Multimedia Tools and Applications
https://doi.org/10.1007/s11042-023-17435-x

Simplifying YOLOv5 for deployment in a real crop monitoring setting

Emmanuel C. Nnadozie[1,2,3] · **Pablo Casaseca-de-la-Higuera**[1] · **Ogechukwu Iloanusi**[2] · **Ozoemena Ani**[3] · **Carlos Alberola-López**[1]

Received: 3 August 2022 / Revised: 25 September 2023 / Accepted: 3 October 2023

Abstract
Deep learning-based object detection models have become a preferred choice for crop detection tasks in crop monitoring activities due to their high accuracy and generalization capabilities. However, their high computational demand and large memory footprint pose a challenge for use on mobile embedded devices deployed in crop monitoring settings. Various approaches have been taken to minimize the computational cost and reduce the size of object detection models such as channel and layer pruning, detection head searching, backbone optimization, etc. In this work, we approached computational lightening, model compression, and speed improvement by discarding one or more of the three detection scales of the YOLOv5 object detection model. Thus, we derived up to five separate fast and light models, each with only one or two detection scales. To evaluate the new models for a real crop monitoring use case, the models were deployed on NVIDIA Jetson nano and NVIDIA Jetson Orin devices. The new models achieved up to 21.4% reduction in giga floating-point operations per second (GFLOPS), 31.9% reduction in number of parameters, 30.8% reduction in model size, 28.1% increase in inference speed, with only a small average accuracy drop of 3.6%. These new models are suitable for crop detection tasks since the crops are usually of similar sizes due to the high likelihood of being in the same growth stage, thus, making it sufficient to detect the crops with just one or two detection scales.

Keywords Object detection · Model simplification · Crop monitoring · YOLOv5 · Deep learning

1 Introduction

1.1 Background

Plant identification is an integral step in crop monitoring and management tasks such as crop yield estimation, weed and pest control, disease prevention and control, etc. The tedious and error-prone manual approach to crop identification has given rise to the need for autonomous techniques for crop identification. Computer vision models have been

Extended author information available on the last page of the article deployed on embedded devices and fitted to different types of farm vehicles—including ground and airborne – to realize autonomous crop detection. Traditional image processing techniques were first used for crop detection [1], however the rapidly changing field conditions such as natural lighting, crop density, leaf occlusions, etc. undermine the accuracy and robustness of those models [2, 3]. Recently, deep learning models are fast becoming the preferred choice for object detection tasks such as crop detection [4, 5]. This growing popularity of deep learning architectures is due to their superior accuracy and ability to generalize to previously unseen data. They are also capable of end-to-end learning of hierarchical features of the images, thus jettisoning the laborious task of manual feature engineering [6, 7]. Despite the shiny attractiveness of deep learning-based computer vision models, they are notoriously resource-greedy. Computational cost and memory footprint are usually high for deep learning models, thus requiring high-end graphics processing units (GPU) and large storage capacity. This computational and memory costs present a challenge when deploying deep learning-based crop detection models on embedded devices with constrained resource availability. Minimizing the resource-demand for deep learning models is currently an active research area to which we intend to contribute.

1.2 Model simplification

In time-critical applications such as self-driving cars, UAV-based crop detection and spraying, etc., real-time fast detection and high performance are important issues. Devices such as single board computers and mobile devices, on which real-time detection models are deployed are often resource constrained [8]. Conventional deep learning-based detection models are often very large if they must have high accuracy, thus requiring more computational power from the processors. On the other hand, small object detection models that exhibit less computational complexity and fast detection speed often trade accuracy for computational complexity. For real-time object detection, the ideal model performance will entail capacity for fast detections and high accuracy, with minimal computational power [9].

Model simplification and compression methods have been proposed to reduce the computational requirement of deep learning object detection models and increase detection speed, while maintaining high detection accuracy. Available model simplification and compression techniques include tensor decomposition [10–13], network pruning [14–19], knowledge distillation [20–23], and neural architecture search (NAS) [24–27]. Tensor decomposition techniques such as low-rank matrix decomposition and tensorized decomposition simplify complex models by reducing a weight matrix or high-dimensional tensor to multiple low-rank matrices or low-dimensional tensors respectively [28]. While tensor decomposition may reduce the memory and computational requirement, it does not increase the detection speed. Moreover, the introduction of many layers as a result of the multiplied low-dimensional tensors makes parallel processing difficult [29].

Network pruning can be divided into unstructured and structured pruning. Unstructured pruning removes filter weights individually, thus requiring a separate software library or special hardware so as not to carry out operations on the weights that have been previously removed [30]. Since this approach does not change the feature map pre- and post-pruning, little model compression is achieved. Structured pruning on the other hand prunes at the layer or channel level, and requires no special software or hardware. Since this approach prunes all the weights in the layer or channel, it often leads to performance reduction [29].

Knowledge distillation trains a simplified model (student network) to achieve an output similar to that of a more complex model (teacher network). This similar output is achieved by means of different loss functions including the conventional knowledge distillation loss [29, 31], Kulback-Leibler divergence[32], and angular distillation loss [31]. Whereas knowledge distillation realizes a lightweight model, the generation of an effective student model which is compatible with the teacher model is still a challenge. Moreover, accounting for the effect of different layers of the teacher on the student model is difficult [30].

Neural architecture search generates a simplified model by repeatedly searching the predefined space and evaluating the architecture until an optimal model is realized [30]. The large searching space makes this iterative procedure time-consuming [33]. NAS is currently faced with the three-pronged challenge of proper search space definition, quickly finding the optimal architecture, and the best way to evaluate the candidate network [29].

The above-mentioned model simplification methods are generic approaches as opposed to model-specific. High-performing state-of-the-art detection models owe their improved performance to the introduction of new techniques during model development. For instance, the introduction of different detection scales in YOLO deep learning detection models, improved the ability of the models to better detect objects of all sizes [34]. As proposed in this work, such model-specific peculiarities can be explored to realize simplified models.

1.3 Related work

Deep learning-based object detection models are broadly classed into two-stage and one-stage detection models. Two-stage detection models such as Fast-RCNN [35], Faster-RCNN [36], Mask-RCNN [37], etc., propose regions of interest as a first stage, followed by pooling of features for classification and localization of objects. While two-stage detectors are known for very high accuracy, they are often too slow and unsuitable for real-time applications. In contrast to two-stage detectors, one-stage detectors like the YOLO series [34, 38–40] single-shot detector (SSD) [41], classify and localize images in one single step, utilizing grid boxes and anchors, and without the need for region proposals. One-stage detectors often fall behind two-stage detectors in accuracy, but are much faster and lighter; thus, making them preferable for real-time applications. In particular, the YOLO series has gained popularity for its speed and accuracy. After the first introduction of the first YOLO object detection model by Redmon et al. [42], attempts have been made by researchers to improve the speed, accuracy, and model size with each evolution of the model [34, 38–40]. Many YOLO models have been released with different improvement strategies. The underlying focus for all YOLO models is to optimise the trade-off between accuracy and speed, thus making YOLO models highly suitable for real-time applications. Table 1 below summarizes the major existing YOLO models and their unique advantages.

The simple architecture and efficiency of YOLO models have made them a focus of optimization efforts for deployment in crop monitoring activities. The potential for model optimization using model pruning was demonstrated by Wang and He [52]. Using channel pruning, the authors reduced the size of YOLOv5s model by 9.5% of the original model size. The fine-tuned model, which was developed for apple fruitlet detection, also achieved 9.5% reduction in the number of parameters and 9.8% decrease in latency. The same technique was applied by Wu et al. [53] to reduce the number of parameters of YOLOv4 by 96.74% which yielded 39.47% increase in inference speed for apple flower detection. In the work by Wang et al. [54], another pruning method, layer pruning, was combined with

Table 1 A chronology of YOLO versions showing features/improvements and performance based on Average Precision (AP) and Frames Per Second (FPS)

Model	Year	Features/Improvements	Performance
YOLOv1 [42]	2016	Single stage detection, simple architecture, fast detection	63.4 AP on Paschal VOC2007
YOLOv2 [39]	2017	High-resolution classifier, anchor box-based to prediction, dimension clusters, direct location prediction, finner-grained features, multi-scale training	78.6 AP on Paschal VOC2007
YOLOv3 [34]	2018	class Prediction, new backbone, modified spatial pyramid pooling (SPP), multi-scale predictions,	36.2 AP and 60.6 AP50 of at 20 FPS on MS COCO dataset
YOLOv4 [40]	2020	Enhanced architecture, bag of freebies (BoF) and bag of specials (BoS) integration, self-adversarial training (SAT), hyperparameter optimisation with Genetic Algorithm	43.5 AP and 65.7 AP50 on MS COCO dataset 2017 at more than 50 FPS on NVIDIA V100
YOLOv5 [38]	2020	Modified CSPDarknet53 backbone, autoanchors, developed in Pytorch framework	50.7 AP on MS COCO dataset 2017 at 200 FPS on NVIDIA V100
Scaled-YOLOv4 [43]	2021	Pytorch framework, scaling up and scaling down techniques	Up to 56 AP on MS COCO dataset
YOLOX [44]	2021	Multi-task learning	55.4% AP and 73.3% AP50 on MS COCO at 30 FPS on an NVIDIA V100
YOLOR [45]	2021	Anchor-free architecture, multiple positives, decoupled head, advanced label assignment, strong augmentations	50.1% AP at 68.9 FPS on Tesla V100
PP-YOLOE [46]	2022	Anchor-free, new backbone and neck, Task Alignment Learning (TAL), Efficient Task-aligned Head (ET-head), Varifocal (VFL) and Distribution focal loss (DFL)	51.4% AP at 78.1 FPS on NVIDIA V100
YOLOv6 [47]	2022	A new backbone based on RepVGG, label assignment using the Task alignment learning, new classification and regression losses, self-distillation, quantization scheme	57.2% AP at around 29 FPS on NVIDIA Tesla T4
YOLOv7 [48]	2022	Extended efficient layer aggregation network (E-ELAN), Model scaling for concatenation-based models, Bag of Freebies	55.9% AP and 73.5% AP50 on MS COCO dataset at 50 FPS on NVIDIA V100
DAMO-YOLO [49]	2022	Neural architecture search (NAS), large neck, small head, AlignedOTA label assignment, knowledge distillation	50.0% AP at 233 FPS on NVIDIA V100
YOLOv8 [50]	2023	Multiple vision tasks, anchor-free, decoupled head, modified backbone	53.9% AP at 280 FPS on an NVIDIA A100
YOLO-NAS [51]	2023	Quantization aware modules, automatic architecture design, hybrid quantization, self-distillation	52.2% AP on MS COCO

channel pruning and detection head searching to optimize YOLOv5 for real-time apple stem/calyx recognition. After optimization, the authors reported up to 71% reduction in model weight, and inference speed of about 25 frames per second (fps), at the cost of 1.6% decrease in mean average precision. Yin et al. [55] replaced the Darknet53 backbone in YOLOv3 network with a joint network of deep random kernel convolutional extreme learning machine (DRKCELM) and a double hidden layer extreme learning machine auto-encoder (DLELM-AE) to realize a simplified feature extraction backbone. The approach resulted in reduced training time, improved detection speed over the YOLOv3 model, with a slight reduction in mean average precision. YOLO models are deep learning networks generally consisting of a backbone for feature extraction, a neck for aggregating and fusing features and a "head" with three scales of detection for image classification and bounding box predictions. While the referenced works on YOLO models optimization for crop monitoring achieved acceptable improvements in speed and model weight reduction, we hypothesise that dropping one or more detection scales of the network can lead to reduction in computational cost and model weight as well as to acceleration of inference. Often, for many crop detection applications, the objects of interest are of similar size, as the crops would normally be at the same growth stage. Thus, all three detection scales would not be in principle necessary for crop detection—only one or two detection scales would be sufficient to detect the crops in the images. Performance evaluation of the optimised models on the type of embedded devices similar to those deployed in the field is also lacking.

At the time of first writing, YOLOv5 was the current state-of-the-art detection model, which informed the choice of YOLOv5 for optimization. At the time of manuscript revision, the authors compared YOLOv5's performance on the custom dataset using with the latest YOLO models, YOLOv6, YOLOv8, and YOLO-NAS. Interestingly, the comparison showed (see Sections 3.1 and 4.1) similar accuracies for YOLOv5, YOLOv8, and YOLO-NAS, but a much lower accuracy for YOLOv6. In addition, YOLOv5 showed faster detection speed than YOLOv8 and YOLO-NAS. Thereafter, YOLOv5 was retained as the candidate model for optimisation based on the obtained results.

1.4 Contributions and paper structure

This work is driven by the pressing need for accurate yet lightweight models deployable in real-time object detection scenarios, particularly in fields like crop monitoring employing unmanned aerial vehicles (UAVs). While model development has progressed over time, resulting in increasingly lighter and more precise models, the incorporation of algorithm-specific acceleration methods has demonstrated remarkable advancements. In this paper, we introduce an acceleration technique that enables the creation of swift and lightweight standalone models, adaptable to various monitoring scenarios. The main contributions of this work are summarised as follows:

- Design of an acceleration methodology for the well-established YOLOv5 object detector, centered around a systematic approach involving the selective exclusion of various detection scales. This methodology balances performance and efficiency by not only enhancing inference speed but also ensuring that detection accuracy remains consistently high.
- Implementation and testing of the acceleration technique at algorithm level on different machine learning platforms.

- Deployment of the derived models on close-to-device hardware such as NVIDIA Jetson nano and NVIDIA Jetson Orin and evaluation of their performance for real crop monitoring settings using unmanned aerial vehicles UAV.

A comparison with pruned versions of YOLOv5 is provided to show how our model-specific strategy compares with currently existing high-performing generic simplification approaches. We also show that YOLOv5, which is a purely detector procedure, compares favourably with later Yolo versions in terms of accuracy.

The paper is structured as follows: in Section 2 we present the methodology of the research, which includes a detailed overview of the YOLOv5 object detection model and its comparison with other recent YOLO models; here we also present our network simplification approach, our custom dataset, model deployment platforms, and comparison of our approach with network pruning. We present the research results in Section 3 and detailed discussion of the results in Section 4. Section 5 concludes the paper and provides our future plans for the research.

2 Methodology

2.1 YOLOv5

YOLOv5 is an improvement over the YOLOv3 network having better accuracy, speed, and size [38, 56]. As shown in Fig. 1, the backbone of YOLOv5 consists of a focus module, CBS (Convolution, Batch Normalization, SiLU activation) modules, C3_*n* (CBS, BottleneckCSP1, Concatenation) modules (*n* is the residual units), and a Spatial Pyramid Pooling (SPP) module.

The focus module preserves all the input image information for better feature extraction. Figure 2 shows the layers that make up the focus module, including the sizes of the kernel (k), strides (s), padding (p), and channels (c).

The CBS, is a basic block merely consisting of a convolutional layer that uses batch normalisation and SiLU activation to extract image features.

The C3 module is designed to enhance the learning abilities of the network. YOLOv5 is largely composed of repeated stacking of C3 blocks. Figure 3 shows the configuration of the C3 module.

YOLOv5 has one type of Cross-stage Partial Network configuration in the backbone referred to as BottleneckCSP1, which improves the learning ability of the CNN. The CSP at the neck, BottleneckCSP2, uses a skip connection to better integrate features. Figure 4 shows the configurations of BottleneckCSP1 and BottleneckCSP2.

The SPP module, which consists of maximum pooling, CBS, and concatenation layers, helps to fuse multiscale features. Figure 5 highlights more details of the SPP configuration.

The neck of YOLOv5 utilizes the Path Aggregation Network (PANet) to fuse extracted features. The PANet is a configuration of CBS, upsampling, concatenation, and C3 layers. The neck of YOLOv5 outputs three feature scales which are used for prediction at the head. Finally, the Non-maximum suppression (NMS) is used to select the best bounding box predictions in the event that multiple predictions exist for one target. The YOLOv5 currently has five versions, the extra-large (YOLOv5x), large (YOLOv5l), medium (YOLOv5m), small (YOLOv5s), and nano (YOLOv5n). Their differences lie in the depth and width of the networks, with decreasing order of depth and width from YOLOv5x to YOLOv5n. The accuracy, complexity, and size of the models increase with increase in the width and depth, whereas the speeds of the different versions decrease as the depth and width increase. The YOLOv5n was selected for this work because the detection speed is suitable for real-time applications and the accuracy is acceptable. Moreover, the same techniques developed in this work are applicable to the other YOLOv5 versions [57].

2.2 YOLOv5 vs YOLOv6 vs YOLOv8 vs YOLO-NAS

The latest models of the YOLO series, YOLOv6, YOLOv8, and YOLO-NAS, were compared with YOLOv5 on the custom dataset presented in this work. The smallest versions of the three models which are YOLOv5n, YOLOv6n, YOLOv8n, and YOLO-NASs were chosen as they are most suited for real-time detection. The compared metrics included mAP, detection speed, size of the weights file, and training. Results are presented in Section 3.

2.3 Model design

Prediction in YOLOv5 takes place at the YOLOv5 head. Here, features from the YOLOv5 neck are used to make anchor-based predictions at three granularity levels. Thus, the YOLOv5 head consists of three output layers. The first granularity layer predicts small objects using grid size of 80×80. The second granularity layer predicts medium-sized objects with 40×40 grid size. The third layer predicts large objects using 20×20 grids. Basically, the head for YOLOv3, YOLOv4, and YOLOv5 follow a very similar structure; details of this structure are explained by Martinez-Alpiste et al. [58].

Figure 6 shows how the YOLOv5 architecture was modified to output only the detection scale for the small objects. The details of the backbone are omitted for conciseness.

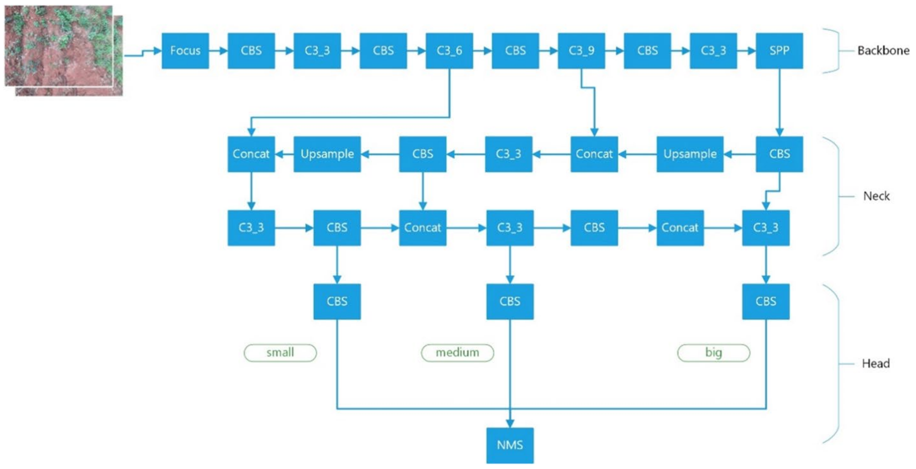

Fig. 1 Overview of YOLOv5 architecture

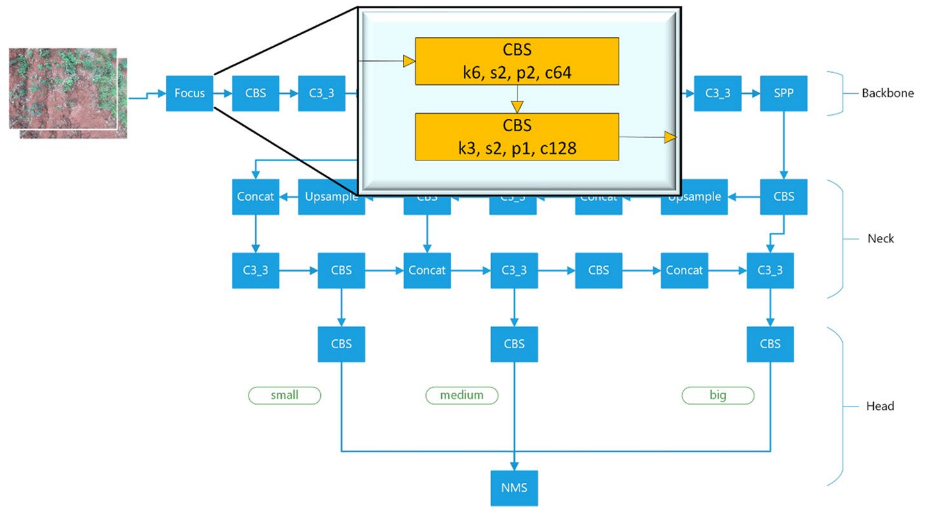

Fig. 2 Composition of the Focus module of YOLOv5

The faded layers indicate the layers that were discarded from the YOLOv5 head. Here, the medium and big scales were dropped.

Similarly, Fig. 7 describes how the YOLOv5 architecture was modified to drop the small and big scales and output only the medium detection scale.

In Fig. 8, we see how only the detection scale for big objects was outputted while discarding the small and medium detection scales.

To output more than one detection scale, but not all three scales, the appropriate scale was dropped. For instance, to output the small and medium scales, the detection scale for big objects was dropped.

Table 2 summarizes the resulting models after scales were systematically dropped.

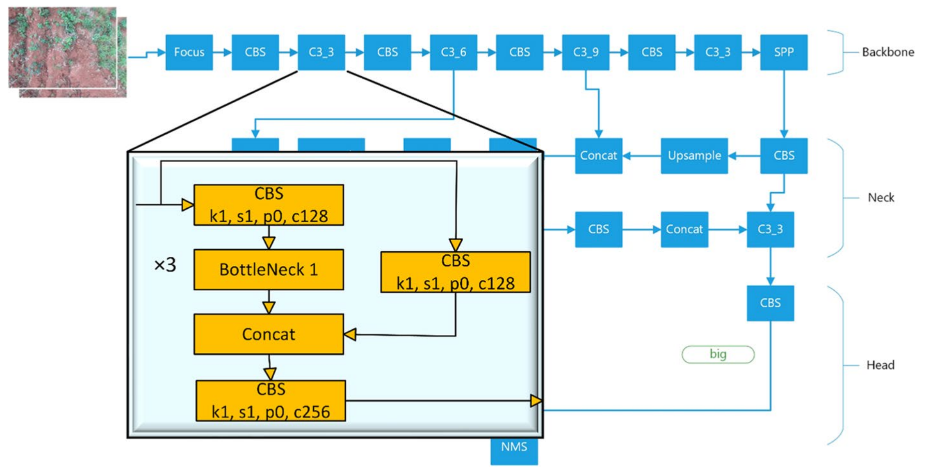

Fig. 3 Configuration of the YOLOv5 C3 block

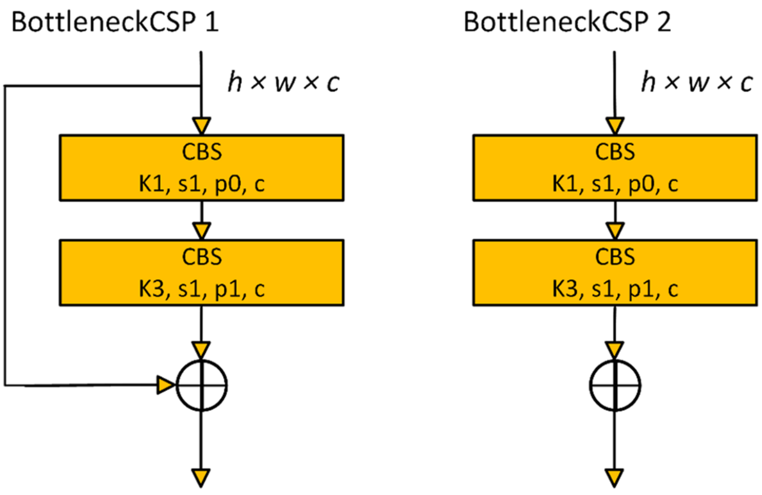

Fig. 4 Configuration of YOLOv5 BottleneckCSP 1 and BottleneckCSP 2

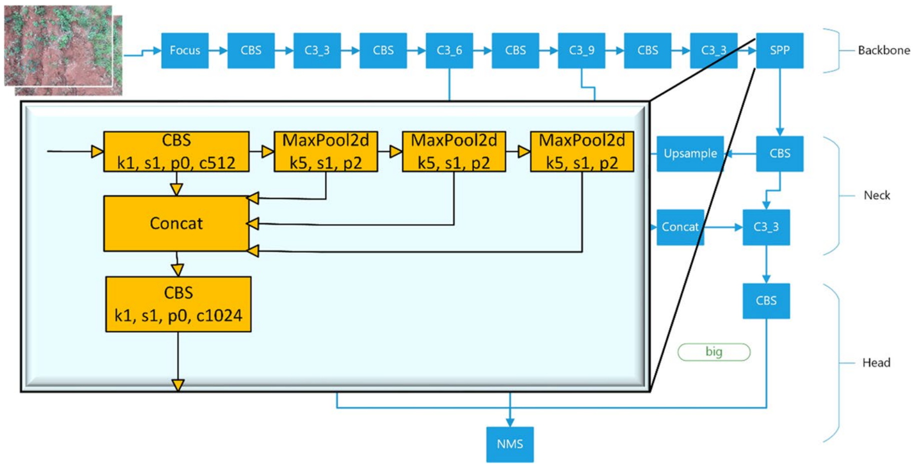

Fig. 5 Configuration of the YOLOv5 SPP module

We would henceforth refer to the modified models by the outputted scale. Thus, for the model where all but the small scale is dropped, we will call it the "s" model; and the same goes for the other models.

2.4 Dataset

The object detection task adopted for this work was cassava crop detection under real-life field conditions. The training, validation, and inference dataset consisted of RGB images captured from an experimental farm at Nsukka, Nigeria. The images were captured using a custom-made UAV and a GoPro Hero 7 camera under variations in field conditions like lighting, weed density, crop growth stages, etc. Figure 9 contains sample images from the dataset showing various field conditions including illumination, shadows, weed density, leaf occlusions, and crop growth stages. The captured images had an original resolution of 4000 × 3000 pixels. However, we resized the images to 960 × 720 pixels while maintaining the original aspect ratio and ensuring the new image height and width were multiples of 32 as recommended for YOLO models. The images were annotated using the python-based LabelImg [59] annotation tool.

The training and validation images containing 1788 and 475 cassava objects respectively, were used for training and validation respectively. As for the latter, we measured mean average precision (mAP) for the customary value of intersection over union equal to 0.5 [60]. This will jointly be denoted as mAP@0.5.

To measure the detection speed of the models for real-time scenarios, we created an inference set. Inferences were run on the inference set to simulate real-time field deployment and monitor the detection speed of the models. The inference set contained up to 246 cassava objects.

2.5 Model training, accuracy assessment and inference

The six derived models and the base model were each trained on a workstation powered by NVIDIA GeForce RTX 3060, 12 GB RAM GPU. The deep learning framework used was Pytorch. A uniform epoch size and batch size of 400 and 8 were respectively used. The learning rate was set to 0.01 and momentum was 0.937. The Generalised Intersection over Union (GIoU) loss function was used for training the models, and the optimiser used was Stochastic Gradient Descent (SGD) [61]. To improve the model accuracy given the relatively small size of our custom dataset, transfer learning was employed using pretrained weights of the YOLOv5n model trained on the Microsoft COCO dataset.

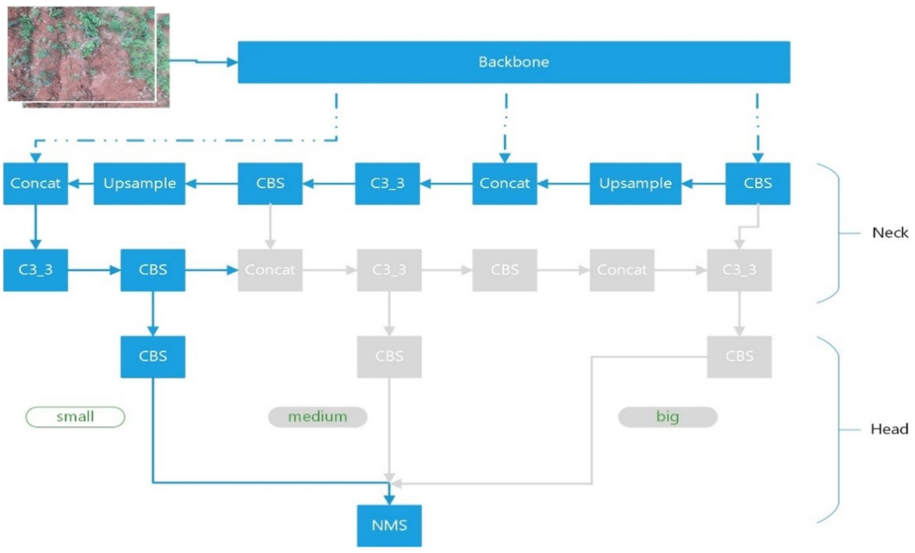

Fig. 6 Modified YOLOv5 architecture for small scale detection. Faded blocks means discarded blocks

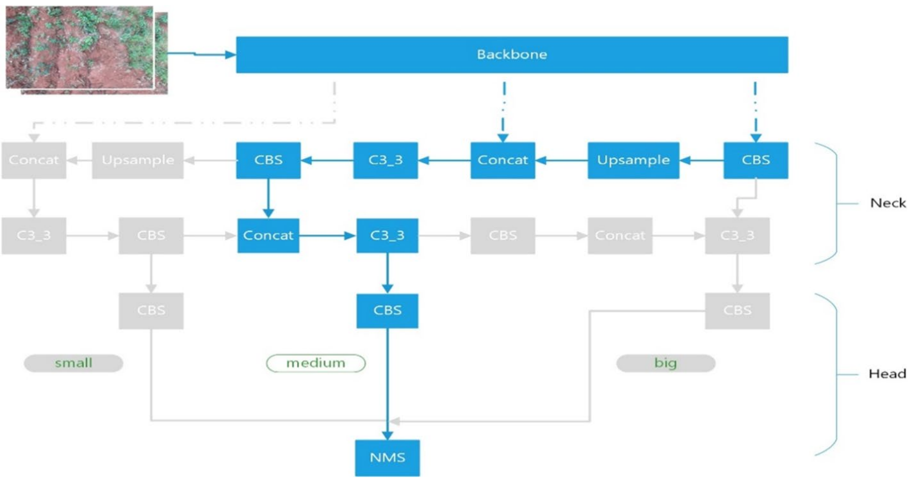

Fig. 7 Modified YOLOv5 architecture for medium scale detection

For performance comparison and analysis, we calculated training time on the training set and mAP@0.5 on the validation set. To understand the complexity reduction in the models, the number of layers, number of parameters, and GFLOPS were documented. The sizes of the weight files were also noted to highlight the improvements in the memory footprints of the models.

The latency (the inverse of which is frames per second, FPS) across all the models was compared, using the inference set.

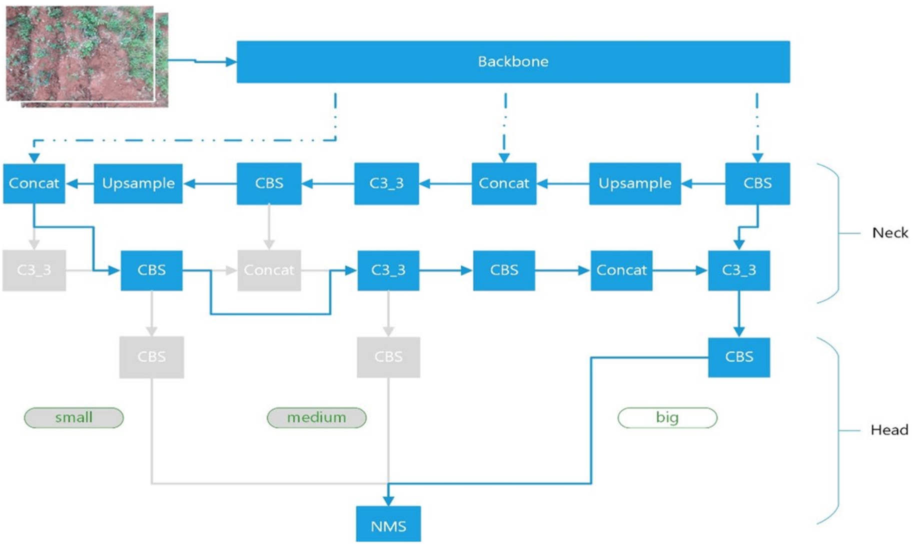

Fig. 8 Modified YOLOv5 architecture for big scale detection

Table 2 Summary of the models showing which scales were dropped and which were retained, where "s", "m", and "b" represent the small, medium, and big scales respectively

Model alias	Dropped scale(s)	Retained scale(s)	Comment
YOLOv5n_s	m, b	s	
YOLOv5n_m	s, b	m	
YOLOv5n_b	s, m	b	
YOLOv5n_s + m	b	s, m	
YOLOv5n_s + b	m	s, b	
YOLOv5n_m + b	s	m, b	
YOLOv5n_all	None	s, m, b	Base model

2.6 Model deployment on Jetson devices

Many real-life applications of object detection are resource-constrained. This is especially true for drone applications such as real-time plant detections using UAVs, where the payload capacity may not accommodate very high-computing devices like the GPUs installed on workstations. To evaluate the performance of the simplified models for such applications, the models were deployed on the NVIDIA Jetson nano 4 Gb RAM running Ubuntu 18.04 LTS. The Jetson nano is a small GPU-powered computer manufactured by NVIDIA and suitable for running deep learning applications for tasks such as image classification, image segmentation, object detection, etc. Neural network models can be deployed to Jetson nano, which in turn can be mounted on mobile platforms like UAVs and self-driving vehicles for different computer vision tasks.

The models were also deployed on NVIDIA Jetson Orin 32 GB RAM. The Jetson Orin packs more computing power and provides an option for realizing detection speeds over 6 times faster than the Jetson nano. As a result, it is much heavier than the Jetson nano. Jetson nano runs on Ubuntu 20.04.6 LTS.

2.6.1 Technical specifications of the NVIDIA Jetson nano and Jetson AGX Orin

Jetson embedded devices such as the Jetson Orin and Jetson Nano are modular devices developed by NVIDIA for mobile applications. Equipped with CUDA cores, the Jetson is able to run deep learning models seamlessly to allow real-time computer vision tasks such as object detection and image segmentation on mobile platforms.

The technical specifications of the Jetson nano and Orin are given in Table 3.

2.6.2 Jetson setup and package installation

The NVIDIA® Jetson Nano™ Developer Kit consisted of the Jetson nano module and the reference carrier board with required accessories. The YOLOv5 package and the attendant requirements were installed on the Jetson nano. To get the best model inference speed and size on the Jetson nano, three target formats for model deployment suitable for embedded systems were initially considered, which included the native Pytorch format of the YOLOv5, the TensorRT developed by NVIDIA®, and the ONNX developed for mobile applications. Preliminary results showed that the performance of the ONNX format of the

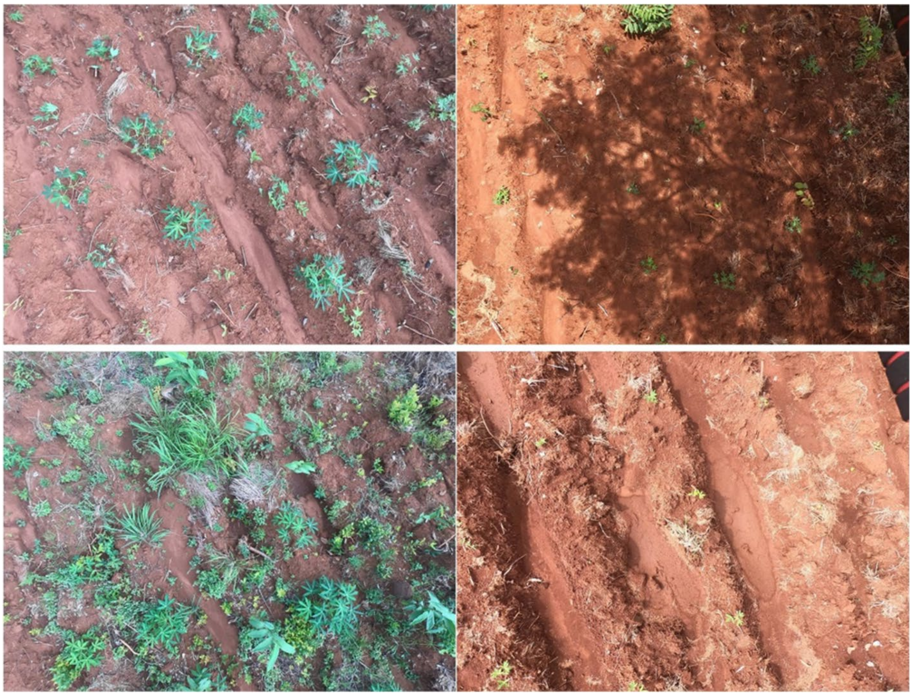

Fig. 9 Sample images from the dataset showing various field conditions including illumination, shadows, weed density, leaf occlusion, and crop growth stages

models on the Jetson was not acceptable and the format was consequently dropped. Thus, inferences on the Jetson were only conducted and compared for the Pytorch and TensorRT formats of the models. The Pytorch framework, TensorRT, and all dependent packages were installed on the Jetson.

For the Pytorch inferences on the Jetson, it was not required to convert the models trained on the workstation since they were already in the native Pytorch format. But for the TensorRT format, it was necessary to convert the models to TensorRT engine for inference on the Jetson nano. The conversion of the detection models to TensorRT was done using the code provided by the authors of the YOLOv5 model.

The setup and installation procedures for the NVIDIA Jetson Orin were similar to those of the Jetson Nano. However, for the Jetson Orin, the models were only deployed in the Pytorch format.

2.6.3 Inference on Jetson devices

For each model and for both the Pytorch and TensorRT formats, 20 inference runs were carried out. The speed improvements of the derived models over the base model were highlighted. The results of the two deployment formats—Pytorch and TensorRT—were also compared. Similarly, 20 inference runs were performed on the Jetson Orin for each model. The results are presented in Section 3.

2.7 Our approach vs network pruning

The YOLOv5 base model was pruned to a sparsity of 0.2 and 0.3 and compared with our optimised models. The pruned models were tested on Google Colab, running on a Tesla T4 GPU with 16 GB RAM. Consequently, additional inferences were conducted for all optimised models on Google Colab for fair comparison. The results are presented and compared in Section 3.

3 Results

3.1 YOLOv5 vs YOLOv6 vs YOLOv8 vs YOLO-NAS

To ensure a fair comparison of YOLOv5, YOLOv6, YOLOv8, and YOLO-NAS on our dataset, the same platform for training and inference was necessary. The chosen platform was Google Colab, using Tesla T4 GPUs with 16 GB RAM. Figure 10 shows how the models perform in terms of accuracy and inference speed. Training time and memory footprint of the models are presented in Table 4.

3.2 Model properties

In Table 5, the attributes of the new and base models including the GFLOPS, number of parameters, model size, and number of layers are presented. We note that for the 'b' model, the training was stopped at 102 epochs after seeing no improvements in the last 100 epochs. Therefore, there are no results for the 'b' model. The possible reasons for this outcome are discussed in Section 4.

The percentage improvements in the attributes of the derived models are presented in Table 6.

Figure 11 shows how reduction in the number of model parameters affects the model size.

The weight sizes of all six models are given in megabytes (mb).

3.3 Performance comparison and analysis

The results documented during training and validation included training time, performance metrics –mean average precision, precision, recall, – and GFLOPS. Given that the mean average precision (mAP) has become the preferred performance metric in object detection research, this paper will present and discuss the mAP results from the model training. The training results are presented in Table 7.

Figure 12 highlights the models' accuracy through the training epochs. The regions at which the models achieve maximal performance can also be observed.

Figure 13 shows a plot of Training time vs GFLOPs to highlight how the improvements in GFLOPS affects training time of the models.

Table 3 Technical specifications of NVIDIA Jetson nano and Jetson Orin

Specification	Jetson nano	Jetson AGX Orin
GPU	128-core Maxwell	1792-core NVIDIA Ampere architecture GPU with 56 Tensor Cores
CPU	Quad-core ARM A57 @ 1.43 GHz	8-core Arm® Cortex®-A78AE v8.2 64-bit CPU 2 MB L2 + 4 MB L3
Memory	4 GB 64-bit LPDDR4 25.6 GB/s	32 GB 256-bit LPDDR5, 204.8 GB/s
Storage	microSD (not included)	64 GB eMMC 5.1
Camera	2 × MIPI CSI-2 DPHY lanes	Up to 6 cameras (16 via virtual channels) 16 lanes MIPI CSI-2 D-PHY 2.1 (up to 40Gbps) | C-PHY 2.0 (up to 164Gbps)
Network	Gigabit Ethernet (GbE), M.2 Key E	1 × GbE 1 × 10GbE
Mechanical	69 mm × 45 mm, 260-pin edge connector	100 mm x 87 mm, 699-pin Molex Mirror Mezz Connector, Integrated Thermal Transfer Plate
Other	GPIO, I2C, I2S, SPI, UART	4 × UART, 3 × SPI, 4 × I2S, 8 × I2C, 2 × CAN, PWM, DMIC & DSPK, GPIOs

Table 4 Memory footprint and training time of YOLOv5, YOLOv6, YOLOv8, and YOLO-NAS

Model	Weight size (MB)	Training time (hrs)
YOLOv5	4.0	1.207
YOLOv6	10.0	1.897
YOLOv8	6.3	0.651
YOLO-NAS	244.5	3.100

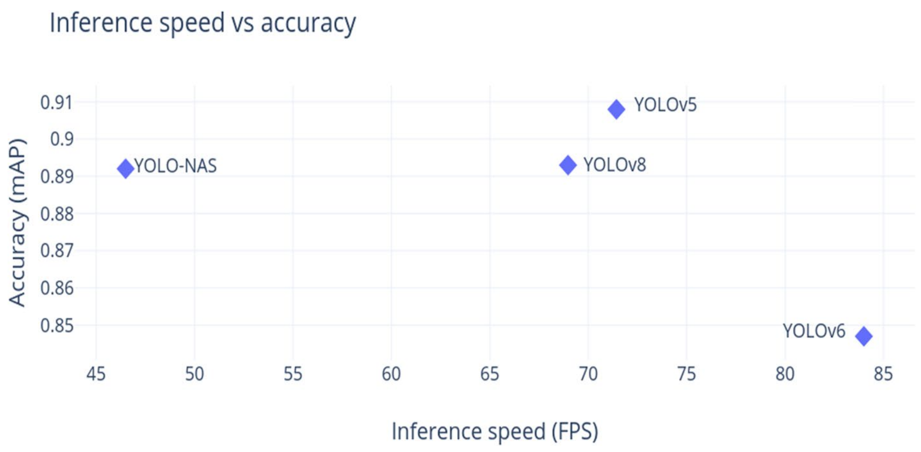

Fig. 10 Inference speed vs Accuracy of YOLOv5, YOLOv6, YOLOv8, and YOLO-NAS

3.4 Inference

The inference results are in two parts – the inference results on the workstation GPU, and the inference results on the Jetson devices. The inference on the Jetson is further divided into inference runs for the Pytorch models and for the TensorRT models. The inference results are presented in Table 8. The inference latency (in milliseconds) and its inverse, the inference speed (in frames per second, fps), are given for all inferences.

In Table 9, we present the percentage improvements in the inference latency and speed of the models. Only the latency improvements on the workstation GPU are shown, since we can make our analysis with the speed improvements alone.

In Table 10, we present, side by side, the inference results of the Pytorch and TensorRT implementation on the Jetson nano for the purpose of highlighting the advantages of the versions over each other.

3.5 Our approach vs pruning

Table 11 shows how our optimisation approach compares with network pruning.

Table 5 A summary of the attributes of the modified and base models

Model	All	s	m	s+m	s+b	m+b
GFLOPS	4.2	3.4	3.3	3.8	3.8	3.8
No of parameters	1,760,518	1,199,266	1,243,618	1,312,628	1,450,420	1,736,628
Size (MB)	3.9	2.8	2.7	3.1	3.7	3.7
No of layers	213	167	165	190	193	193

Table 6 Percentage improvements in the attributes of the modified models over the base model

Model	All	s	m	s+m	s+b	m+b
GFLOPS reduction (%)	base	19.0	21.4	9.5	9.5	9.5
No of parameters reduction (%)	base	31.9	29.4	25.4	17.6	1.4
Size reduction (%)	base	28.2	30.8	20.5	5.1	5.1
No of layers reduction (%)	base	21.6	22.5	10.8	9.4	9.4

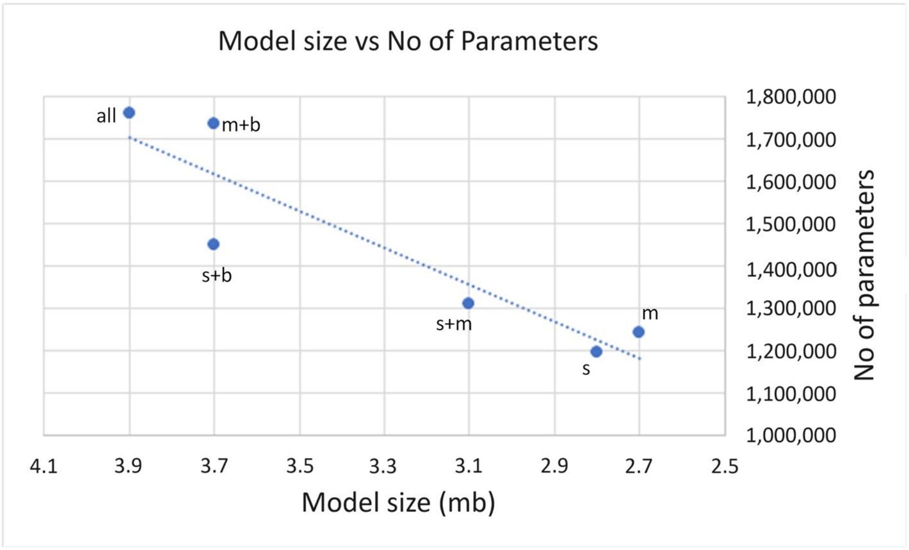

Fig. 11 A plot of the model size vs the number of parameters showing that reduction in number of parameters yields reduction in model size

4 Discussion

In this section the research results are analysed and discussed in the light of the degree to which the research objectives were achieved.

Table 7 Summary of the training results showing the training time and mean average precision of all the modified and base models and the percentage drop in mAP

Model	All	s	m	s+m	s+b	m+b
Training time (hrs)	0.433	0.323	0.319	0.377	0.363	0.367
mAP@0.5	0.951	0.940	0.821	0.947	0.948	0.926
mAP@0.5 drop (%)	base	1.2	13.7	0.4	0.3	2.6

4.1 YOLOv5 as candidate model for optimization

From Fig. 10, despite the improvements in the YOLOv6, YOLOv8, and YOLO-NAS architectures, the accuracy of the four models were similar, with YOLOv5 showing only a slightly higher accuracy of 1.68% over YOLOv8 and 1.79% over YOLO-NAS, and up to 7.2% over YOLOv6. The lack of significant accuracy improvements obtained fromY-OLOv8 and YOLO-NAS models compared to YOLOv5 could be due to the fact that the developers focused on optimising the models for multitask (classification, detection, and segmentation) learning. On the other hand, YOLOv5 is purely a detection model. Just like the accuracy metric, the detection speed followed a similar trend, with YOLOv5 showing a slightly faster detection than YOLOv8. YOLOv5 has the lowest weight size at 4.0 MB, whereas YOLOv6 and YOLOv8 has 10.0 MB and 6.3 MB respectively, while YOLO-NAS' size is 60 times that of YOLOv5 (see Table 4). YOLOv8 has the lowest training time. While training time might be a factor to be considered in deploying models for real-time detection, it is not so important as accuracy and detection speed. YOLOv6 recorded the fastest inference time, but with the lowest accuracy of the four models. Thus, YOLOv5 was retained as the candidate model for optimization in this work.

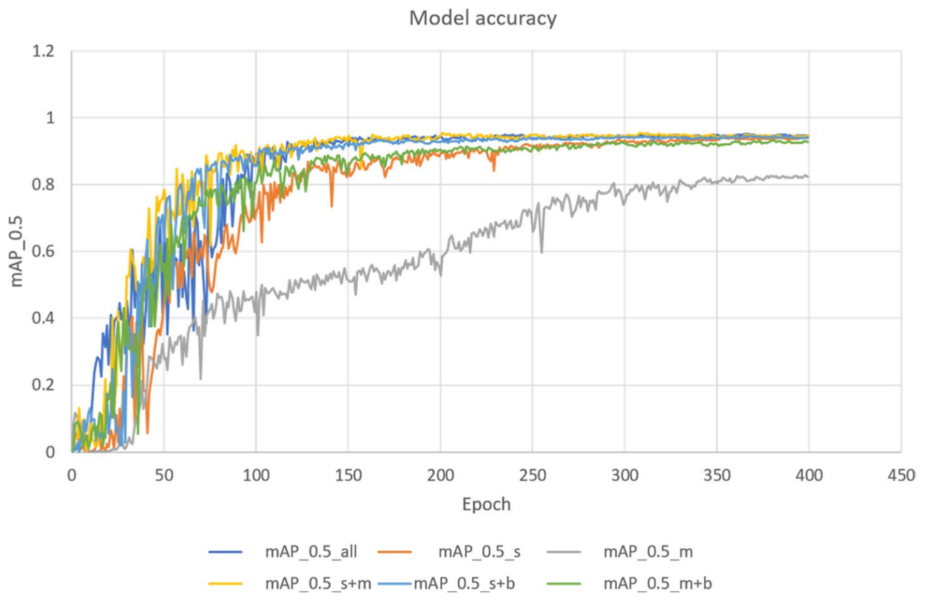

Fig. 12 A graph of the mAP of all models

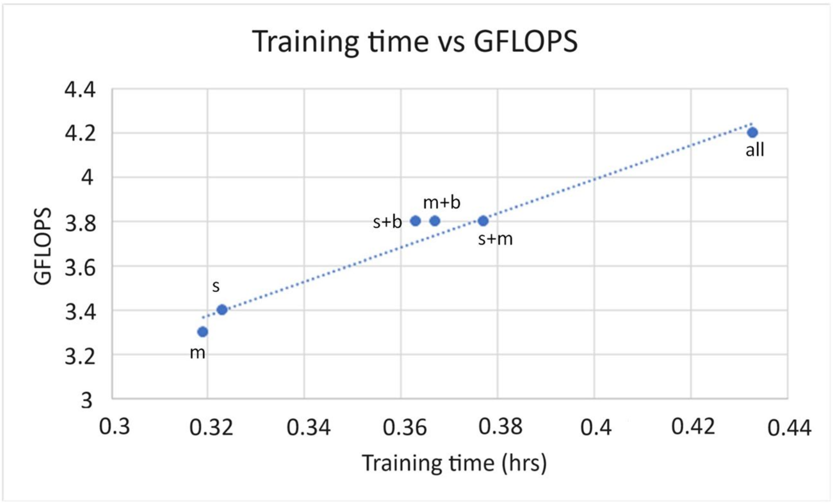

Fig. 13 A plot of Training time vs GFLOPS showing how reduction in GFLOPS corresponds to decrease in training time of the models

Table 8 A summary of the inference results on the workstation GPU and Jetson nano showing the inference latency and speed

Platform	Model	All	s	m	s + m	s + b	m + b
Workstation GPU	Inference latency (ms)	32.7	25.6	25.5	27.0	28.7	27.2
	Inference speed (fps)	30.6	39.0	39.2	37.0	34.8	36.8
Jetson nano Pytorch	Inference latency (ms)	178.7	158.8	142.5	161.5	158.7	155.0
	Inference speed (fps)	5.6	6.4	7.0	6.2	6.3	6.5
Jetson nano TensorRT	Inference latency (ms)	123.9	108.1	96.2	117.1	110.9	118.3
	Inference speed (fps)	8.1	9.3	10.5	8.7	9.1	8.5
Jetson Orin	Inference latency (ms)	24.56	21.41	22.19	23.91	24.30	22.78
	Inference speed (fps)	40.84	47.14	46.19	42.09	41.48	44.47

4.2 Resources demand reduction

The resource-demand of a detection model is an important consideration when evaluating models for the purpose of deployment for real-world applications. These resources include the computational cost and memory footprints of the models. A popular measure of the computational cost of neural networks is the FLOPS. FLOPS gives the number of floating-point operations for one forward pass. Reduction in FLOPS implies less computations are required for a given model. Table 6 shows the improvements in GFLOPS of the derived models over the base model. The modified models show a minimum GFLOPS reduction of 9.5%, while the models where more than one scale was discarded – s, m—boast of a reduction of about one-fifth of the GFLOPS of the base model. This implies that with the new models we cut down the computational cost for detecting objects by up to one-fifth. The benefit of these improvements is reflected in the training time for the models. In Fig. 13,

Table 9 Percentage reduction in latency of the new models over the base models

Platform	Model	All	s	m	s + m	s + b	m + b
Workstation GPU	Inference latency reduction (%)	base	21.7	22.0	17.4	12.2	16.8
	Inference speed increase (%)	base	27.8	28.1	20.9	13.7	20.3
Jetson nano Pytorch	Inference speed increase (%)	base	13.2	25.5	10.7	12.5	15.2
Jetson nano TensorRT	Inference speed Increase (%)	base	14.5	29.0	6.8	11.7	4.9
Jetson Orin	Inference speed increase (%)	base	15.4	13.1	3.1	1.6	8.9

Table 10 Comparison of the speed and memory footprints of Pytorch and TensorRT implementations of the models

Model on Jetson nano	All	s	m	s + m	s + b	m + b
Pytorch inference speed (fps)	5.61	6.35	7.04	6.21	6.31	6.46
TensorRT inference speed (fps)	8.11	9.29	10.46	8.66	9.06	8.51
Pytorch model size (MB)	3.9	2.8	2.7	3.1	3.7	3.7
TensorRT model size (MB)	14.1	7.8	10.2	12.1	11.4	11.0

Table 11 Comparison of the performance of the optimised model with the performance of network pruning technique

Model	Accuracy (mAP)	Inference speed (fps)
YOLOv5_all (base)	0.908	59.50
Prune 0.3	0.824	76.92
Prune 0.2	0.885	86.20
s	0.881	101.01
m	0.776	111.11
s + m	0.895	62.89
s + b	0.899	63.29
m + b	0.563	72.99

a plot of training time vs GFLOPS indicates that reduction in GFLOPS corresponds to a reduction in training time required for the models. This becomes useful in applications where minimal training time is desired.

The reduction in computational cost is also reflected in the reduction of number of parameters the model must learn. The approach of discarding unwanted detection scales cuts the number of learnable parameters in the derived models by as much as 31%, as shown in Table 6.

Reduction in the number of parameters can result in a more compressed model. This is important in applications such as computer vision tasks with embedded systems on mobile platforms like UAVs where the memory footprint of the model is critical. Figure 11 shows a plot of the model size vs the number of parameters. Less number of parameters correspond to a more compressed network. Furthermore, as indicated in Table 6 our simplified models show up to 30% model compression. Thus, we achieve models that are significantly smaller than the base model.

Table 6 also shows that the derived models have less number of layers compared to the base model. We note that number of layers is not generally a dependable measure of the computational cost, especially when comparing different network architectures. However, given that our modified models were derived by discarding scales and layers from one base model, it is logical to conclude that a reduction in number of layers involves a reduction in the number of computations required in the new models when compared to the base model.

4.3 Model accuracy

Model accuracy often comes at the expense of computational complexity and speed. Therefore, it is desired that improvements in speed and computational cost should come at a minimal drop in accuracy. Table 7 shows the percentage drop in accuracy of the models. The accuracy metric discussed in this paper is the mAP@0.5. All but one of the new models impressively show less than 3% drop in accuracy compared with the base model. The least accuracy drop is recorded by the models that retained the small detection scale – ‘s’, ‘s + m’, and ‘s + b’, whereas the models where the small scale was discarded had the most drop in accuracy. This suggests that the dataset contains a high proportion of small objects, and thus presents the models with very sufficient small objects to learn. This could also explain why we had no results for the ‘b’ model as there was likely insufficient number of big objects to learn, hence the lack of improvement in the model precision. Therefore, upgrading the dataset to have sufficient proportion of all object sizes could improve the accuracy of all the models. Nonetheless, we conclude that the reported mAP is good enough for many applications including our case study. We note that, as observed from Fig. 12, the accuracy of the models stopped increasing significantly after 350 epochs. Thus, the models could take a shorter time to train and still achieve good accuracy.

4.4 Speed improvements

Next, we examine the speed improvements of the new models. The inference latency indicates the amount of time taken for one detection starting from the pre-processing of the image to the actual inference and then the non-maximum suppression. Parameter fps shows how many image frames can be inferenced on in one second.

4.4.1 Inference on workstation GPU

From Table 9, we see that the technique of discarding detection scales according to need leads in significant reduction in latency. A minimum of 12.2% latency reduction is recorded, and as much as 22% latency reduction is achieved, which corresponds to 13.7% and 28.1% speed improvements respectively. It is particularly observed that the models with only one detection scale – ‘s’ and ‘m’ – recorded the best latency and speed improvements. This is explained by the fact that in these models, more scales were dropped, leading to more reduction in computational and memory requirements, and thus making them faster. This observation supports the approach of discarding detection scales to increase model speed – the more scales are dropped, the more the potential for a faster network.

4.4.2 Inference on the Jetson nano

Table 9 also shows the speed improvements of the new models over the base model for the Pytorch implementation. Similar improvements in speed are recorded with up to 25.5% speed gain over the base model. The speed gains of the TensorRT implementation of the models are also given in Table 9. Speed gains of up to 29% are observed.

The speed of the models can be increased by dropping a few more layers from the neck of the network. However, this comes at the expense of accuracy. For applications where accuracy is not critical and speed is desired, specific convolutional layers may be dropped from the "neck" of any of the models to reduce the computational requirements, thereby increasing detection speed. To highlight this point, we dropped an extra upsampling layer, a CBS, and a C3 block from the neck of the "m + b" model. This resulted in 7.5% increase in speed for the TensorRT implementation, which is greater than the 4.9% earlier recorded for the "m + b" model. However, discarding the extra layers resulted in 38.4% drop in mAP.

4.4.3 Inference on Jetson Orin

As expected, the speed improvements of the optimised models over the base model are similar to the results from the Jetson nano. However, detection speeds on the Orin are much higher than those on Jetson nano. The higher speed of the Orin is due to its higher computing capacity, but at the cost of a higher energy requirement. Thus, the Jetson Orin constitutes a preferable option for applications that can accommodate increased power requirements.

4.4.4 Suitability of our models for real-time use

Having established the gains of our models over the base model, even on the Jetson nano, we will go on to examine the suitability of our models for real-time applications. We already have a use-case of cassava detection from UAV images. We take a UAV altitude of 2.5 m, which is the average height at which the images in the dataset were captured; and which is a typical altitude for real-time crop detection and control actions such as selective pesticide or herbicide spraying. This altitude corresponds to a ground sampling distance (GSD) of 5.14 m by 3.86 m based on the properties of the GoPro Hero 7 camera which was used for image capturing in our case. For herbicide or pesticide spraying using drones, the typical speed is a slow speed of about 5 m/s. Note that the GoPro camera captures images at 60 frames per second, thus, the UAV needs not stop at each point of image capture. If, during flight, the UAV must capture images such that there are no overlaps in the images given the above GSD, then the models must process the captured images and output the detection results at a minimum frames per second given by Eq. 1:

$$Detection\ speed = {}^{5ms^{-1}}\!\big/_{3.86m} = 1.3fps \tag{1}$$

Given the minimum speed derived above, we see that even the slowest of our models would comfortably run inferences on the captured image while leaving sufficient time for the system to carry out other processes such as triggering a sprayer system. The advantage of our faster models in this scenario is that our models provide a wider margin for increasing the speed of the UAV for faster completion of flight missions. This wider speed margin also implies that our models can better accommodate the processing time of other activities in the flight mission that are dependent on the detection results such as variable control of the herbicide sprayers.

4.4.5 Pytorch vs TensorRT implementation

Here we highlight the advantages of the two Jetson nano implementations of our models over each other. Table 10 shows the comparison of the speed and memory footprints of the models. Note that the accuracy of the models is preserved during conversion from Pytorch to TensorRT. Clearly, the TensorRT versions show a better performance in terms of inference speed. However, the storage requirement for the Pytorch versions is less. The preferred implementation depends on the application requirements. For applications that prioritize detection speed over storage space, then the TensorRT versions of the models are recommended. However, if the storage requirements are more critical, then the Pytorch implementation is recommended.

4.5 Comparison with pruning

Pruning is a generic model simplification technique that has shown significant performance improvements in a number of applications. However, faster inference times often come at the cost of lower accuracies. The comparison of our approach with pruning showed (see Table 11), that just like our models, pruning slightly degrades model accuracy while increasing detection speed. As a matter of fact, the higher the pruning sparsity, the lower the accuracy and the higher the inference speed. When the base model was pruned to 0.2 sparsity, the resulting model accuracy surpassed three out of five of our optimised models. When pruned to 0.3 sparsity, the resulting model accuracy surpassed only two of our optimised models. In terms of speed, two of our models performed better that the pruned models by at least 17%. Higher detection speeds may be obtained with pruning, but that would further degrade model accuracy [18]. Overall, our model compares favourably with the existing high performing pruning approach.

4.6 Limitations of the work

One limitation of the model simplification approach in this paper is that the models may have difficulty with accurate detection of objects sizes that fall within the range of the discarded detection scale. It is therefore important that when choosing which variants of the optimised model to use, one should ensure that the size of the objects of interest do not fall within the range of the discarded scale. This can be easily accomplished by feeding the system back with the altitude from the UAV flight, which can provide the expected object size. Secondly, whereas the ideal situation would be to simplify the model without accuracy loss, our method records slight reductions in accuracy.

5 Conclusions

In this work, model simplification for object detection was achieved by exploiting algorithm-specific optimisation, namely, identification of the scales that are useful for the problem to be solved and discarding the out-of-scale parts of the detector architecture. Faster models suitable for real-time applications were realized while maintaining model accuracy. Compared to the popular pruning technique, our approach showed similar model accuracy with some of the derived models recording better detection speeds. The suitability of the models for real-time applications was shown in the results from the deployment of the models on the NVIDIA Jetson nano and Jetson Orin embedded devices for crop detection. Limitations come up naturally from the design premise (out-of-scale objects) but some ideas have been identified to appropriately tune the detector with additional sensing information.

For future work, we will be deploying the Jetson devices with the models on a UAV for in-field experiments and validation of the models. We also look to further optimize the models by additionally implementing optimization techniques such as knowledge distillation or low-rank factorization. We also consider conducting experiments to determine the exact range of the object sizes for each of the three detection scales, thus we can train the modified models with only images containing objects of sizes within the range of the chosen scales. This could save training time and improve accuracy.

Author contributions Conceptualization, E.C.N., P.C.H., and C.A.L.; methodology, E.C.N., P.C.H., and C.A.L.; data curation, E.C.N.; writing—original draft preparation, E.C.N.; writing—review and editing, E.C.N., P.C.H., C.A.L., O.I., and O.A.; supervision, P.C.H., C.A.L., O.I., and O.A.; funding acquisition, E.C.N., O.I., O.A., C.A.L., and P.C.H.; All authors have read and agreed to the published version of the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This research was funded by TETFUND NRF 2020 with grant number TETF/ES/DR&D-CE/NRF2020/SETI/88/VOL.1, ERASMUS + KA107 (ICM) PhD Mobility Scholarship, and DAAD In-country/In-region PhD scholarship. Thanks are also given to Agencia Estatal de Investigación for grant PID2020-115339RB-I00 and grant CPP2021-008880. The work was additionally supported in part by the EU Horizon 2020 Research and Innovation Programme under the Marie Sklodowska-Curie grant agreement No 101008297. This article reflects only the authors' view. The European Union Commission is not responsible for any use that may be made of the information it contains.

Data availability The cassava dataset is available from the authors upon reasonable request. The data is not publicly available since the dataset is still being expanded and is a part of an ongoing funded project and will be published after its completion.

Declarations

Conflicts of interest The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Mustafa MM, Hussain A, Ghazali KH, Riyadi S (2007) Implementation of image processing technique in real time vision system for automatic weeding strategy. In: 2007 IEEE international symposium on signal processing and information technology. IEEE, pp 632–635. https://doi.org/10.1109/ISSPIT.2007.4458197
2. Romeo J, Pajares G, Montalvo M et al (2013) A new expert system for greenness identification in agricultural images. Expert Syst Appl 40:2275–2286. https://doi.org/10.1016/j.eswa.2012.10.033
3. López-Granados F (2011) Weed detection for site-specific weed management: mapping and real-time approaches. Weed Res 51:1–11. https://doi.org/10.1111/j.1365-3180.2010.00829.x
4. Kamilaris A, Prenafeta-Boldú FX (2018) A review of the use of convolutional neural networks in agriculture. J Agric Sci 156:312–322. https://doi.org/10.1017/S0021859618000436
5. Moazzam SI, Khan US, Tiwana MI, et al (2019) A review of application of deep learning for weeds and crops classification in agriculture. In: 2019 international conference on robotics and automation in industry (ICRAI) Rawalpindi, Pakistan, pp 1–6. https://doi.org/10.1109/ICRAI47710.2019.8967350
6. Tang J, Wang D, Zhang Z et al (2017) Weed identification based on K-means feature learning combined with convolutional neural network. Comput Electron Agric 135:63–70. https://doi.org/10.1016/j.compag.2017.01.001
7. Czymmek V, Harders LO, Knoll FJ, Hussmann S (2019) Vision-based deep learning approach for real-time detection of weeds in organic farming. In: 2019 IEEE international instrumentation and measurement technology conference (I2MTC), Auckland, New Zealand, pp 1–5. https://doi.org/10.1109/I2MTC.2019.8826921
8. Kirchhoffer H, Haase P, Samek W et al (2022) Overview of the neural network compression and representation (NNR) standard. IEEE Trans Circuits Syst Video Technol 32:3203–3216. https://doi.org/10.1109/TCSVT.2021.3095970
9. Mazumder AN, Meng J, Rashid H-A et al (2021) A Survey on the optimization of neural network accelerators for micro-AI on-device inference. IEEE J Emerg Sel Top Circuits Syst 11:532–547. https://doi.org/10.1109/JETCAS.2021.3129415
10. Sun W, Chen S, Huang L et al (2021) Deep convolutional neural network compression via coupled tensor decomposition. IEEE J Sel Top Signal Process 15:603–616. https://doi.org/10.1109/JSTSP.2020.3038227
11. Oymak S, Soltanolkotabi M (2021) Learning a deep convolutional neural network via tensor decomposition. Inf Inference 10:1031–1071. https://doi.org/10.1093/imaiai/iaaa042
12. Wu G, Wang S, Liu L (2021) Fast video summary generation based on low rank tensor decomposition. IEEE Access 9:127917–127926. https://doi.org/10.1109/ACCESS.2021.3112695
13. Nekooei A, Safari S (2022) Compression of deep neural networks based on quantized tensor decomposition to implement on reconfigurable hardware platforms. Neural Netw 150:350–363. https://doi.org/10.1016/j.neunet.2022.02.024
14. Qi Q, Lu Y, Li J et al (2021) Learning low resource consumption cnn through pruning and quantization. IEEE Trans Emerg Top Comput 1–1. https://doi.org/10.1109/TETC.2021.3050770
15. Camci E, Gupta M, Wu M, Lin J (2022) QLP: Deep Q-learning for pruning deep neural networks. IEEE Trans Circuits Syst Video Technol 32:6488–6501. https://doi.org/10.1109/TCSVT.2022.3167951
16. Knight A, Lee BK (2020) Performance analysis of network pruning for deep learning based age-gender estimation. In: 2020 International conference on computational science and computational intelligence (CSCI), Las Vegas, NV, USA, pp. 1684–1687. https://doi.org/10.1109/CSCI51800.2020.00310
17. Lin Y, Tu Y, Dou Z (2020) An improved neural network pruning technology for automatic modulation classification in edge devices. IEEE Trans Veh Technol 69:5703–5706. https://doi.org/10.1109/TVT.2020.2983143
18. Hoefler T, Alistarh D, Ben-Nun T et al (2021) Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. J Mach Learn Res 22(241):1–124. http://jmlr.org/papers/v22/21-0366.html
19. Boateng VA, Yang B (2023) A global modeling pruning ensemble stacking with deep learning and neural network meta-learner for passenger train delay prediction. IEEE Access 11:62605–62615. https://doi.org/10.1109/ACCESS.2023.3287975
20. Li J, Chen X, Zheng P et al (2023) Deep generative knowledge distillation by likelihood finetuning. IEEE Access 11:46441–46453. https://doi.org/10.1109/ACCESS.2023.3273952
21. Feng Z, Lai J, Xie X (2021) Resolution-aware knowledge distillation for efficient inference. IEEE Trans Image Process 30:6985–6996. https://doi.org/10.1109/TIP.2021.3101158
22. Tao Z, Xia Q, Cheng S, Li Q (2023) An efficient and robust cloud-based deep learning with knowledge distillation. IEEE Trans Cloud Comput 11:1733–1745. https://doi.org/10.1109/TCC.2022.3160129

23. Sepahvand M, Abdali-Mohammadi F, Taherkordi A (2023) An adaptive teacher–student learning algorithm with decomposed knowledge distillation for on-edge intelligence. Eng Appl Artif Intell 117:105560. https://doi.org/10.1016/j.engappai.2022.105560
24. Zoph B, Le QV (2017) Neural architecture search with reinforcement learning. https://doi.org/10.48550/arXiv.1611.01578
25. Chitty-Venkata KT, Emani M, Vishwanath V, Somani AK (2023) Neural architecture search benchmarks: insights and survey. IEEE Access 11:25217–25236. https://doi.org/10.1109/ACCESS.2023.3253818
26. Thomas JB, Shihabudheen KV (2023) Neural architecture search algorithm to optimize deep Transformer model for fault detection in electrical power distribution systems. Eng Appl Artif Intell 120:105890. https://doi.org/10.1016/j.engappai.2023.105890
27. Khan S, Rizwan A, Khan AN et al (2023) A multi-perspective revisit to the optimization methods of Neural Architecture Search and Hyper-parameter optimization for non-federated and federated learning environments. Comput Electr Eng 110:108867. https://doi.org/10.1016/j.compeleceng.2023.108867
28. Ghimire D, Kil D, Kim S (2022) A survey on efficient convolutional neural networks and hardware acceleration. Electronics (Basel) 11:945. https://doi.org/10.3390/electronics11060945
29. Choi K, Wi SM, Jung HG, Suhr JK (2023) Simplification of deep neural network-based object detector for real-time edge computing. Sensors 23:3777. https://doi.org/10.3390/s23073777
30. Neill JO (2020) An overview of neural network compression. https://doi.org/10.48550/arXiv.2006.03669
31. Jeon ES, Choi H, Shukla A, Turaga P (2023) Leveraging angular distributions for improved knowledge distillation. Neurocomputing 518:466–481. https://doi.org/10.1016/j.neucom.2022.11.029
32. Kim T, Oh J, Kim N et al (2021) Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, Montreal, pp 2628–2635. https://doi.org/10.24963/ijcai.2021/362
33. Ding Z, Chen Y, Li N et al (2022) BNAS: efficient neural architecture search using broad scalable architecture. IEEE Trans Neural Netw Learn Syst 33:5004–5018. https://doi.org/10.1109/TNNLS.2021.3067028
34. Redmon J, Farhadi A (2018) YOLOv3: an incremental improvement. https://doi.org/10.48550/arXiv.1804.02767
35. Girshick R (2015) Fast R-CNN. In: 2015 IEEE international conference on computer vision (ICCV), Santiago, pp 1440–1448. https://doi.org/10.1109/ICCV.2015.169
36. Ren S, He K, Girshick R, Sun J (2017) Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell 39:1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031
37. He K, Gkioxari G, Dollár P, Girshick R (2020) Mask R-CNN. IEEE Trans Pattern Anal Mach Intell 42:386–397. https://doi.org/10.1109/TPAMI.2018.2844175
38. Jocher G, Stoken A, Borovec J et al (2020) YOLOv5. https://doi.org/10.5281/ZENODO.4154370
39. Redmon J, Farhadi A (2017) YOLO9000: better, faster, stronger. Proceedings - 30th IEEE Conference on computer vision and pattern recognition, CVPR 2017 2017-Janua: 6517–6525. https://doi.org/10.1109/CVPR.2017.690
40. Bochkovskiy A, Wang C-Y, Liao H-YM (2020) YOLOv4: Optimal speed and accuracy of object detection. https://doi.org/10.48550/arXiv.2004.10934
41. Liu W, Anguelov D, Erhan D et al (2016) SSD: Single shot multibox detector. Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) 9905 LNCS:21–37. https://doi.org/10.1007/978-3-319-46448-0_2/FIGURES/5
42. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: Unified, real-time object detection. Proceedings of the IEEE computer society conference on computer vision and pattern recognition 2016-Decem:779–788. https://doi.org/10.1109/CVPR.2016.91
43. Wang CY, Bochkovskiy A, Liao HYM (2021) Scaled-YOLOv4: scaling cross stage partial network. In: 2021 IEEE/CVF conference on computer vision and pattern recognition (CVPR). IEEE, pp 13024–13033. https://doi.org/10.1109/CVPR46437.2021.01283
44. Ge Z, Liu S, Wang F et al (2021) YOLOX: exceeding YOLO series in 2021. arXiv. https://doi.org/10.48550/arXiv.2107.08430
45. Wang CY, Yeh IH, Liao HYM (2021) You only learn one representation: unified network for multiple tasks. arXiv. https://doi.org/10.48550/arXiv.2105.04206
46. Xu S, Wang X, Lv W et al (2022) PP-YOLOE: An evolved version of YOLO. arXiv. https://doi.org/10.48550/arXiv.2203.16250
47. Li C, Li L, Jiang H et al (2022) YOLOv6: a single-stage object detection framework for industrial applications. arXiv. https://doi.org/10.48550/arXiv.2209.02976
48. Wang CY, Bochkovskiy A, Liao HYM (2022) YOLOv7: trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv. https://doi.org/10.48550/arXiv.2207.02696

49. Xu X, Jiang Y, Chen W et al (2022) DAMO-YOLO: a report on real-time object detection design. arXiv. https://doi.org/10.48550/arXiv.2211.15444
50. Jocher G, Chaurasia A, Qiu J (2023) YOLO by Ultralytics (Version 8.0.0). https://github.com/ultralytics/ultralytics
51. DECI AI (2023) YOLO-NAS. https://github.com/Deci-AI/super-gradients/blob/master/YOLONAS.md
52. Wang D, He D (2021) Channel pruned YOLO V5s-based deep learning approach for rapid and accurate apple fruitlet detection before fruit thinning. Biosyst Eng 210:271–281. https://doi.org/10.1016/j.biosystemseng.2021.08.015
53. Wu D, Lv S, Jiang M, Song H (2020) Using channel pruning-based YOLO v4 deep learning algorithm for the real-time and accurate detection of apple flowers in natural environments. Comput Electron Agric 178. https://doi.org/10.1016/J.COMPAG.2020.105742
54. Wang Z, Jin L, Wang S, Xu H (2022) Apple stem/calyx real-time recognition using YOLO-v5 algorithm for fruit automatic loading system. Postharvest Biol Technol 185:111808. https://doi.org/10.1016/j.postharvbio.2021.111808
55. Yin Y, Li H, Fu W (2020) Faster-YOLO: an accurate and faster object detection method. Digit Sign Process: Rev J 102. https://doi.org/10.1016/J.DSP.2020.102756
56. Thuan D (2021) Evolution of Yolo algorithm and Yolov5: the state-of-the-art object detection algorithm. Oulu University of Applied Sciences. https://www.theseus.fi/handle/10024/452552
57. Sirisha U, Praveen SP, Srinivasu PN et al (2023) Statistical analysis of design aspects of various YOLO-based deep learning models for object detection. Int J Comput Intell Syst 16:126. https://doi.org/10.1007/s44196-023-00302-w
58. Martinez-Alpiste I, Golcarenarenji G, Wang Q, Alcaraz-Calero JM (2021) A dynamic discarding technique to increase speed and preserve accuracy for YOLOv3. Neural Comput Appl 33:9961–9973. https://doi.org/10.1007/s00521-021-05764-7
59. Tzutalin (2015) LabelImg. Git code. https://github.com/HumanSignal/labelImg
60. Zaidi SSA, Ansari MS, Aslam A et al (2022) A survey of modern deep learning based object detection models. Digit Signal Process 126:103514. https://doi.org/10.1016/J.DSP.2022.103514
61. Rezatofighi H, Tsoi N, Gwak J et al (2019) Generalized intersection over union: a metric and a loss for bounding box regression. In: 2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR). IEEE, pp 658–666. https://doi.org/10.1109/CVPR.2019.00075

Authors and Affiliations

Emmanuel C. Nnadozie[1,2,3] · Pablo Casaseca-de-la-Higuera[1] · Ogechukwu Iloanusi[2] · Ozoemena Ani[3] · Carlos Alberola-López[1]

✉ Emmanuel C. Nnadozie
ennadozie@lpi.tel.uva.es

Pablo Casaseca-de-la-Higuera
jcasasec@tel.uva.es

Ogechukwu Iloanusi
ogechukwu.iloanusi@unn.edu.ng

Ozoemena Ani
ozoemena.ani@unn.edu.ng

Carlos Alberola-López
caralb@tel.uva.es

[1] Laboratorio de Procesado de Imagen, ETSI Telecomunicación, University of Valladolid, Valladolid, Spain

[2] Department of Electronic Engineering, University of Nigeria, Nsukka, Nigeria

[3] Department of Mechatronic Engineering, University of Nigeria, Enugu, Nigeria