



ISSN ONLINE: 2447-0228



RESEARCH ARTICLE

OPEN ACCESS

EDGE-AI EMPOWERED WASTE DETECTION: OPTIMIZING YOLOV11 FOR RESOURCE-LIMITED DEVICES

Selma Azizi^{*1}, Abderrahmane Bendahmane² and Redouane Tlemsani³^{1,2}Signal Image Speech Laboratory - Computer Sciences Department, University of Sciences and Technology of Oran Mohamed Boudiaf, Algeria.³Advanced Data Science and Cognitive Applications Laboratory - Computer Sciences Department, University of Sciences and Technology of Oran Mohamed Boudiaf, Algeria.¹<https://orcid.org/0009-0002-3518-4965>, ²<https://orcid.org/0000-0003-0801-9550>, ³<http://orcid.org/0000-0002-2426-5113>Email: *selma.azizi@univ-usto.dz, abderrahmane.bendahmane@univ-usto.dz, redouane.tlemsani@univ-usto.dz

ARTICLE INFO

Received: February 7, 2026

Reviewed: April 15, 2026

Accepted: May 15, 2026

Published: June 30, 2026

Keywords:YOLOv11,
Knowledge distillation,
Optimization,
Deep learning,
Waste Detection.

ABSTRACT

This work optimizes the YOLOv11 object detector to enable real-time, high-precision waste sorting on resource-constrained embedded devices. The optimization is carried out in four steps, replacing standard convolution blocks with Ghost Conv layers and swapping the C2PSA and C3K2 modules for lighter alternatives, applying knowledge distillation from a full YOLOv11 model to a compact model to improve the accuracy of the compressed model, fusing neighboring layers to reduce memory access, and quantifying to 16-bit floating point (FP16) while generating builds for ONNX and NCNN. This optimization reduces the number of parameters by 46%. Compared to the full YOLOv11 model, the optimized version runs faster and uses fewer resources on the Raspberry Pi 4. On the TrashNet dataset, the inference latency is reduced from 1,077.1 ms to 329 ms, and further to 296.5 ms with the NCNN FP16 version. On the Drinking Waste Detection dataset, the latency decreases from 1,019.5 ms to 299.4 ms. Memory usage is reduced from 5.23 MB to about 3 MB, and energy consumption drops from 0.74 Wh to 0.29 Wh. In terms of performance, the mAP50 of the PyTorch model remains high, with 95.9% on TrashNet and 99.3% on Drinking Waste, closely matching the original YOLOv11 model's performance. These results show that our optimization approach creates a compact and accurate model suitable for embedded recycling systems running on modest hardware like the Raspberry Pi 4.



Copyright ©2026 by authors and Galileo Institute of Technology and Education of the Amazon (ITEGAM). This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

I. INTRODUCTION

The increase in the global population, coupled with changes in lifestyle and individual behaviors, has led to a significant rise in waste production. The report published by the United Nations Environment Program (UNEP) in 2024 highlights that more than two billion tons of municipal solid waste are produced annually. If this waste were stored in standard maritime containers and lined up end to end, they could encircle the Earth's equator 25 times, thus illustrating the urgency and necessity of effectively managing waste to meet major environmental, social, and economic challenges worldwide [1]. In response to this growing issue, recycling emerges as one of the most effective solutions [2]. This process begins with a rigorous waste sorting system that facilitates the separation of materials such as plastic, paper, glass, and metal, thereby aiding their subsequent processing. However, manual sorting heavily relies on increased public awareness, which is not always guaranteed. It is in this context that modern technologies, such as deep learning, can play a crucial role in improving waste management processes. In this regard, deep learning architectures have made significant advances in waste detection, particularly with the You Only Look Once (YOLO) model, which stands out for its real-time performance and proven effectiveness in various scenarios [3]. The YOLOv11 model, released in 2024, represents one of the latest developments in this series, offering an excellent compromise between accuracy and speed, making it particularly suitable for waste detection applications in real-world environments [4].

However, despite its performance, the implementation of YOLOv11 on embedded devices with limited resources, such as the Raspberry Pi, remains a major challenge. These platforms face constraints in terms of computing power and cost, necessitating the optimization of the model to make it compatible with resource-constrained environments. Several optimization techniques, commonly used in this context, allow these models to be adapted to embedded platforms. One of the most widely used approaches is to replace certain complex layers, which are responsible for significant computational load, with lighter alternatives. However, this simplification can sometimes lead to a loss in performance. This is where knowledge distillation comes in: by transferring the knowledge from a more complex model to a lighter one, this method helps maintain or even improve the performance of the optimized model. The main contributions of this work are:

- An efficient redesign of YOLOv11 for embedded devices, using streamlined modules, layer fusion and FP16 quantization to reduce parameters, memory usage and computational complexity.
- Integration of knowledge distillation from the full YOLOv11 model to enhance the accuracy of the compressed models.
- Deployment-ready builds (ONNX and NCNN) with comparative evaluation on a Raspberry Pi 4B.

Following this introduction, Section 2 reviews deep learning models for waste detection and discusses the associated optimization techniques with an emphasis on the technical characteristics of YOLOv11. Section 3 details our methodology by explaining how layers were replaced and knowledge distillation was utilized to optimize YOLOv11. Section 4 presents the results obtained after optimization through a comparison with other YOLO versions. Finally, the paper concludes with a discussion of the key findings and suggestions for future research.

II. THEORETICAL REFERENCE AND RELATED WORK

The application of object detection techniques has significantly evolved over the years. Traditional approaches such as R-CNN and Faster R-CNN relied on generating multiple region proposals before classification, resulting in computational inefficiencies for real-time applications. With the development of single-stage detectors, particularly the You Only Look Once model (YOLO), object detection has become faster and more accurate, making it suitable for real-time tasks [3]. As its name suggests, YOLO analyzes the entire image in a single pass to identify objects and their positions [5], [6]. For waste detection, YOLO is extensively used to identify and sort various types of waste. For instance, [7] used YOLOv3, trained on the TrashNet and Taiwan Recycled Waste Database, achieving an mAP@0.5 of 81.36% with the TrashNet dataset. By [8] proposed a system based on YOLOv4 for the automated detection and classification of recyclable materials. Evaluated in a Greek waste management facility, the model demonstrated a precision of 92.43%, utilizing the TACO dataset enhanced with 2,500 additional photos. In turn [9] used YOLOv5 to classify waste with 94.5% accuracy using a 1,195-image dataset from the TACO dataset. By [10] proposed Skip-YOLO, an improved YOLOv3 model with dense convolution blocks for extracting high-dimensional feature maps, achieving an mAP@50 of 90.38%, outperforming YOLOv3, YOLOv5, and YOLOv7. Given the effectiveness of the different versions of YOLO, several researchers have aimed to optimize these models to make them lighter, in order to adapt them to low-resource hardware, improve their performance, or make them faster than the original versions. For example, [11] developed an intelligent system using YOLOv4-tiny for waste detection and recyclable classification via a webcam on a Raspberry Pi, achieving 91% accuracy with a dataset of 1,000 images. Ren, [12] proposed MRS-YOLO, a lightweight version of YOLOv8, which reduced the model size by 15.09% while improving mAP@50% by 3.6% compared to YOLOv8. These results were achieved using a dataset of 12,072 images spanning 10 waste categories. Similarly, [13] developed FE-YOLO, an enhanced YOLOv8 algorithm for detecting construction waste in complex scenarios. FE-YOLO replaces the C2f module with Faster_C2f and integrates the ECA attention mechanism, improving mAP@50 by 3%, reducing parameters by 12%, and floating-point operations by 13% compared to YOLOv8n. The model was evaluated on a custom construction waste dataset collected from the Recycling Center in Xiangyang City.

Lastly, [14], proposed Waste-YOLO, an enhanced version of YOLOv5s, for real-time detection of abnormal waste in waste-to-energy power plants. Key improvements included the integration of Coordinate Attention, CARAFE upsampling, BiFPN feature fusion, and the SCYLLA-IoU loss function, boosting accuracy and efficiency. Tested on a custom dataset of 6,283 image, Waste-YOLO achieved an mAP@50 of 96.2% and mAP@50-95 of 77.4%, surpassing YOLOv5s by 2.0% and 2.1%, respectively. In addition to architectural modifications, several optimization techniques have been applied to YOLO in various domains. One of the key optimization techniques is pruning, which is used in deep neural networks to reduce the model size by removing elements such as neurons and connections that are considered non-essential. This helps decrease computational complexity and lower memory usage, all without significantly affecting the model's performance [15]. For example, [16] employed a hybrid pruning approach by combining channel pruning and layer pruning. This resulted in a 97.5% reduction in parameters and an 82.3% decrease in computational load, while maintaining detection performance similar to the original YOLOv3 model. Another important technique is quantization, which involves reducing the precision of the weights and activation values in a neural network to lower bit-width representations. This helps make the model more efficient by decreasing the memory needed and speeding up computation, which is crucial for running models on devices with limited resources. There are two main approaches: Quantization-Aware Training (QAT), which requires retraining the model to optimize for low-precision, and Post-Training Quantization (PTQ), which quantizes the model after it has been fully trained. PTQ is especially useful for real-time object detection as it allows for faster inference without needing extra training resources [17]. The DoReFa-Net method was used in this context by [16], effectively reducing the model's size and improving computation speed. Finally, knowledge distillation (KD) is a technique that involves transferring the knowledge learned by a large, complex model (the teacher) to a smaller, simpler model (the student). The smaller model is trained to mimic the behavior of the larger model by using a combination of two types of loss functions: hard loss, which compares the student's predictions to the ground truth, and soft loss, which encourages the student to match the teacher's output. This method helps the smaller model achieve better performance, particularly in tasks like object detection, without needing the same computational resources as the larger model [18]. By [19] applied KD to optimize the YOLOX model for object detection in side-scan sonar images used in underwater robotics.

By transferring knowledge from a larger model to a smaller one, they achieved a more efficient deployment of YOLOX without sacrificing accuracy. In our research, we aim to optimize the YOLOv11n model, by reducing its parameters and complexity while preserving similar performance. Our proposed optimization follows a four-stage framework that includes: architectural simplification using lighter modules, knowledge distillation, layer fusion, and precision reduction to FP16

III. MATERIALS AND METHODS

III.1 DATASET DESCRIPTION

In this study, we selected two datasets focused on waste classification and detection. This choice was made to optimize a model for an automated sorting system, like a smart trash bin, where the camera often faces a plain, consistent background.

III.1.1 Trashnet Dataset

Designed for waste classification [20], consisting of images categorized into six classes: glass, metal, trash, paper, plastic, and cardboard. Each image contains a single item captured against a plain background under sunlight or artificial lighting. In this study, we used an online version of the dataset [21] containing 2,524 images, divided into 505 images for validation, 252 images for testing, and 1,767 images for training.

III.1.2 Drinking Waste Dataset

This dataset contains over 4,800 images of drinking-related waste, divided into four categories: aluminium cans, glass bottles, PET bottles, and HDPE containers [22]. The photos were taken with a 12-megapixel phone camera [23], so the data was collected in simple, realistic conditions. This dataset is useful for building systems that can recognize and sort these types of waste automatically, especially in controlled settings like smart recycling bins. To prepare the training data, data augmentation techniques were applied, including horizontal and vertical flipping, 90° rotations (clockwise, counterclockwise, and upside down), cropping with a zoom range of 0% to 20%, random rotations between -45° and +45°, and brightness adjustments between -15% and +15%.

III.2 YOLOV11 ARCHITECTURE

In this study, YOLOv11 was chosen as the base model to optimize. YOLOv11 [4], consists of three main components: the Backbone, the Neck, and the Head, as shown in Figure.1.a. Each of these components plays a crucial role in extracting, processing, and predicting features from the input image. The Backbone uses several convolutional blocks, including the recently introduced C3k2 block, which provides a more efficient feature extraction mechanism. This block allows the model to extract complex features with fewer parameters, thus improving both accuracy and processing speed. Next, the Neck aggregates and refines the information using blocks like SPPF (Spatial Pyramid Pooling Fast) and C2PSA (C2 Partial Self-Attention). These blocks are strategically placed to capture multi-scale features and enhance the model's ability to focus on relevant parts of the image. The SPPF manages information from different scales to better detect objects at various resolutions, while the C2PSA applies partial spatial attention to better process the important regions of the image. Finally, the Head performs classification and bounding box regression, using the information passed from the previous stages to make accurate predictions. The YOLOv11 architecture is designed to efficiently handle multi-scale features, and the attention mechanisms within the structure help the model focus on important regions of the image. As seen in the YOLOv11 architecture, the use of C3k2 blocks ensures that the model can extract complex features while reducing the number of parameters, thereby improving both accuracy and processing speed.

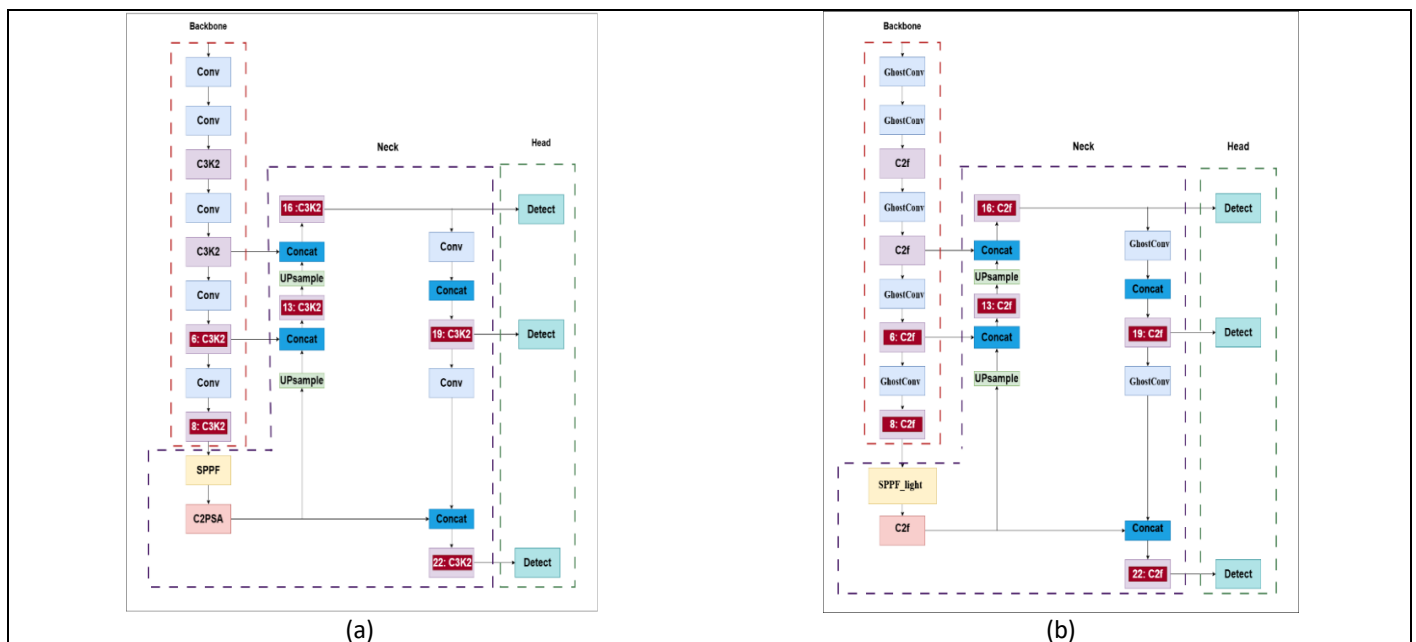


Figure 1: (a) Original YOLOv11 architecture, (b) Optimized YOLOv11 architecture.

Source: Authors, (2026).

III.3. YOLOV11 OPTIMIZATION

In this subsection, we provide a detailed overview of all the optimizations made to enhance the YOLOv11 model's efficiency and effectiveness.

III.3.1 Replacing Convolutional Blocks

To simplify the YOLOv11 model, we replaced its standard convolutional blocks (convolution, Batch Normalization, and SiLU activation) with GhostConv blocks. The Ghost module creates extra feature maps using inexpensive linear operations on intrinsic ones, cutting down on redundant computations and reducing both parameters and computational complexity [24]. A GhostConv block (see Figure. 2.a) comprises two convolutions. The first (cv1) uses a 1x1 kernel to halve the channel count while retaining key input information. The second (cv2) uses a 5x5 kernel to refine these features. Concatenating the outputs restores the desired channel count, leading to faster execution with fewer parameters.

III.3.2 Replacing the SPPF Block

The Spatial Pyramid Pooling Fast (SPPF) block in YOLOv11 (Figure. 2.b) enhances feature extraction at multiple scales by applying pooling operations with 3x3, 5x5, and 9x9 kernels [4]. To boost performance while reducing complexity, we modified the original SPPF to a lightweight version, SPPF_light (Figure.2.c). This version uses a single MaxPooling branch followed by a 1x1 convolution, retaining multi-scale capability with improved efficiency and lower computational costs.

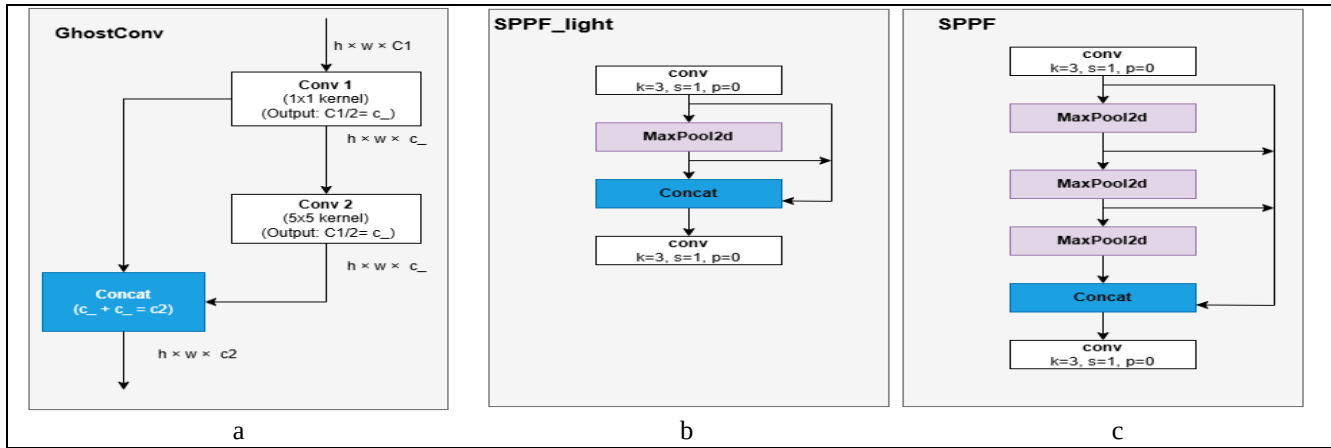


Figure 2: GHOSTConv architecture, (b) SPPF_Light architecture, (c) SPPF architecture.

Source: Authors, (2026).

III.3.3 Replacing the C2PSA Block

The C2PSA module (Figure. 3.b) in YOLOv11 enhances feature extraction through stacked PSABlocks (Figure. 3.a) and multi-head attention, allowing the model to focus on key image regions in complex scenes [4]. However, its use of multi-head attention and 1x1 convolutions increases complexity. To address this, we replaced C2PSA with a simplified, more efficient C2f module (Figure. 3.c). The C2f begins with a 1x1 convolution to reduce channel dimensions, splits the output, processes one branch through repeated Bottleneck layers, and then merges the features with another 1x1 convolution. This design reduces parameters and computational load while retaining effective feature extraction and gradient flow.

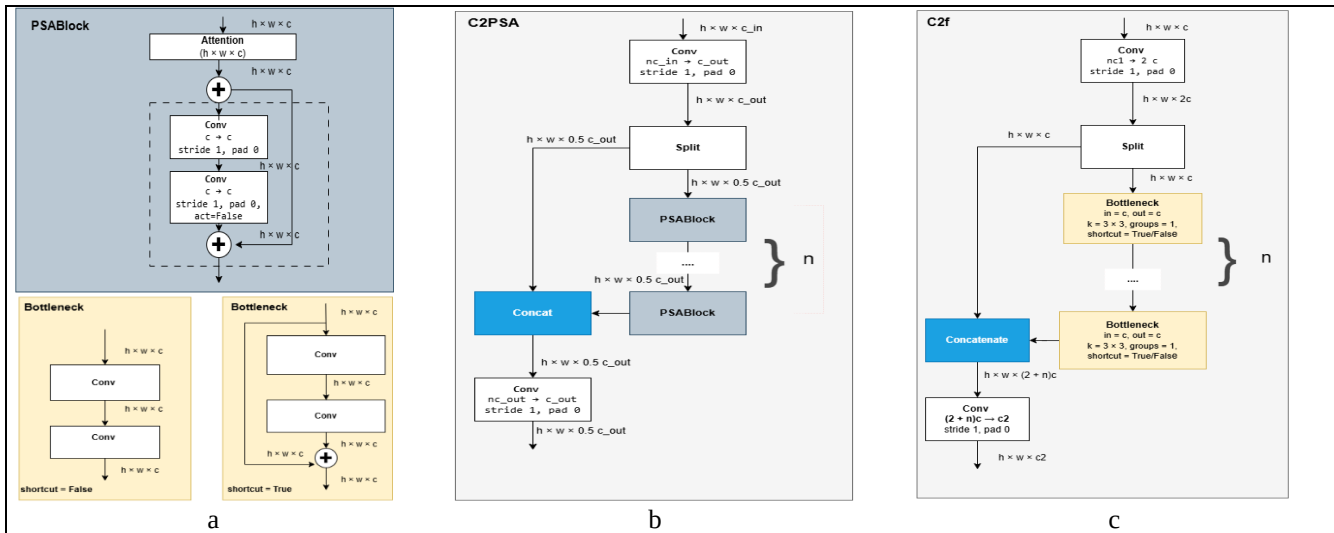


Figure 3: (a) PSABlock architecture, (b) C2PSA architecture, (c) C2f architecture.

Source: Authors, (2026).

III.3.4 Replacing C3k2 with C2f

The C3k2 module (Figure. 4.a) in YOLOv11 significantly boosts object detection by extracting robust features. It is integrated in the backbone to reduce image size while increasing channels, in the neck to aggregate multi-scale information, and in the head to refine feature maps before predictions [25]. C3k2 introduces an extra 1x1 convolution (cv3) and employs a more complex C3k block (Figure. 4.b) in place of some Bottleneck blocks from C2f. Using three 1x1 convolutions instead of two allows finer feature management but increases computational load. To reduce the model size and computational complexity, the width multiplier was reduced to 0.20, and the expansion factor in the C2f blocks was adjusted. These adjustments allow the model to maintain good performance while reducing computational costs. Due to this reduction in complexity and improved parameter efficiency, C2f was chosen to replace C3k2, offering a better balance between performance and efficiency. The final architecture of the optimized model is shown in Figure. 1.b.

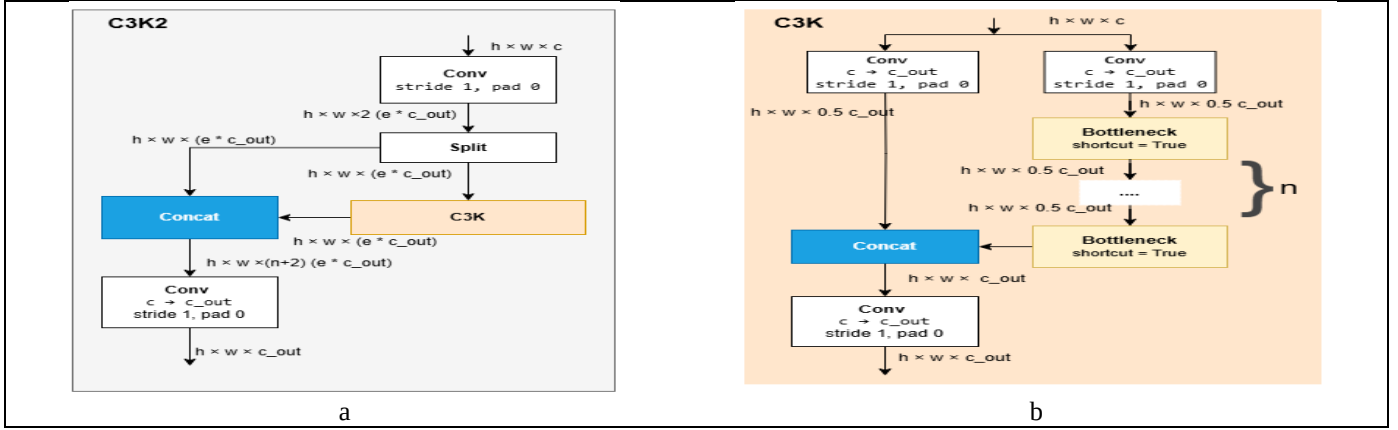


Figure 4: (a) C3K2 architecture, (b) C3K architecture.

Source: Authors, (2026).

III.3.5 Reducing the Width Multiplier in Scales

As part of the YOLO model optimization to lighten its structure and reduce the number of parameters, we made a strategic modification in the "scales" section of the configuration YAML file. Specifically, we adjusted the width multiplier of the 'n' variant from 0.25 to 0.20. This modification aims to proportionally reduce the number of channels in each model layer. By lowering this value, we manage to reduce the model's complexity while maintaining reasonable efficiency, which allows us to lighten the computation without excessively compromising performance.

III.4 KNOWLEDGE DISTILLATION (KD)

In this study, we applied Knowledge Distillation (KD) to enhance the performance of our optimized model, which initially underperformed compared to the YOLOv11n model. KD transfers knowledge from a complex, high-performing teacher model to a simpler, more lightweight student model [26]. In our experiment, YOLOv11n serves as the teacher model, and the optimized model functions as the student model. We attach forward hooks to layers 6, 8, 13, 16, 19, and 22 (outputs of the cv2 sub-modules). These layers correspond exactly to the stages that were lightened/modified in the optimized architecture (GhostConv, C2f, SPPF_LIGHT). Distilling at points where the capacity was reduced helps the student recover lost representational power by mimicking the teacher's multi-scale features (from mid to high levels), rather than distilling only the final logits. Figure 5 illustrates this process. The student learns to replicate the teacher's intermediate activations, capturing both low-level and high-level features while reducing complexity [27].

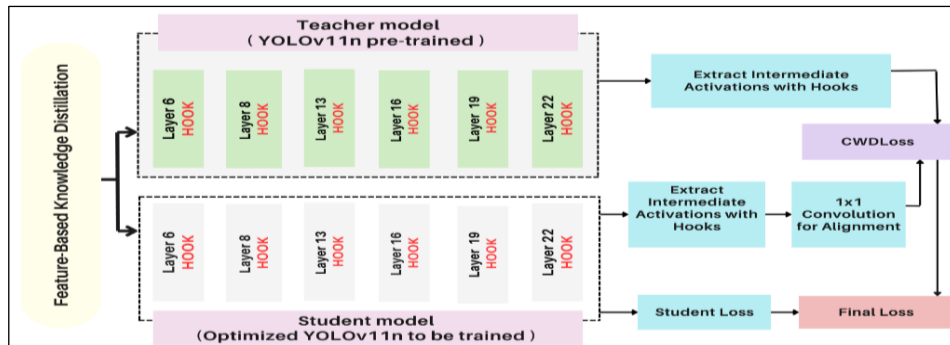


Figure 5: Feature-based knowledge distillation technique.

Source: Authors, (2026).

Before comparing the intermediate activations between the student and teacher models, their dimensions must first be aligned for compatibility. To achieve this, we apply a 1×1 convolution followed by BatchNorm on the student's outputs. Formally, after alignment, the intermediate activations for both models are spatially flattened by channel and are defined by $S_i, T_i \in \mathbb{R}^{H \times W}$, where $i = 1, 2, \dots, N \times C$. Feature-based knowledge distillation relies on comparing intermediate activations between models.

To perform effective comparisons even when dimensions differ, we compute the Channel-Wise Distillation Loss (CWD Loss) using a temperature parameter τ . The Softmax probabilities (with temperature τ) for intermediate activations of each model $\{T, S\}$ (T : teacher, S : Student) is defined by Eq. (1):

$$p_j^{\{T,S\}i} = \frac{\exp\left(\frac{\{T,S\}i(j)}{\tau}\right)}{\sum_{k=1}^{HW} \exp\left(\frac{\{T,S\}i(k)}{\tau}\right)} \quad (1)$$

The divergence between distributions for each channel (i) is computed according to Eq. (2):

$$L_{CWD}^{(i)} = \tau^2 \sum_{j=1}^{HW} \left(p_j^{Ti} \log(p_j^{Ti}) - p_j^{Si} \log(p_j^{Si}) \right) \quad (2)$$

The knowledge distillation loss $L_{KD}(t)$ for an intermediate layer activation for the current epoch t is computed by averaging these channel losses as defined by Eq. (3):

$$L_{KD}(t) = w(t) \times \frac{1}{N.C} \sum_{i=1}^{N.C} L_{CWD}^{(i)} \quad (3)$$

The dynamic weighting for the current epoch, t , evolves relative to the total number of epochs, T , during training according to Eq. (4):

$$w(t) = 1 - 0.45 \left(1 - \cos\left(\frac{\pi t}{T}\right) \right) \quad (4)$$

A detailed description of all symbols used in these equations is provided in Table 1, and the KD hyperparameters used are summarized in Table 2

Table 1: List of Symbols and Notations.

Notation	Description
$N \times C$	Batch size $\times$ Number of channels
H / W	Spatial (Height / Width) of the feature maps
τ	Temperature parameter used in softmax to smooth the distributions.
S_i, T_i $\in \mathbb{R}^{H.W}$	Flattened activations for the student S and teacher T models, with flattened spatial dimensions for channel i .
$p_j^{\{T,S\}i}$	Softmax probabilities (channel i , pixel j) for the teacher T or the Student S model after applying temperature τ

Source: Authors, (2026).

Table 2: KD Hyperparameters.

Hyperparameter	Value
Distillation type	Feature-based CWD (Channel-Wise Distillation)
Temperature τ	3.0
KD weight α	1.0 (KD term added to YOLO loss)
Distilled layers	[6, 8, 13, 16, 19, 22], cv2 outputs

Source: Authors, (2026).

III.5 MODEL DEPLOYMENT

In this study, we aim to optimize and deploy the model for devices with limited resources using portable formats such as TensorRT [28], ONNX [29], and NCNN [30]. ONNX provides excellent portability and cross-platform compatibility by enabling seamless transfers between frameworks like PyTorch and TensorFlow, while also supporting FP16 quantization for enhanced efficiency [31]. Notably, NCNN is particularly recommended for Raspberry Pi devices [30] due to its lightweight design and efficient performance. The deployment process also involves converting weights and activations from FP32 to FP16, which reduces the model size and enables faster inference. Performance metrics were computed on various hardware, including the CUDA:0 GPU (Tesla T4, 15095 MiB), the CPU (Intel Xeon 2.20 GHz), and a Raspberry Pi 4 (Cortex-A72).

III.6 TRAINING AND EVALUATION SETUP

The training configuration was kept consistent across both datasets and all models to ensure reproducibility and fair comparison is showed in Table 3

Table 3: Training Hyperparameters and Hardware Setup.

Parameter	Value
Batch size	16
Optimizer	AdamW
Learning rate	0.01
Momentum	0.937
Weight decay	0.0005
pretrained weights	yolo11n.pt
Epochs (TrashNet)	100 (extended to 200 with distillation)
Epochs (Drinking Waste Detection)	50
Hardware used for training	NVIDIA Tesla L4 GPU

Source: Authors, (2026).

Evaluation was performed on three hardware platforms to assess inference performance:

- GPU: NVIDIA Tesla T4 (16 GB memory)
- Embedded device: Raspberry Pi 4 model B (Cortex-A72 processor)

IV. RESULTS AND DISCUSSIONS

This section presents the results obtained from our model optimization approach. The models used in this approach were evaluated on several performance criteria, including mAP50, mAP50-95, the number of parameters, complexity in GFLOPs, and inference time. The models were tested on the test set of each dataset, consisting of previously unseen images. To ensure the stability of the results, multiple runs were conducted for each configuration. Performance metrics such as mAP50, mAP50-95 and inference times demonstrated significant stability, with consistent results and minimal variability, indicating the robustness of the model. For inference times, the minor differences between runs are due to natural fluctuations in computational resources.

IV.1 ABLATION EXPERIMENTS

IV.1.1 Resource Ablation

The original YOLOv11n model consists of 319 layers with 2.6 million parameters and a complexity of 6.4 GFLOPs. However, its relatively high parameter count and complexity limit its applicability to embedded platforms. To quantify the gains in compactness, we sequentially integrate GhostConv, followed by C2f + SPPF_LIGHT into YOLOv11n, and measure the number of parameters, memory size, and GFLOPs (using identical input and hardware configurations). Table 4 summarizes these measurements and highlights a significant reduction in computational complexity.

Table 4: YOLOv11n resource ablation.

	Parameters (M)	Complexity (GFLOPs)
YOLOv11n	2.6	6.4
GhostConv	1.6	4.8
+ C2f+ SPPF_LIGHT	1.4	4.6

Source: Authors, (2026).

As shown in Table 4, replacing standard convolutions with GhostConv reduces the number of parameters by 38% (from 2.6M to 1.6M) and GFLOPs by 25% (from 6.4 to 4.8). Substituting the C2PSA and C3K2 modules with the simpler C2f module, and replacing the SPPF module with its lightweight variant, SPPF_LIGHT, further reduces the parameters by 12.5% and GFLOPs by 4.2%. This results in a total reduction of 46% in parameters (from 2.6M to 1.4M) and 28% in GFLOPs (from 6.4 to 4.6) compared to the original model. These findings demonstrate that each proposed block effectively reduces computational complexity. This size/complexity trade-off is relevant for embedded deployment.

IV.1.2 Training and Ablation on TrashNet Dataset

Figure 6.a illustrates the mAP50 curves for the four variants. The baseline YOLOv11n model starts with a high mAP50 since all of its layers load pre-trained weights that are well-suited for its original architecture (complete transfer learning). In contrast, the GhostConv and GhostConv + C2f + SPPF_LIGHT variants replace entire blocks. As a result, only the unchanged parts benefit from transferred weights, while the new layers are initialized randomly, leading to a lower starting point followed by gradual improvement. The student model with knowledge distillation (KD) benefits from "soft" targets provided by the teacher, which accelerates convergence and reduces the gap. Visually, its curve stabilizes near the teacher's performance (≈ 0.93 – 0.95) after approximately 100 epochs. Noting that the KD curve maintains a positive slope towards the end of the interval (80–100 epochs), we extended the training by an additional 100 epochs. Figure 6.b presents the results of this extension: mAP50 continues to rise until plateauing at ≈ 0.96 , while mAP50-95 progresses to ≈ 0.94 . Meanwhile, recall stabilizes high at ≈ 0.93 , and precision reaches ≈ 0.89 . All the curves indicate late stabilization (around ~ 170 epochs) and confirm that the extension to 200 epochs was crucial to achieve convergence and maximize the benefits of KD.

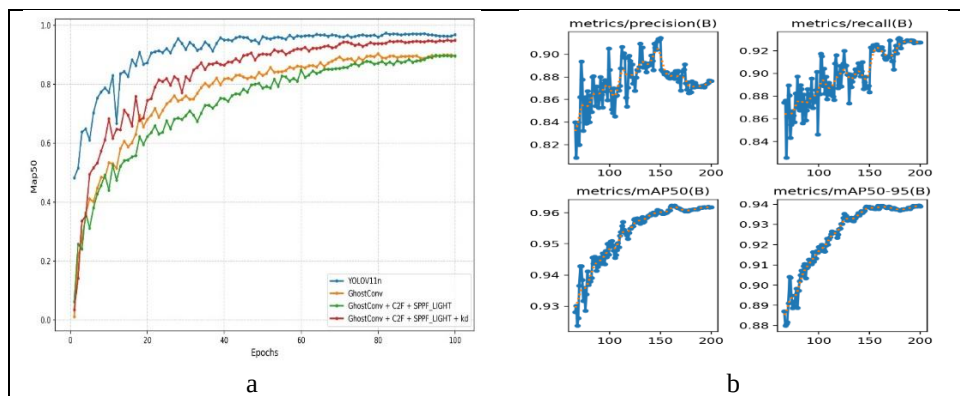


Figure 6: (a) Ablation on TrashNet—mAP50 versus epochs for YOLOv11n (baseline), +GhostConv, +C2f+SPPF_LIGHT, and the KD-trained student. (b) Extended KD training on TrashNet precision, recall, mAP50, and mAP50-95 over 200 epochs.

Source: Authors, (2026).

IV.1.3 Test Evaluation TrashNet & Drinking Waste

To validate the improved detection accuracy of the modified model, we tested it on a custom test dataset for TrashNet and on the Drinking Waste test set to assess the behavior of our approach on a different dataset. The table 5 presents, for each variant, the GPU inference latency (ms) and accuracy metrics (mAP50, mAP50-95).

Table 5: Performance of optimized YOLOv11n and ablation study on TrashNet and Drinking Waste datasets.

	YOLOv11n optimizations					Inference Time on GPU (ms)	mAP50	mAP50-95
	GhostConv	C2f	SPPF_ LIGHT	kd epochs	Conv + Batch Norm Fusion			
TrashNet dataset						6.2	96.4	94.0
	✓					6.0	92.1	86.4
	✓	✓	✓			5.2	90.1	85.3
	✓	✓	✓	100		5.2	91.7	88.2
	✓	✓	✓	200		5.2	95.9	93.1
	✓	✓	✓	200	✓	4.8	95.9	93.1
Drinking waste dataset						5.0	99.7	85.3
	✓					4.8	99.1	82.4
	✓	✓	✓			3.7	98.2	81
	✓	✓	✓	50		3.7	99.3	83.4
	✓	✓	✓	50	✓	3.1	99.3	83.4

Source: Authors, (2026).

The original YOLOv11n achieves a mAP50 of 96.4% with an inference time of 6.2 ms on TrashNet, and a mAP50 of 99.7% with an inference time of 5.0 ms on Drinking Waste. After applying a series of optimizations to reduce the model size while maintaining performance, the results reveal a consistent pattern: model compression without distillation decreases latency but leads to a slight reduction in accuracy. Table 5 presents the detailed results for each optimization and its impact on both datasets. Initially, replacing standard convolutions with GhostConv reduces the inference time (from 6.2 ms to 6.0 ms on GPU for TrashNet, and from 5.0 ms to 4.8 ms for Drinking Waste), while detection accuracy, measured by mAP50 and mAP50-95, also drops (from 96% to 92.1% on TrashNet, and marginally on Drinking Waste, where mAP50 remains above 99%). Substituting the C2PSA and C3K2 modules with the simpler C2f module, and replacing the SPPF module with its lightweight variant, SPPF_LIGHT, further reduces the inference time for both models to 5.2 ms for TrashNet and 3.7 ms for Drinking Waste. This leads to a decrease in mAP50, as expected, since the new modules are less performant than the original ones. The final mAP50 values are 90% for TrashNet and 98% for Drinking Waste. Introducing knowledge distillation (KD) restores much of the performance while maintaining the efficiency gains. After 200 epochs, TrashNet reaches a mAP50 of 95.9% and a mAP50-95 of 93.1%, while Drinking Waste achieves a mAP50 of 99.3% and a mAP50-95 of 83.4% after 50 epochs. Finally, the Conv-BN fusion reduces latency further without compromising accuracy.

For TrashNet, the inference time decreases from 5.2 ms to 4.8 ms, and for Drinking Waste, from 3.7 ms to 3.1 ms, compared to the original YOLOv11n model, which had an inference time of 6.2 ms and 5.0 ms, respectively. These results demonstrate that the compression → KD → fusion pipeline produces a model that is substantially faster and nearly as accurate in mAP50 as the original model, with a slight decrease in mAP50-95 (0.52% on TrashNet). A 22.58% gain in speed over the original YOLOv11n model on TrashNet confirms that this trade-off is well-suited for embedded platforms, where latency and robustness are key priorities. For the Drinking Waste dataset, the optimized model achieves the same performance in mAP50 as the original, while significantly reducing the inference time, making it equally effective for real-time applications. The optimizations lead to a total reduction of 46% in parameters (from 2.6M to 1.4M) and 28% in GFLOPs, significantly reducing computational complexity, which is essential for embedded systems with limited resources. These improvements align with the goal of enhancing computational efficiency without major sacrifices in performance. By strategically applying these modifications, we have developed a model that operates faster and is more suitable for real-time applications on constrained hardware, meeting the specific requirements of embedded systems.

IV.2 PERFORMANCE COMPARISON WITH RECENT LIGHTWEIGHT OBJECT DETECTORS

In this subsection, we compare our approach and optimized model with several recent lightweight models known for their good performance in deep learning and object detection. To do so, we selected two popular versions of the YOLO model: YOLOv8 [32] and YOLOv9 [33], choosing the smallest variants of each, YOLOv8n and YOLOv9t. We also include a comparison with a different type of detector, NanoDet [34], a one-stage anchor-free model, by evaluating two of its lightweight variants: NanoDet-G and NanoDet-Efficient. The comparative metrics, including the number of parameters, computational complexity (GFLOPs), mean average precision (mAP), and inference time, are summarized in Table 6. Our optimized model stands out by having the lowest number of parameters (1.4 million) and a moderate computational complexity (4.6 GFLOPs), placing it favorably compared to other architectures. This significant reduction in parameters is paired with a very low memory footprint (3.2 MB). By comparison, NanoDet-g, which has a similar number of parameters (3.8 million), uses much more memory (15.10 MB). This makes our model the most memory-efficient among all those compared, which strengthens its suitability for embedded environments or resource-constrained devices. In terms of detection quality, our model achieves a mAP50 of 95.9% on TrashNet, slightly below the original YOLOv11n's 96.4%, but close to YOLOv8n (95.8%) and YOLOv9t (95.6%), and better than both NanoDet variants.

For mAP50-95, our model remains competitive at 93.1%, a bit lower than YOLOv11n's 94.0%. On the Drinking Waste Detection dataset, precision is very high (mAP50 of 99.3%), comparable to the top models (YOLOv11n at 99.7%, YOLOv9t at 99.4%), confirming the robustness of our optimization despite the smaller model size. Our optimized model also benefits from very fast inference times, suited for real-time applications: 4.8 ms on TrashNet and only 3.1 ms on Drinking Waste Detection. These are the best times among all models presented, reflecting excellent efficiency without sacrificing detection quality, a key factor for deployment on embedded devices or edge computing platforms. This analysis shows that our optimized model successfully combines lightness, speed, and performance. With roughly half the parameters of the closest competing architectures and controlled complexity, it delivers minimal memory use while maintaining high accuracy and some of the fastest inference times.

Table 6: Performance Comparison of Optimized and State-of-the-Art Lightweight Detection Models.

Model	Parameters (M)	Complexity (GFLOPs)	Memory MB	TrashNet			Drinking waste		
				mAP50 %	mAP50-95 %	Inference time (ms)	mAP50 %	mAP50-95 %	Inference time (ms)
YOLOv8n	3.2	8.2	6	95.8	93.8	5.5	99.2	87.9	4.3
YOLOv9t	2.0	9.9	5	95.6	94.8	5.7	99.4	88.5	5.1
NanoDet-g	3.8	4.2	15	85.4	77.0	9.7	97.4	75.9	5.3
NanoDet-Lite0	3.1	1.7	12	90.4	83.1	8.3	98.7	78.3	5.8
YOLOv11n	2.6	6.4	5	96.4	94.0	6.2	99.7	85.3	5.0
Our model	1.4	4.6	3.2	95.9	93.1	4.8	99.3	83.4	3.1

Source: Authors, (2026).

IV.3 DEPLOYMENT OF OUR OPTIMIZED MODEL

In this subsection we evaluate our optimized YOLOv11 model across several formats (PyTorch, ONNX, NCNN). Key measurements included RAM and storage usage, maximum CPU temperature, energy consumption (relative to a powerbank of 20,000 mAh 5V/3A) and inference time per image (speed) all recorded on a Raspberry Pi 4B platform, as well as mAP50 and mAP50-95. The results of this evaluation are summarized in Table 7.

Table 7: Comparison of performance metrics across different model formats.

Model	Format	Storage MB	Precision	CPU temp (°C)	RAM (MB)	Energy %	TrashNet			Drinking waste		
							Speed (ms)	mAP50 %	mAP50-95 %	Speed (ms)	mAP50 %	mAP50-95 %
YOLOv11n	PyTorch	5.23	FP32	66.70	127.6	0.7440	1328.6	96.4	94.0	1019.5	99.7	85.3
	ONNX	10.1	FP16	82.77	150.7	0.4568	602.2	94.7	91.8	439.9	99.7	85.3
Optimized YOLOv11n	PyTorch	3.02	FP32	71.08	121.5	0.5795	1077.1	95.9	93.1	787.8	99.3	83.4
	ONNX	5.73	FP16	82.28	116.5	0.3702	329.8	94.8	90.4	345.9	99.1	83
	NCNN	3	FP16	73.03	89.01	0.2893	296.5	94.8	90.4	299.4	99.1	83

Source: Authors, (2026).

The model optimization led to a significant reduction in inference times. The improvements on the Raspberry Pi platform were noteworthy, with times decreasing from 1328.6 ms to 1077.1 ms in PyTorch format, and down to 296.5 ms in NCNN on TrashNet. For Drinking Waste Detection, inference times reduced from 1019.5 ms to 787.8 ms, and further to 299.4 ms in NCNN. Additionally, the optimized model's memory footprint decreased significantly, from 5.23 MB to 3.02 MB in PyTorch FP32. The NCNN format maintained this compact size at around 3 MB. Energy consumption also showed meaningful reductions: on the Raspberry Pi 4, the optimized model consumed 0.58 Wh versus 0.74 Wh for the original, with the lowest consumption recorded at 0.29 Wh using NCNN. These gains bode well for the battery life and efficiency of mobile and embedded systems. However, the optimization did cause a slight increase in CPU temperature, rising from 66.7 °C to between 71 °C and 73 °C depending on the format used. This increase was anticipated due to the higher computational load but remains within safe operational limits for sustained use on the Raspberry Pi. Finally, the model's accuracy, measured by mAP50 and mAP50-95, remained high despite the optimizations. On TrashNet, the optimized model in PyTorch format reached an mAP50 of 95.9%, slightly below the original's 96.4%. Conversion to other optimized formats stabilized accuracy around 94.8%. This minor performance drop is expected, resulting from the trade-offs made to speed up inference. In contrast, on the Drinking Waste Detection dataset, performance remained stable and excellent, with an mAP50 exceeding 99%, confirming the robustness of the optimized model in this context. Our YOLOv11 optimization balances performance, energy efficiency, and inference time, making it ideal for resource-limited devices like Raspberry Pi and enabling smart waste management applications.

IV.4 MODEL PERFORMANCE UNDER VARIOUS CONDITIONS

To perform real-time waste detection outside controlled settings, a detector must recognize objects across diverse visual conditions. We therefore evaluate our model training with trashnet on unseen smartphone images that differ markedly from TrashNet (which contains more controlled, uniform backgrounds). The selected images are part of a dataset available online [35].

Covering complex backgrounds, low or uneven illumination, specular highlights and transparency, non-rigid deformation, mild blur, and limited texture. As shown in Figure 7, the model reliably identifies the principal waste categories across these heterogeneous scenarios. Subfigures (a)–(b) illustrate cardboard detected under oblique viewpoints and background clutter, including a deformed shipping box. Subfigure (e) shows paper recognized under low light with tilt and motion blur, indicating tolerance to orientation and sharpness variations. In (d), a metal object is separated cleanly from a highly textured grassy background, demonstrating resilience to visual noise. For plastics, (c) presents a translucent, low-texture container, while (f) depicts a shiny printed package—both known to introduce challenges due to transparency and strong reflections. Taken together, these results indicate robustness and adaptability under changing illumination, texture, and scene complexity, while limiting background confusions. Importantly, the consistency of these predictions despite training on TrashNet’s controlled imagery highlights the model’s generalization capability beyond its training domain and its stable behavior in real-world conditions, a prerequisite for reliable, real-time waste detection.



Figure 7: Qualitative detections under diverse conditions.

Source: Authors, (2026).

V. CONCLUSIONS

In this study, we optimized the YOLOv11n model for efficient waste classification by reducing its size, parameter count, and computational complexity while preserving high detection accuracy. Our approach involved replacing standard convolutional layers with Ghost Conv layers, substituting heavier modules such as C2PSA and C3K2 with lighter alternatives, and applying knowledge distillation to recover any lost precision. Additionally, we fused Conv2D and BatchNorm layers and converted model weights from FP32 to FP16 for ONNX and NCNN to further boost performance. Empirical results demonstrate the effectiveness of our optimization strategy, with GPU inference latency reduced from 6.2 ms to 4.8 ms and mAP50 reaching 95.9%, closely approaching the original YOLOv11n performance of 96.4%. On the Raspberry Pi 4, latency decreased from 1,077.1 ms to 329 ms, with mAP50 maintained at 94% on the TrashNet dataset. Similar improvements were observed on the Drinking Waste dataset, where GPU latency decreased from 5 ms to 3.1 ms, and Raspberry Pi latency dropped from 1,019.5 ms to 299.4 ms, while maintaining mAP50 at 99%. Overall, the optimized model achieves a 43% reduction in parameter count and a 26% decrease in computational complexity, with consistent mAP performance across both datasets. The memory footprint was reduced from 5.23 MB to approximately 3 MB, and energy consumption on the Raspberry Pi was improved, decreasing from 0.74 Wh to 0.29 Wh. These optimizations make the YOLOv11n model more efficient and better suited for resource-constrained environments, easing deployment across various hardware platforms. This work is particularly important for waste management, where real-time and efficient waste detection is crucial for effective sorting and recycling. Our approach paves the way for the broader adoption of deep learning models for waste classification in real-world application.

VI. AUTHOR’S CONTRIBUTION

Conceptualization: Selma Azizi, Abderrahmane Bendahmane and Redouane Tlemsani.

Methodology: Selma Azizi, Abderrahmane Bendahmane and Redouane Tlemsani.

Investigation: Selma Azizi, Abderrahmane Bendahmane and Redouane Tlemsani.

Discussion of results: Selma Azizi, Abderrahmane Bendahmane and Redouane Tlemsani.

Writing – Original Draft: Selma Azizi, Abderrahmane Bendahmane and Redouane Tlemsani.

Writing – Review and Editing: Selma Azizi, Abderrahmane Bendahmane and Redouane Tlemsani.

Resources: Selma Azizi, Abderrahmane Bendahmane and Redouane Tlemsani.

Supervision: Selma Azizi, Abderrahmane Bendahmane and Redouane Tlemsani.

Approval of the final text: Selma Azizi, Abderrahmane Bendahmane and Redouane Tlemsani.

VII. REFERENCES

- [1] UNEP, Global Waste Management Outlook 2024 : Beyond An Age Of Waste. 2024. [Online]. Available: <https://wedocs.unep.org/20.500.11822/44939>.
- [2] M. Cathelineau, Recycling, a challenge for the circular economy. ISTE Group, 2023. [Online]. Available: <https://www.istegroup.com/en/produit/le-recyclage-enjeu-pour-leconomie-circulaire/>.
- [3] A. Sharma, V. Kumar, and L. Longchamps, “Comparative performance of YOLOv8, YOLOv9, YOLOv10, YOLOv11 and Faster R-CNN models for detection of multiple weed species,” *Smart Agric. Technol.*, vol. 9, no. November, p. 100648, 2024, doi: 10.1016/j.jatech.2024.100648.
- [4] R. Khanam and M. Hussain, “YOLOv11: An Overview of the Key Architectural Enhancements,” vol. 2024, pp. 1–9, 2024, [Online]. Available: <http://arxiv.org/abs/2410.17725>.
- [5] 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. Extr.*, vol. 5, no. 4, pp. 1680–1716, 2023, doi: 10.3390/make5040083.
- [6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.
- [7] W. L. Mao, W. C. Chen, H. I. K. Fathurrahman, and Y. H. Lin, “Deep learning networks for real-time regional domestic waste detection,” *J. Clean. Prod.*, vol. 344, no. October 2021, p. 131096, 2022, doi: 10.1016/j.jclepro.2022.131096.
- [8] D. Ziouzos, N. Baras, V. Balafas, M. Dasygenis, and A. Stimoniariis, “Intelligent and Real-Time Detection and Classification Algorithm for Recycled Materials Using Convolutional Neural Networks,” *Recycling*, vol. 7, no. 1, Feb. 2022, doi: 10.3390/recycling7010009.
- [9] G. Yang et al., “Garbage Classification System with YOLOV5 Based on Image Recognition,” 2021 6th Int. Conf. Signal Image Process. ICSIP 2021, pp. 11–18, 2021, doi: 10.1109/ICSIP52628.2021.9688725.
- [10] Z. Lun, Y. Pan, S. Wang, Z. Abbas, M. S. Islam, and S. Yin, “Skip-YOLO: Domestic Garbage Detection Using Deep Learning Method in Complex Multi-scenes,” *Int. J. Comput. Intell. Syst.*, vol. 16, no. 1, pp. 1–16, 2023, doi: 10.1007/s44196-023-00314-6.
- [11] A. Wahyutama and M. Hwang, “YOLO-Based Object Detection for Separate Collection of Recyclables and Capacity Monitoring of Trash Bins,” *Electron.*, vol. 11, no. 9, 2022, doi: 10.3390/electronics11091323.
- [12] Y. Ren, Y. Li, and X. Gao, “An MRS-YOLO Model for High-Precision Waste Detection and Classification,” *Sensors*, vol. 24, no. 13, 2024, doi: 10.3390/s24134339.
- [13] Y. Yang, Y. Li, and M. Tao, “FE-YOLO: A Lightweight Model for Construction Waste Detection Based on Improved YOLOv8 Model,” *Buildings*, vol. 14, no. 9, p. 2672, 2024, doi: 10.3390/buildings14092672.
- [14] H. Wang, L. Wang, H. Chen, X. Li, X. Zhang, and Y. Zhou, “Waste-YOLO: Towards High Accuracy Real-time Abnormal Waste Detection in Waste-to-Energy Power Plant for Production Safety,” *Meas. Sci. Technol.*, 2023.
- [15] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, “Pruning and quantization for deep neural network acceleration: A survey,” *Neurocomputing*, vol. 461, pp. 370–403, 2021, doi: 10.1016/j.neucom.2021.07.045.
- [16] F. Xu, L. Huang, X. Gao, T. Yu, and L. Zhang, “Research on YOLOv3 model compression strategy for UAV deployment,” *Cogn. Robot.*, vol. 4, no. June 2023, pp. 8–18, 2024, doi: 10.1016/j.cogr.2023.11.001.
- [17] M. Wang et al., “Q-YOLO: Efficient Inference for Real-Time Object Detection,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 14408 LNCS, pp. 307–321, 2023, doi: 10.1007/978-3-031-47665-5_25.
- [18] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” pp. 1–9, 2015, [Online]. Available: <http://arxiv.org/abs/1503.02531>
- [19] M. Aubard, L. Antal, A. Madureira, and E. Ábrahám, “Knowledge Distillation in YOLOX-ViT for Side-Scan Sonar Object Detection,” 2024, [Online]. Available: <http://arxiv.org/abs/2403.09313>
- [20] G. Thung, “Trashnet,” 2016. <https://github.com/garythung/trashnet> (accessed Feb. 05, 2025).
- [21] “TRASHNET,” 2024. <https://universe.roboflow.com/test-4fd7k/trashnet-fg9gt/dataset/1> (accessed Jan. 20, 2023).
- [22] Arkadiy Serezhkin, “Drinking Waste Classification.” <https://www.kaggle.com/datasets/arkadiyhacks/drinking-waste-classification> (accessed May 20, 2025).
- [23] S. Majchrowska et al., “Deep learning-based waste detection in natural and urban environments,” *Waste Manag.*, vol. 138, no. December 2021, pp. 274–284, 2022, doi: 10.1016/j.wasman.2021.12.001.
- [24] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, “GhostNet: More features from cheap operations,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 1577–1586, 2020, doi: 10.1109/CVPR42600.2020.00165.
- [25] P. Hidayatullah, N. Syakrani, M. R. Sholahuddin, T. Gelar, R. Tubagus, and P. N. Bandung, “YOLOv8 to YOLO11 : A Comprehensive Architecture In-depth Comparative Review,” 2025.
- [26] S. Li and Z. Wu, “Intelligent Systems with Applications Learning how to transfer : A lifelong domain knowledge distillation framework for continual MRC,” *Intell. Syst. with Appl.*, vol. 26, no. January, p. 200497, 2025, doi: 10.1016/j.iswa.2025.200497.
- [27] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge Distillation: A Survey,” *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, 2021, doi: 10.1007/s11263-021-01453-z.
- [28] “NVIDIA Corporation, Nvidia tensorrt.” <https://github.com/NVIDIA/TensorRT> (accessed Mar. 22, 2025).
- [29] “ONNX Contributors, Open neural network exchange.” <https://github.com/onnx/> (accessed Mar. 22, 2025).

- [30] Ultralytics, “How to Export to NCNN from YOLO11 for Smooth Deployment.” <https://docs.ultralytics.com/fr/integrations/ncnn/> (accessed Apr. 16, 2025).
- [31] Ultralytics, “ONNX Exportation for YOLO11 Models.” <https://docs.ultralytics.com/fr/integrations/onnx/> (accessed Mar. 22, 2025).
- [32] D. Reis, J. Kupec, J. Hong, and A. Daoudi, “Real-Time Flying Object Detection with YOLOv8,” 2023, [Online]. Available: <http://arxiv.org/abs/2305.09972>
- [33] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, “YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information,” 2024, [Online]. Available: <http://arxiv.org/abs/2402.13616>
- [34] RangiLyu, “NanoDet-Plus: Super fast and high accuracy lightweight anchor-free object detection model,” 2021.
- [35] DataCluster Labs, “Household Trash.” <https://www.kaggle.com/datasets/dataclusterlabs/household-trash> (accessed Jun. 01, 2025).