Tan, F. Sun, T. Kong, et al. A survey on deep transfer learning. In *Proceedings International conference on artificial neural networks (ICANN'2018)*, pages 270–279, 2018.

[41] C. Phillip, M. Hang, V. Nym, et al. A review of medical image data augmentation techniques for deep learning applications. *Australasian Radiology*, 65(5):545–563, August 2021.

[42] X. Zhu and A. B. Goldberg, editors. *Introduction to Semi-Supervised Learning*. Morgan & Claypool Publishers, 2009.

[43] C. Zhang and Y. Ma, editors. *Ensemble Machine Learning: Methods and Applications*. Springer, 2012.

[44] C. Peng, T. Xiao, Z. Li, et al. MegDet: A Large Mini-Batch Object Detector . In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'18, pages 6181–6189, 2018.

[45] J. Redmon, S. Divvala, R. Girshick, et al. You Only Look Once: Unified, Real-Time Object Detection . In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'16, pages 779–788, 2016.

[46] M. Shafiq and Z. Gu. Deep residual learning for image recognition: A survey. *Applied Sciences*, 12(18), 2022.

[47] J. Xu, W. Wang, H. Wang, and J. Guo. Multi-model ensemble with rich spatial information for object detection. *Pattern Recognition*, 99:107098, 2020.

[48] K. He, X. Zhang, S. Ren, et al. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'16, pages 770–778, 2016.

[49] I. Sirazitdinov, M. Kholiavchenko, T. Mustafaev, et al. Deep neural network ensemble for pneumonia localization from a large-scale chest x-ray database. *Computers and Electrical Engineering*, 78:388–399, 2019.

[50] N. Vo, K. Duy, T. V. Nguyen, et al. Ensemble of Deep Object Detectors for Page Object Detection. In *Proceedings of the International Conference on Ubiquitous Information Management and Communication (IMCOM'2018)*, pages 1–5, 2018.

[51] H. Rezatofighi, N. Tsoi, J. Gwak, et al. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'2019)*, 2019.

[52] J. Li, J. Qian, and Y. Zheng. Ensemble R-FCN for Object Detectio. In *Proceedings of the International Conference on Computer Science and its Applications (CSA'2017)*, volume 474, pages 400–406, 2017.

[53] Z. Cai and N. Vasconcelos. Cascade R-CNN: Delving Into High Quality Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'2018, pages 6154–6162, 2018.

[54] J. Guo and S. Gould. Deep CNN Ensemble with Data Augmentation for Object Detection. *CoRR*, abs/1506.07224, 2015.

[55] J. Huang, V. Rathod, Chen Sun, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'17, pages 3296–3305, 2017.

[56] J. Lee, S. Lee, and S-I. Yang. An Ensemble Method of CNN Models for Object Detection . In *Proceedings of the IEEE Conference on Information and Communication Technology Convergence (ICTC'2018)*, pages 898–901, 2018.

[57] J. Hendrik Hosang, R. Benenson, and B. Schiele. Learning non-maximum suppression. *CoRR*, abs/1705.02950, 2017.

[58] N. Bodla, B. Singh, R. Chellappa, et al. Soft-NMS: improving object detection with one line of code. In *Proceedings of the IEEE Conference on*

Computer Vision and Pattern Recognition, CVPR'2017, pages 5561–5569, 2017.

[59] H. Zhou, Z. Li, C. Ning, et al. CAD: Scale Invariant Framework for Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'17, pages 760–768, 2017.

[60] P. Wei, J. E. Ball, and D. T. Anderson. Fusion of an Ensemble of Augmented Image Detectors for Robust Object Detection. *Sensors*, 18(3):1–21, 2018.

[61] S. Roman and W. Wang. Weighted Boxes Fusion: ensembling boxes for object detection models. *CoRR*, abs/1910.13302, 2019.

[62] P. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the International Conference on Document Analysis and Recognition*, volume 2 of *ICDAR'03*, pages 958–964, 2003.

[63] P. Simard, B. Victorri, Y. LeCun, et al. Tangent prop – a formalism for specifying selected invariances in an adaptive network. In *Proceedings of the International Conference on Neural Information Processing Systems*, volume 4 of *NIPS'91*, pages 895–903, 1992.

[64] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR'2015)*, 2015.

[65] B. Zoph, E. D. Cubuk, G. Ghiasi, et al. Learning Data Augmentation Strategies for Object Detection. pages 566–583, 2020.

[66] D. Ballabio, R. Todeschini, and V. Consonni. Recent Advances in High-Level Fusion Methods to Classify Multiple Analytical Chemical Data. *Data Handling in Science and Technology*, 31:129–155, 2019.

[67] I. Radosavovic, P. Dollár, R. Girshick, et al. Data Distillation: Towards Omni-Supervised Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'18, pages 4119–4128, 2018.

[68] C. Bucila, R. Caruana, and A. Niculescu-Mizil. Model compression: making big, slow models practical. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, KDD'2006, pages 535–541, 2006.

[69] T. Huang, J. A. Noble, and A. I. L. Namburete. Omni-Supervised Learning: Scaling Up to Large Unlabelled Medical Datasets. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, MICCAI'17, pages 572–580, 2018.

[70] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[71] J. Minichino and J. Howse. *Learning OpenCV 3 Computer Vision with Python*. Packt Publishing, 2015.

[72] A. Casado-García, C. Domínguez, M. García-Domínguez, et al. CLoDSA: A Tool for Image Augmentation in Classification, Localization, Detection and Semantic Segmentation Tasks. *BMC in Bioinformatics*, 20(323), 2019.

[73] V. Sarcar. *Abstract Factory Patterns*, pages 109–114. Apress, Berkeley, CA, 2016.

[74] Joseph Redmon. Darknet: Open source neural networks in c, 2013. Accessed: June 20, 2023.

[75] T. Chen, M. Li, Y. Li, et al. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *CoRR*, abs/1512.01274, 2015.

[76] W. Abdulla. Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. `https://github.com/matterport/Mask_RCNN`, 2017.

[77] T. Lin, P. Goyal, R. Girshick, et al. Keras retinanet. `https://github.com/fizyr/keras-retinanet`, 2017. Accessed: June 20, 2023.

[78] A. Casado-García, J. Heras, and A. Sanz-Saez. Google colaboratory for quantifying stomata in images. In *Proceedings of the International Conference on Computer Aided Systems Theory (EUROCAST'2019)*, volume 12014, pages 231–238. Lecture Notes in Computer Science, 2020.

[79] B. Coüasnon and A. Lemaitre. *Handbook of Document Image Processing and Recognition*, chapter Recognition of Tables and Forms, pages 647–677. Springer International Publishing, 2014.

[80] M. C. Gobel, T. Hassan, E. Oro, et al. ICDAR2013 Table Competition. In *12th ICDAR Robust Reading Competition*, ICDAR'13, pages 1449–1453. IEEE, 2013.

[81] M. Li, L. Cui, S. Huang, et al. TableBank: Table Benchmark for Image-based Table Detection and Recognition. *CoRR*, abs/1903.01949:1918–1925, 2019.

[82] F. Chollet. Keras: Deep learning for humans. `https://github.com/fchollet/keras`, 2015.

[83] L. Gao, Y. Huang, H. Déjean, et al. Icdar 2019 competition on table detection and recognition (ctdar), 2019.

[84] S. S. Abbas Zaidi, M. Samar Ansari, A. Aslam, et al. A survey of modern deep learning based object detection models. *Digital Signal Processing*, 126:103514, 2022.

[85] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[86] C. Y. Wang, A. Bochkovskiy, H. Liao, and H.Y. Mark. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2023.

[87] Y. Wu, A. Kirillov, F. Massa, et al. Detectron2. `https://github.com/facebookresearch/detectron2`. Accessed: June 20, 2023.

[88] AI Fast Track. Icevision: An agnostic object detection framework. `https://github.com/airctic/icevision`, 2020. Accessed: June 20, 2023.

[89] F. Massa and R. Girshick. maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. `https://github.com/facebookresearch/maskrcnn-benchmark`, 2018.

[90] T. Chen, M. Li, Y. Li, et al. Mxnet object detection. `https://gluon-cv.mxnet.io/build/examples_detection/index.html`, 2019.

[91] K. Chen, J. Wang, J. Pang, et al. MMDetection: Open MMLab Detection Toolbox and Benchmark. *CoRR*, abs/1906.07155, 2019.

[92] Y. Chen, C. Han, Y. Li, et al. Simpledet: A simple and versatile distributed framework for object detection and instance recognition. *CoRR*, abs/1903.05831, 2019.

[93] Y. Wu. Tensorpack. `https://github.com/tensorpack/`, 2016. Accessed: June 20, 2023.

[94] D. Tzutalin. LabelImg. https://github.com/tzutalin/labelImg, 2015. Accessed: June 20, 2023.

[95] Intel. Computer vision annotation tool (cvat). `https://github.com/openvinotoolkit/cvat`, 2016. Accessed: June 20, 2023.

[96] A. Dutta. Vgg image annotator (via). `http://www.robots.ox.ac.uk/~vgg/software/via/`, 2016. Accessed: June 20, 2023.

[97] R. Sharma. Rectlabel. `https://rectlabel.com/`, 2017. Accessed: June 20, 2023.

[98] A. B. Alexey. YOLO mark, 2018.

[99] Microsoft. Visual object tagging tool (vott). `https://github.com/microsoft/VoTT`, 2018. Accessed: June 20, 2023.

[100] Labelbox. Labelbox. `https://labelbox.com/`. Accessed: June 20, 2023.

[101] Supervisely. Supervisely. `https://supervise.ly/`. Accessed: June 20, 2023.

[102] P. Metzger. Anno-mage. `https://github.com/metazet/anno-mage`, 2019. Accessed: March 7, 2023.

[103] CV-Toolkit. Cv-toolkit. `https://github.com/ingconti/cvtoolkit`. Accessed: June 20, 2023.

[104] T. Kluyver, B. Ragan-Kelley, F. Perez, et al. Jupyter notebooks — a publishing format for reproducible computational workflows. In *20th International Conference on Electronic Publishing*, pages 87–90. IOS Press, 2016.

[105] Colaboratory team. Google colaboratory, 2017.

[106] M. Tan, R. Pang, and Q. V. Le. EfficientDet: Scalable and Efficient Object Detection. *CoRR*, abs/1911.09070, 2019.

[107] C. Zhu, Y. He, and M. Savvides. Feature selective anchor-free module for single-shot object detection. *CoRR*, abs/1903.00621:840–849, 2019.

[108] Z. Tian, C. Shen, H. Chen, et al. Fcos: Fully convolutional one-stage object detection. pages 9627–9636, 2019.

[109] D. H. Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 07 2013.

[110] J. Anderson, J. McRee, and R. Wilson. *Effective UI: The Art of Building Great User Experience in Software*. O'Reilly Media, 2010.

[111] E. David, S. Madec, P. Sadeghi-Tehran, et al. Global Wheat Head Detection (GWHD) Dataset: A Large and Diverse Dataset of High-Resolution RGB-Labelled Images to Develop and Benchmark Wheat Head Detection Methods. *Plant Phenomics*, 2020, 2020.

[112] L. Ramalho. *Fluent Python: Clear, Concise, and Effective Programming*. O'Reilly Media, 2015.

[113] B. R. Buttery, C. S. Tan, R. I. Buzzell, et al. Stomatal numbers of soybean and response to water stress. *Plant and Soil*, 149(2):283–288, 1993.

[114] A. M. Hetherington and F. I. Woodward. The role of stomata in sensing and driving environmental change. *Nature*, 424(6951):901–908, 2003.

[115] J. Hughes, C. Hepworth, C. Dutton, et al. Reducing Stomatal Density in Barley Improves Drought Tolerance without Impacting on Yield. *Plant Physiology*, 174(2):776–787, 2017.

[116] C. T. Rueden, J. Schindelin, M. C. Hiner, et al. ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics*, 18:529, 2017.

[117] L. T. Bertolino, R. S. Caine, and J. E. Gray. Impact of stomatal density and morphology on water-use efficiency in a changing world. *Frontiers Plant Science*, 10(225), 2019.

[118] R.S. Caine, X. Yin, J. Sloan, et al. Rice with reduced stomatal density conserves water and has improved drought tolerance under future climate conditions. *New Phytologist Trust*, 221:371–384, 2019.

[119] C. Dutton, H. Hõrak, C. Hepworth, et al. Bacterial infection systemically suppresses stomatal density. *Plant, Cell & Environment*, 42(8):2411–2421, 2019.

[120] K. Omasa and M. Onoe. Measurement of Stomatal Aperture by Digital Image Processing . *Plant and Cell Physiology*, 25(8):1379–1388, 1984.

[121] G. Karabourniotis, D. Tzobanoglou, D. Nikolopoulos, et al. Epicuticular Phenolics Over Guard Cells: Exploitation for in situ Stomatal Counting by Fluorescence Microscopy and Combined Image Analysis. *Annals of Botany*, 87(5):631–639, 2001.

[122] P. Sanyal, U. Bhattacharya, and S. K. Bandyopadhyay. Analysis of SEM Images of Stomata of Different Tomato Cultivars Based on Morphological Features. In *Proceedings of the Asia International Conference on Modeling & Simulation*, AICMS'08, pages 890–894, 2008.

[123] M. Oliveira and N. da Silva. Automatic Counting of Stomata in Epidermis Microscopic Images. In *Proceedings of the Workshop de Visao Computacional (WVC'2014)*, pages 253–257, 2014.

[124] S. Liu, J. Tang, P. Petrie, et al. A Fast Method to Measure Stomatal Aperture by MSER on Smart Mobile Phone . In *Proceedings of the Conference on Imaging and Applied Optics (AIO'2016)*, page AIW2B.2, 2016.

[125] K. T. N. Duarte, M. A. G. de Carvalho, and P. S. Martins. Segmenting High-quality Digital Images of Stomata using the Wavelet Spot Detection and the Watershed Transform . In *Proceedings of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, VISAPP'17, pages 540–547, 2017.

[126] T. Higaki, N. Kutsuna, and S. Hasezawa. CARTA-based semi-automatic detection of stomatal regions on an Arabidopsis cotyledon surface. *Plant Morphology*, 26(1):9–12, 2014.

[127] H. Laga, F. Shahinnia, and D. Fleury. Image-based Plant Stomata Phenotyping . In *Proceedings of the International Conference on Control, Automation, Robotics & Vision*, ICARCV'14, pages 217–222, 2014.

[128] P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[129] S. Vialet-Chabrand and O. Brendel. Automatic measurement of stomatal density from microphotographs. *Trees*, 28(6):1859–1865, 2014.

[130] H. Jayakody, S. Liu, M. Whitty, et al. Microscope image based fully automated stomata detection and pore measurement method for grapevines. *Plant Methods*, 13(94):1–12, 2017.

[131] Y. Toda, S. Toh, G. Bourdais, et al. DeepStomata: Facial Recognition Technology for Automated Stomatal Aperture Measurement. *bioRXiv*, page 365098, 2018.

[132] A. H. Aono, J. S. Nagai, G. da S. M. Dickel, et al. A Stomata Classification and Detection System in Microscope Images of Maize Cultivars. *PLoS ONE*, 16(10), 2019.

[133] A. Meheus. Deep learning for stomata detection: from research to education. Master's thesis, Ghent University, Belgium, 2019.

[134] K. C. Fetter, S. Eberhardt, R. S. Barclay, et al. StomataCounter: a neural network for automatic stomata identification and counting. *New Phytologist*, 223(3):1671–1681, 2019.

[135] S. Bhugra, D. Mishra, A. Anupama, et al. Deep Convolutional Neural NetworksBased Framework for Estimationof Stomata Density and Structurefrom Microscopic Images. In *Proceedings of the European Conference on Computer Vision*, volume 11134 of *ECCV'2018*, pages 412–423, 2018.

[136] K. Li, J. Huang, W. Song, et al. Automatic segmentation and measurement methods of living stomata of plants based on the CV model. *Plant Methods*, 15(67):1–12, 2019.

[137] K. Sakoda, T. Watanabe, S. Sukemura, et al. Genetic Diversity in Stomatal Density among Soybeans Elucidated Using High-throughput Technique Based on an Algorithm for Object Detection. *Scientific Reports*, 9(7610):1–9, 2019.

[138] S. B. Cowling, H. Soltani, S. Mayes, and E. H. Murchie. Stomata detector: High-throughput automation of stomata counting in a population of african rice (oryza glaberrima) using transfer learning. *bioRxiv*, 2021.

[139] R. S. Caine, X. Yin, J. Sloan, et al. Rice with reduced stomatal density conserves water and has improved drought tolerance under future climate conditions. *New Phytologist*, 221(1):371–384, 2018.

[140] Z. Xu, Y. Jiang, B. Jia, et al. Elevated-CO_2 Response of Stomata and Its Dependence on Environmental Factors. *Frontiers in Plant Science*, 7(657):1–15, 2016.

[141] C. L. Wilson, P. Lawrence, and B. E. Otto. Plant epidermal sections and imprints using cyanoacrylate adhesives. *Canadian Journal of plant Science*, 61:781–782, 1981.

[142] J. L. Griffin. Quantification of the effects of water stress on corn growth and yield. Master's thesis, Univ. of Missouri, Columbia, USA, 1980.

[143] V. Hoyos-Villegas and F. B. Fritschi. Relationships Among Vegetation Indices Derived from Aerial Photographs and Soybean Growth and Yield. *Crop Science*, 53(6):1756–1768, 2013.

[144] S. K. Singh, V. Hoyos-Villegas, J. H. Houx, et al. Influence of artificially restricted rooting depth on soybean yield and seed quality. *Agricultural Water Manage*, 105:38–47, 2012.

[145] W. R. Fehr, C. E. Caviness, D. T. Burmood, et al. Stage of development descriptions for soybeans, Glycine-Max (L) Merrill. *Crop Science*, 11:929–931, 1971.

[146] A. B. Alexey. YOLO darknet, 2018.

[147] L. Jiao, F. Zhang, F. Liu, et al. A Survey of Deep Learning-Based Object Detection. *IEEE Access*, 7:128837–128868, 2019.

[148] Y. Tian, G. Yang, Z. Wang, et al. Apple detection during different growth stages in orchards using the improved YOLO-V3 model. *Computers and Electronics in Agriculture*, 157:417–426, 2019.

[149] H. Murat Unver and E. Ayan. Skin Lesion Segmentation in Dermoscopic Images with Combination of YOLO and GrabCut Algorithm. *Diagnostics*, 9:72, 2019.

[150] Dhiraj and D. K. Jain. An evaluation of deep learning based object detection strategies for threat object detection in baggage security imagery. *Pattern Recognition Letters*, 120:112–119, 2019.

[151] E. Kondrateva, M. Pominova, E. Popova, et al. Domain shift in computer vision models for MRI data analysis: an overview, 2021.

[152] A. Casado-García and J. Heras. Ensemble Methods for Object Detection. In *European Conference on Artificial Intelligence (ECAI'2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2688–2695, 2020.

[153] J. Howard, S. Gugger, and S. Chintala. *Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD*. O'Reilly Media, Incorporated, 2020.

[154] S. V. Isayenkov. Genetic sources for the development of salt tolerance in crops. *Plant Growth Regulation*, 89:1–17, 2020.

[155] S. Shabala. Learning from halophytes: physiological basis and strategies to improve abiotic stress tolerance in crops. *Annals of Botany*, 112:1209–1221, 2013.

[156] B. J. Barkla and R. Vera-Estrella. Single cell-type comparative metabolomics of epidermal bladder cells from the halophyte *Mesembryanthemum crystallinum*. *Frontiers of Plant Science*, 6, 2015.

[157] M. Dassanayake and J. C. Larkin. Making plants break a sweat: The structure, function, and evolution of plant salt glands. *Frontiers of Plant Science*, 8, 2017.

[158] S. Agarie, T. Shimoda, Y. Shimizu, et al. Salt tolerance, salt accumulation, and ionic homeostasis in an epidermal bladder-cell-less mutant of the common ice plant *Mesembryanthemum crystallinum*. *Journal of Experimental Botany*, 58:1957–1967, 2007.

[159] A. Kiani-Pouya, U. Roessner, N.S. Jayasinghe, et al. Epidermal bladder cells confer salinity stress tolerance in the halophyte quinoa and Atriplex species. *Plant, Cell & Environment*, 40:1900–1915, 2017.

[160] T. Imamura, Y. Yasui, H. Koga, et al. A novel WD40-repeat protein involved in formation of epidermal bladder cells in the halophyte quinoa. *Communications Biology*, 513, 2020.

[161] L. Shabala, A. Mackay, Y. Tian, et al. Oxidative stress protection and stomatal patterning as components of salinity tolerance mechanism in quinoa (*Chenopodium quinoa*). *Physiologia Plantarum*, 146:26–38, 2012.

[162] A. Kiani-Pouya, F. Rasouli, N. Bazihizina, et al. A large-scale screening of quinoa accessions reveals an important role of epidermal bladder cells and stomatal patterning in salinity tolerance. *Environmental and Experimental Botany*, 168:103885, 2019.

[163] F. Orsini, M. Accorsi, G. Gianquinto, et al. Beyond the ionic and osmotic response to salinity in *Chenopodium quinoa*: functional elements of successful halophytism. *Functional Plant Biology*, 38:818–831, 2011.

[164] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020.

[165] S. Garcia, A. Fernández, J. Luengo, et al. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180:2044–2064, 2010.

[166] D. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, London, 2011.

[167] S. S. Shapiron and M. B. Wilk. An analysis for variance test for normality (complete samples). *Information Sciences*, 180:2044–2064, 1965.

[168] H. Levene. *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, chapter Robust tests for equality of variances, pages 278–292. Stanford University Press, 1960.

[169] O.J. S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[170] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Academic Press, USA, 1969.

[171] J. Cohen. Eta-squared and partial eta-squared in fixed factor anova designs. *Educational and Psychological Measurement*, 33:107–112, 1973.

[172] I. Arvidsson, N. C. Overgaard, F.-E. Marginean, et al. Generalization of prostate cancer classification for multiple sites using deep learning. In *IEEE 15th International Symposium on Biomedical Imaging (ISBI'2018)*, pages 191–194. IEEE, 2018.

[173] M. H. Saleem, J. Potgieter, and K. M. Arif. Automation in agriculture by machine and deep learning techniques: A review of recent developments. *Precision Agriculture*, pages 1–39, 2021.

[174] T. Ad ao, J. Hruška, L. Pádua, et al. Hyperspectral imaging: A review on uav-based sensors, data processing and applications for agriculture and forestry. *Remote Sensing*, 9(11), 2017.

[175] J. Kim, S. Kim, C. Ju, and H. I. Son. Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications. *IEEE Access*, 7:105100–105115, 2019.

[176] J. Zhou, J. Zhou, H. Ye, et al. Classification of soybean leaf wilting due to drought stress using uav-based imagery. *Computers and Electronics in Agriculture*, 175:105576, 2020.

[177] J. Dyson, A. Mancini, E. Frontoni, and P. Zingaretti. Deep learning for soil and crop segmentation from remotely sensed data. *Remote Sensing*, 11(16), 2019.

[178] W. Guo, B. Zheng, A. B. Potgieter, et al. Aerial imagery analysis–quantifying appearance and number of sorghum heads for applications in breeding and agronomy. *Frontiers in Plant Science*, 9, 2018.

[179] L. P. Osco, K. Nogueira, A. P. M. Ramos, et al. Semantic segmentation of citrus-orchard using deep neural networks and multispectral uav-based imagery. *Precision Agriculture*, 22:1–18, 2021.

[180] H. Wu, T. Wiesner-Hanks, E. L. Stewart, et al. Autonomous detection of plant disease symptoms directly from aerial imagery. *The Plant Phenome Journal*, 2(1):1–9, 2019.

[181] M. Yang, H. Tseng, Y. Hsu, and H. P. Tsai. Semantic segmentation using deep learning with vegetation indices for rice lodging identification in multi-date uav visible images. *Remote Sensing*, 12(4):633, 2020.

[182] J. Das, G. Cross, C. Qu, et al. Devices, systems, and methods for automated monitoring enabling precision agriculture. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 462–469. IEEE, 2015.

[183] H. Tian, T. Wang, Y. Liu, et al. Computer vision technology in agricultural automation—a review. *Information Processing in Agriculture*, 7(1):1–19, 2020.

[184] P. Jiang, Y. Chen, B. Liu, et al. Real-time detection of apple leaf diseases using deep learning approach based on improved convolutional neural networks. *IEEE Access*, 7:59069–59080, 2019.

[185] J. Ma., K. Du, L. Zhang, et al. A segmentation method for greenhouse vegetable foliar disease spots images using color information and region growing. *Computers and Electronics in Agriculture*, 142:110–117, 2017.

[186] K. Yang, W. Zhong, and F. Li. Leaf segmentation and classification with a complicated background using deep learning. *Agronomy*, 10(11):1721, 2020.

[187] M. Afonso, H. Fonteijn, F. S. Fiorentin, et al. Tomato fruit detection and counting in greenhouses using deep learning. *Frontiers in Plant Science*, 11, 2020.

[188] I. Sa, Z. Ge, F. Dayoub, et al. Deepfruits: A fruit detection system using deep neural networks. *Sensors*, 16(8):1222, 2016.

[189] J. Mack, C. Lenz, J. Teutrine, and V. Steinhage. High-precision 3d detection and reconstruction of grapes from laser range data for efficient phenotyping based on supervised learning. *Computers and Electronics in Agriculture*, 135:300–311, 2017.

[190] P. Bosilj, E. Aptoula, T. Duckett, and G. Cielniak. Transfer learning between crop types for semantic segmentation of crops versus weeds in precision agriculture. *Journal of Field Robotics*, 37(1):7–19, 2020.

[191] F. J. Knoll, V. Czymmek, S. Poczihoski, et al. Improving efficiency of organic farming by using a deep learning classification approach. *Computers and Electronics in Agriculture*, 153:347–356, 2018.

[192] A. Milioto, P. Lottes, and C. Stachniss. Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in cnns. In *Proceedings 2018 IEEE international conference on robotics and automation (ICRA'2018)*, pages 2229–2235. IEEE, 2018.

[193] A. Wang, Y. Xu, X. Wei, and B. Cui. Semantic segmentation of crop and weed using an encoder-decoder network and image enhancement method under uncontrolled outdoor illumination. *IEEE Access*, 8:81724–81734, 2020.

[194] N. Chebrolu, P. Lottes, A. Schaefer, et al. Agricultural robot dataset for plant classification, localization and mapping on sugar beet fields. *The International Journal of Robotics Research*, 36(10):1045–1052, 2017.

[195] J. Naranjo-Torres, M. Mora, R. Hernández-García, et al. A review of convolutional neural network applied to fruit image processing. *Applied Sciences*, 10(10):3443, 2020.

[196] O. Wosner, G. Farjon, and A. Bar-Hillel. Object detection in agricultural contexts: A multiple resolution benchmark and comparison to human. *Computers and Electronics in Agriculture*, 189:106404, 2021.

[197] Y. Tian, G. Yang, Z. Wang, et al. Instance segmentation of apple flowers using the improved mask r–cnn model. *Biosystems Engineering*, 193:264–278, 2020.

[198] F. Gao, L. Fu, X. Zhang, et al. Multi-class fruit-on-plant detection for apple in snap system using faster r-cnn. *Computers and Electronics in Agriculture*, 176:105634, 2020.

[199] Z. Song, Z. Zhou, and W. Wang others. Canopy segmentation and wire reconstruction for kiwifruit robotic harvesting. *Computers and Electronics in Agriculture*, 181:105933, 2021.

[200] A. Barriguinha, M. de Castro Neto, and A. Gil. Vineyard yield estimation, prediction, and forecasting: A systematic literature review. *Agronomy*, 11(9), 2021.

[201] Y. Majeed, M. Karkee, and Q. Zhang. Estimating the trajectories of vine cordons in full foliage canopies for automated green shoot thinning in vineyards. *Computers and Electronics in Agriculture*, 176:105671, 2020.

[202] R. Berenstein, O. B. Shahar, A. Shapiro, and Y. Edan. Grape clusters and foliage detection algorithms for autonomous selective vineyard sprayer. *Intelligent Service Robotics*, 3(4):233–243, 2010.

[203] N. Behroozi-Khazaei and M. R. Maleki. A robust algorithm based on color features for grape cluster segmentation. *Computers and Electronics in Agriculture*, 142:41–49, 2017.

[204] T. T. Santos, L. L. de Souza, A. A. dos Santos, and S. Avila. Grape detection, segmentation, and tracking using deep neural networks and three-dimensional association. *Computers and Electronics in Agriculture*, 170:105247, 2020.

[205] L. Fu, F. Gao, J. Wu, et al. Application of consumer rgb-d cameras for fruit detection and localization in field: A critical review. *Computers and Electronics in Agriculture*, 177:105687, 2020.

[206] R. Barnea, R. Mairon, and O. Ben-Shahar. Colour-agnostic shape-based 3d fruit detection for crop harvesting robots. *Biosystems Engineering*, 146:57–70, 2016.

[207] A. Gongal, A. Silwal, S. S. Amatya, et al. Apple crop-load estimation with over-the-row machine vision system. *Computers and Electronics in Agriculture*, 120:26–35, 2016.

[208] L. Fu, Y. Majeed, X. Zhang, et al. Faster r–cnn–based apple detection in dense-foliage fruiting-wall trees using rgb and depth features for robotic harvesting. *Biosystems Engineering*, 197:245–256, 2020.

[209] T. T. Nguyen, K. Vandevoorde, N. Wouters, et al. Detection of red and bicoloured apples on tree with an rgb-d camera. *Biosystems Engineering*, 146:33–44, 2016.

[210] S. Paulus, J. Behmann, A.-J-Mahlein, et al. Low-cost 3d systems: suitable tools for plant phenotyping. *Sensors*, 14(2):3001–3018, 2014.

[211] Y. Tao and J. Zhou. Automatic apple recognition based on the fusion of color and 3d feature for robotic fruit picking. *Computers and Electronics in Agriculture*, 142:388–396, 2017.

[212] J. Zhang, L. He, M. Karkee, et al. Branch detection for apple trees trained in fruiting wall architecture using depth features and regions-convolutional neural network (r-cnn). *Computers and Electronics in Agriculture*, 155:386–393, 2018.

[213] R. Marani, A. Milella, A. Petitti, and G. Reina. In-field high throughput grapevine phenotyping with a consumer-grade depth camera. *Computers and electronics in agriculture*, 156:293–306, 2019.

[214] Y. W. Kuan, N. O. Ee, and L. S. Wei. Comparative study of intel r200, kinect v2, and primesense rgb-d sensors performance outdoors. *IEEE Sensors Journal*, 19(19):8741–8750, 2019.

[215] H. Kang and C. Chen. Fruit detection, segmentation and 3d visualisation of environments in apple orchards. In *Computers and Electronics in Agriculture*, volume 171, page 105302, 2020.

[216] R. Marani, A. Milella, A. Petitti, and G. Reina. Deep neural networks for grape bunch segmentation in natural images from a consumer-grade camera. *Precision Agriculture*, 22(2):387–413, 2021.

[217] X. A. Wang, J. Tang, and M. Whitty. Side-view apple flower mapping using edge-based fully convolutional networks for variable rate chemical thinning. *Computers and Electronics in Agriculture*, 178:105673, 2020.

[218] J. Heras, R. Marani, and A. Milella. Semi-supervised semantic segmentation for grape bunch identification in natural images. In *Proceedings of Precision Agriculture'2021*, pages 65–84. Wageningen Academic Publishers, 2021.

[219] R. Marani, A. Milella, A. Petitti, and G. Reina. Deep learning-based image segmentation for grape bunch detection. In *Proceedings of the 12th European Conference on Precision Agriculture, Precision Agriculture'19*, pages 3320–3328. Wageningen Academic Publishers, 2019.

[220] A. S. Razavian, H. Azizpour, J. Sullivan, et al. CNN features off-the-shelf: An astounding baseline for recognition. In *27th Conference on Computer Vision and Pattern Recognition Workshops*, CVPRW'14, pages 512–519, 2014.

[221] K. Sun., Y. Zhao, B. Jiang, et al. High-resolution representations for labeling pixels and regions. *CoRR*, abs/1904.04514, 2019.

[222] M. Yang, K. Yu, C. Zhang, Z. Li, and K. Yang. Denseaspp for semantic segmentation in street scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3684–3692, 2018.

[223] P. K. R. Poudel, U. Bonde, S. Liwicki, and C. Zach. Contextnet: Exploring context and detail for semantic segmentation in real-time. In *Proceedings of the 29th British Machine Vision Conference*, 2018.

[224] T. Wu, S. Tang, R. Zhang, and Y. Zhang. Cgnet: A light-weight context guided network for semantic segmentation. *IEEE Transactions on Image Processing*, 30:1169–1179, 2020.

[225] L.C. Chen, Y. Zhu, G. Papandreou, et al. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV'2018)*, pages 833–851, 2018.

[226] H. Li, P. Xiong, J. An, and L. Wang. Pyramid attention network for semantic segmentation. In *Proceedings of the 29th British Machine Vision Conference*, 2018.

[227] M. Liu and H. Yin. Feature pyramid encoding network for real-time semantic segmentation. In *Proceedings of the 30th British Machine Vision Conference*, 2019.

[228] Z. Zhou, Md M. Rahman Siddiquee, N. Tajbakhsh, and J. Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: 4th International Workshop, DLMIA 2018, and 8th International Workshop, ML-CDS 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 20, 2018, Proceedings 4*, volume 11045, pages 3–11. Springer, 2018.

[229] C. Yu, J. Wang, C. Peng, et al. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV'2018)*, pages 334–349. Springer, 2018.

[230] Y. Wang, Q. Zhou andJ. Liu, J. Xiong, et al. Lednet: A lightweight encoder-decoder network for real-time semantic segmentation. pages 1860–1864, 2019.

[231] Y. Yuan and J. Wang. Ocnet: Object context network for scene parsing. *arXiv preprint arXiv:1809.00916*, 2018.

[232] R. Li, S. Zheng, C. Duan, et al. Multi-attention-network for semantic segmentation of fine resolution remote sensing images. *ArXiv preprint arXiv:2009.02130*, 2020.

[233] A. Paszke, S. Gros, F. Massa, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

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

[235] S. Poni, M. Gatti, and A. Garavani. Grape maturity assessment in vineyards using unmanned aerial vehicles and machine learning. *Precision Agriculture*, 19(5):789–801, 2018.

[236] Y. Zhang, H. Sun, and J. Zhang. Development of an online near-infrared spectroscopy system for real-time monitoring of grape maturation during harvest. *Journal of Food Engineering*, 183:67–73, 2016.

[237] A. Casado-García, J. Heras, A. Milella, and R. Marani. Semi-supervised deep learning and low-cost cameras for the semantic segmentation of natural images in viticulture. *Precision Agriculture*, 23:2001–2026, 2022.

[238] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Indoor semantic segmentation using depth information. In *Proceedings of International Conference on Learning Representations (ICLR'2013)*, pages 1–8, 2013.

[239] M. D. Ansari, A. Husada, and D. Stricker. Scale invariant semantic segmentation with rgb-d fusion, 2022.

[240] C. Hazirbas, L. Ma, C. Domokos, and D. Cremers. Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *Computer Vision – (ACCV'2016)*, volume 10111 of *Lecture Notes in Computer Science*. Springer, 2017.

[241] D. Kim, W. Ga, P. Ahn, et al. Global-local path networks for monocular depth estimation with vertical cutdepth. *arXiv preprint arXiv:2201.07436*, 2022.

[242] Pillow contributors. Pillow documentation. `https://pillow.readthedocs.io/en/stable/`, 2023.

[243] A. Kirillov, E. Mintun, N. Ravi, et al. Segment anything, 2023.