IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 24 May 2024, accepted 9 June 2024, date of publication 13 June 2024, date of current version 21 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3413979

RESEARCH ARTICLE

Enhancing Precision Agriculture Pest Control: A Generalized Deep Learning Approach With YOLOv8-Based Insect Detection

MARIO VILAR-ANDREU, LAURA GARCÍA, ANTONIO-JAVIER GARCIA-SANCHEZ, RAFAEL ASOREY-CACHEDA, (Member, IEEE), AND JOAN GARCIA-HARO, (Member, IEEE)
Department of Information and Communications Technologies, Universidad Politécnica de Cartagena, 30202 Cartagena, Spain

Corresponding author: Laura García (laura.garcia@upct.es)

This work was supported in part by MCIN/AEI/10.13039/501100011033 and European Union NextGenerationEU/PRTR under Grant FJC2021-047073-I, Grant PID2020-116329GB-C22, and Grant TED2021-129336B-I00; in part by Fundación Séneca under Grant 22236/PDC/23; in part by ThinkInAzul and AgroAlNext Program, funded by MICIU from European Union NextGenerationEU under Grant PRTR-C17.I1; in part by Fundación Séneca from Comunidad Autónoma Región de Murcia (CARM); and in part contextualized to DAIMon, a cascade funding action deriving from the Horizon Europe project aerOS, funded by the European Commission under grant number 101069732.

ABSTRACT Precision Agriculture (PA) is gaining new momentum due to its ability to accurately adjust the resources to a crop's needs while maintaining/enhancing quality levels. However, crop-damaging pests compromise yields, jeopardizing the benefits of PA. Computer vision-based pest detection techniques offer promising avenues to overcome potential losses for farmers. The recent object detection framework, YOLOv8 (You Only Look Once) applied to real-time insect monitoring is an open-source, cutting-edge PA approach based on Convolutional Neural Network (CNN) models that enables precise and quick decision making in agricultural crops. Under this umbrella, traditional pest studies using YOLO or other deep-learning solutions focused on only one or a few insects for specific crops provide an excessively narrow solution. In this paper, we propose a new form of using YOLO for pest detection with a generalist perspective by intensively testing a YOLOv8-based tool implementing a single insect category. The goal is to detect the presence of any type of insect in any type of crop in real time. A comprehensive performance evaluation is carried out using a well-known dataset. The results of the training, validation, and testing phases are then discussed, obtaining an mAP_{50} value of 0.967 for the *m* model and an mAP_{50-95} value of 0.632 for the *l* model. Finally, we also identify the premises to elaborate a complete and useful dataset able to unleash the full potential of YOLOv8.

INDEX TERMS Insect detection, object detection, pest control, you only look once (YOLO).

I. INTRODUCTION

Agriculture is one of the world's main economic activities, with 1.23 billion workers, including on-farm and off-farm operations [1]. Furthermore, the growing demand to produce more and higher quality products has resulted in an increased use of resources, which can be detrimental to the environment. Precision agriculture (PA) has emerged as a solution to reduce resource consumption, improving both crop yield and quality levels by deploying communication-enabled and monitoring sensor devices. The data acquired by these sensors are then analyzed to determine the specific needs of the crop. However, the benefits of PA are being questioned due to the presence of crop pests. Annually, the FAO [2] estimates that pests are responsible for 20-40% of losses from damaged crops. These losses do not only affect crop yields but also have significant economic implications for farmers and increase the cost of end products to consumers. Pests are often treated or prevented with potent chemicals that can be harmful to humans and the environment. In an effort to solve this problem, some researchers are designing alternative applications, such as on-demand spot spraying [3], but these

The associate editor coordinating the review of this manuscript and approving it for publication was Zhongyi Guo.

solutions may be insufficient in advanced pest infestations. Therefore, early detection is essential to mitigate the resulting economic, environmental, and health damage.

Pest monitoring employs a multiple strategy, combining various control mechanisms including the use of chemical agents. An effective approach to minimizing the undesirable effects of crop pests is to deploy traps [4]. They are conceived to attract insects and are divided into several types, such as sex pheromone traps, yellow sticky traps, and glow traps. Selecting a trap depends on the specific crop or type of pest being targeted [5]. Traps are routinely monitored by trained personnel to assess the insect population captured. However, this supervision process often requires periodic field visits, increasing costs [6]. Integrating advanced technology in pest detection solutions has made it possible to remotely monitor insects through sensing, enhancing agricultural practices and enabling real-time surveillance [7]. Video cameras are strategically positioned in fields to monitor insects in the existing traps, or they directly focus on the crops to assess the presence of pests on leaves or stems. However, these crop images require further processing for insect detection and classification in order to further automate the procedure.

Computer vision is a field in Artificial Intelligence (AI) that deploys devices equipped with cameras to process and analyze images as if they were human eyes [8]. This allows objects in an image to be detected and classified. Introducing computer vision techniques in the agriculture field, also called agro-vision [9], has provided more tools for PA deployment. Examples of this are automated farming tasks when paired with robots and adapted farming machinery used for weeding [10] and harvesting ripe crops [11]. Computer vision is also an excellent pest detection tool for crops in the field and in greenhouses. Different techniques based on machine-learning or deep-learning algorithms have been studied and tested, with those based on deep learning obtaining the best outcomes [12]. As a result, current works focus on employing deep-learning algorithms such as Convolutional Neural Networks (CNN) [13], [14]. CNN-based techniques have achieved notable success in many computer vision challenges since they do not require manual feature extraction. Their architecture is designed as visual perception, carrying out automatic image classification to later perform object detection tasks.

YOLO [15] is a real-time, open-source object detection system based on CNN models and algorithms. From the first version, which outperformed other CNN-based algorithms such as Fast Region-CNN [16], YOLO has been updated with new frameworks, improving its predecessors [17]. Pest detection studies using YOLO have also been enhanced [18], [19] [20]. However, current YOLO pest works have an important drawback: they involve detecting just one or a small set of insects, resulting in solutions only valid for one specific crop or insect species. This restricts the applicability of pest detection systems, making it difficult to generalize them as a de facto feature of broader PA systems.

Recent literature about YOLO is based on classifying the insect into established categories of a reduced subset of insect types, which limits its applicability. Not all insects can be found in every part of the world. Some are distinctive of certain regions and specific crops. This is not useful for farmers located in other regions or cultivating different crops. In this paper, one of the main contributions is approaching the problem in a generic manner, allowing the farmer to detect insects regardless of the particular type of insect or crop. This way, our approach can be used as all-purpose by farmers, minimizing the limitations of other solutions that only address a reduced number of insects and can miss other damaging pests, which leads to important problems not being detected. Through the introduction of a generalist use of YOLO for pest detection, we provide a solution that considers all insect species as a single category to detect the presence of any insect in any type of crop in real time. To this end, we exhaustively study and evaluate the different models of the latest version of YOLO, YOLOv8 [21], to comprehensively apply it to insect detection. Our major contribution to the scientific community and end-users in general is the development of a generic and powerful pest detection deep-learning tool to be included in PA solutions that leverage YOLOv8.

For this purpose, we use experimentation to analyze the outcomes of the training, validation, and testing phases, evaluating the model's performance in different scenarios. The results reveal not only the strengths but also the potential weaknesses of YOLOv8 in the context of insect detection. This has allowed us to design a complete and robust dataset and establish the starting point for future accurate and reliable insect detection solutions.

The rest of the paper is organized as follows. Section II describes the related work. The YOLOv8 architecture is detailed in section III. The datasets and the metrics used to evaluate the performance of our proposal are introduced in section IV. Section V discusses the results obtained. Lastly, section VI concludes and offers some remarks on future work.

II. RELATED WORK

The evolution of information processing in PA brought about by recent advances in AI, such as machine-learning and deep-learning, has led authors to evaluate its performance for pest detection and classification. Artificial Neural Networks (ANN), naïve bayes (NB), k-Nearest neighbors (KNN), Support Vector Machine (SVM), and CNN models were compared for insect detection and classification tasks by [22]. In particular, insect detection was performed using foreground extraction and contour identification. The validation results showed that the highest accuracy values were obtained for CNN with 91.5% classification accuracy for 9 different insect classes and 90% for 24 different insect classes. Similarly, [12] compared machine-learning and deep-learning techniques for computer vision-based pest identification in tomato plants. This pest identification solution was intended for agricultural autonomous robots.

The authors generated their own dataset with affected tomato plants. Their results from the validation tests determined that deep-learning techniques perform better in terms of accuracy, speed, and pest-type discrimination.

Analyzing the recent scientific literature, there seems to be a consensus about the superior performance of deep-learning for pest detection. Reference [13] presented a deep-learning solution for tiny pest detection employing the images obtained from cameras on sticky traps. The focus of the study was on thrips and whiteflies in greenhouses. A model called TPest-RCNN based on Faster Regional-CNN was developed, with a mean average precision (mAP) of 0.952 and a mean F1 score, which is the harmonic mean of precision and recall, of 0.944. Their results demonstrated better performance using the solution proposed by the authors than the Faster R-CNN model it was based on. A deep learning-based system for pest detection in oilseed rape crops to be employed in mobile devices was introduced by [23]. Firstly, the authors created a database by gathering images from the 12 most common pests in oilseed rape. Five different models were evaluated to determine their performance considering the model loss. These models were Faster R-CNN, R-FCN, and SSD (Single-Shot Detector), with a combination of different feature extractors such as RestNet101, Inception, or MobileNet. Data augmentation techniques as well as a dropout layer, which prevented overfitting, were added to improve the resulting mAP. The authors determined that the best solution was SSD with Inception as it provided a balance between execution time, accuracy, and memory usage. Their results showed an improvement in mAP, which achieved a value of 77.14%. The trained model was implemented in an Android platform for use in mobile phones. Furthermore, [14] developed a mobile application for pest identification based on Faster R-CNN deep-learning techniques. The application was also connected to a database with pesticide recommendations, providing users with specific information on how to eliminate the detected pests. The pests considered were red spider, flax budworm, aphids, flea beetles, and Cicadellidae, all receiving recognition results of 99.0%.

Within the options of deep-learning techniques available, YOLO has been gaining popularity due the performance demonstrated in several studies. Reference [24] reviewed insect detection techniques using deep learning techniques. The authors highlighted the importance of the quality of the dataset to obtain suitable detection and classification results. Specifically, object detection was affected by dataset size and number of classes, whereas classification was most affected by the dataset size. Furthermore, the conclusions indicated that modified Faster R-CNN and YOLOv5 exhibited the best performance. Reference [25] suggested a deep learning-based solution to detect brown rice planthoppers. Due to the small size of this insect, the authors considered an algorithm comprised of two layers, using Faster RCNN in both of them. The performance of their proposal, which relies on two-step object detection, was tested and compared with YOLOv3, which follows a single-stage detection approach. The outcomes showed that their approach obtained a recall rate of 81.92%, compared to 57.12% obtained by YOLOv3. However, it is important to note that YOLOv3 received higher average accuracy than the proposed solution, 97.36% and 94.64%, respectively. A method for pest identification based on the deep-learning YOLO object detection algorithm was presented by [15]. The system included a smartphone IP-camera handled by farmers that collected videos and images of the insects. Different versions of the YOLO algorithm were compared for precision. Specifically, YOLO-Lite, YOLOR, YOLOv3, and YOLOv5 were compared at scales of "n," "s," "m," "l," and "x." These scales indicate the model size, where "n" stands for nano, "s" for small, "m" for medium, "l" for large, and "x" for extra large. The outcomes from tests where 23 different pest species were identified showed the best precision using YOLOv5x with 98.3%. Another solution intended for smartphones was provided by [26]. Their application for scale insect detection compared YOLOv4, SSD, and Faster R-CNN models in terms of classification accuracy. Their results revealed that YOLOv4 attained the best performance with a precision of 89%, 97%, and 100% for Coccidae, Diaspididae, and mealybugs, respectively. Lastly, [27] combined IoT and deep-learning to implement a pest detection system for smart agriculture. The deployed IoT devices were also equipped with sensors for environmental monitoring. YOLOv3 was used to analyze the combination of pest images and environmental data. The results demonstrated a pest identification precision of 90%. Additionally, the system provided the user with information on the location and virulence of the pest.

In contrast to these studies, we used the latest release from the YOLO family in the field of precision agriculture and, specifically, intelligent pest detection systems, YOLOv8. Furthermore, we present an innovative form of using YOLO for pest detection with a generalist perspective by treating all instances of insects as a single class instead of using separate classes for different types of insects, which is the main use case in the related literature. This way, we acquire more knowledge about deep-learning for pest-detection and how to train the available models to obtain results applicable to a wider set of crops and insect types.

III. YOLOv8 ARCHITECTURE

The YOLO (You Only Look Once) series has gained widespread recognition in the field of computer vision due to its remarkable accuracy and compact model size. Originating in 2015 with its first version [28], in its early iterations (versions 1-4), YOLO was programmed in C language and embedded in a custom deep learning framework called Darknet. The transition to YOLOv5 marked a significant difference. YOLOv5 enabled more versatile and Pythonic structural adaptations and fostered a collaborative environment for the scientific community to innovate and swiftly share modeling improvements through repositories. This structure facilitated the exploration of novel models, and each iteration in the YOLO series has introduced more

advanced techniques, enhancing the model's precision and efficiency. The latest iteration, YOLOv8, is distinguished by its cutting-edge performance in terms of accuracy and speed.

A brief summary of YOLOv8's architecture as described by [29] can be seen in Figure 1. YOLOv8 uses blocks to construct its architecture. These blocks are comprised of layers, and each layer applies distinct operations and transformations to the input, generating varied outputs with unique parameters. Furthermore, YOLOv8, as with most object detection models, is divided into two parts, the backbone and the head, which are described below.

A. BACKBONE

The backbone is the main component of an object detection model. It is responsible for capturing hierarchical features from the input image. The backbone processes the input image through multiple convolutional layers, extracting features on different spatial scales. These features are then used to represent the image hierarchically, that is with dependent relationships among features, enabling the model to understand both low-level and high-level patterns.

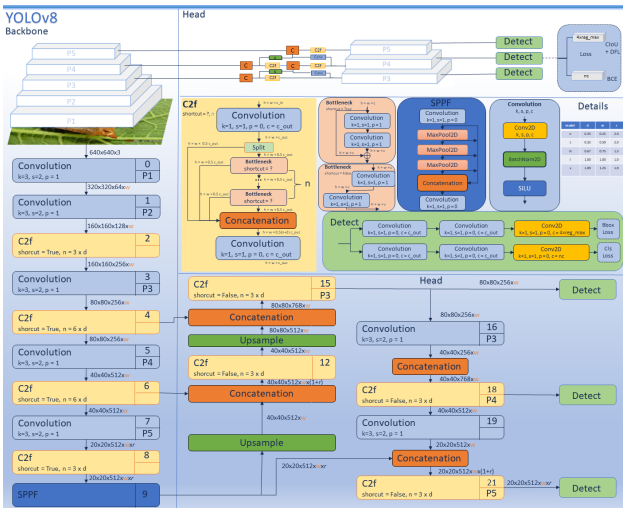

FIGURE 1. YOLOv8 architecture.

The first two blocks of the YOLOv8 backbone are Convolutional blocks. They are the most basic blocks of the YOLOv8 architecture. In Figure 1, they are the light blue blocks. These blocks perform a 2D convolution to extract features, apply batch normalization to enhance training stability, and use the *SiLU* equation as an activation function to introduce non-linearities in the network. The Convolutional block is composed of a Convolutional layer, a Batch Normalization layer, and a Sigmoid Linear Unit (*SiLu*) [30] as an activation function.

The Convolutional layer applies a convolution operation to the input, using filters as parameters. The goal is to systematically analyze the input, region by region, identifying patterns and extracting features, which are aggregated into activation maps to obtain the depth of the input (filter outputs). After the Convolutional layer, a non-linear activation function is mandatory. Activation functions compute the output, element-by-element, based on the input, preserving both spatial resolution and depth. These functions are non-linear, giving the model the capability to capture and represent non-linear patterns in the data.

Moreover, Batch Normalization layers are employed to normalize the mean and standard deviation of the activation batch according to the following equation:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{1}$$

where x represents the current batch, μ the mean of the activations, σ the standard deviation of the batch, and ϵ a small value close to zero. These layers contribute to training deeper models, enhancing their robustness and stability.

The Convolutional blocks are followed by a *C2f* block, which is the light yellow block in Figure 1. The *C2f* block initiates with a 2D convolutional layer, followed by splitting the resulting output into n Bottleneck blocks to obtain an unaltered output that is then concatenated, leading to the final 2D convolutional layer.

Bottleneck blocks are used in the *C2f* blocks, occasionally incorporating a shortcut. The concept behind these blocks is to reduce the volume of information, thereby enhancing computational efficiency and alleviating the computational load. This is achieved by employing 1×1 Convolutional layers to compress and subsequently expand the volume of the input. They are the light orange blocks in Figure 1.

Concatenation layers are employed to merge inputs from different depths within the network. However, as the input traverses the neural network, dimensions often change due to depth expansion or spatial resolution reduction. To facilitate the addition of inputs from varying depths, Upsample layers come into play. These layers interpolate values to align with a desired target resolution.

The *C2f* block leverages Bottleneck blocks to enhance computational efficiency, thus enabling the learning of complex and hierarchical representations.

After the *C2f* block, the process leads to a final Spatial Pyramid Pooling block, shown as the medium blue block in Figure 1. The objective of Spatial Pyramid Pooling is to gather information from different regions of an image, regardless of their size or position. This is achieved by dividing the input image into several levels and subsequently pooling each of them, assuring contextual information is captured across multiple scales. This block employs a 2D convolutional layer to compress the input, then the output is split into three pathways. These outputs are then addressed by the Max Pooling layer and concatenated with a non-pooled output.

Pooling layers are frequently used to decrease the dimensions of the input, resulting in a loss of information but expediting computations. The pooling function, normally either maximum or average, is independently applied to each channel of the input, exclusively reducing the spatial dimensions.

After this, the combined result is fed into a 2D convolutional layer to expand the dimensions once again. This process allows the model to extract features from different spatial dimensions and refine them to obtain relevant information.

B. HEAD

The head is responsible for generating predictions based on the features extracted by the backbone. It consists of several *C2f* blocks, along with Upsample and Concatenation blocks and Detection blocks. All these blocks typically consist of additional layers, including fully connected layers and bounding box regression or classification layers.

The Detection block divides the output from the head into two pathways: one for classification loss and another for detection loss. Both pathways follow a similar structure, consisting of two Convolutional blocks followed by a 2D Convolutional layer. The outer dimension of this final Convolutional layer corresponds to either the number of classes (for classification) or the number of outputs per box (for detection). This can also be observed in Figure 1 as the light green block.

The head refines the feature representations obtained from the backbone and carries out the final predictions for the object bounding boxes and their associated class probabilities. It is also important to note that one of the main differences between YOLOv5 and YOLOv8 is that the latter is anchor free. This means that YOLOv8 does not predict the offset from a known anchor box, but it predicts the center of the object directly. Anchor-free detection decreases the number of box predictions, expediting the inference process, particularly Non-Maximum Suppression (*NMS*). *NMS* is a post-processing step that reviews candidate detections and eliminates overlapping boxes.

The YOLOv8 architecture offers several models that are distinguished by the number of layers and parameters employed. The models used in this study and their respective characteristics are summarized in Table 1.

Each model is characterized by the number of layers, parameters, and GFLOPs (Giga Floating Point Operations Per Second). Models *n* and *s* both consist of 225 layers, but *s* has a significantly higher parameter count at 11,166,560, compared to *n*, which computes up to 3,157,200 parameters. The computational complexity, measured in GFLOPs, is also notably higher for *s*, at 28.3 GFLOPs. Models *m* and *l* further escalate in complexity with 295 and 365 layers, 25,857,478 and 43,631,382 parameters, and 79.3 and 165.7 GFLOPs, respectively. These metrics provide insights into the scale and computational demands associated with each YOLOv8 variant, offering a comprehensive view of their architecture.

TABLE 1. Number of layers, parameters, and GFLOPS for the different YOLOv8 models.

Model	Layers	Parameters	GFLOPs
n	225	3157200	8.9
s	225	11166560	28.3
m	295	25857478	79.3
l	365	43631382	165.7

IV. DATASET ANALYSIS AND METRICS

In this section, we evaluate the selected dataset and detail the metrics employed to obtain the performance of the object detection model. The images and labels are studied in Section IV-A. The data augmentation techniques applied to these images are described in Section IV-B. Lastly, the adopted metrics are specified in Section IV-C.

A. DATASET ANALYSIS

To efficiently train an object detection model to any specific task, it is necessary to properly feed images to this model. Insects are highly biodiverse [31], making their detection a challenge as it is complex to generalize physical characteristics, species similarities, and morphological differences in such a heterogeneous group. The scarce availability of public datasets referring to insects, along with the low number of images in these datasets, have intensified the effort to develop this work. Furthermore, laboratory imagery, with good light conditions and white backgrounds, does not suitably adapt to real life conditions. Some approaches to insect detection rely on traps, and the images available in these datasets are only appropriate for studying this specific situation. Even with images obtained with cameras or drones in the field, the datasets tend to focus on specific insects or pests of interest in particular crops. These circumstances only add more difficulty to an already complex issue.

In this work, the employed dataset is IP102 [32]: a large scale dataset of insects. This dataset contains 102 classes, which is appropriate for classification and detection tasks. Our main goal, however, is not detecting specific instances of insects but the presence of pests. Samples of the dataset images can be found in Figure 2.

FIGURE 2. Example images of the IP102 dataset. Each image belongs to a different species of insect.

All the classes in the original dataset have been merged into a superclass called "insect." This approach solves the problem of class imbalance caused by a different number of samples per insect type, but introduces significant variance, that is, the resulting class becomes too heterogeneous. This is caused by the substantial difference in the number of images belonging to each class, and the model tends to predict one image as belonging to the class with the most images. To accomplish this merging process, all the images are assigned the same *class id* in their labels. The labels take the YOLO format, encompassing essential information for object classification and detection. This format includes the following key parameters: class id, a unique number assigned to each class; *x center* and *y center*, the coordinates of the bounding box center; and the width and height of the bounding box. These parameters play a crucial role in object classification and detection within an image. The detection process implies creating a bounding box, a rectangular enclosure that precisely delineates the object of interest. This bounding box serves as a spatial reference, encapsulating the detected object and facilitating subsequent analyses or actions based on its identified location.

Information relative to bounding box distribution can be found in Figure 3, where it is possible to observe how the bounding boxes behave in the dataset, with most boxes having their *x center* and *y center* right in the middle of the image, while the height and width of the labels vary widely from one to another. The values of the axes range from 0 to 1, and these values are obtained from the following assumptions: i) the YOLO bounding box format; ii) the number of instances of each class, which in our case is only one; and iii) each bounding box in the dataset.

The image size distribution, shown in Figure 4, presents a similar pattern. This is due to the fact that most labels have their centers in the middle of the image and tend to occupy most of the frame. In addition to the similar placing of the bounding boxes, most images have comparable lighting conditions, times of day, and angles.

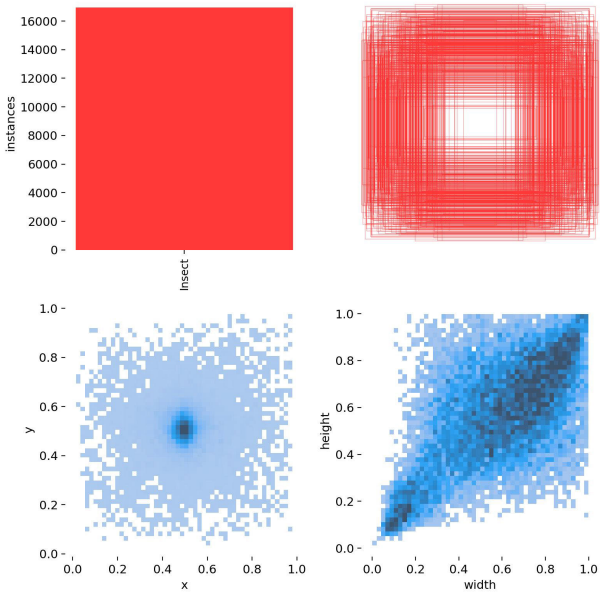

FIGURE 3. Bounding box analysis. Top left graph is the number of instances of each class, top right is every bounding box, bottom left is the *y center* against the *x center* of each label and bottom right is the height of each label against its width.

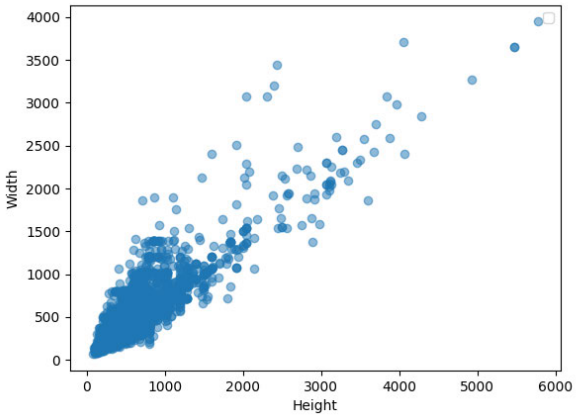

FIGURE 4. Image distribution analysis, height against width of the dataset images.

B. DATA AUGMENTATION

This subsection describes the different transformations applied to the dataset and the effects of these operations.

An appropriate dataset is crucial to successful training. However, publicly available datasets often have limitations in terms of number of images and diversity. Many of them were obtained from laboratory environments where the insects are in the forefront of the picture and the background is white and uniform, which is not representative of the real environment where the model is expected to be deployed. Furthermore, the complexity of the agricultural environment leads to difficulties in obtaining good quality images, for instance due to obstructions from leaves and branches, as well as the continuous changes in lighting conditions caused by the varying weather. To mitigate these challenges, data augmentation techniques have been applied to the images of the dataset. Data augmentation involves transforming the images in a dataset to increase their quantity or quality. Data augmentation techniques are dynamically applied in the image training phases for each epoch. This exposes the model to slightly different variations from the original images, contributing to improving the model's generalization and robustness. As a result, the diversity and scope of the training data set is effectively broadened, allowing the model to learn more versatile and generalized features.

The operations applied to the images are as follows: hue, saturation, and value random adjustments; image translation; image scaling; horizontal flipping; and mosaic augmentation [33]. The HSV (Hue, Saturation, Value) color model [34] represents colors based on three intuitive visual

attributes: hue, saturation, and value. Hue denotes the color tone, usually represented as an angle on the color wheel ranging from $0^{\underline{o}}$ to $360^{\underline{o}}$. Saturation measures the intensity of a color, ranging from 0% (shades of gray) to 100% (pure color with no mixture of white). Value represents the brightness of the color, also expressed as a percentage. A value of 0% refers to black (absence of light), while 100% indicates the color at its highest intensity. By applying random transformations to these channels, the resulting images can exhibit a broader range of color variations, intensities, and brightness levels. This helps obtain the diversity of the dataset, improving the model's ability to recognize objects under different color and lighting conditions.

Image translation [35] involves displacing the image content along the x and y axes, introducing a shift in the position of the image without altering its structure or appearance. This manipulation can include horizontal, vertical, or diagonal movements within the image plane, simulating changes in object positions or different camera viewpoints. By systematically shifting the image contents within the frame, this augmentation technique aims to enhance the robustness and generalization of the model. Additionally, these artificially created variations enable the model to learn patterns from multiple perspectives and spatial contexts, thereby increasing its capacity to recognize objects despite positional changes.

Horizontal flipping [36] generates a mirrored representation of an image by swapping the pixel values from the left side to the right side across a vertical axis. This augmentation technique helps simulate changes in perspective as, for instance, when objects are viewed from the opposite side. By applying this transformation, the model is exposed to additional instances of the same objects but from a different viewpoint, contributing to better recognition and understanding of the objects regardless of their orientation.

Image scaling entails resizing images to increase or decrease their dimensions while preserving the original aspect ratio. Scaling is performed through transformations that adjust the size of images, allowing models to learn from variations in object sizes and providing robustness against different scales in real-world scenarios. When scaling images, pixel values are reorganized to fit a new set of dimensions. Enlarging an image involves interpolation methods to estimate new pixel values, resulting in an expanded view, while decreasing the image size uses down-sampling techniques. Scaling allows several representations of the same objects to be modeled at various sizes, contributing to the recognition of objects regardless of their scale.

The mosaic image augmentation technique [33] is a method used to enrich dataset training in machine learning by constructing composite images from multiple source images. It creates a new image by combining four images into one, where a central region of the main image is replaced by a section of each of the other images. This process is intended to expose the model to various scenarios and contexts, providing diverse scenes and object configurations in a single image. Introducing mosaic images to the training set effectively enlarges the range of situations and scenarios the model encounters, improving its ability to recognize and generalize across many contexts. This augmentation technique encourages the model to adapt and learn from a wide variety of compositions and object arrangements, fostering robustness and better performance in real-world applications.

Figure 5 shows several examples of data augmentation techniques applied to the images in the dataset.

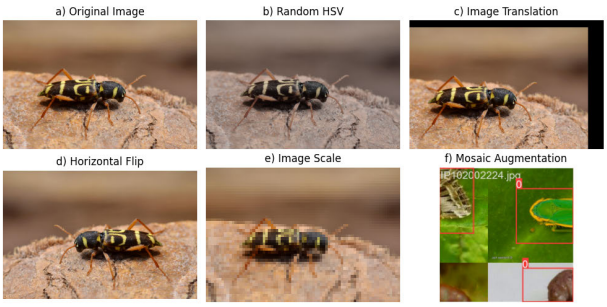

FIGURE 5. Samples of data augmentation. a) original image, b) random HSV adjustments, c) image translation, d) horizontal flip, e) image scaling, and f) a sample of mosaic augmentation.

C. ADOPTED METRICS

In this subsection, we explain the loss functions, along with pertinent metrics such as precision, recall, and mean average precision *mAP*. YOLOv8 uses three loss metrics to characterize the total loss function of the model. They are Box loss, Distribution Focal Loss (*DFL*) [37], and Classification Loss. Box loss is the loss associated with bounding box regression and will be referred to as Bounding Box Loss [21]. Classification Loss is the loss function used for the classification task, also denoted as Cross Entropy Loss [38].

Cross Entropy is a measure of the difference between two probability distributions for a given random variable or set of events. As such, Cross Entropy Loss determines how well the model predictions fit the real class of the object. Cross Entropy Loss is defined as:

$$Cross\ Entropy\ Loss(cls\_loss) = -\frac{1}{N}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}) \tag{2}$$

where N is the number of predictions, C is the number of classes, y_{ij} is 1 if prediction i belongs to class j and 0 otherwise, and $\hat{y}_{ij}$ is the model's predicted probability of object i belonging to class j.

Bounding Box Loss is a metric for object detection. It is used to measure how much the bounding box prediction diverges from the ground-truth box label. It is calculated using the following equation:

$$Bounding\ Box\ Loss(bbox\_loss) = \frac{\sum_{i=1}^{n}(1 - IoU) \times w_i}{n} \tag{3}$$

where n is the number of bounding boxes in an image, w_i is a weight applied to each bounding box and IoU is the Intersection Over Union, a metric denoting how much a predicted bounding box overlaps with the real, ground-truth label. IoU can be determined as:

$$IoU = \frac{Area\ of\ Intersection}{Area\ of\ Union} \tag{4}$$

where the Area of Intersection is the area of overlapping between two bounding boxes, and the Area of Union is the combined area encompassed by both bounding boxes.

Distribution Focal Loss (DFL) is another common metric in the object detection field. It is used to address the issue of class imbalance when detecting objects of different classes. This imbalance occurs when certain classes are more represented than others in a dataset. The definition of Distribution Focal Loss (dfl_loss) is:

$$DFL(P_i, P_{i+1}) = -((y_{i+1} - y)\log(P_i) + (y - y_i)\log(P_{i+1})) \tag{5}$$

where y_i stands for the IoU score of the i-th predicted bounding box, y_{i+1} stands for the IoU score of the next i+1 predicted bounding box, y is the target IoU score, and P_i is the underlying General Distribution of said bounding boxes.

Additionally, we consider the precision, recall, and mean average precision metrics to further evaluate the performance of the model. Precision and recall are computed as follows:

$$P = \frac{T_p}{T_p + F_p} \qquad R = \frac{T_p}{T_p + F_n} \tag{6}$$

where T_p is a true positive, meaning the model correctly identifies a class object; F_p is a false positive, the model predicts an object where there is not one; and F_n is a false negative, the model does not predict an object where there is one. Precision is intuitively the proportion of true positives over total detections, while recall is the proportion of true positives over total objects.

Mean average precision, which is computed considering a confidence threshold, is defined as $m_{0.5}$ [39] and $m_{0.95}$ [40], using as confidence threshold $\tau = 0.5$ and $\tau = 0.95$, respectively. This metric quantifies the area under the precision-recall curve, taking a single confidence value, such as $m_{0.5}$, or using a range, such as $m_{0.95}$, that goes from $\tau = 0.5$ to $\tau = 0.95$ in increments of 0.05. This is a common metric to evaluate the performance of object detection models, and it generally defines how well a model performs in the detection task.

V. RESULTS

In this section, we evaluate the different YOLOv8 models employed in the detection tasks to later discuss their outcomes. In Section V-A, we analyze the training phase, then we analyze the validation results in Section V-B. Finally, the model is tested using images taken from the internet in Section V-C.

A. TRAINING

Every model of YOLOv8 has been trained with the same dataset and same hyper-parameters. The specifications of the computing server are provided in Table 2. Stochastic gradient descent (SGD) is the selected optimizer, and it is influenced by several hyperparameters. An initial learning rate of 0.01 is employed to regulate the step size during optimization, ensuring a balanced convergence. Then momentum, set at 0.937, determines the contribution of the previous gradient to the current step. A weight decay of 0.0005 is applied to act as a regularization term to control overfitting. The models train for 200 epochs, although a patience parameter defining the number of epochs without improvement before terminating the process is set to 50. Only model l sets patience to 5.

TABLE 2. Specifications of the computing server.

Element	Description
OS	Linux
RAM	128 GB
Hard Disk	2 TB
Processor	Intel Xeon Gold 5220 2.2G 18C/36T, 10.4GT/s, 24.75M Cache (2 units)

For all the cases, the loss functions are computed during training, then the same loss functions are computed together in the validation phase with the metrics described in Section IV-C.

For the n model, the values for bounding box loss, classification loss, and distribution focal loss for each epoch are shown in Figure 6. The bounding box loss and the distribution focal loss reach a peak around epoch 3 and decrease from this point. Classification loss (Cross Entropy) declines steadily through the whole training. The final values obtained are $bbox\_loss = 1.01$, $cls\_loss = 0.567$, and $dfl\_loss = 1.32$. Classification loss is the smallest of the three due to the fact that this task is simplified by only having one class.

The s model is the earliest to stop training, as no improvement occurred for 50 epochs. The model trained for 141 epochs, obtaining the best results at epoch 91. The values of loss functions are plotted in Figure 7. There is not much difference between this model's training results and model n's: the loss functions have similar behavior. At the beginning of the training, they reach a peak and then slowly decrease. The final values are $bbox\_loss = 0.99$, $cls\_loss = 0.549$,

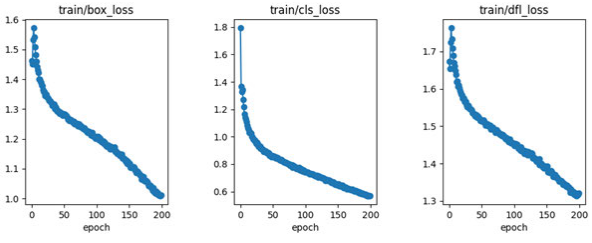

FIGURE 6. Training bounding box loss, training classification loss, and training distribution focal loss for model *n*.

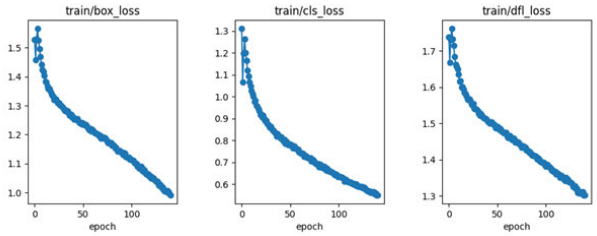

FIGURE 7. Training bounding box loss, training classification loss, and training distribution focal loss for model *s*.

and *dfl_loss* = 1.3, representing a small improvement over model *n*.

Previously, models *n* and *s* triggered Early Stopping, indicating that, as per the patience parameter, training concluded before reaching completion as they did not improve in performance. However, model *m* did not, due to repeated failures during training. These failures involved manually resuming training, interfering with the Early Stopping procedure and, as a result, concluding with the completion of the 200 epochs, as can be seen in Figure 8.

Analyzing the results for model *m*, if we compare them with the validation phase, it is clear that after epoch 80, there is no improvement and performance worsens to the point of overfitting. The values for the training loss functions are as follows: *bbox_loss* = 0.675, *cls_loss* = 0.263, and *dfl_loss* = 1.16.

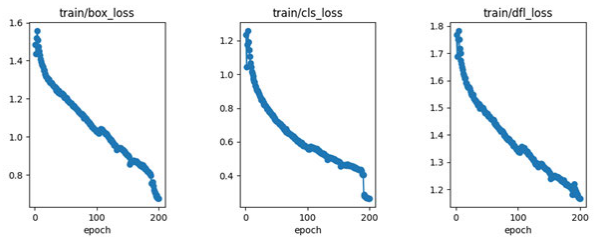

FIGURE 8. Training bounding box loss, training classification loss, and training distribution focal loss for model *m*.

For model *l*, the values for bounding box loss, classification loss, and distribution focal loss for each epoch are shown in Figure 9. The last values registered are the following: *bbox_loss* = 1.14, *cls_loss* = 0.66, and *dfl_loss* = 1.45. Early Stopping was triggered in epoch 71, with the best values obtained in epoch 66. Model *l* has similar performance to the previous models. The training shows that as complexity and depth increase, the results improve. However, this enhancement is accompanied by longer training times.

B. VALIDATION

In this subsection, the validation results are presented and discussed. Specifically, we present the loss functions, precision, recall, and mean average precision (*mAP*) metrics.

Concerning model *n*, the values for bounding box loss, classification loss, and distribution focal loss for each epoch

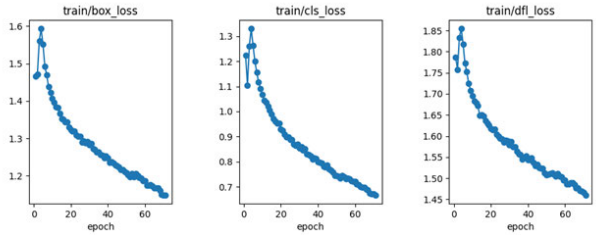

FIGURE 9. Training bounding box loss, training classification loss, and training distribution focal loss for model *l*.

are shown in Figure 10, while precision, recall and *mAP* are shown in Figure 11.

The behavior of the loss functions is similar to the training results. However, while the training values continued to decline, the validation values reached an inflection point around epoch 90, where the bounding box loss and distribution focal loss increase. Classification loss decreases, indicating that the model is improving in the classification task, while it is less accurate in the detection task. This reflects a small overfitting in the bounding box prediction.

This apparent overfitting in the detection task does not remain in the *mAP* values or in accuracy and recall, which generally increase across epochs or stabilize without experiencing a significant decline.

The performance of the model on the validation dataset can be observed in Figure 12. The model accurately predicts 95.5% of the insect instances out of the 5,365 insects available in the validation dataset. Specifically, the model made 5,127 correct predictions (true positives), while 238 insect instances were missed, representing only 4.5% of the total instances. Calculating the percentage of false positives is challenging, as the dataset lacks instances of other classes, and the false positives were cases where the model mistook the background for an object. This circumstance occurred 412 times out of 5,777 predictions made by the model, constituting 7.1% of the total predictions.

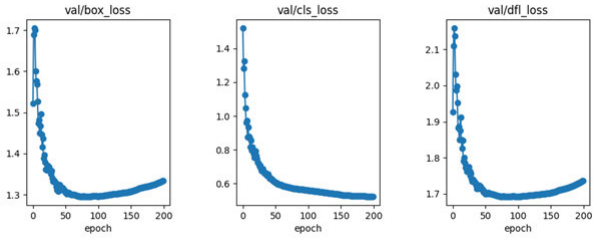

FIGURE 10. Validation bounding box loss, validation classification loss, and validation distribution focal loss for model *n*.

Regarding model *s*, the values for bounding box loss, classification loss, and distribution focal loss for each epoch can be found in Figure 13, while precision, recall, and *mAP* are represented in Figure 14. Similar values and behavior can be observed in models *n* and *s*, with a slight insignificant improvement in performance seen for the latter, resulting in higher $mAP_{0.5}$ and $mAP_{0.95}$ values.

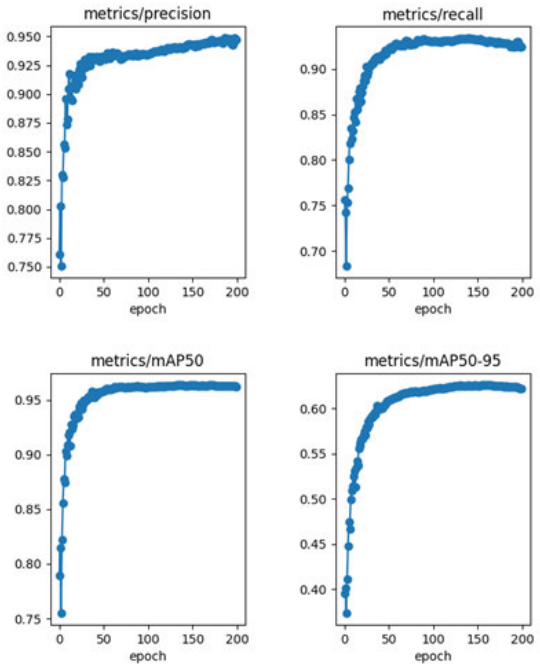

FIGURE 11. Precision, recall, $mAP_{0.5}$, and $mAP_{0.95}$ for model *n*.

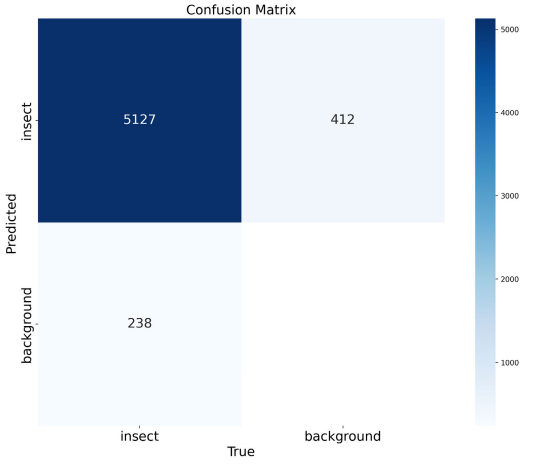

FIGURE 12. Confusion matrix for model *n*.

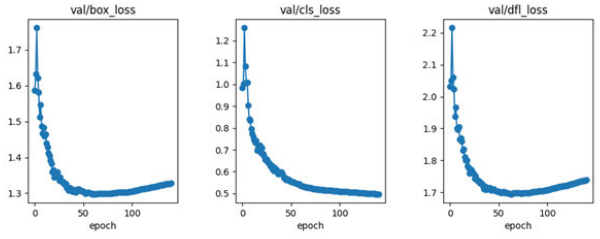

FIGURE 13. Validation bounding box loss, validation classification loss, and validation distribution focal loss for model *s*.

Figure 15 illustrates the confusion matrix for the *s* model, which yields valuable insights. The model obtains 5,166 true positives, 199 false negatives, and 432 false positives.

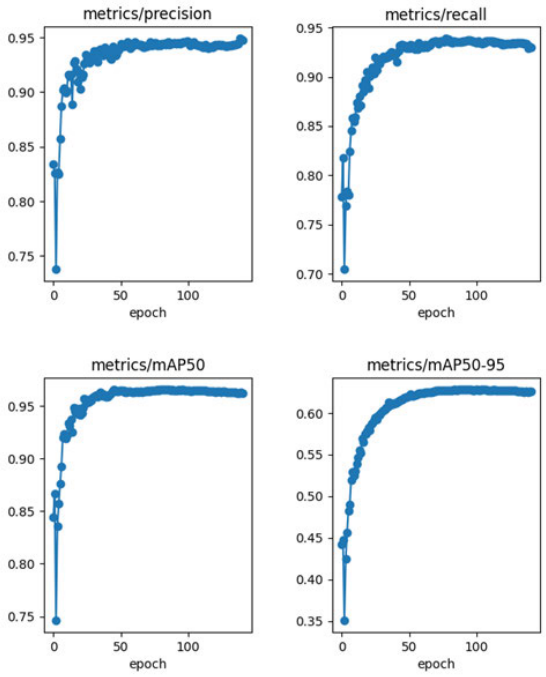

FIGURE 14. Precision, recall, $mAP_{0.5}$, and $mAP_{0.95}$ for model *s*.

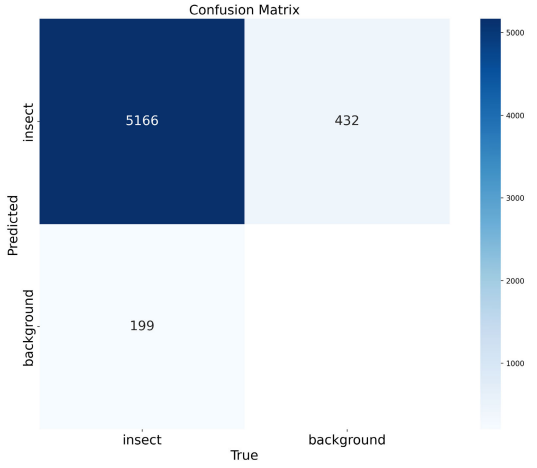

FIGURE 15. Confusion matrix for model *s*.

In comparison to model *n*, this model achieves a higher number of detections. However, given the constant number of objects "insect" included in the dataset, it is more susceptible to identifying backgrounds as insects, with 7.4% of the predictions resulting in false positives. This represents an increase of 0.3% compared to model *n*. The true positives and false negatives exhibit similar results to model *n*, with 96.29% and 3.71%, respectively.

With regard to model *m*, the values for bounding box loss, classification loss, and distribution focal loss for each epoch can be found in Figure 16, while precision, recall, and *mAP* are depicted in Figure 17. Model *m* continues with the trend of slight improvement in the $mAP_{0.5}$ and $mAP_{0.95}$ values. The overfitting is much more evident, while the classification loss

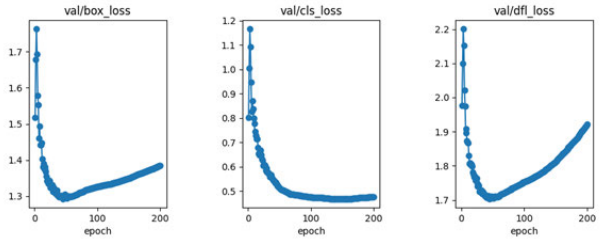

FIGURE 16. Validation bounding box loss, validation classification loss, and validation distribution focal loss for model *m*.

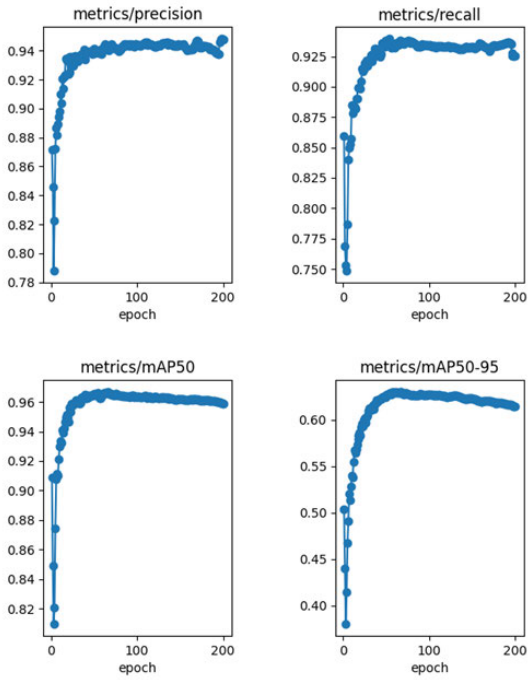

FIGURE 17. Precision, recall, $mAP_{0.5}$, and $mAP_{0.95}$ for model *m*.

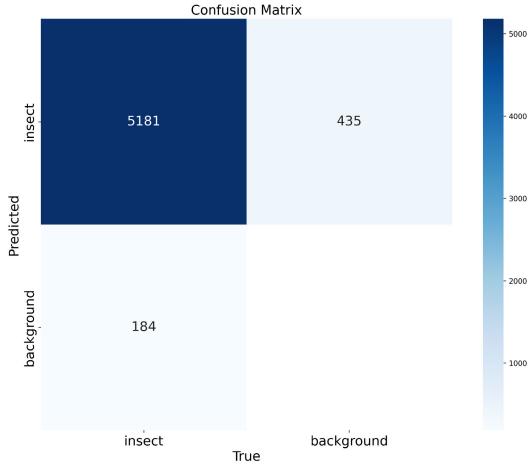

FIGURE 18. Confusion matrix for model *m*.

stagnates, showing that further training does not bring about any improvement.

Now, we delve into the confusion matrix for model *m*. Figure 18 gives a detailed breakdown of the model's performance. Model *m* achieves 5,181 true positives, 184 false negatives, and 435 false positives. So, model *m* records 3 more false positives than model *s*, verifying that both models are equally prone to confusing backgrounds with insects. However, model *m* performs better in predicting insects, achieving a success rate of 96.57% with only 3.48% of the cases undetected. This performance aligns with the observed trend of slight improvements in the models as they scale in size.

Finally, model *l* obtained the best $mAP_{0.95}$ and, thanks to the Early Stopping procedure, minimal overfitting. Values for bounding box loss, classification loss, and distribution focal loss for each epoch are represented in Figure 19 and precision, recall, and *mAP* values are shown in Figure 20. Furthermore, in Figure 21, we plot the confusion matrix, where, due to the shorter training duration for this model, its behavior differs from that of models *m* and *s*. While the above two models: i) tend to increase the number of false positives and ii) also increase the cases in which the model accurately predicted insects, model *l* erroneously predicts the background as an insect in only 377 cases, the lowest value among the models tested. With 5,157 true positives and 208 false negatives, the model correctly predicted 96% of the cases and failed 4%. This represents the second-worst performance among all the models, which contrasts with the fact that model *l* obtained the highest $mAP_{0.95}$ value.

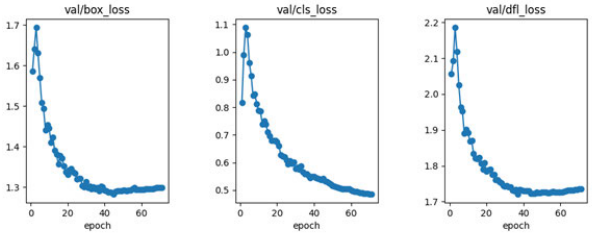

FIGURE 19. Validation bounding box loss, validation classification loss, and validation distribution focal loss for model *l*.

A summary of each model's performance is given in Table 3, where the best results for each metric are highlighted. A visual representation is also shown in Figure 22.

TABLE 3. YOLOv8 performance metrics.

Model	Precision	Recall	mAP50	mAP50-95	F1 Score
n	0.944	0.93	0.963	0.626	0.934
s	**0.945**	0.936	0.965	0.628	**0.94**
m	0.942	0.938	**0.967**	0.63	0.939
l	0.943	**0.939**	0.966	**0.632**	**0.94**

C. TEST

In this section, we aim to assess the robustness and generalization capabilities of the CNN presented here by providing images not available in the training dataset as input. The objective is to validate the model's performance by using

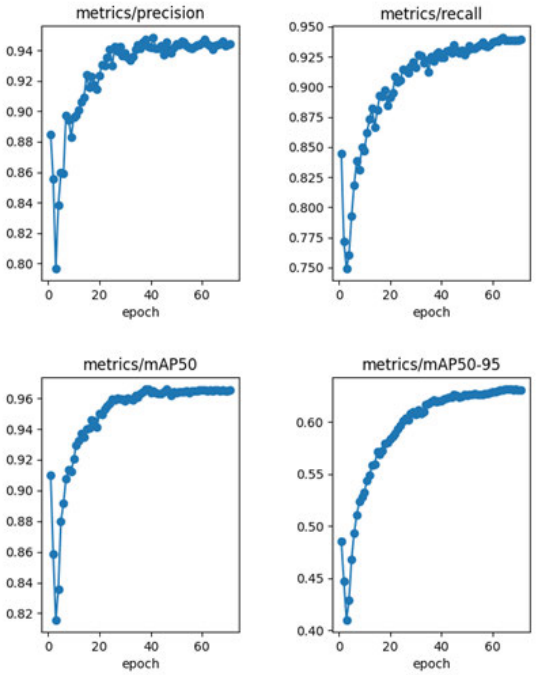

FIGURE 20. Precision, recall, $mAP_{0.5}$, and $mAP_{0.95}$ for model *l*.

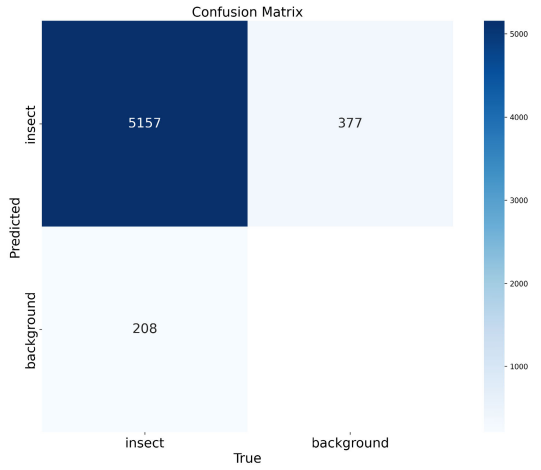

FIGURE 21. Confusion matrix for model *l*.

new, different images, thereby testing its functionality in real-world scenarios. Our analysis of the results will suggest the generalization capacity of the models, identify potential for improvement, and contribute valuable insights to enhance the model's applicability in practical scenarios not covered by the dataset.

The model executes inference on images of different types of insects with varying conditions. The predictions obtained by the Neural Network can be observed in Figure 23.

The model performs as predicted, detecting insects in images that were not included in the original dataset, although the confidence of the predictions is not high. To further evaluate the performance of the model, it is tested against even more non-dataset images, as shown in Figure 24.

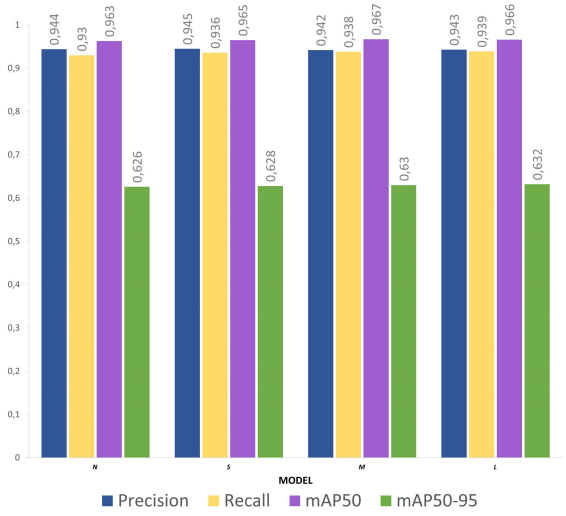

FIGURE 22. Precision, recall, mAP_{50}, and mAP_{50-95} per model.

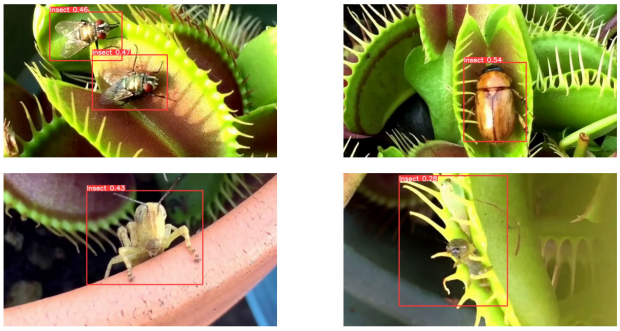

FIGURE 23. Sample 1 of test data. Bounding box predictions in red with the confidence of the prediction.

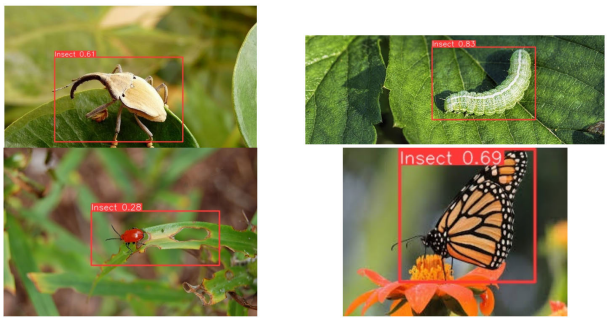

FIGURE 24. Sample 2 of test data. Bounding box predictions in red with the confidence of the prediction.

Although the model performs satisfactorily, some false positives and false negatives remain. Samples of these failures can be seen in Figure 25, in which the upper right and upper left images are examples of false positives and overlapping bounding boxes. In the upper right image, the model mistakes the shape and color of the leaf for an insect, giving even more confidence to this prediction than to the actual fly captured in the image. The upper left image has two valid

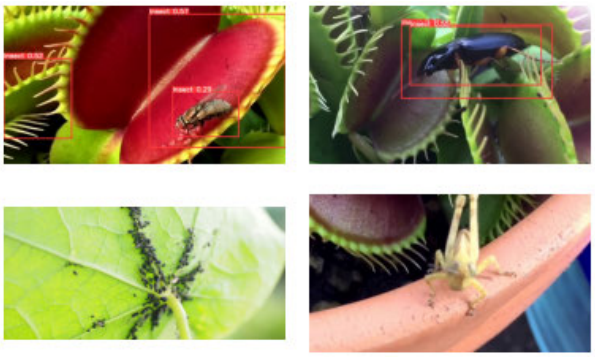

FIGURE 25. Samples of failed predictions on test data. Bounding box predictions in red with the confidence of the prediction.

overlapping bounding boxes that should have been removed during post-processing using the Non Maximum Supression method, although this is a minor failure as the insect is properly detected.

The bottom two images are false negatives, meaning that the model is not able to recognize the insects that are in the picture. In the case of the lower right image, the model does not detect the grasshopper in this particular position, but as observed in Figure 23, it is detected in other cases. It is therefore beneficial to supplement the training datasets with a wider variety of insect poses since their detection requires capturing the target as it moves, adopting different perspectives, and the model may have difficulties in this regard.

As for the lower left image, this false negative is surprising since aphids, the insects in the image, are one of the most represented species in the dataset. This contradiction between the representation of the species in the dataset and the failed detection is due to the distance at which the image is taken. The vast majority of the images in the dataset are close-ups of insects that feed the model with detailed views for accurate identification. However, the lower left image deviates from this trend by presenting a more distant perspective. This variation in distance has contributed to the inability of the model to recognize the aphids in the image. The dataset, which is dominated by close-ups, suggests that the model may have problems with insect detection in scenarios involving greater distances. This underscores the importance of extending the dataset to incorporate a wider range of perspectives, specifically including images captured from more distant viewpoints. By enriching the dataset with such variations, the model can be trained to better generalize across different spatial contexts, improving its performance in scenarios where insects are observed at varying distances.

This generic approach for insect detection is appropriate for real-time use in agricultural fields as shown by the average inference time of the set of test images that was recorded for each model (See Table 4). The test was performed in a desktop computer with the specifications detailed in Table 5. As it is expected, lighter models (*n* and *s*) have lower inference time values, which make them more suitable for computation at the edge in devices with limited resources.

TABLE 4. Inference time recorded for each model.

Model	Inference time (*ms*)
n	103.5
s	140.8
m	297.1
l	521.5

TABLE 5. Specifications of the desktop computer.

Element	Description
OS	Microsoft Windows 10 Pro
RAM	16 GB
HHD	2 TB
SSD	240 GB
Processor	Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz, 3001MHz (4 core)

VI. CONCLUSION AND FUTURE WORK

Pests in crops can cause serious damage that reduce yields and calls into question further investment in PA systems. In this paper, we explored the challenges associated with PA insect detection, testing the efficacy of the YOLOv8 architecture, an open-source deep-learning (CCN) framework, for this specific domain. The complexities inherent in insect detection, arising from variations in species, environmental conditions, and spatial perspectives have been exhaustively explored. Furthermore, there is an added difficulty in acquiring images in real environments, without resorting to traps and other type of devices to capture insects, due to several factors, such as the scale of the insect compared to the plant leaves, obstructions from leaves and branches, inaccessible positioning of insects on the underside of leaves and lower parts of the plants, variations in lighting, and weather conditions. These factors can contribute to a reduced visibility and clarity of the resulting picture that interferes with the adequate operation of an AI model. This can be mitigated using data augmentation techniques to transform the images into a more suitable input for the model. Additionally, even when an adequate input image from the data acquisition device or resulting from data augmentation techniques is fed to an AI model, if the solution is based on classifying the insect from a reduced number of possible insect types, there is an added difficulty that can render the model useless for farmers. If an insect is captured in an image but the model fails to classify it into one of the insect classes, the output would suggest that there is no presence of insects when the truth is the opposite. The approach presented in this paper consists of following a generic strategy for insect detection instead of classification. This would avoid giving farmers a false sense of security as they would be alerted of the presence of any type of insect. All the insect classes in the dataset have been grouped into a single insect dataset to facilitate insect detection for general purpose. This generalization provides more flexibility and enables the model to adapt to a wider range of scenarios, extending

its real-time detection capabilities to: i) encompass insect species beyond those included in the dataset and ii) consider any arbitrary crop type. The comprehensive analysis of the dataset has provided valuable insights into the limitations and potential deviation that may influence the model's performance. Although YOLOv8 demonstrated appropriate performance in certain scenarios, with an $mAP_{0.95}$ metric value of 0.632 for model *l* and an $mAP_{0.5}$ value of 0.967 for model *m*, challenges arose when dealing with variations in spatial distances. This highlights the importance of incorporating diversity into the dataset to generalize the model under multiple conditions. The findings underscore the need to continuously refine detection models, considering the dynamic nature of insect-related datasets. Therefore, our study contributes valuable insights to the ongoing research on insect detection methodologies, emphasizing the importance of the type of information contained in the dataset and model architecture considerations.

Future work will focus on improving the detection of insects as a general class by creating a dataset with much more variability in aspects such as insect perspectives, location, and distance. In addition, we are working to create an automated pest-detection system for PA that gathers real-time images from cameras directly deployed in the fields, analyzes them, and makes intelligent decisions, issuing appropriate alerts when insects are detected.

REFERENCES

[1] B. Davis, E. Mane, L.Y. Gurbuzer, G. Caivano, N. Piedrahita, K. Schneider, N. Azhar, M. Benali, N. Chaudhary, R. Rivera, R. Ambikapathi, and P. Winters, *Estimating Global and Country Level Employment in Agri-Food Systems* (FAO Statistics Working Paper Series). Rome, Italy: FAO, 2023, doi: 10.4060/cc4337en.

[2] *Pest and Pesticide Management, Food and Agriculture Organization of the United Nations*. Accessed: Jan. 10, 2024. [Online]. Available: https://www.fao.org/pest-and-pesticide-management/about/understanding-the-context

[3] R. Kestur, S. N. Omar, and S. Subhash, "Unmanned aerial system technologies for pesticide spraying," in *Innovative Pest Management Approaches for the 21st Century*. Singapore: Springer, 2020, pp. 37–60, doi: 10.1007/978-981-15-0794-6_3.

[4] M. Abbas, M. Ramzan, N. Hussain, A. Ghaffar, K. Hussain, S. Abbas, and A. Raza, "Role of light traps in attracting, killing and biodiversity studies of insect pests in Thal," *Pakistan J. Agricult. Res.*, vol. 32, no. 4, pp. 684–690, 2019.

[5] P. Trematerra and M. Colacci, "Recent advances in management by pheromones of thaumetopoea moths in urban parks and woodland recreational areas," *Insects*, vol. 10, no. 11, p. 395, Nov. 2019, doi: 10.3390/insects10110395.

[6] T. R. E. Southwood and P. A. Henderson, *Ecological Methods*. Hoboken, NJ, USA: Wiley, 2009.

[7] K. Ennouri, S. Smaoui, Y. Gharbi, M. Cheffi, O. Ben Braiek, M. Ennouri, and M. A. Triki, "Usage of artificial intelligence and remote sensing as efficient devices to increase agricultural system yields," *J. Food Qual.*, vol. 2021, pp. 1–17, Jun. 2021, doi: 10.1155/2021/6242288.

[8] H. Tian, T. Wang, Y. Liu, X. Qiao, and Y. Li, "Computer vision technology in agricultural automation—A review," *Inf. Process. Agricult.*, vol. 7, no. 1, pp. 1–19, Mar. 2020, doi: 10.1016/j.inpa.2019.09.006.

[9] Y. Lu and S. Young, "A survey of public datasets for computer vision tasks in precision agriculture," *Comput. Electron. Agricult.*, vol. 178, Nov. 2020, Art. no. 105760, doi: 10.1016/j.compag.2020.105760.

[10] W. Zhang, Z. Miao, N. Li, C. He, and T. Sun, "Review of current robotic approaches for precision weed management," *Current Robot. Rep.*, vol. 3, no. 3, pp. 139–151, Jul. 2022, doi: 10.1007/s43154-022-00086-5.

[11] H. C. Bazame, J. P. Molin, D. Althoff, and M. Martello, "Detection, classification, and mapping of coffee fruits during harvest with computer vision," *Comput. Electron. Agricult.*, vol. 183, Apr. 2021, Art. no. 106066, doi: 10.1016/j.compag.2021.106066.

[12] A. Gutierrez, A. Ansuategi, L. Susperregi, C. Tubío, I. Rankić, and L. Lenža, "A benchmarking of learning strategies for pest detection and identification on tomato plants for autonomous scouting robots using internal databases," *J. Sensors*, vol. 2019, pp. 1–15, May 2019, doi: 10.1155/2019/5219471.

[13] W. Li, D. Wang, M. Li, Y. Gao, J. Wu, and X. Yang, "Field detection of tiny pests from sticky trap images using deep learning in agricultural greenhouse," *Comput. Electron. Agricult.*, vol. 183, Apr. 2021, Art. no. 106048, doi: 10.1016/j.compag.2021.106048.

[14] M. E. Karar, F. Alsunaydi, S. Albusaymi, and S. Alotaibi, "A new mobile application of agricultural pests recognition using deep learning in cloud computing system," *Alexandria Eng. J.*, vol. 60, no. 5, pp. 4423–4432, Oct. 2021, doi: 10.1016/j.aej.2021.03.009.

[15] I. Ahmad, Y. Yang, Y. Yue, C. Ye, M. Hassan, X. Cheng, Y. Wu, and Y. Zhang, "Deep learning based detector YOLOv5 for identifying insect pests," *Appl. Sci.*, vol. 12, no. 19, p. 10167, Oct. 2022, doi: 10.3390/app121910167.

[16] J. Du, "Understanding of object detection based on CNN family and YOLO," *J. Phys., Conf. Ser.*, vol. 1004, Apr. 2018, Art. no. 012029, doi: 10.1088/1742-6596/1004/1/012029.

[17] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Mach. Learn. Knowl. Extraction*, vol. 5, no. 4, pp. 1680–1716, Nov. 2023, doi: 10.3390/make5040083.

[18] K. Li, J. Zhu, and N. Li, "Insect detection and counting based on YOLOv3 model," in *Proc. IEEE 4th Int. Conf. Electron. Technol. (ICET)*, Chengdu, China, Sep. 2021, pp. 1229–1233, doi: 10.1109/ICET51757.2021.9450898.

[19] E. Önler, "Real time pest detection using YOLOv5," *Int. J. Agricult. Natural Sci.*, vol. 14, no. 3, pp. 232–246, 2021.

[20] Z. Yang, H. Feng, Y. Ruan, and X. Weng, "Tea tree pest detection algorithm based on improved YOLOv7-tiny," *Agriculture*, vol. 13, no. 5, p. 1031, May 2023, doi: 10.3390/agriculture13051031.

[21] G. Jocher, A. Chaurasia, and J. Qi. *YOLO By Ultralytics*. Accessed: Nov. 6, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[22] T. Kasinathan, D. Singaraju, and S. R. Uyyala, "Insect classification and detection in field crops using modern machine learning techniques," *Inf. Process. Agricult.*, vol. 8, no. 3, pp. 446–457, Sep. 2021, doi: 10.1016/j.inpa.2020.09.006.

[23] Y. He, H. Zeng, Y. Fan, S. Ji, and J. Wu, "Application of deep learning in integrated pest management: A real-time system for detection and diagnosis of oilseed rape pests," *Mobile Inf. Syst.*, vol. 2019, pp. 1–14, Jul. 2019, doi: 10.1155/2019/4570808.

[24] A. C. Teixeira, J. Ribeiro, R. Morais, J. J. Sousa, and A. Cunha, "A systematic review on automatic insect detection using deep learning," *Agriculture*, vol. 13, no. 3, p. 713, Mar. 2023, doi: 10.3390/agriculture13030713.

[25] Y. He, Z. Zhou, L. Tian, Y. Liu, and X. Luo, "Brown Rice planthopper (*Nilaparvata lugens* Stal) detection based on deep learning," *Precis. Agricult.*, vol. 21, no. 6, pp. 1385–1402, Dec. 2020, doi: 10.1007/s11119-020-09726-2.

[26] J.-W. Chen, W.-J. Lin, H.-J. Cheng, C.-L. Hung, C.-Y. Lin, and S.-P. Chen, "A smartphone-based application for scale pest detection using multiple-object detection methods," *Electronics*, vol. 10, no. 4, p. 372, Feb. 2021, doi: 10.3390/electronics10040372.

[27] C.-J. Chen, Y.-Y. Huang, Y.-S. Li, C.-Y. Chang, and Y.-M. Huang, "An AIoT based smart agricultural system for pests detection," *IEEE Access*, vol. 8, pp. 180750–180761, 2020, doi: 10.1109/ACCESS.2020.3024891.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, Jun. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[29] (2023). *YOLOv8 Architecture Visualization, RangeKing*. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/189

[30] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," 2016, *arXiv:1606.08415*.

[31] N. E. Stork, "World of insects," *Nature*, vol. 448, no. 7154, pp. 657–658, Aug. 2007, doi: 10.1038/448657a.

[32] X. Wu, C. Zhan, Y. Lai, M. M. Cheng, and J. Yang, "IP102: A large-scale benchmark dataset for insect pest recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Long Beach, CA, USA, Jun. 2019, pp. 8787–8796, doi: 10.1109/CVPR.2019.00899.
[33] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.
[34] G. H. Joblove and D. Greenberg, "Color spaces for computer graphics," *ACM SIGGRAPH Comput. Graph.*, vol. 12, no. 3, pp. 20–25, Aug. 1978, doi: 10.1145/965139.807362.
[35] H. Zhao, H. Li, and L. Cheng, "Data augmentation for medical image analysis," in *Biomedical Image Synthesis and Simulation*, N. Burgos and D. Svoboda, EdS. Cambridge, MA, USA: Academic, 2022, pp. 279–302.
[36] Y. Cha, W. Choi, G. Suh, S. Mahmoudkhani, and O. Büyüköztürk, "Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 33, no. 9, pp. 731–747, Sep. 2018, doi: 10.1111/mice.12334.
[37] X. Li, C. Lv, W. Wang, G. Li, L. Yang, and J. Yang, "Generalized focal loss: Towards efficient representation learning for dense object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 3139–3153, Mar. 2023, doi: 10.1109/TPAMI.2022.3180392.
[38] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics), 1st ed. New York, NY, USA: Springer, 2007.
[39] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn and A. Zisserman. (2010). *The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results*. [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html
[40] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. Lawrence Zitnick, and P. Dollár, "Microsoft COCO: Common objects in context," 2014, *arXiv:1405.0312*.

MARIO VILAR-ANDREU received the bachelor's degree in telematic engineering and the master's degree in telecommunications engineering from Universidad Politécnica de Cartagena, in 2020 and 2022, respectively. His research interests include computer vision, machine learning, object detection, deep learning, and the IoT.

LAURA GARCÍA received the bachelor's degree in telecommunications technology engineering and the master's degree in digital post production from the Polytechnic University of Valencia, in 2015 and 2016, respectively, the master's degree in business administration from Universidad Católica San Antonio de Murcia, in 2020, the Ph.D. degree in telecommunications from the Polytechnic University of Valencia, and the Ph.D. degree in computer science from Université de Mulhouse Haute-Alsace, in September 2021. She is the author or coauthor of 23 articles in international journals listed in the J.C.R., and 11 articles in other international journals. She is the coauthor of two book chapters. She has published 23 conference papers. She has been involved in several organization committees of international conferences, since 2016. Her research interests include precision agriculture and the IoT-based environmental monitoring systems, and the design of communication architectures and protocols for the aforementioned purposes.

ANTONIO-JAVIER GARCIA-SANCHEZ received the M.S. degree in industrial engineering from Universidad Politécnica de Cartagena (UPCT), Cartagena, Spain, in 2000, and the Ph.D. degree from the Department of Information Technologies and Communications (DTIC), UPCT, in 2005. Currently, he is a Full Professor with UPCT. He is the coauthor of more than 100 conference papers and journal articles, 60 of them indexed in *Journal Citation Report* (JCR). He has been the Head of several EU research projects in the field of communication networks and optimization. He is also an inventor/the co-inventor of 12 patents or utility models. He has been a Visiting Scholar with Bologna University, Bologna, Italy, in 2007; Wageningen University, Wageningen, The Netherlands, in 2012; and Cali University, Cali, Colombia, in 2019. His main research interests include wireless sensor networks (WSNs), streaming services, artificial intelligence, the Internet of Things (IoT), and nanonetworks. He has been a TPC member or the chair in about 40 international congresses or workshops. He is also a reviewer of several journals listed in the ISI-JCR.

RAFAEL ASOREY-CACHEDA (Member, IEEE) received the M.Sc. degree in telecommunication engineering, major in telematics, and the Ph.D. degree (cum laude) in telecommunication engineering from Universidade de Vigo, Spain, in 2006 and 2009, respectively. He was a Researcher with the Information Technologies Group, Universidade de Vigo, until 2009. From 2008 to 2009, he was a Research and Development Manager with Optare Solutions, a Spanish telecommunications company. He was a Visiting Scholar with New Mexico State University, USA, from 2007 to 2011, and Universidad Politécnica de Cartagena, Spain, from 2011 to 2015. From 2009 to 2012, he held an Ángeles Álvari no position with Xunta de Galicia, Spain. From 2012 to 2018, he was an Associate Professor with Centro Universitario de la Defensa en la Escuela Naval Militar, Universidade de Vigo. He is currently an Associate Professor with Universidad Politécnica de Cartagena. He is the author or coauthor of more than 70 journal and conference papers, mainly in the fields of switching, wireless networking, and content distribution. His research interests include content distribution, high-performance switching, peer-to-peer networking, wireless networks, and nano-networks. He received the Best Master Thesis Award for the M.Sc. degree and the Best Ph.D. Thesis Award for the Ph.D. degree.

JOAN GARCIA-HARO (Member, IEEE) received the M.S. and Ph.D. degrees in telecommunication engineering from Universitat Politécnica de Catalunya, Barcelona, Spain, in 1989 and 1995, respectively. He has been a Visiting Scholar with Queen's University, Kingston, ON, Canada, from 1991 to 1992, and Cornell University, Ithaca, NY, USA, from 2010 to 2011. He is currently a Professor with Universidad Politécnica de Cartagena (UPCT), Cartagena, Spain. He is the author or coauthor of more than 70 journal articles mainly in the fields of switching, wireless networking, and performance evaluation. He received the Honorable Mention for the IEEE Communications Society Best Tutorial Paper Award, in 1995. He served as the Editor-in-Chief for the IEEE Global Communications Newsletter, including in the *IEEE Communications Magazine*, from April 2002 to December 2004. He has been a Technical Editor of the *IEEE Communications Magazine*, from March 2001 to December 2011.