IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 5 June 2024, accepted 21 June 2024, date of publication 27 June 2024, date of current version 8 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3419835

RESEARCH ARTICLE

Palm Oil Counter: State-of-the-Art Deep Learning Models for Detection and Counting in Plantations

MARTINUS GRADY NAFTALI[1], GREGORY HUGO[1], AND SUHARJITO[2], (Member, IEEE)
[1]Computer Science Department, BINUS Graduate Program–Master of Computer Science, Bina Nusantara University, Jakarta 11480, Indonesia
[2]Industrial Engineering Department, BINUS Graduate Program–Master of Industrial Engineering, Bina Nusantara University, Jakarta 11480, Indonesia

Corresponding author: Suharjito (suharjito@binus.edu)

This work was supported in part by the Ministry of Education, Culture, Research, and Technology in Indonesia; and in part by the Directorate General of Higher Education under Contract 1165/LL3/AL.04/2023.

ABSTRACT Traditional palm oil production methods for evaluating fruit bunches (FFBs) are inefficient, costly, and have limited coverage. This study evaluates the performance of various YOLO models and other state-of-the-art object detection models using a novel dataset of oil palm fresh fruit bunches in plantations, captured in the plantation regions of Central Kalimantan Province, Indonesia. The dataset includes five ripeness classes (abnormal, ripe, underripe, unripe, and flower) and presents challenges such as partially visible objects, low contrast scenes, occluded and small objects, and blurry images. The proposed YOLOv8s Depthwise model was compared with other YOLO models, including YOLOv6s, YOLOv6l, YOLOv7 Tiny, YOLOv7l, YOLOv8s, and YOLOv8l. YOLOv8s Depthwise demonstrated a balanced performance, with a compact size (10.6 MB), fast inference time (0.027 seconds), and strong detection accuracy (mAP50 at 0.75, mAP50-95 at 0.481). Its rapid convergence and low training loss highlighted its efficiency, completing training in the shortest time of 2 hours, 18 minutes, and 30 seconds. Furthermore, it achieved low Mean Absolute Error (MAE) of 0.164 and Root Mean Square Error (RMSE) of 0.4, indicating precise counting capability. Hyperparameter tuning revealed that the YOLOv8s Depthwise model achieved optimal performance using the SGD optimizer with a batch size of 16 and a learning rate of 0.001, showing the best balance between accuracy and training efficiency. Data augmentation positively impacted model performance, resulting in improved performance metrics across various models. When evaluated against other state-of-the-art models on the same dataset, including Faster RCNN, SSD MobileNetV2, YOLOv4, and YOLOv9, YOLOv8s Depthwise surpassed other state-of-the-art models, including Faster R-CNN, SSD MobileNetV2, YOLOv4, and EfficientDet-D0 from previous research, in terms of speed, accuracy, and efficiency, making it ideal for real-time palm oil harvesting applications.

INDEX TERMS Deep learning, object counting, palm oil ripeness, real-time object detection, YOLO.

I. INTRODUCTION

Palm oil, one of the essentials of agriculture in Southeast Asia, is also a global market leader. Its affordability and versatility have fueled its rise, driven by increasing global demand for oleochemicals [1], [2], [3]. These come from oil palm trees, whose fruit bunches (FFBs) are crucial for oil extraction. Peak OER (Oil Extraction Rate) relies heavily on harvesting FFBs at optimal ripeness [4]. Indonesia and Malaysia dominate palm oil production, together accounting for 80% globally. It's a significant economic driver for these nations, including Thailand, Colombia, and Nigeria. Notably, Malaysia holds a 25.8% production share and a 34.3% export share [5], [6]. Indonesia boasts an even larger 45.6 million tons annually, translating to a 59% global production share [5], [6], [7].

Delving in further, palm oil stands out as one of the world's most productive oilseed crops. It surpasses other options like soybean, sunflower, and rapeseed in terms of oil yield per hectare. This translates to a cost-effective

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

advantage, making palm oil a highly attractive choice for the oleochemical industry [8], [9]. Beyond its economic benefits, palm oil offers unique characteristics that contribute to its widespread use. It is virtually odourless and boasts remarkable versatility, finding its way into a vast array of everyday products lining supermarket shelves. From essential cooking oils and delectable snacks like Oreos to creamy margarines, shampoos, soaps, and lotions, palm oil's applications are truly diverse. To add to that, palm oil is also refined for biodiesel and other fatty compounds, highlighting the industry's complexity and scale [10].This intricate nature is further emphasized by the structural characteristics of FFBs. Each bunch consists of a central stem holding individual fruits situated on its outer layer [11], which requires for accurate identification and collection for enhancing production efficiency and contribute significantly to a higher OER [12].

Traditionally, the evaluation of palm oil fruit bunches (FFBs) involves a manual classification technique introduced by [13]. This method assesses the maturity stages of FFBs based on objective parameters such as colour, fruitlets moisture, and oil content. Additionally, a method for black bunch counting entails surveying bunches approximately 1.5 months after the flowering stage at a selected oil palm tree. The recommended sampling involves 5 to 10 percent of the total trees within a 0.3 to 0.5 km2 block area. To estimate FFB yield, multiply the total counted bunches with the estimated bunch weight [13]. However, the traditional method, which relies on manual counting, encounters inherent challenges, including high operational costs, time inefficiency, and limited coverage [2]. Moreover, manual assessment of harvest yield tends to be problematic due to inconsistent and subjective ripeness assessment, leading to frequent disputes between harvesters and customers [14]. The complexity arises from the diverse shapes and ripeness stages of FFBs, making precise visual detection challenging. On top of that, one of the oil palm industry's current practice of using commercial colour meters adds another layer of inefficiency. Not only are they inaccurate, but they also damage the fruits themselves [15].

The palm oil yield estimation process requires innovative automated solutions to improve accuracy and streamline the process. Advancements in artificial intelligence, particularly machine vision-based technologies, offer a more efficient alternative to manual harvesting [16]. In spite of conventional machine learning classifiers having exhibited potential in identifying crops, their efficacy is often hampered by the labour-intensive process of feature extraction [17], [18], [19], [20]. Conventional machine learning methods rely on techniques like colour model conversion, thresholding, histogram equalization, spatial filtering with Laplace and Sobel operators, and Gaussian blur to extract features, with some models explicitly incorporating colour space as a feature [19]. In contrast, deep learning, particularly Convolutional Neural Networks (CNNs), revolutionizes the feature extraction task, presenting a substantial advancement over traditional methods [21], [22]. Drawing inspiration from the human brain's mechanisms, deep learning has found widespread applications in visual imagery analysis [23].

Exploring deeper into [23], it becomes evident that in deep learning, automated feature extraction through deep convolutional neural networks (DCNNs) eradicates the necessity for manual feature engineering. Through iterative learning, DCNNs learn and extract relevant features from input data, offering distinct advantages over conventional machine learning methods. This automated approach enhances classification accuracy and avoids labour-intensive and time-consuming manual feature extraction processes. Techniques employed in automating feature extraction in DCNNs include visual saliency modelling, unsupervised pre-training, and specialized modules like the Inception module. These techniques empower the network to autonomously learn and extract pertinent features from the input data, resulting in superior performance compared to conventional machine learning methods.

In recent years, there has been substantial growth in utilizing deep learning techniques for detecting objects in agricultural images [24], [25]. This is due in part to the fact that deep learning models have achieved significantly higher accuracy in object detection tasks. Additionally, these models excel at handling large and complex datasets [26]. However, existing deep learning models for FFB detection still face limitations, such as constrained ripeness level classification and a scarcity of comprehensive training datasets [27], [28]. Previous studies have primarily explored datasets of harvested fruits rather than those still attached to trees [3]. Although there is one study from 2024 that focuses on a plantation setting, it only addresses classification, making it unsuitable for real-time detection and counting [29]. This lack of diverse data, particularly for fruits still on trees, further hinders the ability of these models to accurately detect FFBs in real-world scenarios.

For models designed to support the FFB harvesting process, real-time execution and efficient resource utilization are vital requirements. Certain systems, like YOLOv4, demonstrate diminished performance when adapted for real-time mobile devices [3], [4], but it is still resource-efficient, nonetheless. While models like Mask R-CNN and Faster R-CNN boast high precision capabilities, their computational demands render them impractical for real-time mobile applications [30], [31].

Within the agricultural domain, a growing body of research highlights the application of real-time object detection for various tasks. Notably, these studies demonstrate a strong preference for YOLO models as the preferred solution [32], [33], [34], [35], [36]. Particularly for FFB harvesting, this emphasis on real-time and efficient object detection has led to a strong inclination for YOLO models, as evidenced by research focusing on FFB ripeness detection using these models [3], [4], [16], [37]. In addition, YOLO's single-stage

architecture streamlines object detection, enabling real-time performance through efficient processing in one network pass. This contrasts with two-stage detectors and traditional machine learning models, like [30], [31], [37], offering a significant advantage for resource-constrained environments.

Researchers have also developed automatic counting systems for counting oil palm tree yield, utilizing remote sensing imagery and deep learning techniques, achieving high accuracy rates [2]. Smartphone technology has also been harnessed for fruit bunch enumeration, addressing challenges like embezzlement during harvesting seasons [16]. Accurately counting FFBs on trees is a challenge due to overlapping bunches, tight clusters, and obstructed views (occlusion). Developing solutions for efficient FFB counting can significantly improve the harvesting process.

This paper proposes an innovative approach to palm oil bunch estimation, focusing on the recently developed YOLOv8 Depthwise model. This model leverages advancements in deep learning techniques, combining the strengths of previous YOLO versions with depthwise separable convolutions to enhance efficiency and accuracy. The choice to propose YOLOv8 Depthwise stems from its significant improvements over its predecessors, making it well-suited for addressing the complexities and challenges of our novel dataset of oil palm fruits in plantations for real-time, resource-efficient applications in agricultural monitoring which captured real-world challenges in the plantation regions of Central Kalimantan Province, Indonesia.

The decision to compare YOLOv8 Depthwise with YOLOv6 [38], YOLOv7 [39], and YOLOv8 [40], coupled with the capability to count objects and evaluate counting error, is grounded in their status as newer and enhanced models, each featuring distinctive improvements over their predecessors. These models are not incremental iterations of each other but represent distinct advancements over the earlier YOLO versions. The choice of these specific versions is based on their individual enhancements and optimizations, making them particularly effective for our specific palm oil yield estimation task. Across various domains, YOLOv6, YOLOv7, and YOLOv8 have demonstrated versatility and effectiveness in diverse applications beyond their primary use in object detection [41], [42], [43], [44], [45], [46].

Recent research has highlighted the advantages of YOLOv8. According to [47], [48], YOLOv8 not only runs faster but also scores higher on mAP metrics than YOLOv5, YOLOv6, and YOLOv7. [49] found that YOLOv8 excels at detecting small objects on camera sensors, outperforming older versions like YOLOv3, YOLOv5, and YOLOv7.

The decision to incorporate depthwise separable convolutions in YOLOv8 Depthwise is based on their proven ability to reduce model size and computational complexity while maintaining detection accuracy. Previous applications in YOLOv5 demonstrated significant reductions in model size and complexity, as noted by [50], [51]. In YOLOv4, [52] showed that the use of depthwise separable convolutions reduced parameters by 40%, increased speed by 20%, and even improved detection accuracy by 1%.

Prior research has demonstrated the feasibility of automated counting in various agricultural domains. For instance, [17] introduces an automated counting system for tomatoes. In a similar vein, [18] presents an automatic counting approach for green oranges on trees. Qiao et al. [53] employ a deep learning model for automated counting, focusing on red jujubes. Their algorithm adds a detected single fruit to the overall count.

Multiple contributions are made to the field of palm oil fruit yield estimation, encompassing the following: (1) Utilization of a self-collected dataset with video recordings of Fresh Fruit Bunches (FFBs) still attached to trees. The dataset includes five ripeness classes (abnormal, ripe, underripe, unripe, and flower) and presents challenges like occlusion, blur, and low contrast. Comprehensive processing includes frame extraction, cleaning, annotation, splitting, and augmentation. (2) Introduction and fine-tuning of the YOLOv8 Depthwise model, leveraging depthwise separable convolutions for enhanced efficiency and accuracy. Performance is compared with other YOLO models (YOLOv6s, YOLOv6l, YOLOv7tiny, YOLOv7l, YOLOv8s, YOLOv8l) and state-of-the-art models (Faster RCNN, SSD MobileNetV2, YOLOv4). (3) The implementation of evaluation on counting error integrated into each model. This increases the practicality and real-time applicability of the models. (4) Optimization of model inference time using converters and quantization, ensuring suitability for mobile devices with limited computational resources. (5) Deployment of models in a web application, enhancing accessibility across various devices, including mobile phones, for real-time FFB counting in the field.

The proposed approach reduces human labor, inconsistencies, and errors, making it suitable for large-scale operations. It enhances oil quantity and quality, boosts efficiency, lowers labour costs, and promotes environmental sustainability by reducing waste from unripe FFBs. The subsequent sections of this paper will delve into an explanation of materials and methods, results and discussion of the experiment, and conclusion of the proposed automated counting systems potential to revolutionize palm oil detection and count estimation.

II. MATERIALS AND METHODS

A. DATASET

The dataset for this research was meticulously compiled in Central Kalimantan Province, Indonesia, through video recordings at 30 frames per second within an oil palm plantation. We have collected a total of 440 videos, each with varying lengths ranging from 8 seconds to 1 minute and 31 seconds. These videos are captured in a resolution of 320×640 pixels, providing a portrait orientation.

All videos are stored in the MP4 format, a standard in digital media, which ensures compatibility and ease of

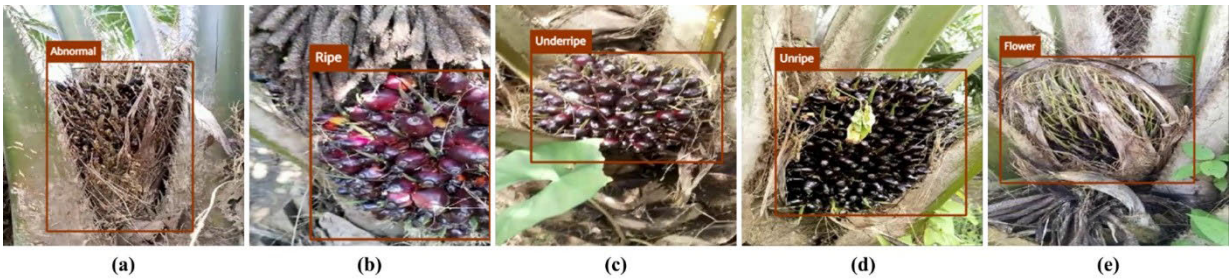

FIGURE 1. Annotated ground- truth images of each class: (a) Abnormal, (b) Ripe, (c) Underripe, (d) Unripe, and (e) Flower.

FIGURE 2. Dataset complexity.

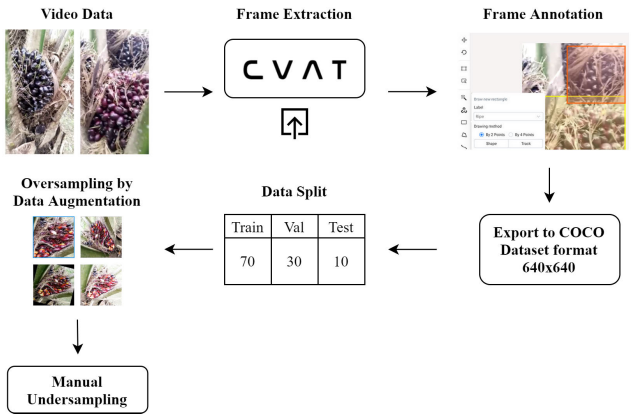

FIGURE 3. Data collection and pre-processing flow.

use in various object detection frameworks. As illustrated in Figure 1, the dataset focuses on categorizing oil palm fruits into five stages: Unripe, Underripe, Ripe, Flower, and Abnormal. The Unripe stage (early-stage fruits) is characterized by fruit bunches with a blackish hue. The Underripe stage (transitional phase fruits) features fruit bunches that vary in color from dark purple to dark red. The Ripe stage (mature fruits ready for harvest) showcases fruit bunches that range from dark red to bright red, often with yellow spots. In the Flower stage (representing the flowering stage), the structures lack fruits and have long, brown fibers resembling petals. The Abnormal stage (fruits with atypical characteristics due to potential disease or irregularities) includes fruits with irregular shapes, often appearing less uniform and with fewer fruits in the bunch.

The dataset involved presents a multitude of intricate challenges that require thorough examination. A primary obstacle lies in the dataset's intrinsic complexity, where a single frame may encompass multiple instances

Occlusion and partial visibility are significant challenges in detecting Fresh Fruit Bunches (FFBs) in the dataset. Occlusion occurs when portions of fruits are obscured from view by other objects, while partial visibility refers to objects that are only partially within the frame. These challenges are compounded by factors such as substantial distances between the camera and objects, which can lead to diminished perceptual acuity. Furthermore, the challenges are worsened by the presence of slight blurring and reduced contrast, which may cause one class to resemble another. This combination, presented in Figure 2, highlights the need for increased sensitivity in model recognition, as each of these factors contributes to the overall complexity of the challenges present in the dataset.

B. DATA PRE-PROCESSING

In our research, Figure 3 illustrates the data collection and pre-processing pipeline. This pipeline plays a crucial role in preparing the dataset for effective object detection model training. The details of this process can be further dissected, as shown in the following pseudocode, which reveals the specific steps involved in data pre-processing.

Algorithm 1 Dataset Preprocessing

```
procedure PreprocessDataset
    Input: Raw videos
    Output: Preprocessed train, val, and test dataset
    extracted_frames ←
ExtractAndAnnotateFrames(dataset)
    train_data, val_data, test_data ←
SplitDataByObjectCount(extracted_frames)
    AugmentTrainData(train_data)
    Resize val_data and test_data to 640 × 640
```

```
    Manually undersample majority classes by deleting images containing those objects
    return Preprocessed Train, Val, Test Dataset
end procedure

procedure ExtractAndAnnotateFrames
    Input: Raw videos
    Output: Annotated frames
    for each video in dataset do
        while there are remaining frames do
            Extract frame
            if frame quality is acceptable and not similar to the last then
                Annotate frame content
            end if
            Save annotated frame
        end while
    end for
    return Annotated frames
end procedure

procedure SplitData
    Input: Annotated frames
    Output: Train, validation, and test data
    Group annotated frames by video sources
    for each group in groups do
        assigned ← false
        while not assigned do
            Randomly choose train, val, or, test
            if train dataset is chosen and ratio of objects < 0.7 then
                Add group to train dataset
                assigned ← true
            else if val dataset is chosen and ratio of objects < 0.2 then
                Add group to val dataset
                assigned ← true
            else if test dataset is chosen and ratio of objects < 0.1 then
                Add group to test dataset
                assigned ← true
            end if
        end while
    end for
    sets ← [train, val, test]
    for each class in FFB classes do
        for each set in sets do
            if number of class in set is low then
                Move groups containing class from other sets to set until minimal presence of class is achieved
            end if
        end for
    end for
    return Train, val, test data
end procedure

procedure AugmentTrainData
    Input: Train data
    Output: Augment train data
    for each image in train data do
        Resize image to 640 × 640
        augmentations ← [(Horizontal flip, 0.5), (Vertical flip, 0.5), (Rotate 90 degrees, 0.5), (Adjust brightness, 0.5), (Adjust contrast, 0.5), (Add Gaussian noise, 0.5), (RGB shift, 0.5), (HSV shift, 0.5), (Random crop, 0.8)]
        for (augmentation, probability) in augmentations do
            if random chance < probability then
                Apply augmentation to image
            end if
        end for
    end for
    return Augmented train data
end procedure
```

1) FRAME EXTRACTION AND CLEANING

The initial step involves converting the 440 collected videos into individual frames in.png format using Computer Vision Annotation Tool (CVAT) [54]. This transformation ensures uniformity in resolution and aspect ratio across all frames. To enhance dataset quality, frames with blue tint or distortions, indicative of lighting anomalies or lens aberrations, are manually removed using CVAT's frame deletion feature.

2) ANNOTATION

Annotating these frames is critical for supervised learning. Using CVAT's ''draw new rectangle'' feature, we thoroughly label each frame, assigning classes like unripe, underripe, ripe, flower, and abnormal. The ''track'' feature in CVAT aids in maintaining object consistency across frames, ensuring accurate temporal tracking.

3) FORMAT CONVERSION FOR MODEL TRAINING

The annotation results are converted into a format suitable for YOLO-based models. The dataset is initially structured in the YOLOv8 format from CVAT, including a configuration file and separate folders for images and labels. This format is then transformed into the COCO dataset.

4) DATA SPLITTING

We ensure no overlap of videos between training, validation, and testing datasets, avoiding temporal information redundancy. Additionally, a class-wise distribution approach ensures balanced representation across different classes. At the onset of our data splitting process, we initialize parameters by setting a split ratio of 0.7, 0.2, and 0.1 for training, validation, and testing, respectively. Alongside this, we establish a dictionary of the total class count, to keep track of the count of each class across all frames. In the first pass of our procedure, we iterate over each video frame, counting frames and reading annotations to update the total class count.

5) DATA AUGMENTATION AND CLASS BALANCING

To address class imbalance and enrich dataset diversity, we leverage Albumentations [55] for data augmentation. This process involves various techniques applied with specific probabilities: random cropping (80%), horizontal/vertical flipping (50% each), 90° rotations (50%), adjustments to brightness/contrast (50%), and the introduction of Gaussian noise, RGB shift, and HSV shift (all 50%). Importantly, all images are standardized to 640 × 640 pixels before

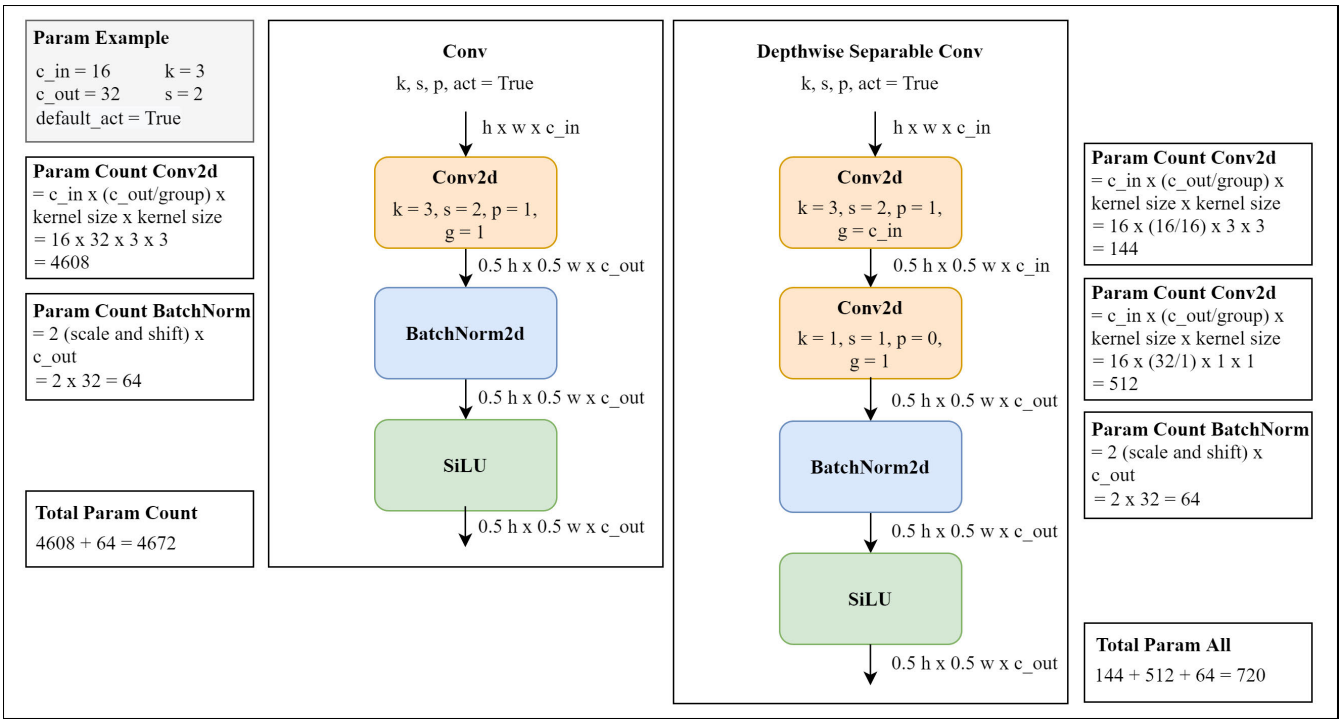

FIGURE 4. Detailed architecture of conv and depthwise separable conv module with parameter count for each layer.

augmentation. This augmentation pipeline aims to bolster the representation of underrepresented classes (targeting 2480 images per class) without resorting to excessive oversampling. In cases of severe class imbalance, we employ manual undersampling to mitigate the dominance of overrepresented classes. This undersampling focuses on eliminating frames with high visual similarity.

TABLE 1. Class distribution by object.

Class	Train	Valid	Test
Ripe	2354	443	192
Underripe	1324	298	181
Unripe	4990	1699	764
Flower	1121	251	182
Abnormal	418	205	81
Total	10207	2896	1400
Split (%)	70.04	19.97	9.65

C. PROPOSED METHOD

This study proposed a method for estimating palm oil production using advanced deep learning models, particularly the YOLOv8 Depthwise model on a novel dataset. The comparative analysis of the proposed model is streamlined using the MMYOLO benchmark toolbox from OpenMMLab [56], while the enumeration process will be handled simultaneously while the detection is being run. Finally, every model's performance is going to be accelerated with ONNX Runtime to acquire faster inference time.

YOLO (You Only Look Once) series [57], a one-stage object detection networks. Introduced by Joseph Redmon et al. in 2016, instigated a significant shift in the paradigm of real-time, end-to-end object detection. Setting itself apart from preceding methodologies, YOLO performs a single forward pass to simultaneously predict class probabilities and bounding box coordinates, a stark contrast to the two-tiered approach of methods like R-CNN [58], fast R-CNN [59], and Faster R-CNN [60], consequently placing YOLO at the vanguard of real-time applications [61]. Over time, YOLO has undergone several enhancements, notably in terms of speed and precision [38], [39], [40], [56], [62], [63], [64], [65]. In essence, the YOLO architecture segments the input image into a grid, within which it predicts numerous bounding boxes and class probabilities for each cell. These boxes encompass coordinates and a confidence score. The network's output, characterized as a tensor with dimensions $S \times S \times (B \times 5 + C)$, is refined via non-maximum suppression (NMS) for heightened precision in detection [56].

YOLOv6 marks a significant leap, especially for industrial applications, prioritizing efficiency post-deployment [38]. It employs RepOptimizer and channel-wise distillation [66], [67], and integrates both Post-training Quantization (PTQ) and Quantization-aware Training (QAT) to balance speed and accuracy. The architecture of YOLOv6 amalgamates the EfficientRep backbone, based on RepVGG [68], with a neck composed of the Path Aggregation Network (PAN) [69], enhanced using RepBlocks [68] or CSPStackRep [70]

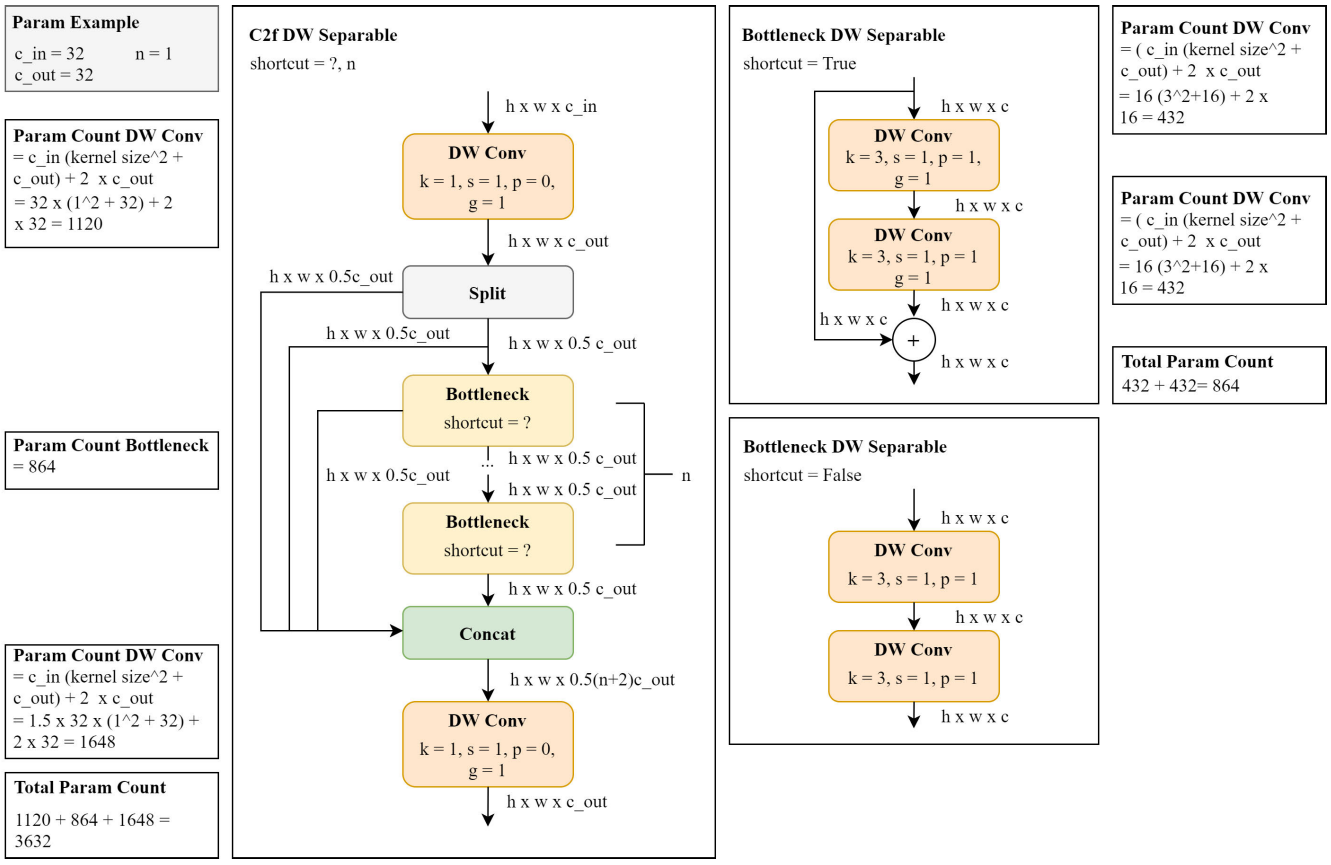

FIGURE 5. Detailed architecture of C2f and its bottlenecks modified with depthwise separable convolutions with parameter count for each layer.

for its larger variants, and culminates with an Efficient Decoupled Head. Its other distinct innovations include the adoption of a Task alignment learning approach from Task-aligned One-stage Object Detection (TOOD) for label assignment [71]. It also introduces novel classification and regression losses, specifically a classification VariFocal loss [72] and SIoU/GIoU regression loss [73], [74].

YOLOv7 introduces numerous architectural changes and a ''bag-of-freebies,'' enhancing accuracy while reducing parameter count and computational load compared to YOLOv4 [64]. A key improvement is the Extended Efficient Layer Aggregation Network (E-ELAN) [75], which enhances learning capacity by shuffling and merging cardinality without disrupting the gradient path [38]. YOLOv7 also pioneers model scaling for concatenation-based models, preserving optimal structure [38]. Its ''bag-of-freebies'' includes Planned Reparametrized Convolution (RepConvN) without the identity connection typical in standard RepConv [68], Coarse and Fine Label Assignment for auxiliary and lead heads, and the use of Exponential Moving Average for final inference [39]. These enhancements make YOLOv7 an efficient and precise object detector.

The most recent iteration, YOLOv8, builds upon YOLOv5's strengths while introducing significant novelties [40]. It retains a similar backbone but revamps the CSPLayer to the C2f module, which integrates high-level features with contextual data to boost detection accuracy [76]. The C2f module improves efficiency by reducing one convolutional layer [77]. YOLOv8 also incorporates the SPPF module [78], combining a CBS module with multiple max-pooling layers to enhance feature map interpretability [79]. Its neck integrates a Feature Pyramid Network (FPN) [80] with PAN [69] for comprehensive feature integration. For loss functions, YOLOv8 uses CIoU [81] and DFL [82] for bounding box loss and binary cross-entropy for classification loss, improving detection capabilities, especially for smaller objects.

For comparative analysis, MMYOLO provides a unified backend for YOLO-based object detection and segmentation, ensuring a consistent and effective framework for benchmarking. This overview, informed by key references [83], [84], [85], [86], [87], elucidates MMYOLO's architecture, implementation, and its role in comparative analyses within the computer vision community.

The proposed model prioritizes a parameter-efficient architecture. This is achieved by replacing standard convolutional blocks with depthwise separable convolutions throughout the YOLOv8 design. YOLOv8, particularly the YOLOv8s variant, offers a compelling balance between speed and accuracy, as it is the lighter variation of YOLOv8.

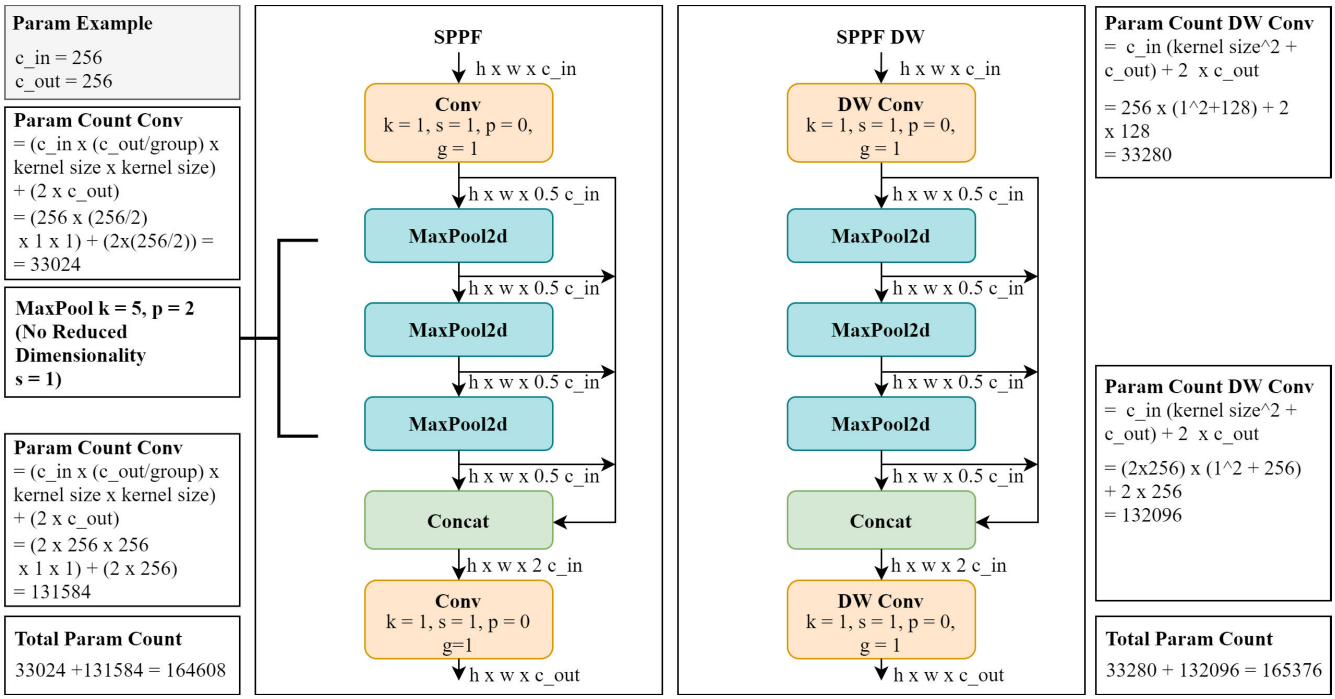

FIGURE 6. Detailed architecture of SPPF and depthwise-separable-convolution-modified SPPF with parameter count for each layer.

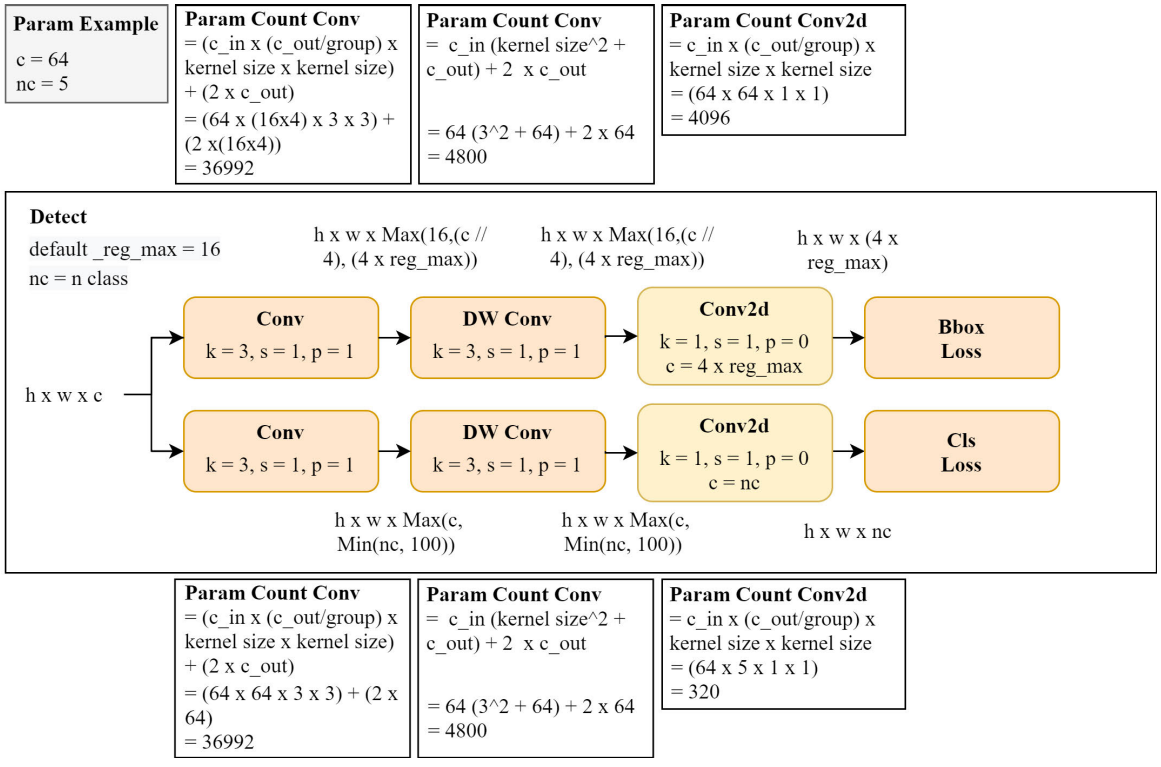

FIGURE 7. Detailed architecture of depthwise separable convolutions in detect module with parameter count for each layer.

This characteristic is suitable for real-time applications where both aspects are crucial. As depicted in Figure 4, a depthwise separable convolution consists of two parts: a depthwise conv block with stride of 2, resulting in halved output width and height. While the group is set the same as channel in, so that each channel is convolved by the filter.

This is followed by a pointwise convolutional block where the kernel size is 1×1 convolving across all the channel processed before. Batch normalization (BatchNorm2d) is then employed to normalize the output using shift and scale operations, followed by a SiLU activation function, which is the default choice in YOLOv8. This approach significantly reduces the number of parameters compared to traditional convolutional blocks. Our calculations based on the architecture in Figure 4 indicate a parameter reduction of 84.589%.

The C2f block in the YOLOv8 architecture is also replaced by C2f DW Separable Block as shown in Figure 5. While the overall structure remains the same as the C2f block, all Conv blocks in C2f are replaced with Depthwise Separable Conv. The number of bottleneck blocks 'n' is configured according to parameter 'd' from YOLOv8s which is 0,33. Overall, the new C2f Ghost block uses 50.65% less parameters than the original C2f block.

The SPPF block used in the neck is also replaced with SPPF DW. As shown in Figure 6, the only difference in this block is that the Conv blocks are replaced with Depthwise Separable Conv from Figure 6. Every ouput from the maxPool2d layer and the first Conv block is concatenated. This configuration is the same as individually maxpooling each input with maxpool kernel sizes of 5, 9, and 13. Here, the first max pooling has a 5×5 receptive field. The second pooling, applied on the already pooled output, increases the receptive field equivalent to a 9×9 area due to overlapping effects. Finally, the third maxpool has a receptive field of 13×13. This allows the network to detect objects at multiple scales.

The detect head in YOLOv8 is also replaced by detect DW (Figure 7) which enhanced its efficiency by replacing original Conv in the middle before bounding box regression and class prediction operation with Depthwise Separable Conv. This helps reduces the parameter by 42.25%. Combining all the modification described earlier, the final architecture of the proposed YOLOv8 model is shown in Figure 8. It also displays the dimensions of feature maps that goes through each module. This dimension specifies the height, width, and channel of the corresponding feature map. The number of input and output channels is determined based on the model size. Here, a width multiple of the model size is used to scale the number of channels. Additionally, this value is capped by a maximum channel number that also correlates with the model size.

Each model was converted to the ONNX Runtime [88] format for cross-platform compatibility, as ONNX Runtime is optimized for accelerating the deployment of machine learning models across various platforms and devices, thereby enabling cross-platform model inference. In addition to the model conversion, we applied FP16 quantization to both the small and large versions of each YOLO model used in our experiment. This process aimed to investigate the impact of quantization on the inference time and mean Average Precision (mAP) of each model. The results of these comparative analyses provide valuable insights into the trade-offs between model size, computational efficiency, and performance accuracy.

Finally, the enumeration of detected oil palms is performed automatically, wherein the number of each object in a frame is quantified. This counting process involves sorting the objects based on their respective classes, allowing for a detailed comparison between the predicted and ground truth counts for all images. Evaluation metrics, including Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) [89] are then employed to assess the accuracy of the counting results. The specific steps of this counting process are detailed in the ensuing pseudocode, which outlines the algorithm used to automate the enumeration of objects based on their classes within each frame.

Algorithm 2 Object Count

```
procedure CountOilPalmPerFrame
    Input: boundingBoxes, IoUThreshold
    Output: countDataPerClass
    classCounts ← {}
    for each boundingBox in boundingBoxes do
        if boundingBox. IoU > IoUThreshold then
            class ← boundingBox.class
            if class in classCounts then
                classCounts[class] ←
            classCounts[class] + 1
            else
                classCounts[class] ← 1
            end if
        end if
    end for
    for each class in classCounts do
        Print "Class:", class, "Count:", classCounts[class]
    end for
    return classCounts
end procedure
```

Following the evaluation of counting errors, the subsequent phase involves the development of a web application tailored for FFB counting. This strategic approach ensures seamless accessibility from a myriad of mobile devices while retaining the capability to conduct real-time fruit detection through camera access. The foundational framework for this application is based on next.js framework [90] and onnxruntime-web package [91], with a deliberate effort to align its functionality with our adopted model's specifications. This necessitates fine-tuning the application's processing capabilities to harmonize with our model's architecture. Specifically, adjustments are made to accommodate the model's distinct shape and facilitate the recognition of the five distinct classes targeted for detection. This intricate calibration ensures a symbiotic relationship between the application and our specialized model, fostering optimal performance in FFB counting.

D. TRAINING MODEL

The training model in the MMYOLO toolbox is executed in the Google Colaboratory environment, leveraging the

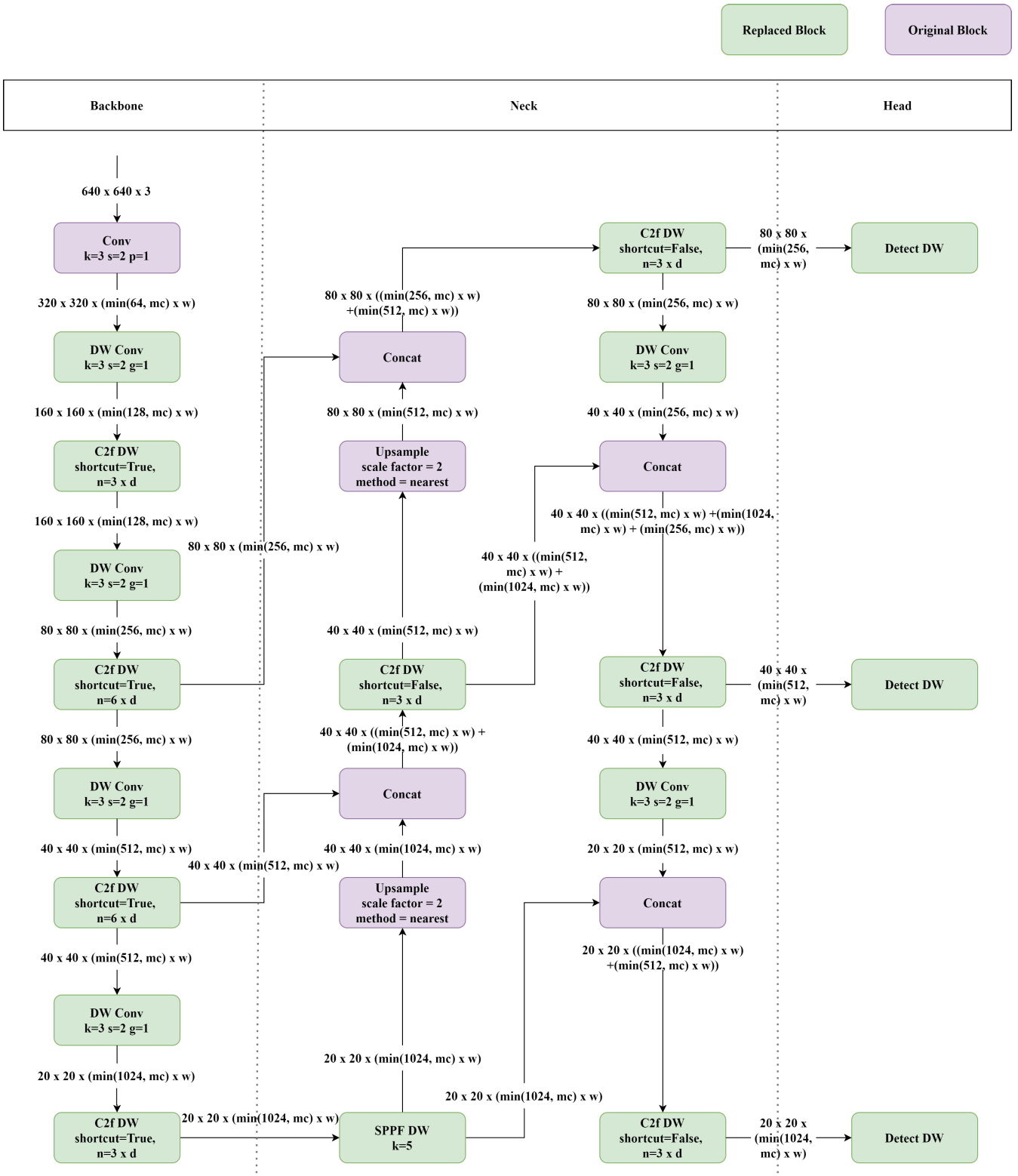

FIGURE 8. Architecture of YOLOv8 modified with depthwise separable convolutions (DW Conv), C2f DW, SPPF DW, and detect DW including input dimensions (height × width × channel).

computational capabilities of an NVIDIA V100 Tensor Core GPU. The choice of the V100 GPU aligns with the need for robust training performance and efficient processing of complex deep learning models. For inferencing tasks, the

trained models are tested on an NVIDIA T4 GPU using the ONNX Runtime, ensuring streamlined and optimized execution in real-world applications.

Regarding hyperparameters, the training process utilizes a learning rate of 0.001 to achieve an optimal balance between the speed of convergence and the stability of the optimization process. A weight decay of 0.0005 is used to prevent overfitting and improve model generalization. The momentum, which controls the model's update direction, is set to 0.03 to stabilize the optimization process.

Additionally, the batch size for all models was uniformly set to 16. This decision was necessitated by the computational limitations of our testing bench. Although smaller models could potentially be run with a batch size of 32, we opted for a batch size of 16 to ensure consistency and fairness across all models.

To identify the optimal base model architecture for further improvement through depthwise separable convolutions, we employed a finetuning strategy detailed in the following pseudocode.

Algorithm 3 Model Training

```
procedure FineTuneBaseModels
    Input: Preprocessed train and validation dataset
    Output: Trained models
    baseline_model_configs ← [
YOLOv6s_config, YOLOv6l_config,
YOLOv7tiny_config, YOLOv7l_config,
YOLOv8s_config, YOLOv8l_config]
    for config in baseline_model_configs do
        model ← InitializeModel(config)
        epoch ← 0
        max_epochs ← Get max epochs from config
        training ← true
        while training do
            for mini batch in train dataset do
                predictions ← Make model predict mini
            batch
                loss ← Calculate loss from predictions
            and mini batch labels
                Perform backpropagation using loss
                Update model weights with optimizer
            end for
            epoch ← epoch + 1
            validation_loss ← Calculate validation loss
            if epoch ≥ max_epochs then
                training ← false
            end if
        end while
        Save trained model
    end for
    return Trained models
end procedure

procedure InitializeModel
    Input: Model configuration
    Output: Initialized model
    Define model architecture using model configuration
    Define hyperparameters using model configuration
    Define classes and class weights using model
configuration
    return Initialized model
end procedure
```

Following the identification of the optimal base model through finetuning, a modified version incorporating depthwise separable convolutions will be trained. The training regimen for this modified model closely resembles the finetuning process as well, with the key difference being the model architecture itself. The InitializeModel procedure within the pseudocode will be modified to reflect these architectural changes, including adjustments to the backbone, neck, and head components. These modifications aim to improve the model's efficiency while maintaining its object detection accuracy.

The approach to hyperparameter tuning mirrors the finetuning process as well. Here, the model's performance is evaluated with different configurations for key hyperparameters that can significantly impact training effectiveness. These hyperparameters include the base learning rate and the optimizer used. The optimal number of images processed together during each training iteration different learning rates (0.0001, 0.001, and 0.01) will be tested to find the rate that allows the model to learn effectively without overshooting the minimum loss. The impact of the optimization algorithm will also be compared by evaluating both SGD (Stochastic Gradient Descent) and Adam optimizers.

Algorithm 4 Model Inference

```
procedure DetectImage
    Input: Image input, Confidence threshold, NMS threshold
    Output: Image with detections
    model ← InitializeModel(YOLOv8s_DW_config)
    Resize image to 640 × 640
    detections ← Perform forward pass on model
    results ← PostprocessDetections(detections,
      confidence_threshold, nms_threshold)
    for (box, confidence, class_label) in results do
        Draw bounding box on image with label and
    confidence
    end for
    Show the image with detections
    return Image with detections
end procedure

procedure PostprocessDetections
    Input: Detections, Confidence threshold, NMS threshold, Classes
    Output: Bounding boxes, Confidences, Classes
    boxes ← []
    confidences ← []
    class_ids ← []
    for each detection in detections do
        scores ← detection's confidence scores
        class_id ← Get class with highest score
        confidence ← scores[class_id]
        if confidence > confidence_threshold then
            box ← Extract bounding box coordinates
        from detection
            Append box to boxes
            Append confidence to confidences
```

```
            Append class_id to class_ids
        end if
    end for
    indices ← Apply NMS to boxes, confidences, nms_threshold
    results ← [(boxes[i], confidences[i],
classes[class_ids[i]]) for i in indices]
    return results
end procedure
```

Moreover, the DetectImage procedure exemplifies how the trained model operates during inference. Inference refers to the stage where the model is used to make predictions on new, unseen data. In this case, the pseudocode showcases how the modified version with depthwise separable convolutions would process an image for object detection. This procedure provides a high-level overview of the steps involved in making predictions using the trained model.

E. MODEL OPTIMIZATION

This section explains the strategies employed to enhance the performance of YOLO-based models. The aim is to improve the efficiency and accuracy of the model in detecting fresh fruit bunches, addressing the unique challenges posed by real-world scenarios. In this discussion, we will examine different optimization techniques that enhance the model's predictive capabilities.

1) CLASS WEIGHT ADJUSTMENT

To address the imbalances in the dataset and improve the model's ability to discern between different classes of fruit bunches, we assign a comparatively lower weight to the majority class, 'Unripe.' The weight assigned to each class is calculated using Equation (1), where it considers the total instances of the respective class in relation to the overall instances across all classes.

$$w_{\text{new}} = 1 - \frac{n_{\text{class}}}{n_{\text{total}}} \tag{1}$$

Here, n_{class} is the number of instances in a class and n_{total} is the number of instances across all class.

2) LAYER-FREEZING

In conjunction with our efforts to optimize the models, we selectively freeze specific stages of the backbone network during model training. This ensures that the parameters associated with them remain unaltered throughout the fine-tuning process, providing stability to the foundational layers of the model. In this implementation, the first stage of the backbone is frozen. By preserving the initial feature extraction layers strategically, we aim to retain valuable low-level features that are critical for accurate identification of fruit bunches. This approach to freezing backbone layers preserves essential foundational features during the model fine-tuning process.

In detail, for the YOLOv8 family, we freeze the ConvModule and CSPLayer_2Conv blocks in the first stage of the backbone. In the YOLOv7 family, the frozen blocks in the first stage consist of ConvModule and ELANBlock. On the other hand, the YOLOv6 family exhibits variations between YOLOv6s and YOLOv6l. For YOLOv6s, the frozen blocks in the first stage of the backbone include RepVGGBlock and RepStageBlock, whereas YOLOv6l employs RepVGGBlock and BepC3StageBlock in the first stage.

3) ANCHOR TUNING

Furthermore, our attention focuses on anchor tuning within the YOLOv7 architecture. Unlike YOLOv6, which uses dynamic anchors [38], YOLOv7 adheres to a pre-set anchor paradigm as stipulated in [92]. To increase adaptability and dynamism, K-means clustering is integrated seamlessly into these anchors, a method widely explained in [93]. This approach allows for adjusting anchor boxes to better fit the various object scales and aspect ratios in the dataset.

4) LOSS FUNCTIONS

Our optimization strategy for object detection models in YOLO-based frameworks, such as YOLOv6, YOLOv7, and YOLOv8, goes beyond anchor tuning. It includes meticulous selection and implementation of loss functions, which are critical for refining model performance. The choice of loss functions plays a pivotal role in guiding the model's learning process. These loss functions serve as metrics determining the disparity between predicted outputs and actual annotations, steering the model toward increased precision in its predictions.

In YOLOv6, the model uses Varifocal Loss (VFL) [72] for classification, a dynamic approach that assigns different weights to different classes based on their importance. Additionally, for regression, YOLOv6 uses both SCYLLA-IoU (SIoU) [73] and Generalized IoU (GIoU) [74] as loss functions. These choices contribute to the model's robust training dynamics and promote improved object recognition performance.

VFL serves as a refinement of Focal Loss [94], introducing additional nuances to improve the training dynamics of object detection models. Rooted in Focal Loss, VFL extends its foundation to incorporate elements of binary cross-entropy loss. It aims to forecast the IoU-Awareness Classification Score (IACS), a scalar classification unit, considering the intersection between the anticipated and actual bounding box. VFL, as shown in Equation (2), is explicitly defined, and its distinctive feature is to selectively scale the loss contribution of negative examples by a factor of p^{γ}, without applying the same down-weighting to positive examples. In this way, VFL aims to prioritize the learning of challenging instances and high-quality positives, ultimately contributing to the model's ability to achieve a higher Average Precision (AP) by focusing on the most informative and relevant examples during training.

$$FL(p, y) = \begin{cases} -\alpha(1-p)^{\gamma}\log(p) & \text{if } y = 1 \\ -(1-\alpha)p^{\gamma}\log(1-p) & \text{otherwise} \end{cases} \tag{2}$$

where p is the predicted IoU-Aware Classification Score, q is the target score, α is the adjustable scaling factor, $(1-p)^{\gamma}$ is the positive class, and p^{γ} is the negative class.

As for SIoU, it consists of four cost functions: angle cost, distance cost, shape cost, and IoU cost. The addition of the angle-sensitive component helps the model converge faster by minimizing the number of variables in distance-related ''wandering''. Through each of these functions, the formula displayed in Equation (3) is formed. Directly involved in this formula, Equation (4) symbolizes the distance cost, while Equation (5) represents the shape cost. To calculate loss functions involving IoU, its value is required, which can be calculated with Equation (6).

$$\mathcal{L}_{\text{SIoU}} = 1 - IoU + \frac{\Delta + \Omega}{2} \tag{3}$$

where

$$\Delta = \sum_{t=x,y} (1 - e^{-\gamma \rho_t}) \tag{4}$$

$$\Omega = \sum_{t=w,h} (1 - e^{-\omega_t}) \tag{5}$$

$$IoU = \frac{\text{Area of prediction box} \cap \text{Area of truth box}}{\text{Area of prediction box} \cup \text{Area of truth box}} \tag{6}$$

To accompany SIoU in YOLOv6, GIoU is calculated by finding the smallest convex shape that encloses both shapes, then measuring the ratio between the volume occupied by this shape excluding the original shapes and the total volume occupied by it. Finally, GIoU is obtained by subtracting this ratio from the IoU value. GIoU has properties like IoU, such as being a distance metric, invariant to the scale of the problem, and always a lower bound on IoU. From computing the value of GIoU, the loss function can be found, as shown in Equation (7). This process entails computing the area of the bounding box enclosure using Equation (8) and determining the area of union (U), as expressed in Equation (9) through the calculation involving the areas of both the predicted and ground-truth bounding boxes.

$$\mathcal{L}_{\text{GIoU}} = 1 - IoU - \frac{A^c - U}{A^c} \tag{7}$$

where

$$A^c = (x_2^c - x_1^c) \times (y_2^c - y_1^c) \tag{8}$$

$$U = A^p + A^g - I \tag{9}$$

In YOLOv7, Binary Cross Entropy Loss [95] is employed for classification loss, while the loss for the detector utilizes CIoU (Complete IoU) [81]. CIoU is a loss function utilized in object detection for bounding box regression. It considers three geometric factors in bounding box regression: overlap area, central point distance, and aspect ratio. Shown in Equation (10), the loss function of CIoU is found from the sum of the penalty between the predicted and ground-truth bounding box with the dot product of the positive trade-off parameter (α) and the aspect ratio consistency (v). Which then, it will be added with the loss.

$$\mathcal{L}_{\text{CIoU}} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v \tag{10}$$

$$\alpha = \frac{v}{(1 - IoU) + v'} \tag{11}$$

$$v = \frac{4}{\pi^2} \left(\arctan \frac{w^{gt}}{h^{gt}} - \arctan \frac{w}{h} \right)^2 \tag{12}$$

In accordance with Binary Cross Entropy Loss, the operation is carried out on the predicted probability score and its corresponding ground truth within a binary classification model. The function treats each problem independently and calculates the loss for each class separately. Equation (13) exhibits how the loss function is assessed. The probability score, denoted as $p(x_i)$, represents the predicted likelihood that the instance x_i is associated with the default class. Here, y_i signifies the true label, taking on values of either 0 or 1.

$$\mathcal{L} = -\left(y_i \cdot \log(p(x_i)) + (1 - y_i) \cdot \log(1 - p(x_i))\right) \tag{13}$$

As discussed in the previous section, YOLOv8 utilizes CIoU [81] and DFL [82] for its loss functions. Equation (10) has already demonstrated the formula of CIoU. Meanwhile DFL, or Distribution Focal Loss, models bounding box locations as general distributions and encourages the network to focus on learning the probabilities of values close to the target coordinates. This loss function, as seen in Equation (14), explicitly increases the probabilities of values around the target, ensuring its accuracy.

$$DFL(S_i, S_{i+1}) = -\left((y_i + 1 - y)\log(S_i) + (-y_i)\log(S_i + 1)\right) \tag{14}$$

5) OPTIMIZER

The preferred optimization method for enhancing the performance of computer vision models, such as those implemented in this study, is Stochastic Gradient Descent (SGD) [96]. It is a common method for optimizing computer vision models. SGD strategically utilizes a single random sample to estimate the gradient of the loss function during each iteration, making it a highly efficient and well-suited solution for large-scale problems. The ability of SGD to achieve rapid convergence, especially in strong convex problem scenarios, is noteworthy. It exhibits a high convergence rate even when the optimization problem's conditions are less strict than the strong convex condition. Additionally, fundamental improvements to SGD, which concentrate on increasing speed and reducing fluctuations, have been crucial in refining its optimization process. In addition to SGD, this experiment also employs Adam for hyperparameter tuning. Adam combines the advantages of both SGD with momentum and RMSprop [97]. Adam is known for its ability to handle sparse gradients and its robustness to noisy data, making it a popular choice for many machine learning applications. The selection of an appropriate base learning rate is crucial because it has a direct impact on how quickly and efficiently

a model learns. By testing base learning rates across different scales like 0.0001, 0.001, and 0.01, we can explore how large or small adjustments the model makes to its weights during training, impacting its ability to find the optimal solution.

6) QUANTIZATION

As part of our model optimization strategy, we prioritize efficiency through quantization during training. This process involves converting the numerical representation of model parameters into the half-precision floating-point format, known as fp16. In fp16, each floating-point number is represented using 16 bits, offering advantages such as reduced memory storage and faster computations. Implementing fp16 quantization significantly reduces the memory footprint of model parameters, enhancing computational efficiency. This is especially beneficial when deploying models on hardware that supports half-precision computations.

F. MODEL EVALUATION

Within the scope of model assessment, our focal point lies in the evaluation conducted during the validation phase, elucidating the nuanced performance of each YOLO variant. This thorough examination is crucial for assessing the effectiveness and precision of our models. We use established metrics such as Average Precision (AP), mean Average Precision (mAP), and mean Average Recall (mAR) to precisely measure and analyse the performance of each model [98]. These metrics are essential to the validation process as they provide a quantitative measure of the models' ability to accurately identify and localize objects within the provided dataset.

Our assessment evaluates the accuracy of instance enumeration within individual frames, in addition to the previously mentioned metrics. The aim is to evaluate the accuracy of instance enumeration within individual frames and determine whether the detected number of instances aligns with the ground truth. This emphasizes the correctness of estimation for each class in the image. Across all images employed in the counting process, the evaluation employs metrics such as MAE and RMSE [89]. MAE represents the average absolute differences between predicted and ground truth values, providing a straightforward measure of accuracy. In contrast, RMSE emphasizes the magnitude of errors by taking the square root of the average squared differences, offering a more sensitive metric to large discrepancies. Moreover, model size and inference time were measured. This was done to compare the efficiency and performance of the models during real-time inference operations.

1) AVERAGE PRECISION (AP)

Calculating AP for object detection differs from the conventional method used in regular precision and recall computations. While the latter typically relies purely on true and false outcomes, object detection introduces an additional element: the confidence level associated with each detection. Equations (15) and (16) outline the computation of recall and precision by integrating a threshold (τ) based on this confidence level. This means that a higher threshold results in a lower computed value.

$$Pr(\tau) = \frac{\sum_{n=1}^{S} TP_n(\tau)}{\sum_{n=1}^{S} TP_n(\tau) + \sum_{n=1}^{N-S} FP_n(\tau)} \tag{15}$$

$$Rc(\tau) = \frac{\sum_{n=1}^{S} TP_n(\tau)}{\sum_{n=1}^{S} TP_n(\tau) + \sum_{n=1}^{G-S} FN_n(\tau)} \tag{16}$$

AP is based on the region below the Precision-Recall (PR) curve and is calculated using a specific IoU threshold. The PR curve is discretely represented, with recall values having a direct and unchanging correspondence to confidence levels (τ). Then, compute the AP using the two ordered sets of reference recall values and the confidence values. The AP itself, expressed in Equation (17) is calculated by a Riemann integral of $Pr_{interp}(R)$ with reference to the K recall values from the set $Rr(k)$.

$$AP = \sum_{k=0}^{K} (R_r(k) - R_r(k+1)) Pr_{\text{interp}}(R_r(k)) \tag{17}$$

2) MEAN AVERAGE PRECISION (MAP)

Through this, we can determine the value of mAP with Equation (17), which is just the mean of AP among all classes. The total AP across all C classes will be aggregated and then divided by the number of classes.

$$mAP = \frac{1}{C} \sum_{i=1}^{C} AP_i \tag{18}$$

The calculation of mean Average Precision (mAP) typically incorporates the utilization of IoU thresholds, often represented as mAP50, mAP@0.5, mAPAvailIoU = 0.5], or some other similar variations, if the threshold is set at 50%. Additionally, another commonly employed threshold is 50-95%.

3) MEAN AVERAGE RECALL (MAR)

To assess the performance of object detection algorithms, Average Recall (AR) measures the model's ability to identify all relevant instances, represented by ground-truth bounding boxes. Given the confidence level $\tau(k)$ and the IOU threshold $t(o)$, where $Pr_{t(o)}(\tau(k))$ and $Rc_{t(o)}(\tau(k))$ represent the Precision-Recall (PR) points, the computation of AR is expressed as:

$$AR = \frac{1}{O} \sum_{o=1}^{O} \max_{k | Pr_{t(o)}(\tau(k)) > 0} \left\{ Rc_{t(o)}(\tau(k)) \right\} \tag{19}$$

In this computation, the process entails determining the mean of the highest recall values while ensuring that precision exceeds zero for every IOU threshold, and $\tau(k)$ is defined according to the set of K different confident. This process provides an approximate integral of the function that retrieves the recall for a given IoU with O values, given that the O

IOU threshold set, $t(o)$, adequately covers a range of overlaps, where O is the size of such set.

Through Equation (20), we can calculate the mAR by averaging the AR with the number of classes.

$$mAR = \frac{1}{C}\sum_{i=1}^{C} AR_i \tag{20}$$

4) INFERENCE TIME

Beyond the established metrics, the inclusion of inference time provides a supplementary dimension to gauge a model's performance. Notably, in the scope of YOLO, renowned for its real-time object detection capabilities, inference time emerges as a pivotal metric. This metric examines the temporal domain, providing insights into the speed at which a model processes and detects objects in real-time scenarios. More specifically, it denotes the duration required for a model to analyse an input image or frame and generate the corresponding output, encompassing the identification and localization of objects. In leveraging inference time as a metric, we gain a comprehensive understanding of a model's practical efficiency, particularly crucial in applications where swift decision-making is paramount.

To solidify our exploration of inference time as a performance metric, we implemented the ONNX Runtime. This strategic choice aims to optimize and accelerate the inference process. The ONNX Runtime acts as a cross-platform model booster and introduces efficiency enhancements, which are particularly beneficial for devices with constrained computational resources, such as mobile devices. Mobile devices can benefit greatly from the expedited inference capabilities facilitated by ONNX Runtime, given their limited capabilities. This optimization ensures a more responsive and swift inference process, aligning with the demands of applications where real-time responsiveness is of paramount importance.

5) MEAN AVERAGE ERROR (MAE)

To evaluate the model over all images, the MAE value is determined by calculating the absolute difference for each class individually. This involves calculating the absolute difference between the ground truth counts (G_i) and the predicted counts, $P_i(\tau)$, for each class (c) in N images. Notably, predicted counts are only considered if they exceed a given confidence threshold (τ). We then infer the MAE, as depicted in Equation (21). The overall MAE, as described in Equation (22), is then obtained by averaging the MAE values across all classes (C), providing a comprehensive evaluation of the model's accuracy in estimating object counts across the entire dataset.

$$MAE(c) = \frac{1}{N}\sum_{i=1}^{N} |G_i - P_i(\tau)| \tag{21}$$

$$\text{Overall MAE} = \frac{1}{C}\sum_{c=1}^{C} MAE(c) \tag{22}$$

6) ROOT MEAN SQUARE ERROR (RMSE)

The overall approach to computing the RMSE closely mirrors that of the MAE in the provided code. For each class (c) and for each of the N images, the squared difference between the ground truth counts (G_i) and the predicted counts that meet the confidence threshold ($P_i(\tau)$) are computed, as portrayed in Equation (23). Once the RMSE for each class has been determined, the overall RMSE, presented in Equation (24), is obtained by averaging these class specific RMSE values across the total number of classes (C). This method ensures a comprehensive evaluation of the model's performance in estimating object counts, taking into account both the magnitude and direction of the errors.

$$RMSE(c) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(G_i - P_i(\tau))^2} \tag{23}$$

$$\text{Overall RMSE} = \frac{1}{C}\sum_{c=1}^{C} RMSE(c) \tag{24}$$

G. WEB APPLICATION

Our web application incorporates WebAssembly (wasm) as a pivotal element to optimize the execution of our deep learning model within a web environment. By designating wasm as the execution provider during the instantiation of the inference session, our application capitalizes on the capabilities of WebAssembly for accelerated model execution. This strategic use of wasm on client devices eliminates the necessity for server-side processing, ensuring seamless integration and compatibility with our web-based application.

In the operational framework of the application, we leverage the ONNX runtime for web applications to execute any YOLO model. To initiate model execution, a session is created for the model. Real-time input is pre-processed to conform to the model's input shape, adhering to the NCHW format (number of batches, channel, height, and width), and adjusting the numerical precision based on the ONNX runtime model employed. Subsequently, the model is invoked to infer the pre-processed input, generating the desired output, including detection results for each identified object along with the corresponding label.

Following the model inference, a post-processing stage is enacted, wherein each output is mapped to its corresponding label, and bounding boxes are displayed in alignment with these labels. The detection results include confidence scores, enabling the application to filter out results below a predetermined threshold. Concurrently, the counting process is integrated into the post-processing stage, estimating the count for each detected class in the frame. This final step enables the display of bounding boxes for each object based on its class, accompanied by an accurate count of objects within the current frame.

TABLE 2. Overall performance for each model (with data augmentation).

Model	mAP_{50}	mAP_{50-95}	mAR_{50-95}	MAE	RMSE
YOLOv6s	0.774	0.503	0.707	0.184	0.45
YOLOv6l	**0.803**	**0.538**	**0.736**	0.194	0.466
YOLOv7tiny	0.736	0.417	0.637	0.16	0.398
YOLOv7l	0.750	0.486	0.677	**0.157**	0.397
YOLOv8s	0.759	0.500	0.658	0.148	**0.394**
YOLOv8l	0.786	0.517	0.684	0.165	0.411

TABLE 3. Overall performance for each model (without data augmentation).

Model	mAP_{50}	mAP_{50-95}	mAR_{50-95}	MAE	RMSE
YOLOv6s	0.729	0.480	0.711	0.198	0.473
YOLOv6l	**0.781**	**0.514**	**0.729**	0.186	0.457
YOLOv7tiny	0.730	0.431	0.641	**0.159**	**0.402**
YOLOv7l	0.733	0.472	0.678	**0.159**	0.403
YOLOv8s	0.719	0.471	0.656	0.167	0.413
YOLOv8l	0.770	0.506	0.676	0.164	0.412

III. RESULTS AND DISCUSSIONS

Table 2 and 3 offer a comprehensive comparison of the impact of data augmentation on the performance of each model. Notably, larger models such as YOLOv6l, YOLOv7l, and YOLOv8l demonstrate a clear benefit when trained with augmented data. YOLOv6l, in particular, exhibits superior performance in terms of mAP50, mAP50-95, and mAR50-95 metrics, both with and without data augmentation. These results indicate its robustness and ability to generalize across different datasets. The 'l' (large) variants typically offer more layers and parameters, enabling them to capture complex features more effectively, which could explain their enhanced performance in our analysis.

Upon examining the impact of data augmentation, we observed a positive trend in mAP50 and mAP50-95 scores across all models. However, the difference in mAR50-95, MAE, and RMSE is relatively insubstantial. This suggests that while data augmentation certainly aids in improving the model's ability to detect objects accurately (as reflected in mAP scores), its impact on the overall recall (mAR) and error metrics (MAE, RMSE) is more nuanced.

As shown in Table 4, the performance of YOLOv6s, YOLOv6l, YOLOv7tiny, YOLOv7l, YOLOv8s, YOLOv8l, and YOLOv8s Depthwise models was evaluated across five distinct classes: Abnormal, Flower, Ripe, Underripe, and Unripe. These classes were assessed based on Average Precision (AP50 and AP50-95) and error metrics (MAE and RMSE). Analyzing the data, it is observed that the YOLOv6 series, particularly YOLOv6s, demonstrates exceptional performance in terms of AP50 (0.874) and AP50-95 (0.543) in the Abnormal class, indicating its robust capability in accurately detecting crop anomalies. In contrast, the YOLOv7tiny model excels in reducing counting error rates, achieving the lowest MAE (0.064) and RMSE (0.253) in the Abnormal class and MAE (0.105) and RMSE (0.323) in the Flower class. This distinction suggests that YOLOv7tiny is particularly effective in tasks requiring precise counting, such as distinguishing subtle variations in crop conditions. The proposed YOLOv8s Depthwise model shows a well-rounded performance, balancing both detection accuracy and counting error metrics across all classes. For instance, in the Abnormal class, it maintains a low MAE and RMSE, indicating its proficiency in accurate detection and counting. Similarly, in the Flower and Ripe performance showcasing its versatility. Notably, the YOLOv8s Depthwise model also excels in the Underripe and Unripe classes, maintaining a balanced performance that makes it a reliable choice for identifying different maturity stages of crops.

Additionally, the YOLOv8s Depthwise model has a significant advantage in terms of model size. With a size of just 21.2 MB, it is the smallest among all the models evaluated, which includes the much larger YOLOv6l at 223.7 MB and YOLOv8l at 174.5 MB. This compact size is particularly beneficial for deployment in resource-constrained environments, offering ease of implementation without compromising on performance. When accuracy in detection (reflected by high AP scores) is prioritized, the YOLOv6 series is preferable due to its superior detection capabilities. However, for tasks where reducing classification and counting errors is crucial, the YOLOv7 series, especially the YOLOv7tiny and YOLOv7l models, provide significant advantages due to their low MAE and RMSE values. Meanwhile, the YOLOv8 series, and particularly the YOLOv8s Depthwise model, presents a well-rounded solution with balanced performance across various classes and a compact model size.

For the proposed model that is YOLOv8s Depthwise, we also performed hyperparameter tuning for its base learning rate, and optimizer, with the performance results shown in Table 5. The Adam optimizer, known for its adaptive learning rate properties, showed robust performance at lower learning rates (0.0001 and 0.001), with stable detection accuracy and moderate error rates. However, at a higher learning rate (0.01), the performance deteriorated, indicating overfitting or training instability. Conversely, the SGD optimizer, which is simpler but can generalize better, required a moderate learning rate (0.001) to achieve optimal performance. At this learning rate, the model demonstrated balanced detection accuracy and low error rates, highlighting effective learning. Lower learning rates (0.0001) with SGD resulted in inadequate learning, while higher rates (0.01) maintained stable performance but with slightly increased errors. Quantization to fp16 consistently resulted in a negligible increase in inference time while maintaining similar performance metrics compared to non-quantized models, proving it as a viable strategy for reducing computational load without sacrificing accuracy.

As outlined in Figure 9, YOLOv6l shows a steady decline in loss, suggesting the effectiveness of its EfficientRep backbone and RepBlocks. However, its larger parameter space leads to a slower convergence rate compared to models

TABLE 4. Class-wise performance comparison for each model: abnormal, flower, ripe, underripe, and unripe.

Model	Abnormal				Flower				Ripe				Underripe				Unripe			
	AP_{50}	AP_{50-95}	MAE	RMSE	AP_{50}	AP_{50-95}	MAE	RMSE	AP_{50}	AP_{50-95}	MAE	RMSE	AP_{50}	AP_{50-95}	MAE	RMSE	AP_{50}	AP_{50-95}	MAE	RMSE
YOLOv6s	**0.874**	**0.543**	0.089	0.298	0.73	0.364	0.142	0.414	0.725	0.546	0.212	0.485	0.673	0.494	0.158	0.4	0.869	0.568	0.319	0.653
YOLOv6l	0.809	0.493	0.138	0.376	**0.731**	**0.39**	0.149	0.418	**0.841**	**0.643**	0.166	0.423	0.757	**0.557**	0.166	0.414	**0.88**	**0.606**	0.35	0.699
YOLOv7 tiny	0.808	0.441	**0.064**	**0.253**	0.56	0.227	**0.105**	**0.323**	0.807	0.586	0.132	0.37	0.71	0.516	0.122	0.352	0.792	0.512	0.376	0.69
YOLOv7l	0.771	0.463	0.1	0.321	0.601	0.29	0.138	0.372	0.797	0.579	**0.118**	**0.347**	**0.758**	0.554	**0.119**	**0.344**	0.825	0.545	0.31	0.6
YOLOv8s	0.77	0.46	0.082	0.291	0.51	0.26	0.158	0.398	0.778	0.562	0.153	0.407	0.682	0.518	0.157	0.403	0.869	0.577	**0.238**	**0.534**
YOLOv8l	0.861	**0.543**	0.082	0.291	0.688	0.332	0.136	0.369	0.79	0.598	0.171	0.426	0.716	0.518	0.14	0.377	0.875	0.591	0.295	0.59
YOLOv8s Depthwise (Proposed)	0.77	0.47	0.097	0.311	0.61	0.279	0.15	0.391	0.787	0.556	0.15	0.39	0.749	0.557	0.153	0.398	0.832	0.55	0.354	0.672

TABLE 5. Performance evaluation of hyperparameter scenarios: optimizer and learning rate.

Batch Size	Optimizer	Learning Rate	Inference Time (s)		mAP_{50}		mAP_{50-95}		MAE		RMSE	
			Non-quantized	Quantized (fp16)	Non-quantized	Quantized (fp16)	Non-quantized	Quantized (fp16)	Non-quantized	Quantized (fp16)	Non-quantized	Quantized (fp16)
16	Adam	0.0001	0.0281	0.028	0.739	0.733	0.469	0.464	0.192	0.198	0.446	0.45
		0.001	0.0283	0.028	0.735	0.733	0.475	0.473	0.190	0.194	0.443	0.446
		0.01	**0.0273**	0.0271	0.669	0.668	0.415	0.414	0.232	0.234	0.492	0.495
	SGD	0.0001	0.0279	0.0276	0.003	0.002	0.001	0.001	0.401	0.401	0.658	0.661
		0.001	0.0275	**0.027**	**0.75**	**0.749**	0.482	0.481	**0.164**	**0.164**	**0.411**	**0.411**
		0.01	0.0277	0.0271	0.742	0.740	**0.483**	**0.482**	0.181	0.183	0.433	0.436

with fewer parameters. The training time for YOLOv6l was 8 hours, 16 minutes, and 20 seconds. In contrast, YOLOv6s demonstrates the most rapid initial loss reduction, likely due to its efficient backbone and task alignment learning. This model completed training in 3 hours, 30 minutes, and 4 seconds.

YOLOv7l also shows a consistent decrease in loss, benefiting from its E-ELAN architecture that optimizes gradient flow and model scaling. The training duration for YOLOv7l was 7 hours, 32 minutes, and 59 seconds. However, YOLOv7tiny exhibits a slower and more fluctuating loss curve, possibly due to its lightweight design being more sensitive to learning rate adjustments. Nevertheless, it still achieves a competitive final loss, with a training time of 3 hours, 30 minutes, and 3 seconds.

Similarly, YOLOv8l displays a smooth and steady loss decrease, highlighting the optimized performance of its C2f module and decoupled head. While its convergence speed is moderate, similar to YOLOv6l, it effectively reduces loss over time. The training time for this model was 5 hours, 12 minutes, and 15 seconds. Interestingly, YOLOv8s shows a slower convergence compared to other models, even though its loss decreases steadily. This suggests that while its architecture is efficient, it might require more training epochs to reach its full potential. YOLOv8s completed its training in 2 hours, 39 minutes, and 41 seconds. Finally, YOLOv8s depthwise stands out with the lowest loss over time, demonstrating the efficiency of depth-wise separable convolutions in achieving rapid convergence and maintaining low loss throughout the training process. This model had the shortest training time of 2 hours, 18 minutes, and 30 seconds.

The comparison shown in Figure 10 and 11 provides a comprehensive evaluation of the counting results obtained by our object detection models. In Figure10, the examination of the ground truth against the counting results reveals different types of errors. Figure 10(a) illustrates instances where the same object is counted twice, highlighting potential challenges in the detection process. Conversely, Figure 10(b) shows situations where the model fails to detect a single object, highlighting the importance of addressing false negatives. On the positive side, Figure 11 provides a reassuring representation of accurate counting results that closely match the ground truth.

As presented in Table 6 of our study, the ONNX runtime converted models which deliver the conversion and optimization of models for deployment across various platforms were analyzed to understand their performance in comparison to their original counterparts. Notably, the metrics mAP50, mAP50-95 and inference time were used as the primary evaluation criteria. The analysis revealed that the mAP50 and mAP50-95 scores of the ONNX runtime converted models were identical to those of the original models, indicating that the conversion process preserved the models' accuracy effectively. This consistency is significant as it ensures that the deployment on various platforms does not compromise the model's detection capabilities.

Further, a comparison was conducted between the quantized (fp16) and non-quantized versions of the ONNX

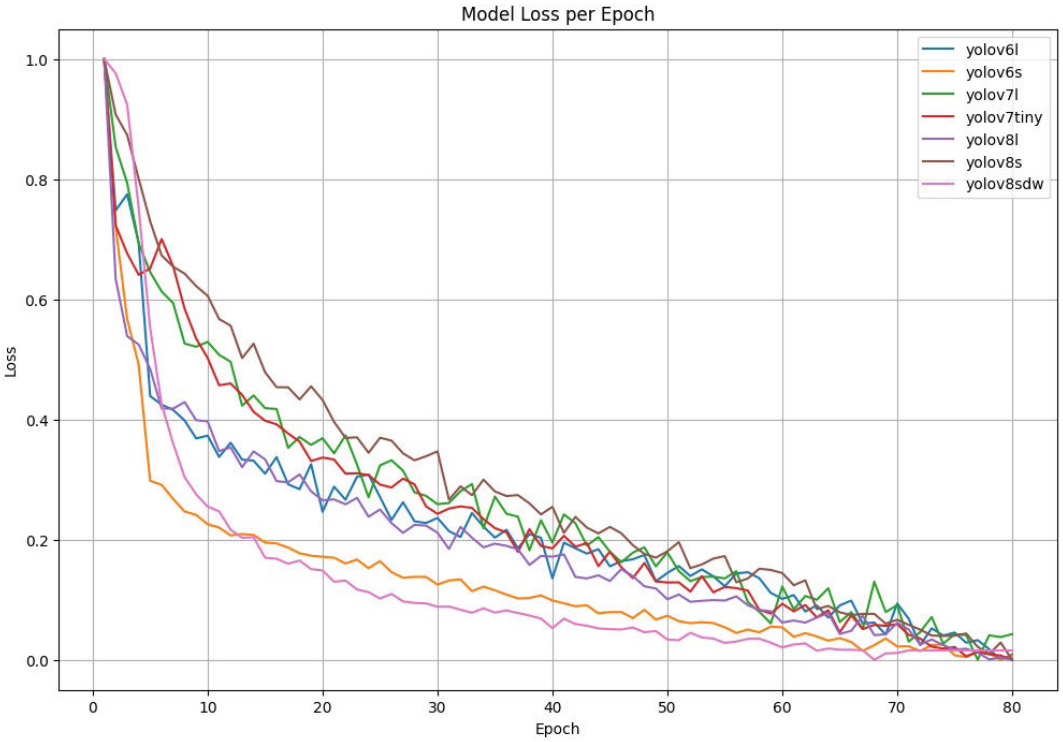

FIGURE 9. Comparison of training loss for different YOLO model variants: YOLOv6s (orange), YOLOv6l (blue), YOLOv7tiny (dark red), YOLOv7l (green), YOLOv8s (brown), YOLOv8l (purple), and YOLOv8s Depthwise (pink).

TABLE 6. ONNX runtime comparison for each model.

Model	Inference Time (s)		mAP_{50}		mAP_{50-95}		Size (MB)		MAE		RMSE	
	Non-quantized	Quantized (fp16)	Non-quantized	Quantized (fp16)	Non-quantized	Quantized (fp16)	Non-quantized	Quantized (fp16)	Non-quantized	Quantized (fp16)	Non-quantized	Quantized (fp16)
YOLOv6s	0.0264	0.0224	0.774	0.774	0.501	0.501	68.8	34.4	0.184	0.185	0.451	0.451
YOLOv6l	0.0632	0.0378	**0.803**	**0.803**	**0.537**	**0.538**	223.7	116.925	0.194	0.194	0.466	0.466
YOLOv7 tiny	**0.0248**	**0.0219**	0.735	0.735	0.414	0.415	24.1	12.1	0.156	0.155	0.398	0.398
YOLOv7l	0.0523	0.0328	0.75	0.75	0.485	0.485	146.1	73.1	0.157	0.157	0.397	0.397
YOLOv8s	0.0276	0.027	0.759	0.762	0.5	0.5	44.5	22.3	**0.148**	**0.147**	**0.394**	**0.392**
YOLOv8l	0.0550	0.0314	0.786	0.786	0.516	0.516	174.5	87.3	0.164	0.164	0.411	0.411
YOLOv8s Depthwise (Proposed)	0.0275	0.027	0.75	0.749	0.482	0.481	**21.2**	**10.6**	0.163	0.164	0.399	0.400

models. Remarkably, the quantization process did not lead to any substantial differences in mAP50 and mAP50-95 scores. The variations were marginal, often as little as 0.001, which is negligible in practical scenarios. This finding is crucial as it demonstrates that model quantization, a process often used to reduce model size and increase computational efficiency, does not adversely affect the model's efficacy in detecting objects.

The most significant observation from the quantization process was its impact on inference time. Despite the minimal effect on accuracy, the inference time was notably reduced, particularly for larger models. For instance, the inference time for the YOLOv6l model was halved from 0.0632s (original) to 0.0378s (fp16), and similar trends were observed across other versions like YOLOv7 and YOLOv8. The YOLOv7tiny model, for example, showed a reduction from 0.0248s to 0.0219s, and the YOLOv8l model from 0.055s to 0.0314s in their fp16 formats. These reductions in inference time are particularly valuable in real-time applications where speed is as critical as accuracy.

YOLOv8s Depthwise model, the quantized version achieved an inference time of 0.027s compared to 0.0275s for the non-quantized version. While this reduction is relatively small, it is consistent with the trend observed across other models. The model maintained an mAP50 of 0.75 and mAP50-95 of approximately 0.481 for both non-quantized

FIGURE 10. Ground truth vs. counting errors: (a) Duplicate counting and (b) Missing object.

FIGURE 11. Ground truth vs. correct counting result.

and quantized versions, confirming that the quantization does not compromise the detection performance. In terms of size, the YOLOv8s Depthwise model exhibited a significant reduction from 21.2 MB (non-quantized) to 10.6 MB (quantized fp16). This reduction is particularly advantageous for initial loading and reduces storage allocation especially for mobile applications.

When comparing the YOLOv8s Depthwise model to other state-of-the-art models in the plantation dataset, such as seen in Table 7, the YOLOv8s Depthwise model achieves an inference time of 0.027 seconds, which is competitive among the models tested. Although SSD MobileNetV2 has a slightly faster inference time at 0.024 seconds, it significantly lags in detection accuracy with mAP50 at 0.546 and mAP50-95 at 0.306, along with higher error rates (MAE 0.321, RMSE 0.543). In terms of accuracy, the YOLOv8s model outperforms the YOLOv8s Depthwise, with mAP50 at 0.759 and mAP50-95 at 0.5. However, YOLOv8s is larger in size (22.3 MB) compared to YOLOv8s Depthwise (10.6 MB), making the latter more suitable for deployment in resource-constrained environments.

TABLE 7. Performance comparison of YOLOv8s depthwise with other SOTA models.

Model	Inference Time (s)	mAP_{50}	mAP_{50-95}	Size (MB)	MAE	RMSE
YOLOv8s Depthwise (Proposed)	0.027	0.75	0.481	**10.6**	0.164	0.400
YOLOv8s	0.027	0.759	0.5	22.3	0.147	0.392
Faster RCNN	0.0497	0.743	0.426	165.4	0.175	0.43
SSD MobileNetV2	**0.024**	0.546	0.306	12.3	0.321	0.543
YOLOv4	0.048	**0.787**	**0.513**	244.3	0.162	0.407
YOLOv9c	0.095	0.744	0.488	101.2	**0.137**	**0.388**

Faster RCNN offers competitive accuracy with an mAP50 of 0.743 but suffers from a longer inference time (0.0497 seconds) and a much larger model size (165.4 MB), limiting its practicality for real-time applications. YOLOv4 shows the highest detection accuracy (mAP50 at 0.787 and mAP50-95 at 0.513) but at the cost of a longer inference time (0.048 seconds) and the largest model size (244.3 MB), making it less feasible for mobile applications. The YOLOv8s Depthwise model balances efficiency and performance with a small size (10.6 MB), competitive inference time (0.027 seconds), and high detection accuracy (mAP50 at 0.75, mAP50-95 at 0.481). Its counting error metrics (MAE 0.164, RMSE 0.4) are also within acceptable ranges, ensuring reliable counting.

TABLE 8. Comparison between proposed model and previous works with harvested fruits dataset.

Model	Inference Time (s)	mAP_{50}	mAP_{50-95}	Size (MB)
YOLOv8s Depthwise (Proposed)	**0.026**	**0.994**	**0.875**	**10.6**
YOLOv4 [3]	0.042	0.985	0.743	244.3
EfficientDet-D0 [99]	0.03	0.993	0.871	40.1
SSD MobileNetV2 [100]	0.027	0.985	0.871	12.3

To validate the performance of the proposed model corresponding to Table 8, the YOLOv8s Depthwise model was validated to previous research that detected harvested oil palm placed on the ground [3]. The results highlight that the YOLOv8s Depthwise model achieves the fastest inference time at 0.026 seconds, outperforming other models such as SSD MobileNetV2 (0.027 seconds), EfficientDet-D0 (0.03 seconds), and YOLOv4 (0.042 seconds). This

efficiency is crucial for real-time applications where rapid processing is essential. In terms of detection accuracy, the YOLOv8s Depthwise model excels with an mAP50 of 0.994 and an mAP50-95 of 0.875, the highest among the compared models. Although EfficientDet-D0 also shows strong performance with an mAP50 of 0.993 and an mAP50-95 of 0.871, it has a larger model size of 40.1 MB. YOLOv4, despite achieving high accuracy in previous research with an mAP50 of 0.985 and an mAP50-95 of 0.743, lags in mAP50-95, indicating less precise detection across varying IoU thresholds. The YOLOv8s Depthwise model maintains a compact size of 10.6 MB, significantly smaller than YOLOv4 (244.3 MB) and EfficientDet-D0 (40.1 MB), and slightly smaller than SSD MobileNetV2 (12.3 MB). This compactness is advantageous for deployment in resource-constrained environments, ensuring efficient storage and faster model loading times.

FIGURE 12. Real-time counting in mobile device-operated web application: (a) Before and (b) After inference.

As illustrated in Figure 12(a), the mobile device-operated web application, developed with the capability for real-time inference, allows users to direct their device camera towards fresh fruit bunches (FFBs) for immediate counting. In Figure 12(a), a virtual camera was employed to replicate the process of capturing FFBs in a real-world setting. Moving to Figure 12(b), the application performs real-time detection after running the inference, displaying each identified object within bounding boxes marked with corresponding class labels and confidence scores. For enhanced clarity, each bounding box is color-coded based on its class. Moreover, the application interface below provides a real-time count of objects in the current frame. In the example of Figure 12(b), the application successfully counted two instances of unripe fruits, one instance of an abnormal fruit, and one instance of an underripe fruit. These outcomes affirm the application's ability to employ resource-efficient YOLO models, particularly YOLOv8s Depthwise, for real-time counting, accessible across various devices due to its web-based nature.

IV. CONCLUSION

The paper presented an innovative approach to palm oil yield estimation, leveraging the advancements in deep learning techniques, specifically YOLOv8 Depthwise, alongside YOLOv6, YOLOv7, and YOLOv8. These models can accurately detect and count Fresh Fruit Bunches (FFBs) in plantations. The research leveraged a self-gathered dataset of high complexity, comprising video recordings of FFBs still attached to trees. A key aspect of this study was the emphasis on estimating the number of FFBs during model inference. To run on devices with limited resources, the models were converted and quantized with ONNX runtime for deployment in a web application.

Our experimental results highlight distinct trends and strengths among the models. Hyperparameter tuning revealed that the YOLOv8 Depthwise model achieved optimal performance using the SGD optimizer with a batch size of 16 and a learning rate of 0.001. The YOLOv8 Depthwise model demonstrated superior performance with a mAP50 of 0.75, mAP50-95 of 0.481, MAE of 0.164, RMSE of 0.4, and an inference time of 0.027 seconds. Quantization further improved efficiency, reducing model size (21.2 MB to 10.6 MB) with minimal impact on speed (0.0275s to 0.0273s), while maintaining high detection accuracy.

When evaluated against other state-of-the-art models on the same dataset, including Faster RCNN, SSD MobileNetV2, and YOLOv4, YOLOv8 Depthwise maintained superior performance in speed, accuracy, and efficiency. Additionally, comparisons with other datasets from previous research further validated the model's robustness and adaptability against models like YOLOv4, EfficientDet-D0, and SSD MobileNetV2.

This research shows that automatically detecting ripeness in fruit bunches can save labor, reduce errors, and improve efficiency for both small and large farms. This not only helps businesses be more competitive but also reduces waste and promotes sustainable practices in the palm oil industry. The YOLOv8 Depthwise model is a particularly good solution for this task because it's fast and accurate, even on devices with low processing power.

Despite these promising results, there are some limitations to this research. The dataset, while comprehensive, is limited to the specific conditions of the Central Kalimantan Province in Indonesia. Variations in environmental conditions, palm species, and agricultural practices in different regions could affect the model's generalizability. Additionally, the deployment of the model in real-world scenarios might face challenges related to hardware constraints, network connectivity for the web application, and the need for continuous updating and retraining of the model to adapt to changing conditions and new data.

Future work will focus on addressing these limitations, expanding the dataset, and further optimizing the model for diverse agricultural environments. Additionally, there is significant potential for the application of YOLO technology in broader agricultural contexts, such as improving crop monitoring, pest detection, yield estimation, and precision farming, enhancing overall productivity and sustainability in agriculture.

REFERENCES

[1] K. Hashim, S. Tahiruddin, and A. J. Asis, "Palm and palm kernel oil production and processing in Malaysia and Indonesia," in *Palm Oil*. Amsterdam, The Netherlands: Elsevier, 2012, pp. 235–250.

[2] M. S. Muna, A. P. Nugroho, M. Syarovy, A. Wiratmoko, and L. Sutiarso, "Development of automatic counting system for palm oil tree based on remote sensing imagery," in *Proc. Int. Conf. Sustain. Environ.*, 2022, pp. 503–508.

[3] M. Asrol, D. N. Utama, and F. A. Junior, "Real-time oil palm fruit grading system using smartphone and modified YOLOv4," *IEEE Access*, vol. 11, pp. 59758–59773, 2023.

[4] J. W. Lai, H. R. Ramli, L. I. Ismail, and W. Z. W. Hasan, "Real-time detection of ripe oil palm fresh fruit bunch based on YOLOv4," *IEEE Access*, vol. 10, pp. 95763–95770, 2022.

[5] U.S. Dept. Agricult. Foreign Agricult. Service (FAS). (2024). *Palm Oil World Production*. Accessed: Jan. 22, 2024. [Online]. Available: https://ipad.fas.usda.gov/cropexplorer/commodityView.aspx?cropid=4243000

[6] Malaysian Palm Oil Council (MPOC). (2023). *Malaysian Palm Oil Industry*. Accessed: Jan. 22, 2024. [Online]. Available: https://mpoc.org.my/malaysian-palm-oil-industry

[7] M. Barrientos and C. Soria. *Agricultural Production Statistics By Country*. Accessed: Apr. 4, 2023. [Online]. Available: https://www.indexmundi.com/agriculture

[8] X. Liu, K. H. Ghazali, F. Han, and I. I. Mohamed, "Automatic detection of oil palm tree from UAV images based on the deep learning method," *Appl. Artif. Intell.*, vol. 35, no. 1, pp. 13–24, Oct. 2020.

[9] F. B. Ahmad, Z. Zhang, W. O. S. Doherty, and I. M. O'Hara, "The outlook of the production of advanced fuels and chemicals from integrated oil palm biomass biorefinery," *Renew. Sustain. Energy Rev.*, vol. 109, pp. 386–411, Jul. 2019.

[10] H. Purnomo, B. Okarda, A. A. Dewayani, M. Ali, R. Achdiawan, H. Kartodihardjo, P. Pacheco, and K. S. Juniwaty, "Reducing forest and land fires through good palm oil value chain governance," *Forest Policy Econ.*, vol. 91, pp. 94–106, Jun. 2018.

[11] W. J. Henderson, O. Purba, H. I. Purba, and T. Juliana, "Oil palm (Elaeis guineensis Jacq.) bunch structure variation and limitations," *Sci. Res. Journal.*, vol. 3, no. 1, pp. 5–10, 2015.

[12] C. L. Chew, C. Y. Ng, W. O. Hong, T. Y. Wu, Y.-Y. Lee, L. E. Low, P. S. Kong, and E. S. Chan, "Improving sustainability of palm oil production by increasing oil extraction rate: A review," *Food Bioprocess Technol.*, vol. 14, no. 4, pp. 573–586, Apr. 2021.

[13] M. S. M. Kassim, W. I. W. Ismail, A. R. Ramli, and S. K. Bejo, "Oil palm fresh fruit bunches (FFB) growth determination system to support harvesting operation," *J. Food Agricult. Environ.*, vol. 10, no. 2, pp. 620–625, 2012.

[14] M. K. Shabdin, A. R. M. Shariff, M. N. A. Johari, N. K. Saat, and Z. Abbas, "A study on the oil palm fresh fruit bunch (FFB) ripeness detection by using Hue, saturation and intensity (HSI) approach," in *Proc. IOP Conf. Series, Earth Environ. Sci.*, vol. 37, 2016, p. 012039.

[15] I. Omar, M. K. Ashhar, and M. W. Basri, "Colour meter for measuring fruit ripeness," *MPOB Inf. Ser.*, no. 182, p. 195, 2003.

[16] A. Aripriharta, A. Firmansah, N. Mufti, G.-J. Horng, and N. Rosmin, "Smartphone for palm oil fruit counting to reduce embezzlement in harvesting season," *Bull. Social Informat. Theory Appl.*, vol. 4, no. 2, pp. 76–82, Sep. 2020.

[17] K. Yamamoto, W. Guo, Y. Yoshioka, and S. Ninomiya, "On plant detection of intact tomato fruits using image analysis and machine learning methods," *Sensors*, vol. 14, no. 7, pp. 12191–12206, Jul. 2014.

[18] W. Maldonado and J. C. Barbosa, "Automatic green fruit counting in orange trees using digital images," *Comput. Electron. Agricult.*, vol. 127, pp. 572–581, Sep. 2016.

[19] R. Hamza and M. Chtourou, "Design of fuzzy inference system for apple ripeness estimation using gradient method," *IET Image Process.*, vol. 14, no. 3, pp. 561–569, Jan. 2020.

[20] W. Castro, J. Oblitas, M. De-La-Torre, C. Cotrina, K. Bazán, and H. Avila-George, "Classification of cape gooseberry fruit according to its level of ripeness using machine learning techniques and different color spaces," *IEEE Access*, vol. 7, pp. 27389–27400, 2019.

[21] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019.

[22] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, no. 1, pp. 1–74, Mar. 2021.

[23] W. Yang, Q. Liu, S. Wang, Z. Cui, X. Chen, L. Chen, and N. Zhang, "Down image recognition based on deep convolutional neural network," *Inf. Process. Agricult.*, vol. 5, no. 2, pp. 246–252, Jun. 2018.

[24] F. Khoroshevsky, S. Khoroshevsky, and A. Bar-Hillel, "Parts-per-object count in agricultural images: Solving phenotyping problems via a single deep neural network," *Remote Sens.*, vol. 13, no. 13, p. 2496, Jun. 2021.

[25] L. Shen, J. Su, R. He, L. Song, R. Huang, Y. Fang, Y. Song, and B. Su, "Real-time tracking and counting of grape clusters in the field based on channel pruning with YOLOv5s," *Comput. Electron. Agricult.*, vol. 206, Mar. 2023, Art. no. 107662.

[26] K. Kipli, S. Osman, A. Joseph, H. Zen, D. N. S. D. Awang Salleh, A. Lit, and K. L. Chin, "Deep learning applications for oil palm tree detection and counting," *Smart Agricult. Technol.*, vol. 5, Oct. 2023, Art. no. 100241.

[27] S. N. A. B. M. Robi, M. A. B. M. Izhar, M. B. Sahrim, and N. B. Ahmad, "Image detection and classification of oil palm fruit bunches," in *Proc. 4th Int. Conf. Smart Sensors Appl. (ICSSA)*, Jul. 2022, pp. 108–113.

[28] M. H. Junos, A. S. Mohd Khairuddin, S. Thannirmalai, and M. Dahari, "Automatic detection of oil palm fruits from UAV images using an improved YOLO model," *Vis. Comput.*, vol. 38, no. 7, pp. 2341–2355, Apr. 2021.

[29] S. Puttinaovarat, S. Chai-Arayalert, and W. Saetang, "Oil palm bunch ripeness classification and plantation verification platform: Leveraging deep learning and geospatial analysis and visualization," *ISPRS Int. J. Geo-Inf.*, vol. 13, no. 5, p. 158, May 2024.

[30] K. Yarak, A. Witayangkurn, K. Kritiyutanont, C. Arunplod, and R. Shibasaki, "Oil palm tree detection and health classification on high-resolution imagery using deep learning," *Agriculture*, vol. 11, no. 2, p. 183, Feb. 2021.

[31] Y.-P. Huang, T.-H. Wang, and H. Basanta, "Using fuzzy mask R-CNN model to automatically identify tomato ripeness," *IEEE Access*, vol. 8, pp. 207672–207682, 2020.

[32] L. Fu, J. Duan, X. Zou, J. Lin, L. Zhao, J. Li, and Z. Yang, "Fast and accurate detection of banana fruits in complex background orchards," *IEEE Access*, vol. 8, pp. 196835–196846, 2020.

[33] O. M. Lawal, "YOLOMuskmelon: Quest for fruit detection speed and accuracy using deep learning," *IEEE Access*, vol. 9, pp. 15221–15227, 2021.

[34] Q. An, K. Wang, Z. Li, C. Song, X. Tang, and J. Song, "Real-time monitoring method of strawberry fruit growth state based on YOLO improved model," *IEEE Access*, vol. 10, pp. 124363–124372, 2022.

[35] J. Guo, Y. Yang, X. Lin, M. Sohail Memon, W. Liu, M. Zhang, and E. Sun, "Revolutionizing agriculture: Real-time ripe tomato detection with the enhanced tomato-YOLOv7 system," *IEEE Access*, vol. 11, pp. 133086–133098, 2023.

[36] J. Meng, F. Kang, Y. Wang, S. Tong, C. Zhang, and C. Chen, "Tea buds detection in complex background based on improved YOLOv7," *IEEE Access*, vol. 11, pp. 88295–88304, 2023.

[37] Z. Chen, S. Elsaid, D. Y. Y. Sim, T. Maul, and I. Y. Liao, "Detection of oil palm fresh fruit bunches (FFBS) with computer vision models," in *Proc. Int. Conf. Mech., Autom. Electr. Eng. (CMAEE)*, Dec. 2022, pp. 86–91.

[38] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, and X. Wei, "YOLOv6: A single-stage object detection framework for industrial applications," 2022, *arXiv:2209.02976*.

[39] C.-Y. Wang, A. Bochkovskiy, and H.-Y. Mark Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 7464–7475.

[40] G. Jocher, A. Chaurasia, and J. Qiu. *Ultralytics YOLOv8 (Version 8.0.0)*. Accessed: Nov. 5, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[41] S. Norkobil Saydirasulovich, A. Abdusalomov, M. K. Jamil, R. Nasimov, D. Kozhamzharova, and Y.-I. Cho, "A YOLOv6-based improved fire detection approach for smart city environments," *Sensors*, vol. 23, no. 6, p. 3161, Mar. 2023.

[42] R. B. Bist, S. Subedi, X. Yang, and L. Chai, "A novel YOLOv6 object detector for monitoring piling behavior of cage-free laying hens," *AgriEngineering*, vol. 5, no. 2, pp. 905–923, May 2023.

[43] Y. Wang, H. Wang, and Z. Xin, "Efficient detection model of steel strip surface defects based on YOLO-V7," *IEEE Access*, vol. 10, pp. 133936–133944, 2022.

[44] L. Cao, X. Zheng, and L. Fang, "The semantic segmentation of standing tree images based on the YOLO v7 deep learning algorithm," *Electronics*, vol. 12, no. 4, p. 929, Feb. 2023.

[45] O. K. T. Alsultan and M. T. Mohammad, "A deep learning-based assistive system for the visually impaired using YOLO-v7," *Revue D'Intell. Artificielle*, vol. 37, no. 4, pp. 901–906, Aug. 2023.

[46] D. Wu, S. Jiang, E. Zhao, Y. Liu, H. Zhu, W. Wang, and R. Wang, "Detection of camellia oleifera fruit in complex scenes by using YOLOv7 and data augmentation," *Appl. Sci.*, vol. 12, no. 22, p. 11318, Nov. 2022.

[47] B. Gasparović, G. Mausa, J. Rukavina, and J. Lerga, "Evaluating YOLOv5, YOLOv6, YOLOv7, and YOLOv8 in underwater environment: Is there real improvement?" in *Proc. 8th Int. Conf. Smart Sustain. Technol. (SpliTech)*, Jun. 2023, pp. 1–4.

[48] A. Hattak, G. Iadarola, F. Martinelli, F. Mercaldo, and A. Santone, "Benchmarking YOLO models for automatic reading in smart metering systems: A performance comparison analysis," in *Proc. Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2023, pp. 2207–2213.

[49] H. Lou, X. Duan, J. Guo, H. Liu, J. Gu, L. Bi, and H. Chen, "DC-YOLOv8: Small size object detection algorithm based on camera sensor," *Electronics*, vol. 12, no. 10, p. 2323, 2023.

[50] R. Hong, X. Wang, Y. Fang, H. Wang, C. Wang, and H. Wang, "YOLO-Light: Remote straw-burning smoke detection based on depthwise separable convolution and channel attention mechanisms," *Appl. Sci.*, vol. 13, no. 9, p. 5690, May 2023.

[51] X. Tang, X. Chen, J. Cheng, J. Wu, R. Fan, C. Zhang, and Z. Zhou, "YOLO-Ant: A lightweight detector via depthwise separable convolutional and large kernel design for antenna interference source detection," *IEEE Trans. Instrum. Meas.*, vol. 73, pp. 1–18, 2024.

[52] T. Liu, B. Pang, L. Zhang, W. Yang, and X. Sun, "Sea surface object detection algorithm based on YOLO v4 fused with reverse depthwise separable convolution (RDSC) for USV," *J. Mar. Sci. Eng.*, vol. 9, no. 7, p. 753, Jul. 2021.

[53] Y. Qiao, Y. Hu, Z. Zheng, H. Yang, K. Zhang, J. Hou, and J. Guo, "A counting method of red jujube based on improved YOLOv5s," *Agriculture*, vol. 12, no. 12, p. 2071, Dec. 2022.

[54] B. Sekachev, N. Manovich, M. Zhiltsov, A. Zhavoronkov, D. Kalinin, B. Hoff, D. Kruchinin, A. Zankevich, M. Markelov, M. Chenuet, A. Melnikov, J. Kim, L. Ilouz, N. Glazov, Priya, R. Tehrani, S. Jeong, V. Skubriev, S. Yonekura, V. Truong, Zliang, Lizhming, and T. Truong. (Aug. 2020). *openCV/CVAT: V1.1.0*. Accessed: Nov. 12, 2023. [Online]. Available: https://doi.org/10.5281/zenodo.4009388

[55] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, p. 125, Feb. 2020.

[56] MMDetection Contributors. *Openmmlab Toolbox and Benchmark*. Accessed: Nov. 8, 2023. [Online]. Available: https://openmmlab.com

[57] 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. (CVPR)*, Jun. 2016, pp. 779–788.

[58] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[59] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[60] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[61] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019.

[62] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6517–6525.

[63] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

[64] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[65] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, Y. Kwon, K. Michael, J. Fang, Z. Yifu, C. Wong, A. V, D. Montes, Z. Wang, C. Fati, J. Nadar, V. Sonck, P. Skalski, A. Hogan, D. Nair, M. Strobel, and M. Jain. (Nov. 2022). *Ultralytics/YOLOv5: V7.0—YOLOv5 SOTA Realtime Instance Segmentation*. Accessed: Nov. 4, 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7347926

[66] X. Ding, H. Chen, X. Zhang, K. Huang, J. Han, and G. Ding, "Re-parameterizing your optimizers rather than architectures," 2022, *arXiv:2205.15242*.

[67] C. Shu, Y. Liu, J. Gao, Z. Yan, and C. Shen, "Channel-wise knowledge distillation for dense prediction," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 5291–5300.

[68] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8759–8768.

[69] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "RepVGG: Making VGG-style ConvNets great again," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 13728–13737.

[70] C.-Y. Wang, H.-Y. Mark Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "CSPNet: A new backbone that can enhance learning capability of CNN," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 1571–1580.

[71] C. Feng, Y. Zhong, Y. Gao, M. R. Scott, and W. Huang, "TOOD: Task-aligned one-stage object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 3490–3499.

[72] H. Zhang, Y. Wang, F. Dayoub, and N. Sünderhauf, "VarifocalNet: An IoU-aware dense object detector," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 8510–8519.

[73] Z. Gevorgyan, "SIoU loss: More powerful learning for bounding box regression," 2022, *arXiv:2205.12740*.

[74] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 658–666.

[75] C.-Y. Wang, H.-Y. Mark Liao, and I.-H. Yeh, "Designing network design strategies through gradient path analysis," 2022, *arXiv:2211.04800*.

[76] 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.

[77] X. Zhai, Z. Huang, T. Li, H. Liu, and S. Wang, "YOLO-drone: An optimized YOLOv8 network for tiny UAV object detection," *Electronics*, vol. 12, no. 17, p. 3664, Aug. 2023.

[78] E. Casas, L. Ramos, E. Bendek, and F. Rivas-Echeverría, "Assessing the effectiveness of YOLO architectures for smoke and wildfire detection," *IEEE Access*, vol. 11, pp. 96554–96583, 2023.

[79] K. Xia, Z. Lv, C. Zhou, G. Gu, Z. Zhao, K. Liu, and Z. Li, "Mixed receptive fields augmented YOLO with multi-path spatial pyramid pooling for steel surface defect detection," *Sensors*, vol. 23, no. 11, p. 5114, May 2023.

[80] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 936–944.

[81] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-IoU loss: Faster and better learning for bounding box regression," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2020, vol. 34, no. 7, pp. 12993–13000.

[82] 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.

[83] L. I. Kuncheva, F. Williams, S. L. Hennessey, and J. J. Rodríguez, "A benchmark database for animal re-identification and tracking," in *Proc. IEEE 5th Int. Conf. Image Process. Appl. Syst. (IPAS)*, vol. Five, Dec. 2022, pp. 1–6.
[84] M. Zhou, Y. Bai, W. Zhang, T. Zhao, and T. Mei, "Look-into-object: Self-supervised structure modeling for object recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11771–11780.
[85] H.-W. Huang, C.-Y. Yang, Z. Jiang, P.-K. Kim, K. Lee, K. Kim, S. Ramkumar, C. Mullapudi, I.-S. Jang, C.-I. Huang, and J.-N. Hwang, "Enhancing multi-camera people tracking with anchor-guided clustering and spatio-temporal consistency ID re-assignment," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2023, pp. 5239–5249.
[86] L. Hu and J. Ren, "YOLO-LHD: An enhanced lightweight approach for helmet wearing detection in industrial environments," *Frontiers Built Environ.*, vol. 9, Nov. 2023, Art. no. 1288445.
[87] Q. Wu, B. Zhang, C. Guo, and L. Wang, "Multi-branch parallel networks for object detection in high-resolution UAV remote sensing images," *Drones*, vol. 7, no. 7, p. 439, Jul. 2023.
[88] ONNX Runtime Developers. (2021). *ONNX Runtime*. [Online]. Available: https://onnxruntime.ai
[89] D. Lu, J. Ye, Y. Wang, and Z. Yu, "Plant detection and counting: Enhancing precision agriculture in UAV and general scenes," *IEEE Access*, vol. 11, pp. 116196–116205, 2023.
[90] Vercel. (2016). *Next.Js*. [Online]. Available: https://nextjs.org
[91] (2021). *ONNX Runtime Developer*. [Online]. Available: https://www.npmjs.com/package/onnxruntime-web
[92] Y. Hui, J. Wang, and B. Li, "DSAA-YOLO: UAV remote sensing small target recognition algorithm for YOLOV7 based on dense residual super-resolution and anchor frame adaptive regression strategy," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 36, no. 1, Jan. 2024, Art. no. 101863.
[93] Y. Deng, X. Hu, D. Teng, B. Li, C. Zhang, and W. Hu, "Dynamic adjustment of hyperparameters for anchor-based detection of objects with large image size differences," *Pattern Recognit. Lett.*, vol. 167, pp. 196–203, Mar. 2023.
[94] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2999–3007.
[95] R. Wali, "Xtreme margin: A tunable loss function for binary classification problems," 2022, *arXiv:2211.00176*.
[96] Y. Tian, Y. Zhang, and H. Zhang, "Recent advances in stochastic gradient descent in deep learning," *Mathematics*, vol. 11, no. 3, p. 682, Jan. 2023.
[97] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[98] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit," *Electronics*, vol. 10, no. 3, p. 279, Jan. 2021.
[99] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10778–10787.
[100] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

MARTINUS GRADY NAFTALI is currently pursuing the master's degree in computer science with Bina Nusantara University, Jakarta.

He is a high-achieving student selected for the accelerated master's program, which recognizes undergraduates with outstanding academic achievements and facilitates a swift transition to their master's degree. He aspires to make significant contributions to this dynamic field through innovative research. Beyond academics, he is actively engaged as a junior programmer at his university, developing a learning management system. Prior to this, he gained valuable practical experience as a software developer intern at a leading global supply chain enterprise in Singapore. His research interests include deep learning and machine learning, particularly within the domain of computer vision.

GREGORY HUGO is currently pursuing the master's degree in computer science with Bina Nusantara University, Jakarta.

He stands out as a participant in the prestigious Master Track program, a carefully selected group of students chosen by the university for an accelerated and integrated transition from bachelor's to master's studies. Aside from his academic aspirations, he works as a junior programmer at one of Indonesia's largest telecommunications and digital companies. In his current position, he is a part of a team tasked with creating and maintaining a content management systems that aids the development of villages in Indonesia. This unique initiative offers an efficient path to an advanced degree in computer science, with a particular focus on artificial intelligence. His research interest includes computer vision, and he hopes to make significant contributions to the field's ongoing advancements. He has been awarded a scholarship by the university in recognition of his academic achievements.

SUHARJITO (Member, IEEE) received the master's degree in information engineering from the Sepuluh Nopember Institute of Technology (ITS), in 2000, and the Ph.D. degree in agro-industrial engineering from Bogor Agricultural University, Indonesia, in 2011.

He is currently a Senior Lecturer with the Department of Master of Industrial Engineering, Binus Graduate Program, Bina Nusantara University. His areas of expertise are computer science, engineering, decision sciences, soft computing, and information engineering. His current research interest includes computer vision, especially the use of computer vision in agriculture with the topic of detecting the maturity level of oil palm. The research, he is currently carrying out is supported by the Indonesian Directorate General of Higher Education, Research and Technology.