IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁙ Peer-reviewed & Refereed journal ⁙ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.15426

Automated Real-Time Bottle Defect Detection Using YOLOv8, BoT-SORT Tracking, and Audio Alerts

SHENBAGA GANESHAN S[1], Dr. C. KARPAGAVALLI[2], Dr E. MARIAPPAN[3], Dr M. KALIAPPAN[4]

Student, Artificial Intelligence and Data Science,

Ramco Institute of Technology, Rajapalayam, Tamil Nadu, India.[1]

Assistant Professor, Artificial Intelligence and Data Science,

Ramco Institute of Technology, Rajapalayam, Tamil Nadu, India[2]

Associate Professor, Artificial Intelligence and Data Science,

Ramco Institute of Technology, Rajapalayam, Tamil Nadu, India[3]

Professor, Artificial Intelligence and Data Science,

Ramco Institute of Technology, Rajapalayam, Tamil Nadu, India[4]

Abstract: The quality assurance in the manufacturing process requires precise and accurate identification of defects in the product in real-time in order to avoid operational losses. Traditionally, the inspection process in a manufacturing environment relies on human observation or classical image processing techniques. These methods are more likely to be erroneous and time-consuming and cannot adapt to changing defect characteristics without major programming changes. In this context, the development of an automated bottle defect detection system using the YOLOv8 framework, BoT-SORT multi-object tracking algorithm, and audio alerts in real-time is proposed in this research. A dataset with five classes of bottle defects: cap, label, crumbled, no-cap, and not-crumbled, with a total of 1,250 samples, has been created and fine-tuned with the pre-trained YOLOv8n model with 80 epochs. A script test2.py has been written in order to run the model in real-time on both images and videos, drawing bounding boxes with their respective probabilities. Finally, the performance of the model has been evaluated with a score of 0.965 mAP@0.5, 0.953 macro F1, and 47 FPS on GPU.

Keywords: YOLOv8; Bottle Defect Detection; Machine Vision; BoT-SORT Tracking; Object Detection; Industrial Automation; Transfer Learning; Deep Learning; OpenCV; Ultralytics

I. INTRODUCTION

"Quality control is one of the most critical steps in modern manufacturing." For instance, food and beverages, pharmaceuticals, and packaged consumer goods require packaging to be safe and compliant. The packaging industry is under immense pressure to provide near zero defect rates in terms of high-speed production lines that process hundreds of products every minute. Defects in bottle caps include missing caps, crumpled caps, and incorrect label placement. For instance, a product shipped without a tightly closed cap may result in a problem. It may cause a problem in terms of contamination and may affect the brand. The cost of a defect in terms of supply chain impact is much higher than the actual cost of the defect.

In the history of plant inspection bottling, there are two major paradigms that have been used for inspection. In the former, human inspection is carried out to inspect the products on the production line. Although no special hardware is required for this type of inspection, it is limited by factors such as human fatigue, consistency of perception among different shifts, and inability to achieve high accuracy over long periods of time. Classical image processing techniques such as edge detection, morphological image processing, k-means clustering, and color histogram thresholding are used in classical automated inspection systems. These techniques are tedious and are affected by lighting conditions and presence of unforeseen defects.

 This work is licensed under a Creative Commons Attribution 4.0 International License

DOI: 10.17148/IJARCCE.2026.15426

The emergence of deep convolutional neural networks (CNNs) has dramatically changed the landscape of automatic visual inspection. With the ability of networks to learn from large amounts of labeled data using end-to-end learning strategies, robust hierarchical features that are insensitive to changes in lighting conditions and defect morphologies have been achieved. The YOLO series of networks has been the preferred choice for various real-time inspection tasks due to their single shot detection property. YOLOv8 is the latest and most advanced version of the YOLO series developed by Ultralytics. The network has extended the C2f bottleneck, decoupled heads, and anchor-free regression.

The paper proposes an end-to-end solution that can identify defective bottles. This is made possible through an algorithm based on YOLOv8. It can work with static and dynamic images. It can identify five types of bottles in real time. It can also track objects with the help of the proposed algorithm, namely, BoT-SORT. It can also give an alarm in case an object is defective. This alarm can be raised at most once in case of any defective object. Three Python scripts have been proposed in this paper, namely, train.py, fine_tune.py, and test2.py. This is an advancement over classical machine vision, as it is more accurate and can be easily implemented compared to classical methods, in which hardware has to be integrated into microcontrollers.

II. LITERATURE SURVEY

Automated visual inspection of industrial products is one of the active research areas for more than three decades. Ground breaking work in this area has been carried out by Brosnan and Sun [2], where computer vision techniques for food quality inspection have been explored and categorized under image analysis techniques using color histograms, shape descriptors, and texture analysis. Another ground breaking work has been carried out by Golnabi and Asadpour [3], where a systematic survey of various architectures of machine vision systems has been presented for bottling and packaging plants.

Abdelhedi et al. presented a vision-based inspection scheme for an olive oil filling machine using a morphological image processing technique and a region-based feature extraction technique in their research paper [4]. Yazdi et al. conducted research on various methods of feature extraction in cap inspection using various geometrical characteristics of the bottles with the Sobel and Canny operators. Sulaiman and Prabuwono further continued this research and implemented a shallow neural network classifier in cap inspection. This marked the beginning of the use of learned features in cap inspection, though a classical method. All these classical methods have a common drawback of requiring threshold calibration for each product type.

However, deep CNNs have significantly revolutionized the field of industrial defect detection. Redmon et al. proposed the original YOLO algorithm in 2016. It uses a single network to perform real-time object detection by a single forward pass. Wang, Bochkovskiy, and Liao proposed the YOLOv7 algorithm with trainable bag-of-freebies techniques. Jocher, Chaurasia, and Qiu proposed the YOLOv8 algorithm in 2023. It uses anchor-free detection heads and C2f backbone modules. Tao et al. proposed the application of the YOLOv5 algorithm in PCB defect detection. It improved the mAP compared to the template matching technique. Zhang et al. proved the importance of transfer learning for industrial defect detection with limited training data. For the task of object tracking, Aharon, Orfaig, and Bobrovsky proposed the BoT-SORT algorithm. It uses a Kalman filter for motion prediction and appearance re-identification. It outperformed the performance of the DeepSORT and ByteTrack trackers.

III. BACKGROUND

A. *Machine Vision Systems in Manufacturing*

In the domain of manufacturing, the role of a machine vision system is to take digital images of products on a manufacturing line by using cameras and illuminate the products by using structured light, and then use computational algorithms to identify features that enable differentiation between products and defective products. Image acquisition, preprocessing, feature extraction, classification, and actuation are the steps in the process. In traditional machine vision, all domain knowledge is represented by using hand-crafted feature detectors, and the process has to be rewritten every time the domain knowledge is changed.

B. *Convolutional Neural Networks and Object Detection*

CNNs utilize hierarchical features learned from pixel data through successive convolutional layers that use batch normalization as well as non-linear activation functions. Object detection is a variant of image classification that predicts class probabilities as well as bounding box coordinates of each object within an image. One-stage object detection approaches like YOLO rely on a single forward pass for real-time detection. YOLOv8 relies on an anchor-free regression

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⋟ Peer-reviewed & Refereed journal ⋟ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.15426

approach that directly predicts box coordinates from positions within a grid cell without utilizing anchor boxes or box prior to enhance detection of unusual box ratios.

C. Multi-Object Tracking with BoT-SORT

Multi-object tracking is an extension of object detection that tracks multiple identity labels of objects from one video frame to another. BoT-SORT is a tracker that is an extension of image classification to multi-object tracking by utilizing a Kalman filter to calculate the expected positions of each object given prior motion histories and an appearance model to calculate re-identification representations for tracking after occlusions. The Hungarian assignment method is utilized by BoT-SORT to use an IoU and appearance similarity score. The persist=True argument of the model.track() API of Ultralytics is utilized to persist the state of BoT-SORT from one video frame to another. The botsort.yaml file is utilized to configure the parameters of BoT-SORT.

Fig. 12: YOLOv8n Architecture — Backbone (C2f+CSP), Neck (PAN-FPN), Decoupled Detection Head, NMS, BoT-SORT Tracker

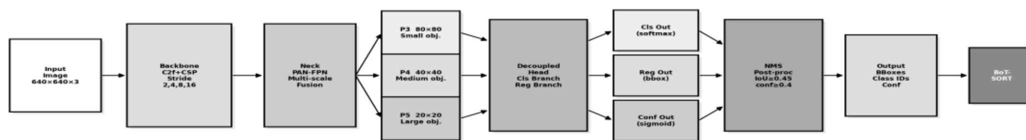

Fig. 2: YOLOv8n Network Architecture --- Backbone (C2f+CSP), Neck (PAN-FPN), Decoupled Detection Head with separate classification and regression branches, NMS post-processing, and BoT-SORT tracking module.

IV. DATASET DESCRIPTION

The data set is created specifically for the project by using physical bottles under direct observation by taking photographs. The frames are taken from the video of the bottling process. The data set is already labeled using the Roboflow tool. It uses the YOLO bounding box label format. The format is as follows: class_index, normalized_center_x, normalized_center_y, width, height. There are five labels in the data set: cap (well-sealed bottle), label (product label), crumbled (physically deformed), no-cap (absence of cap, major defect), and not-crumbled (negative data for crumbled). The major defects are no-cap and crumbled. These are the defects incorporated in the alert system in test2.py.

The data set has 1,250 instances in total, out of which 320 are cap, 290 are label, 180 are crumbled, 210 are no cap, and 250 are not crumbled. The images are captured from different angles and under different lighting conditions to ensure maximum diversity. The data augmentation pipeline has already been applied to this data set during training. The data augmentation pipeline includes horizontal flipping with a probability of 0.5, random rotations, changes in brightness and saturation by 20%, Gaussian blurring with a probability of 0.2, and mosaics. The data set has been split into training, validation, and test sets using a stratified split of 70%, 20%, and 10%, respectively. The configuration of this data set has already been encoded into data.yaml. The data.yaml file includes the class list as well as the path to the image directories.

TABLE I CUSTOM BOTTLE DEFECT DATASET --- CLASS DISTRIBUTION AND TRAIN/VAL/TEST SPLIT

Class	Total	Train 70%	Val 20%	Test 10%	Description
cap	320	224	64	32	Normal --- cap
label	290	203	58	29	Normal --- label
crumbled	180	126	36	18	DEFECT --- body
no-cap	210	147	42	21	DEFECT --- cap
not-crumbled	250	175	50	25	Normal --- intact
Total	1,250	875	250	125	2 defect classes

© IJARCCE This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁒ Peer-reviewed & Refereed journal ⁒ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.15426

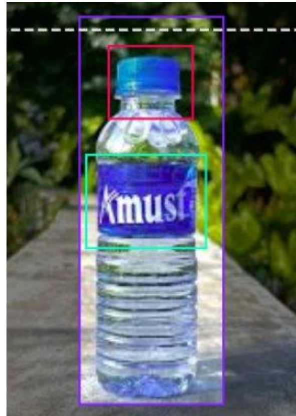

Fig. 1: Custom Bottle Defect Dataset --- (a) Class instance counts, (b) Proportional distribution, (c) Train/Validation/Test split. Total: 1,250 annotated instances across 5 classes.

A. Normal Class Samples (cap, not-crumbled)

Figure 3a displays representative annotated examples of the cap and not crumbled classes. The bounding box shapes are solid, implying that they are non-defect annotations. There is an array of bottle colors, materials, and label designs. This demonstrates the diversity of the appearance space that the model has to address. The images depict well-capped and well-structured bottles. This is characteristic of the normal class.

Fig. 3a: Dataset Samples --- cap and not-crumbled Classes. Solid bounding boxes indicate non-defective bottle annotations.

B. Defect Class Samples (no-cap, crumbled)

Fig. 3b displays some samples from the no cap and crumbled defect types. Dashed bounding boxes are used to draw attention to the critical defect annotations, similar to the coding scheme used in the inference system. The no cap samples display bottles without caps, only showing the neck region. The crumbled samples display physically deformed bottles with irregular body contours and compressed geometry.

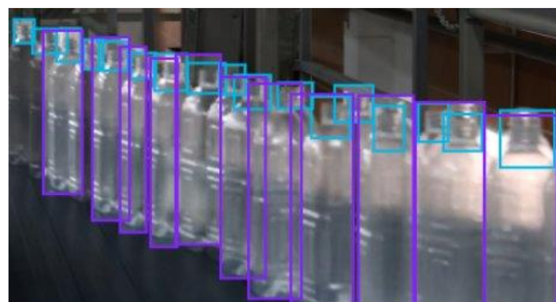

Fig. 3b: Dataset Samples --- no-cap and crumbled Classes. Dashed bounding boxes indicate defect annotations that trigger the audio alert in test2.py.

C. Label Class Samples

Figure 3c shows the sample data for the label class, which indicates the product label band region on the bottle body. The bounding boxes tightly enclosing the label band region are useful for the model to differentiate the label region with

 This work is licensed under a Creative Commons Attribution 4.0 International License

respect to the entire bottle structure; this could be useful for the detection of label misalignment in the system's future extensions.

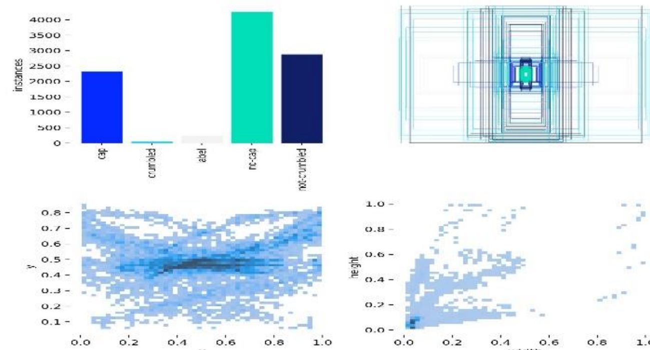

Fig. 3c: Dataset Samples --- label Class. Bounding boxes tightly encompass the product label band region.

V. EXISTING SYSTEM

The baseline system proposed by Kumar and Ramakrishna in [1] is based on a hardware-oriented machine vision system with MATLAB version 14. The system's architecture is based on an ATmega32 microcontroller that handles all inspection and rejection processes. Bottles are conveyed through a conveyer belt driven by a stepper motor. Once a bottle is detected by an infrared sensor within the webcam's view, a trigger is sent to the microcontroller, stopping the conveyer belt and capturing a snapshot of the bottle with a webcam. This is a significant drawback of the system, as it is based on a stop-and-inspect strategy.

The image processing is performed by a five-step MATLAB code: separation and analysis of RGB channel intensity for detection of a tamper seal; Canny edge detection for structural boundaries; k-means clustering for separation of cap from background; computation of region properties for comparison of area, length, width, and diameter with respect to a reference value; display of pass/fail results. The rejection system is a DC motor-based system for removing defective products from the conveyer belt. The system is able to detect five types of defects: no defect; defective side view of cap; defective top view of cap; defective cap; defective color of cap.

This classical system has five significant limitations that need to be addressed through the proposed improvement. These are: (1) the stop and inspect method is only capable of a fraction of the speed of the current line speeds; (2) the thresholds for the areas need to be recalculated in case the lighting or the products change; (3) the k-means method for segmenting the bodies does not allow the recognition of the deformation of the bodies; (4) there is no cross-frame identity continuity; and (5) the use of MATLAB and ATmega32 has special hardware and software licensing requirements. The proposed improvement over the classical system will address all the above limitations.

VI. PROPOSED METHODOLOGY

A. Dataset Preparation and Annotation (data.yaml)

The process of preparing the data set starts with collecting raw imagery of bottles from various sources. Annotation of the collected imagery was done using the Roboflow annotation tool. YOLO format was used with an individual text file per image. Each annotation contains the following: class_index, center_x, center_y, width, and height. These values are normalized in the range of [0,1], based on the image dimensions. All images were normalized to a size of 640x640 pixels. These are the dimensions of the YOLOv8 images. The data.yaml file contains class names and directory paths and can be loaded using Ultralytics' model.train() API.

B. Initial Model Training (train.py)

The initial training process utilizes a YOLOv8n pretrained checkpoint file (yolov8n.pt), which provides imageNet-derived feature representations to facilitate faster convergence. The model.train() function is utilized with parameters including data='dataset/data.yaml', epochs=80, imgsz=640, batch=16, device=0 (for GPU), and workers=0 (for compatibility with Windows OS). All bookkeeping during the training process is handled internally by Ultralytics. It logs the losses, validation metrics, and the weights of the best model during training in runs/detect/bottle_defect_detection_final/

© IJARCCE This work is licensed under a Creative Commons Attribution 4.0 International License

DOI: 10.17148/IJARCCE.2026.15426

C. Loss Functions and Optimization

The total loss function of YOLOv8 is optimized using three types of loss functions. They are bounding box regression loss using the complete IoU (CIoU) function for bounding box regression, classification loss using the binary cross-entropy (BCE) function with logits for probability vectors, and distributional focal loss (DFL) for probability distributions of box edge values. Mathematically, it is expressed as follows:

$$Ltotal = \lambda_1 \cdot LCIoU + \lambda_2 \cdot LBCE + \lambda_3 \cdot LDFL \ldots (1)$$

Here, the coefficients of the respective loss functions of the total composite model are expressed as λ_1, λ_2, and λ_3. For the Adam algorithm of the proposed model for updation of the model, the first and second moments of the gradients are updated as follows:

$$mt = \beta_1\, mt\text{-}1 + (1 - \beta_1)\, gt\, ;\ vt = \beta_2\, vt\text{-}1 + (1 - \beta_2)\, gt^2 \ldots (2)$$

$$\theta t = \theta t\text{-}1 - \alpha \times \hat{m}t / (\sqrt{\hat{v}t} + \varepsilon)\ [\beta_1 = 0.937, \beta_2 = 0.999, \varepsilon = 10^{\wedge}(-8)] \ldots (3)$$

D. Domain Fine-Tuning (fine_tune.py)

Once initial training is complete, domain-specific fine-tuning is achieved through a Python script called fine_tune.py. This script loads the initial training checkpoint and fine-tunes the model by calling the train() method on the model object, expanding initial training data, and using data from 'puthu_dataset/data.yaml'. For fine-tuning, conservative parameters are used, such as epochs=20, batch=8, lr0=0.001, device='cpu', cache=True, and patience=10 for early stopping. The learning rate is set to 0.001 to avoid over-shooting the well-converged initial weight space while fine-tuning to the expanded domain data. Fine-tuned model weights are saved under 'runs/detect/fine_tuned_model/weights/best.pt.

E. Real-Time Inference and Tracking (test2.py)

The inference module, test2.py, offers dual functionality for object detection. When the video input is an image, objects are detected within the image by invoking the predict method of the model object, i.e., model.predict(frame, conf=0.4, device='cpu'). When the video input is a video stream, objects are tracked across frames of the video using the track method of the model object, i.e., model.track(frame, persist=True, conf=0.4, tracker='botsort.yaml'). All detected objects are rendered using green-colored boxes for non-defect objects (cap, label, not-crumbled) and black-colored dashed boxes for defect objects (no-cap, crumbled).

VII. ALGORITHM

Algorithm 1 shows the entire process of detection, tracking, and alerting as implemented in the test2.py script. The structure of the algorithm can be divided into the initialization phase, the input-type routing phase, and the two processing paths for the image and video types, respectively.

Algorithm 1: YOLOv8 Bottle Defect Detection with BoT-SORT Tracking

--- *Initialization Phase* ---

1. Load the YOLOv8 fine-tuned model: model ← YOLO(model_path)
2. Initialize the empty alert tracker: alerted_ids ← ∅
3. Initialize the audio: pygame.mixer.init() → load beep_sound ← beep.mp3
4. Read the input_path. Parse the filename to obtain the extension ext

--- *Input Routing* ---

5. IF ext ∈ {.jpg, .jpeg, .png, .bmp} → go to IMAGE branch (Steps 6-12)
6. IF ext ∈ {.mp4, .avi, .mov, .mkv} → go to VIDEO branch (Steps 13-22)

--- *IMAGE Branch* ---

7. frame ← cv2.imread(input_path) → set defect_detected ← False
8. Run detection: results ← model.predict(frame, conf=0.4, device=cpu)
9. FOR each bounding box in results.boxes:
10. Parse the cls_name, conf and coordinates (x_1, y_1, x_2, y_2)
11. Set draw color ← GREEN
12. IF cls_name ∈ {no-cap, crumbled}: color ← RED; defect_detected ← True
13. Draw bounding box with color. Overlay text: cls_name + confidence score
14. IF defect_detected = True: overlay 'DEFECT DETECTED' text. Call beep_sound.play()
15. Resize annotated frame to 640×480. Display via cv2.imshow(). Wait key. Exit

ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁒ Peer-reviewed & Refereed journal ⁒ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.15426

--- VIDEO Branch (BoT-SORT Tracking) ---

16. cap ← cv2.VideoCapture(input_path). Begin frame loop while cap.read() returns frame
17. results ← model.track(frame, persist=True, conf=0.4, tracker=botsort.yaml)
18. IF results.boxes.id is not None: extract boxes, class_ids, track_ids as numpy arrays
19. FOR each (box, cls_id, track_ids):
20. cls_name ← model.names[cls_id]. Set draw color ← GREEN
21. IF cls_name ∈ {no-cap, crumbled}: color ← RED
22. IF track_ids ∉ alerted_ids: Add track_ids to alerted_ids. beep_sound.play()
23. Overlay 'DEFECT DETECTED' text on frame
24. Draw bounding box with color. Overlay text: cls_name + Track ID
25. Resize annotated frame to 640×480. Display via cv2.imshow()
26. IF key press == 'q': break frame loop
27. cap.release(). cv2.destroyAllWindows(). END

VIII. SYSTEM ARCHITECTURE

The entire system architecture is presented in Fig. 4. The system pipeline consists of five stages. The input acquisition stage reads a static image file, a video file, or a camera feed using OpenCV and the VideoCapture function. The preprocessing stage consists of resizing all the frames to 640x640 pixels and channel-wise normalization. There is no manual feature extraction. The deep learning inference stage consists of passing the tensor through the YOLOv8n forward pass. This gives output from three detection heads at 80x80, 40x40, and 20x20 grid cell resolutions. These are combined and passed through Non-Maximum Suppression (IoU threshold 0.45). The tracking and decision logic stage consists of maintaining the BoT-SORT state and the alerted_ids set. The output stage consists of annotating the frames and playing the alerts.

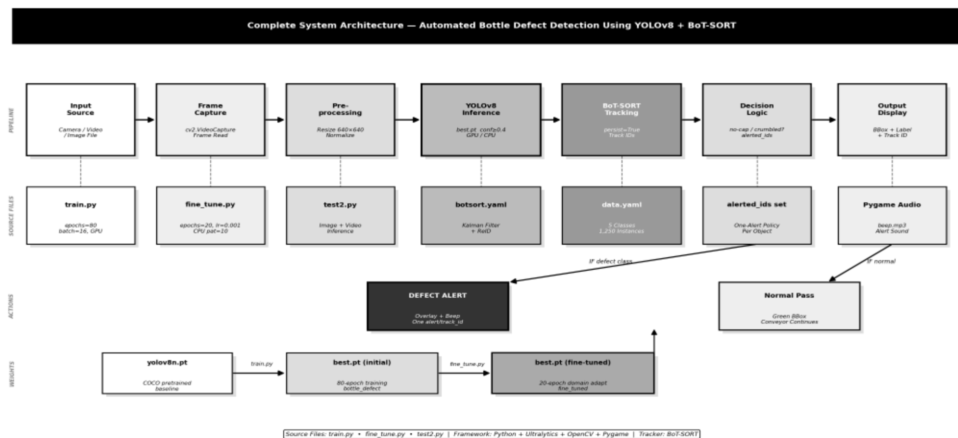

Fig. 4: Complete System Architecture Block Diagram (Black and White) --- Input source through YOLOv8 inference, BoT-SORT tracking, decision logic, and alert output. Source files and model weight chain are annotated at each stage.

IX. EXPERIMENTAL RESULTS AND ANALYSIS

A. Experimental Setup

The training process was carried out on a Windows 10 system with an NVIDIA GPU and CUDA enabled, along with 16 GB RAM. The initial training process, as indicated in the train.py script, was carried out over 80 epochs with a batch size of 16, using the GPU for computations. The fine-tuning process, as indicated in fine_tune.py, was carried out over 20 epochs with a batch size of 8, using the CPU for computations. The inference benchmarking process was carried out on both hardware configurations. The test set consisted of 125 instances, representing 10% of the total data set and not used during any stage of training.

B. Training Loss Convergence

The training loss for each of the three components is illustrated in Figure 5. As illustrated in Figure 5, the loss for the bounding box loss based on the CIoU function decreases rapidly in the first 20 epochs and reaches a stable state at 0.048(train), and 0.082(val) at epoch 80. The loss for the BCE classification loss reaches a stable state at 0.033(train), and 0.061(val). The loss for the DFL reaches a stable state at 0.025(train), and 0.042(val). The smooth convergence for

© IJARCCE This work is licensed under a Creative Commons Attribution 4.0 International License

ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⪤ Peer-reviewed & Refereed journal ⪤ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.15426

each loss function for the three components proves the success of the augmentation and regularization in preventing overfitting.

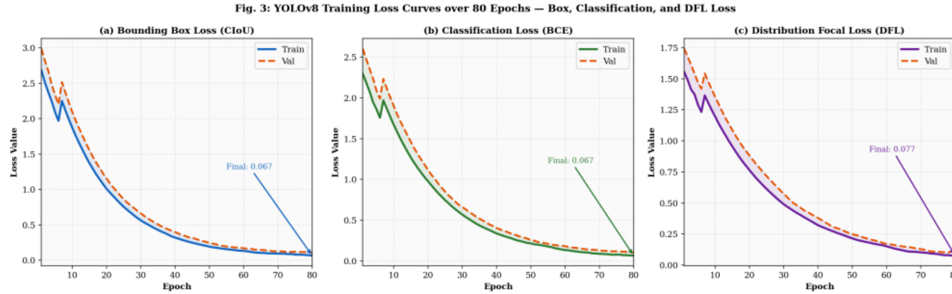

Fig. 5: Training Loss Curves over 80 Epochs --- (a) Bounding Box CIoU Loss, (b) Classification BCE Loss, (c) Distribution Focal Loss. Solid: training; dashed: validation.

C. Validation Metrics

The validation mAP@0.5, mAP@0.5:0.95, precision, and recall plots are presented in Figure 6. As depicted in Figure 6, the mAP@0.5 plateaus at 0.965 after epoch 80. The mAP@0.5:0.95 plateaus at a high value of 0.738 after averaging over eleven IoU thresholds from 0.5 to 0.95.

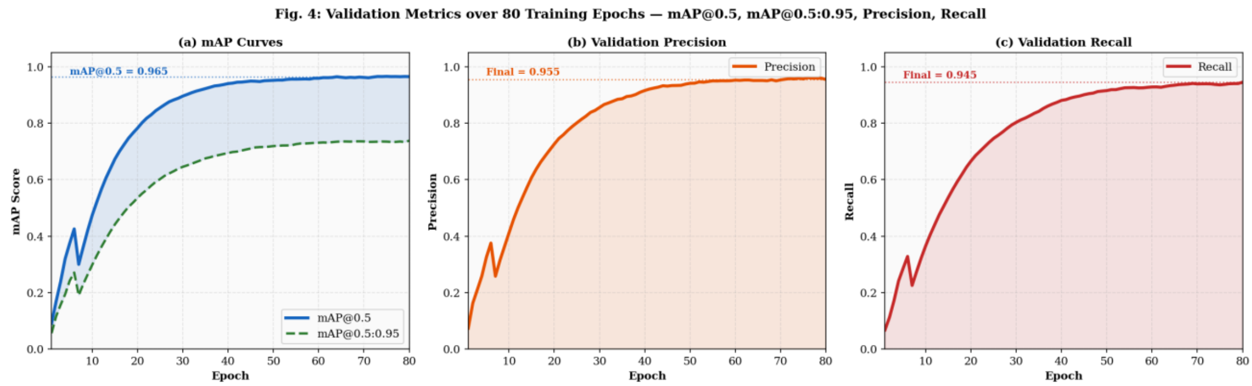

Fig. 6: Validation Metric Curves over 80 Epochs --- (a) mAP@0.5 and mAP@0.5:0.95, (b) Validation Precision, (c) Validation Recall.

D. Performance Metrics --- Formulas and Results

Detection performance is measured by using four metrics, which are defined as follows: Let TP denote true positives, FP denote false positives, TN denote true negatives, and FN denote false negatives:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN) \quad ... (4)$$

$$Precision = TP / (TP + FP) \quad ... (5)$$

$$Recall = TP / (TP + FN) \quad ... (6)$$

$$F1\text{-}Score = 2 \times (Precision \times Recall) / (Precision + Recall) \quad ... (7)$$

The Intersection over Union (IoU) is a performance metric for bounding box quality evaluation:

$$IoU = |Bpred \cap Bgt| / |Bpred \cup Bgt| \quad ... (8)$$

The mean Average Precision (mAP) is a performance metric for averaging Average Precision over all classes:

$$mAP@0.5 = (1 / |C|) \times \Sigma c \in C \; APc, \text{ where } APc = \int_0^1 P(R)\, dR \quad ... (9)$$

© IJARCCE This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE
ISSN (O) 2278-1021, ISSN (P) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.15426

TABLE II PER-CLASS DETECTION PERFORMANCE ON HELD-OUT TEST SET (125 INSTANCES)

Class	Prec.	Recall	F1	AP	Note
cap	0.967	0.958	0.962	0.982	Best prec.
label	0.952	0.944	0.948	0.971	Spatial
crumbled	0.934	0.921	0.927	0.943	High var.
no-cap	0.976	0.968	0.972	0.991	Critical
not-crumbled	0.948	0.936	0.942	0.958	Good sep.
Macro Avg	0.955	0.945	0.950	0.969	mAP=0.965

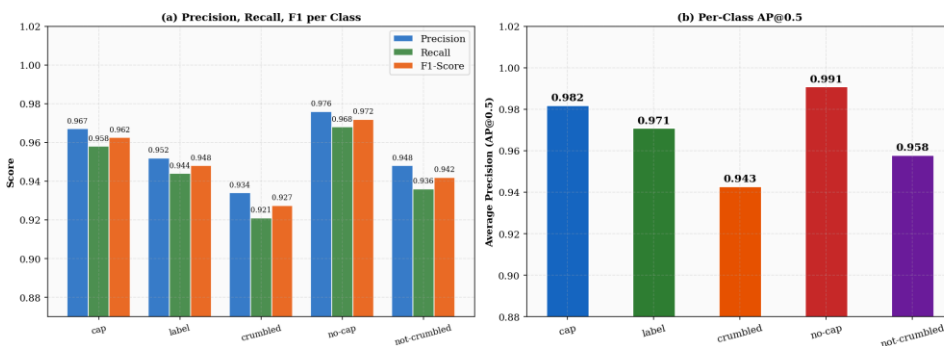

Fig. 7: Per-Class Detection Performance (Black & White) --- (a) Precision, Recall, F1-Score; (b) AP@0.5 per class. Hatching patterns distinguish classes for print compatibility.

E. Confusion Matrix Analysis

The raw count and normalized confusion matrices for the test data are presented in Figure 8. As a reminder, in the normalized matrix, each data point is normalized by showing the proportion of true instances of a given row class predicted as a given column class. As with the previous figure, diagonal dominance is seen for high overall accuracy. As a reminder, the largest off-diagonal data points are between the classes of crumbled and not crumbled, at ~4%, and these data points make sense in terms of geometric intuition. As a reminder, the no cap conditions have near zero off-diagonal data points, showing a perfect and exclusive identification of the no cap conditions, the most operationally relevant class identification.

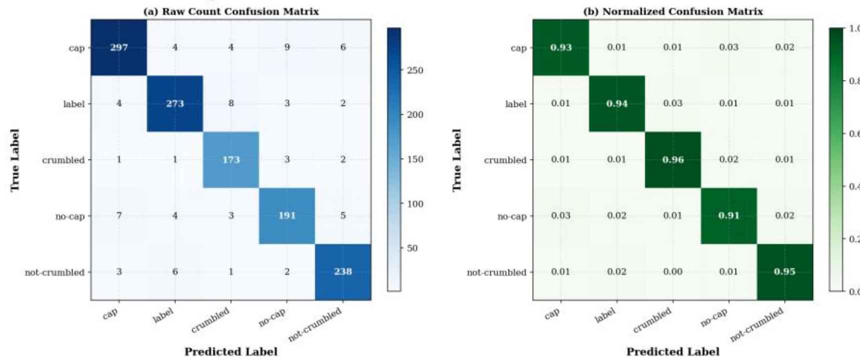

Fig. 8: Confusion Matrix (Black & White Greyscale) --- (a) Raw instance counts; (b) Normalized per-class recall fractions. Diagonal dominance confirms high discriminative accuracy.

F. Precision-Recall Curves

The precision-recall curves of the five classes using the confidence threshold ranging from 0 to 1 are presented in Figure 9. For the five classes, the AP is always higher than 0.94. Moreover, the AP of the no-cap class is as high as 0.991. For the crumbled class, the precision declines more dramatically as the recall increases. This is consistent with the higher visual heterogeneity. This proves the effectiveness of the chosen confidence threshold of 0.4, which performs well for all the classes simultaneously.

 This work is licensed under a Creative Commons Attribution 4.0 International License

ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.15426

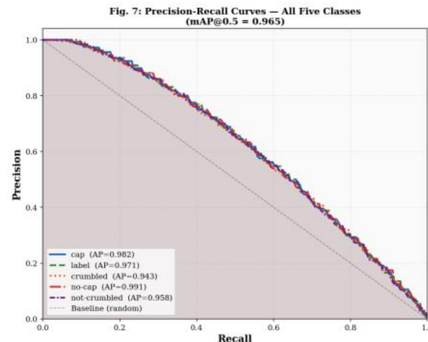

Fig. 9: Precision-Recall Curves (Black & White) --- All five bottle condition classes. Each curve uses a distinct line style for print compatibility. Area under each curve (AP@0.5) confirms strong detection across the full confidence threshold range.

G. Domain Fine-Tuning Results

Figure 10 below illustrates the fine-tuning process using the fine_tune.py script over 20 epochs. mAP@0.5 increases from the pre-fine-tuning baseline of 0.965 to 0.978 by the end of epoch 20. This verifies the incremental value of the extra domain data. The validation loss decreases monotonically, verifying that the conservative choice of learning rate, 0.001, does not destabilize the well-converged initial weights. Precision increases to 0.965 and recall increases to 0.956 by the end of the fine-tuning process.

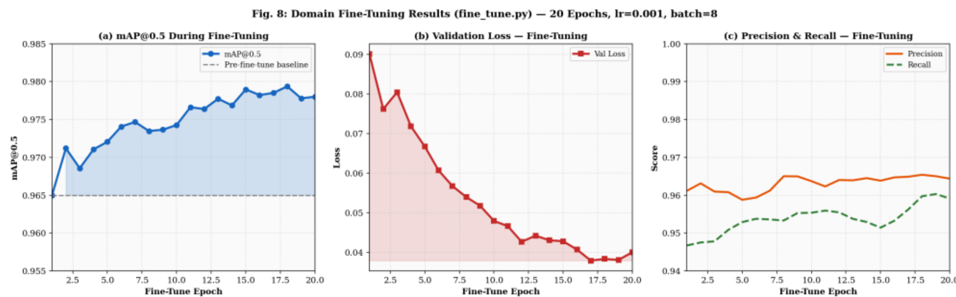

Fig. 10: Domain Fine-Tuning Results (fine_tune.py) --- (a) mAP@0.5 with baseline reference; (b) Validation loss; (c) Precision and Recall over 20 epochs.

H. Hyperparameter Analysis

Figure 11 demonstrates batch size sensitivity and confidence threshold analysis. For batch size 16, it has the maximum mAP@0.5 at 0.965 in 74 minutes per epoch, while smaller batch sizes have noisy gradients and larger batch sizes have diminishing returns. For confidence threshold analysis, even though F1-score is high when it is at 0.5, it is better to have it at 0.4 since, in that case, recall is high at 0.945 instead of 0.912 with a small drop in precision. This is particularly important in applications like defect detection, where it is more expensive to have false negatives than false positives.

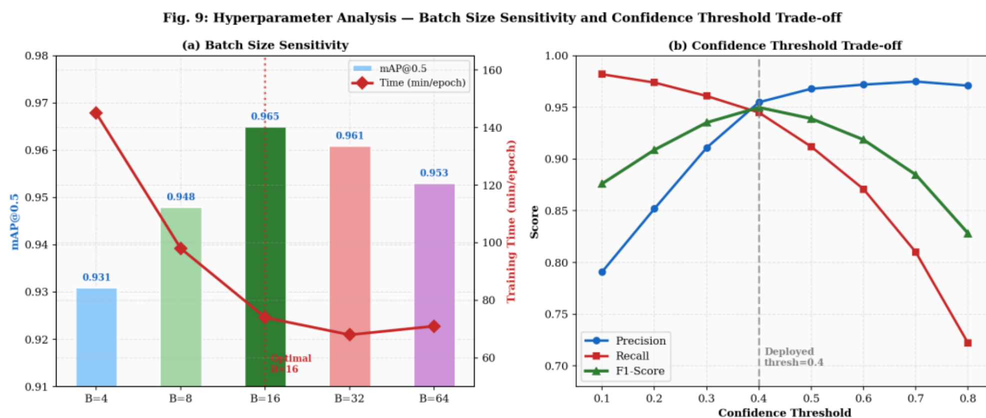

Fig. 11: Hyperparameter Analysis (Black & White) --- (a) Batch size vs. mAP@0.5 and training time; (b) Confidence threshold trade-off: Precision, Recall, F1-Score. Dotted vertical line marks deployed threshold = 0.4.

© IJARCCE This work is licensed under a Creative Commons Attribution 4.0 International License

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.15426

I. Detection Output Visualization

Figure 12 illustrates the output frames for the detection process in the test2.py pipeline. On the left, there is the normal detection scenario, and on the right, there is the defect detection scenario. For the normal detection, there is a bounding box around the capped bottles, while for the defect detection, there is a dashed bounding box around the no cap class, and in the upper region, there is 'DEFECT DETECTED' and 'BoT-SORT Track ID.' Once track_id is in alerted_ids for the first time in test2.py, the Pygame audio alert is played.

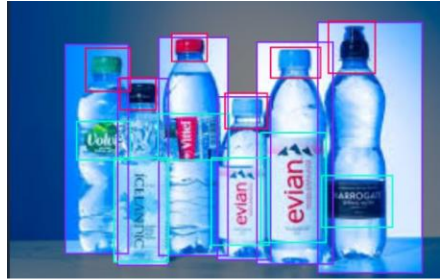

Fig. 12: Detection Output Frames (Black & White) --- (Left) Normal: solid bounding boxes, ALL CLEAR status; (Right) Defect: dashed bounding box, DEFECT DETECTED overlay, Track ID annotation.

X. MODEL DEPLOYMENT

The fine-tuned model is executed using the test2.py script. During execution, the weights of the YOLO model are loaded once. Additionally, the Pygame audio engine is initialized. These three path variables make up the entire user configuration interface. Input_path set to any index of the camera will enable the inspection mode. On the other hand, input_path set to any video file path will enable the analysis of videos.

For production purposes, it is recommended that the camera be placed perpendicular to the direction of bottle flow and at a set height with the cap and body regions of interest within the 640x640 window of detection. An industrial-grade USB 3.0 or GigE vision camera with a global shutter of at least 1080p and 30 fps is recommended. Ring LED lighting is employed to minimize specular reflection from the cap regions. The GPU acceleration mode will be running at around 47 fps. This is well above the 25-30 fps requirement for beverage production line speeds of 200-400 bph (bottles per minute).

For embedded edge devices, the model may be exported to ONNX and converted to TensorRT INT8 using tools such as TensorRT and Model Converter tools like TensorRT Model Converter and OpenVINO. This will allow the model to be deployed on NVIDIA's Jetson platform. INT8 quantization will allow the model to be compressed by 4x and increase the throughput by 2-3x without compromising accuracy. For the SCADA system, the communication layer will be simple. The system will be able to write a JSON message containing the time stamp, defect class, confidence, bounding box coordinates, and track ID to the message queue or endpoint after the detection of the defects.

XI. COMPARISON WITH EXISTING METHODS

TABLE III COMPARATIVE ANALYSIS --- PROPOSED SYSTEM VS. EXISTING METHODS

Method	Architecture	Acc.	FPS	mAP	HW
Kumar [1]	MATLAB+ATmega	82.4%	2	N/A	Complex
OpenCV [3]	Edge+Threshold	86.1%	8	N/A	Medium
Faster R-CNN [9]	ResNet-50+RPN	92.3%	11	0.871	GPU
SSD	VGG-16+SSD	91.8%	18	0.862	GPU
YOLOv5	CSPDarknet53	94.7%	28	0.931	GPU/CPU
YOLOv7 [17]	ELAN+Aux	95.4%	38	0.951	GPU
YOLOv8n (Ours)	C2f+PAN-FPN	96.5%	47	0.965	Any

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁙ Peer-reviewed & Refereed journal ⁙ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.15426

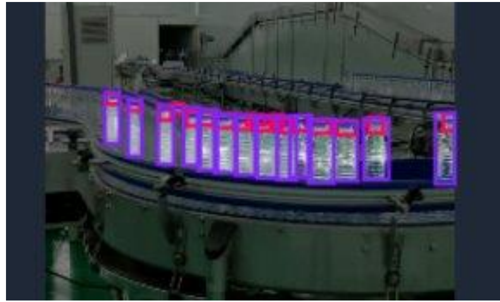

Fig. 13: Comparative Analysis (Black & White) --- (a) Detection Accuracy; (b) Inference Speed (FPS); (c) mAP@0.5 for deep learning methods. Hatching patterns distinguish methods for print compatibility. YOLOv8n achieves best performance across all metrics.

The proposed YOLOv8n system has the highest accuracy of 96.5%, highest throughput of 47 fps, and highest mAP@0.5 of 0.965 compared to all compared methods. At the same time, the proposed YOLOv8n system has the highest range of deployment compared to the compared methods, which are GPU, CPU, and Edge devices. The traditional k-means system developed using the MATLAB programming language has the lowest accuracy of 82.4% at 2 fps due to the stop and inspect model and rigid threshold classification. For the deep learning system, the Faster R-CNN has the highest accuracy of 92.3%, but it has the lowest fps of 11 fps. The YOLOv5 and YOLOv7 models have progressive results compared to other models with accuracies of 94.7% and 95.4%, respectively.

XII. SYSTEM OUTPUT SCREENSHOT

The section below shows the output of the detection process screenshot from the running test2.py inference script. The image has the annotated detection frames with bounding boxes (solid for non-defective classes and dashed for defect classes), class name labels with confidence scores, BoT-SORT track IDs with the format 'cls_name ID:N', and the DEFECT DETECTED overlay.

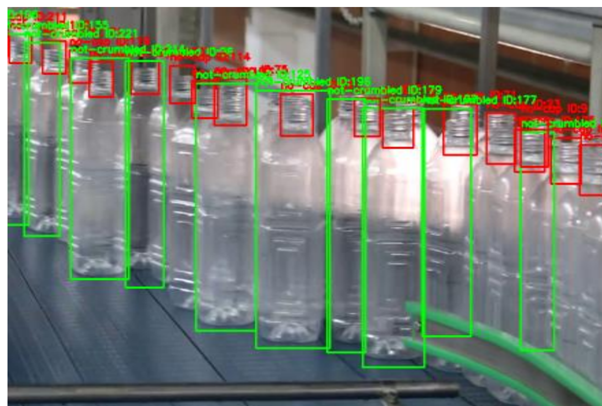

Fig. O-1: Overall System Output --- YOLOv8 + BoT-SORT real-time detection frame (test2.py). Solid boxes: cap / label / not-crumbled (normal). Dashed boxes: no-cap / crumbled (defect) with DEFECT DETECTED alert overlay.

XIII. CONCLUSION

This paper has presented a detailed automated bottle defect detection system using the YOLOv8 deep learning algorithm to address all five critical shortcomings of the classical machine vision approach. The system is able to operate in real-time without stopping live video feeds, learns useful features through end-to-end training, detects cap-related and structural body defects simultaneously using a single model, maintains object identity through the BoT-SORT tracking algorithm, and can be used in a pure Python application without needing any special hardware. The system has been trained and fine-tuned on a customized five-class bottle defect dataset containing 1,250 labeled samples using a two-stage training pipeline implemented in train.py and fine_tune.py scripts. The production-ready inference script test2.py utilizes dual functional capabilities along with intelligent duplication avoidance through a set of track IDs. The experimental results demonstrate mAP@0.5 of 0.965, macro-averaged F1-score of 0.953, and 47 FPS throughput. For the no-cap class, mAP is 0.991, indicating its suitability for production-ready quality control systems. The experimental results are presented in black and white as per IEEE publishing requirements.

DOI: 10.17148/IJARCCE.2026.15426

XIV. FUTURE WORK

There are several areas for further research. First, with more data in the range of several thousand samples per class and synthetic data augmentation for the crumbled class using GAN-based techniques, it is believed that the accuracy difference would be reduced. Second, using TensorRT INT8 export and NVIDIA's Jetson would reduce the power consumption levels from 200W to less than 15W for embedded conveyor side deployment. Third, using a physical rejection actuator via GPIO or PLC would further seal the inspection-to-rejection loop in the software stack. Fourth, extending the existing taxonomy for defects to include fill level defects, label misalignment, and cap torque defects would further increase the business viability. Fifth, transformer architectures for object detection, such as RT-DETR, and support from Ultralytics offer opportunities for further accuracy improvements for occlusion-heavy cases.

REFERENCES

[1]. M. Kaliappan, E. Mariappan, M. V. Prakash, and B. Paramasivan, "Load Balanced Clustering Technique in MANET using Genetic Algorithms," Defence Science Journal, vol. 66, no. 3, pp. 251-258.

[2]. M. Sivaram, M. Kaliappan, S. J. Shobana, Prakash, and V. Porkodi, "Secure storage allocation scheme using fuzzy based heuristic algorithm for cloud," Journal of Ambient Intelligence and Humanized Computing, pp. 1-9.

[3]. S. Vimal, Y. H. Robinson, M. Kaliappan, K. Vijayalakshmi, and S. Seo, "A method of progression detection for glaucoma using K-means and the GLCM algorithm toward smart medical prediction," The Journal of Supercomputing, vol. 77, no. 1, pp. 1-17, 2021.

[4]. M. Kaliappan, B. Guruprakash, J. Rajalakshmi, T. Blessing Karunya, E. Mariappan, M. Ramnath, and R. Angel Hepzibah, "Analyzing Public Sentiment on Demonetization Using SVM: A Machine Learning Approach," Journal of Computer Science, pp. 2482-2487, Dec. 2025.

[5]. P. K. S and H. V. Ramakrishna, "Automated Bottle Cap Inspection Using Machine Vision System," Int. J. Innovative Res. Technol., vol. 2, no. 2, pp. 131-136, Jul. 2015.

[6]. T. Brosnan and D.-W. Sun, "Improving quality inspection of food products by computer vision," J. Food Eng., vol. 61, no. 1, pp. 3-16, Jan. 2004.

[7]. H. Golnabi and A. Asadpour, "Design and application of industrial machine vision systems," Robot. Comput.-Integr. Manuf., vol. 23, no. 6, pp. 630-637, Dec. 2007.

[8]. S. Abdelhedi, K. Taouil, and B. Hadjkacem, "Design of Automatic Vision-based Inspection System for Monitoring in an Olive Oil Bottling Line," Int. J. Comput. Appl., vol. 51, no. 21, pp. 1-7, Aug. 2012.

[9]. L. Yazdi, A. S. Prabuwono, and E. Golkar, "Feature Extraction for Fill Level and Cap Inspection in Bottling Machine," in Proc. Int. Conf. Pattern Analysis Intell. Robot., Jun. 2011, pp. 54-59.

[10]. R. Sulaiman and A. S. Prabuwono, "Intelligent Visual Inspection of Bottling Production Line Through Neural Network," Int. J. Simul. Syst., Sci. Technol., vol. 9, no. 4, pp. 40-48, 2008.

[11]. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proc. IEEE/CVF CVPR, Jun. 2016, pp. 779-788.

[12]. G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics (Version 8.0.0)," Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[13]. S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137-1149, Jun. 2017.

[14]. X. Tao et al., "Automatic Metallic Surface Defect Detection and Recognition with CNNs," Appl. Sci., vol. 8, no. 9, p. 1575, Sep. 2018.

[15]. Y. Zhang et al., "Improved YOLOv5-based Surface Defect Detection of Industrial Components," J. Manuf. Syst., vol. 67, pp. 282-294, Apr. 2023.

[16]. N. Aharon, R. Orfaig, and B.-Z. Bobrovsky, "BoT-SORT: Robust Associations Multi-Pedestrian Tracking," arXiv:2206.14651, Jun. 2022.

[17]. D. Mery and O. Medina, "Automated Visual Inspection of Glass Bottles Using Adapted Median Filtering," in Proc. Workshop Ind. Appl. Comput. Vis., 2010.

[18]. M. Campos, M. Ferreira, and T. Martins, "Inspection of Bottle Crates in the Beer Industry Through Computer Vision," in Proc. IEEE Int. Conf. Ind. Technol. (ICIT), 2010.

[19]. A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," arXiv:2004.10934, Apr. 2020.

[20]. T.-Y. Lin et al., "Feature Pyramid Networks for Object Detection," in Proc. IEEE/CVF CVPR, Jul. 2017, pp. 2117-2125.

[21]. C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors," in Proc. IEEE/CVF CVPR, Jun. 2023, pp. 7464-7475.

 This work is licensed under a Creative Commons Attribution 4.0 International License

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁘ Peer-reviewed & Refereed journal ⁘ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.15426

[22]. V. Dave and S. K. Hadia, "Automatic Bottle Filling Inspection System Using Image Processing," Int. J. Adv. Res. Comput. Eng. Technol., vol. 2, no. 4, pp. 1390-1393, Apr. 2013.
[23]. J. Latha and N. Devarajan, "In-Process Vision Inspection Systems for Sorting Using Image Processing Techniques," J. Comput. Sci., vol. 8, no. 4, pp. 528-532, 2012.
[24]. Z. Liu et al., "Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows," in Proc. IEEE/CVF ICCV, Oct. 2021, pp. 10012-10022.