International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

AI-Based Automated Monitoring and Surveillance System

An Intelligent Approach to Real-Time Object Detection and Video Monitoring

AUTHORS
Project Guide: **Asst. Prof. Arjun Vinod Shinde**
Anshu Gajbhiye, Anurag Dewangan, Ayush Kumar, Jigyasha Dewangan

Department of Computer Science and Engineering
Shri Shankaracharya Technical Campus, Junwani, Bhilai, Chhattisgarh, India

***Abstract*—Conventional video surveillance systems commonly depend on continuous human observation, which can make it difficult to monitor multiple visual events consistently. This paper presents an AI-assisted surveillance prototype that combines Python, OpenCV and the YOLOv8 object-detection model. The proposed workflow accepts frames from a camera or recorded video, uses OpenCV for video capture and frame handling, and applies YOLOv8 to identify supported object categories. Detected objects are represented through class labels, confidence values and bounding boxes overlaid on the video frame. A Flask-based interface is considered as the application layer for presenting processed video and system output, while the project includes database connectivity components using PyMySQL. The design focuses on assisting an operator by highlighting detected objects rather than replacing human judgement or performing identity recognition. The available project information does not include a labelled test set or measured accuracy, precision, recall, latency or frame-rate results; therefore, no quantitative performance improvement is claimed. The paper describes the system workflow, software architecture, implementation considerations, limitations and an evaluation plan for future testing.**

***Index Terms*—AI surveillance, object detection, Python, OpenCV, YOLOv8, computer vision, video monitoring.**

I. INTRODUCTION

Video surveillance is widely used in campuses, offices, public facilities and other environments where continuous observation is required. Traditional camera systems primarily record or display video, leaving an operator to inspect the footage and identify relevant activity. This approach can become demanding when several cameras or long video streams must be monitored. Computer vision offers a way to analyse visual information automatically and to present selected events to a human operator.

Object detection is a computer-vision task in which a model identifies instances of predefined classes and estimates their locations in an image. The YOLO family formulates detection as a single-stage prediction task, producing bounding boxes and class probabilities in one model evaluation [1]. Ultralytics YOLOv8 provides a Python interface for loading a model and running prediction on images, videos and streams [2]. OpenCV complements the detector by providing tools for video capture, frame processing, image display and file handling [3].

The proposed AI Surveillance System integrates these components into a practical software workflow. Python coordinates the processing pipeline, OpenCV obtains and prepares frames, and YOLOv8 performs object detection. The system can display the resulting detections to support human monitoring. A Flask application layer and database connectivity are included in the project technology stack; their exact deployment configuration should be confirmed against the final running build before submission.

The contribution of this work is an application-level integration of established computer-vision components for surveillance assistance. It does not claim a new detection algorithm, face recognition, behaviour understanding or a validated security-alert mechanism. The system is intended to assist an operator, who remains responsible for interpreting detections and deciding whether further action is necessary.

A. RESEARCH OBJECTIVES

The primary objective of this research is to develop an AI-based surveillance system using Python, OpenCV, and YOLOv8 for automated object detection and visual monitoring.
The specific research objectives are:
O1. Develop an intelligent surveillance application using Python by integrating computer vision and deep learning technologies.
O2. Implement video frame acquisition and processing using OpenCV to obtain visual information from the selected video source.
O3. Integrate the YOLOv8 object detection model to identify and classify objects belonging to its supported categories.
O4. Display detected objects using bounding boxes, class labels, and confidence scores to make detection results easily interpretable.
O5. Examine the functional performance of the proposed system under different input conditions and identify factors affecting object detection.
O6. Explore the possibilities of extending the system with additional surveillance features, including event logging, automated alerts, and database integration.

B. RESEARCH QUESTIONS

To achieve the above objectives, this research addresses the following questions:
RQ1. How can Python, OpenCV, and YOLOv8 be integrated to develop an AI-based surveillance system?
RQ2. How effectively can the YOLOv8 model identify and classify supported objects from video frames?
RQ3. How can bounding boxes, class labels, and confidence scores improve the visualization and interpretation of detected objects?

RQ4. What factors, such as lighting conditions, image quality, object occlusion, and hardware limitations, affect the performance of the proposed surveillance system?
RQ5. How can deep learning-based object detection reduce the dependence on continuous manual observation in conventional surveillance environments?
RQ6. What additional features and improvements can enhance the scalability, reliability, and practical usability of the proposed system?

C. RESEARCH CONTRIBUTIONS

The primary contribution of this research is the design and implementation of an AI-based visual surveillance approach that integrates Python, OpenCV, and YOLOv8. The proposed system combines video frame processing with deep learning-based object detection to identify and visually represent supported object categories.
The major contributions include the integration of computer vision and object detection technologies into a unified surveillance workflow, automated identification and localization of objects in video frames, visual presentation of detection results through bounding boxes and class labels, and an examination of the technical challenges associated with deploying object detection in surveillance environments. The proposed architecture also provides a foundation for future enhancements such as event recording, notification mechanisms, and database-supported monitoring.

II. RELATED WORK

The original YOLO method introduced a unified object-detection approach that predicts bounding boxes and class probabilities directly from an image [1]. This single-stage design helped establish the YOLO family as an important direction for real-time object detection. Ultralytics provides the YOLOv8 model family through a Python interface for supported computer-vision tasks [2]. OpenCV is a widely used open-source computer-vision library that supports video capture, image processing and real-time visual applications [3]. Large-scale annotated datasets such as Microsoft COCO have supported the development and evaluation of object-detection systems. COCO contains everyday objects in natural scenes and provides annotations for object localization and recognition [4].

OpenCV is a widely used open-source computer-vision library. Bradski described the library as an infrastructure for real-time computer vision applications [2]. In a surveillance pipeline, OpenCV can acquire frames from a camera or video file, resize or convert frames as required, and render detection annotations. These operations are complementary to the neural-network inference performed by YOLOv8.

Large-scale annotated datasets such as Microsoft COCO have supported the development and evaluation of object-detection systems. COCO contains everyday objects in natural scenes and provides annotations for object localization and recognition [4]. A pretrained model may recognize categories represented in its training data; performance in a particular surveillance environment can nevertheless vary with camera angle, illumination, occlusion, object scale and scene conditions.

In contrast to research that proposes a new detector or reports benchmark results, this project focuses on integrating a pretrained YOLOv8 model with video handling and an application interface. A controlled comparison with other detectors is outside the scope of the current implementation report.

III. PROBLEM STATEMENT AND OBJECTIVES

The central problem addressed is the effort required to visually inspect continuous surveillance footage and locate frames containing objects of interest. A basic camera feed does not automatically identify the visible objects, while manual inspection may be time-consuming. The proposed system addresses this problem by applying object detection to incoming video frames and displaying the detected categories and locations.

A. OBJECTIVES

- Integrate Python, OpenCV and YOLOv8 into a unified video-analysis workflow.
- Support frame acquisition from a camera or recorded video, subject to the configured input source.
- Detect object categories supported by the selected pretrained YOLOv8 weights.
- Display bounding boxes, class labels and confidence values on processed frames.
- Provide an application layer through Flask and retain database connectivity components in the project stack.
- Define a repeatable evaluation procedure for accuracy, processing speed and operational reliability.

B. SCOPE

The scope is limited to object detection in visual input. The system does not identify a person's real-world identity, infer intent, guarantee detection of every security incident, or replace a trained security operator. Any alerting, tracking, access-control or automated response function should be treated as a separate feature unless it is implemented and tested in the submitted application.

IV. PROPOSED METHODOLOGY

The proposed system follows a systematic approach to automated video monitoring and object detection by integrating video acquisition, frame processing, deep learning-based analysis, and visual output generation into a unified workflow. Initially, the system is initialized and the pretrained object detection model is loaded to prepare it for processing visual input. The video source is then selected, which may be a connected camera or a previously recorded video file. OpenCV is used to access the selected source and capture individual frames from the video stream. Each captured frame is checked to ensure that valid image data is available before it is forwarded to the detection stage. The YOLOv8 model processes the input frames and identifies objects belonging to the categories supported by its pretrained weights. For every detected object, the model generates information such as the predicted class, bounding-box coordinates, and confidence score. These detection results are then used to annotate the corresponding video frame by drawing rectangular bounding boxes and displaying the associated class labels and confidence values. The processed frame is presented through the application's output interface, allowing the operator to observe the detected objects and review the visual information. This process continues sequentially for the remaining frames until the video reaches its end or the operator stops the monitoring process. The application then releases the video-capture resources and completes the processing operation. The project also includes Flask and database connectivity components as part of its application technology stack, which can support interface and event-data management according to the features enabled in the final implementation. Overall, the methodology aims to reduce

the effort involved in manually inspecting video footage by providing automated object detection and clear visual representations while retaining human supervision for interpreting the results.

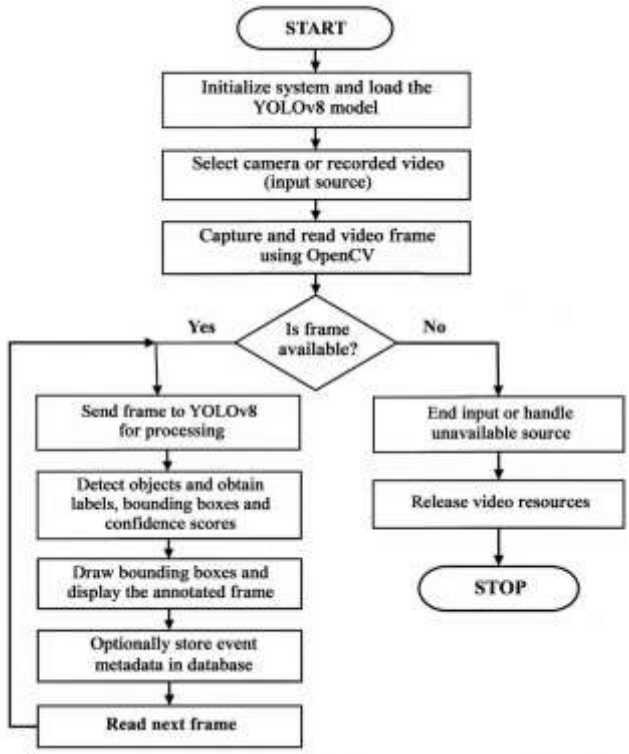


Fig. 1. Flowchart of the proposed AI-based automated monitoring and surveillance system.

A. VIDEO ACQUISITION AND FRAME PREPARATION

OpenCV's video-capture interface can obtain frames from a connected camera or a video file. Each frame is checked for successful acquisition before it is passed to the detector. Preprocessing requirements such as input resizing and colour-channel handling are determined by the model interface. The original frame should be retained where needed so that annotations can be displayed without losing the source visual context.

A robust acquisition loop should handle the end of a video file, a disconnected camera and an unavailable input device. The application should not pass an empty frame to the detection model. When a read operation fails, it should stop or attempt a controlled recovery and provide a clear status message. Camera index, resolution and frame dimensions are environment-dependent settings and should be recorded in the implementation documentation.

Frame sampling is another design consideration. Processing every captured frame provides the most complete temporal coverage but may exceed the processing capacity of the host machine. Processing selected frames can reduce computational load, although events occurring between sampled frames may be missed. The final implementation should document whether it processes every frame or uses a sampling strategy.

B. YOLOV8-BASED OBJECT DETECTION

The project uses the YOLOv8 model weights identified in the project files as yolov8n.pt. The 'n' variant is the nano configuration in the YOLOv8 family and is intended for comparatively lightweight inference. The model receives an image frame and returns a set of candidate detections. Each detection contains a bounding-box location, a predicted class and a confidence score. A confidence threshold can be configured to filter low-confidence predictions; the chosen threshold should be documented during formal testing.

For an input frame, the detector produces a collection of predictions. A predicted bounding box can be represented by its centre coordinates and width and height, or by its upper-left and lower-right corners. The class identifier maps the prediction to one of the categories supported by the loaded weights. The confidence value expresses the model's estimated confidence for that prediction; it is not a calibrated probability that an incident has occurred. The application should preserve this distinction in its labels and operator guidance.

The confidence threshold is an operational parameter. A lower threshold may retain more candidate objects but can also increase false detections, whereas a higher threshold may suppress uncertain predictions while missing some genuine objects. The appropriate value depends on the intended environment and must be selected using validation data rather than chosen solely by visual preference. Non-maximum suppression, which is part of common YOLO detection pipelines, helps remove overlapping predictions for the same object.

The choice of a lightweight model is relevant when the target computer has limited processing resources. However, the 'nano' model designation alone does not establish that the complete application will operate in real time. End-to-end throughput also depends on image resolution, CPU or GPU availability, frame-transfer overhead, video decoding, rendering and any database operations performed during processing.

C. ANNOTATION AND OUTPUT

For each retained detection, the application draws a rectangular bounding box around the predicted object and displays its class label and confidence value. Annotated frames can be shown as a live preview or processed video output depending on the selected input and interface configuration. Detection labels indicate model predictions and should not be interpreted as certain facts.

The visual overlay should remain readable without obscuring important scene details. Consistent line thickness, text size and label placement improve the operator's ability to interpret the frame. If several objects are detected, the display should avoid placing labels on top of one another where practical. The system should also provide a clear indication when no supported object is detected, rather than leaving an operator unsure whether the frame was processed.

For recorded video, output handling may include displaying the annotated stream or saving a processed copy. For a live camera feed, the interface should distinguish the incoming feed from the annotated result. These are presentation options; only the modes present in the final executable application should be described as implemented features.

D. APPLICATION AND DATA LAYER

The project technology inventory includes Flask and PyMySQL, together with a database setup script. Flask can provide the web-facing layer for displaying system status or processed output, while PyMySQL provides Python connectivity to a MySQL-compatible database. The precise tables, fields, retention policy and user-access controls must be described from the final database schema; they are not assumed in this paper. If event logging is enabled, a useful record may

include the event time, detected class, confidence score and source identifier.

V. SYSTEM ARCHITECTURE AND IMPLEMENTATION

The architecture is organized into input, processing, detection, presentation and optional persistence layers. The modular structure separates video acquisition from model inference and user-interface operations, making it easier to test individual components.

At the input layer, the system receives visual data from a configured source. The processing layer validates the frame and prepares it for inference. The detection layer invokes the YOLOv8 model and converts its output into a form that can be drawn or passed to other application components. The presentation layer displays the annotated result and any relevant status information. The persistence layer is optional and should store only the event information required by the application's stated purpose.

This separation supports maintainability. For example, a different camera source can be introduced without redesigning the object detector, while the model can be updated without changing the basic frame-acquisition mechanism. Interfaces between modules should use clearly defined inputs and outputs, and exceptions should be handled at module boundaries so that a camera or database failure does not produce an uncontrolled application crash.

Video Source	OpenCV Frame Capture	YOLOv8 Inference	Flask Interface	Database (PyMySQL)

Fig. 2. High-level processing architecture of the proposed AI surveillance system.

A. SOFTWARE COMPONENTS

Component	Role in the system
Python	Coordinates model loading, frame processing and application logic.
OpenCV	Captures frames and supports image/video processing and annotation.
YOLOv8 (yolov8n.pt)	Performs object detection and returns class, confidence and box coordinates.
Flask	Provides the web application layer, subject to the final interface implementation.
PyMySQL / MySQL	Provides database connectivity; exact stored fields depend on the implemented schema.
NumPy / Pandas	Available supporting libraries for numerical or tabular data handling where required.

Table.1. Software Components.

B. PROCESSING ALGORITHM

- Initialize the Python application and load the YOLOv8 weights.
- Open the configured camera or video source using OpenCV.
- Read the next frame and verify that frame acquisition succeeded.
- Pass the frame to YOLOv8 for object-detection inference.
- Apply the configured confidence threshold to the returned detections.
- Draw bounding boxes and labels on the frame and display the annotated output.
- If event persistence is enabled, store the selected detection metadata through the database layer.
- Repeat until the input ends or the operator stops the process, then release the video resource.

C. IMPLEMENTATION ENVIRONMENT

The project files previously identified include a main Python entry point, a requirements file, YOLOv8 nano weights, a database setup script and a configuration test file. The stated software stack also includes NumPy, Pandas, Flask and PyMySQL. Exact package versions, operating-system details, camera specifications and hardware configuration should be inserted from the environment used for the final demonstration, since these details affect reproducibility and inference speed.

A reproducible setup should record the Python interpreter version and the installed versions of OpenCV, Ultralytics, Flask, PyMySQL, NumPy and Pandas. Dependencies should be installed from the project's requirements file in an isolated virtual environment where possible. The model-weight file should be stored at a known path, and the application should report a clear error if the file is missing or cannot be loaded. Configuration values such as camera source, model path and database connection parameters should be separated from core processing logic.

Database credentials should not be embedded directly in source code or committed to a public repository. They should be supplied through protected configuration, and database operations should use parameterized queries. If the application records detection events, the team should define what information is necessary, how long it is retained, who can access it and how records can be deleted in accordance with institutional requirements.

VI. RESULTS AND EVALUATION

The project is described as an implemented software prototype using Python, OpenCV and YOLOv8. The available information confirms the presence of the main application entry point and model-weight file, but does not provide a labelled evaluation dataset, test protocol, measured output examples or numerical performance results. Accordingly, this paper reports the system workflow and implementation design rather than claiming a measured detection accuracy or real-time frame rate.

VII. OPERATIONAL WORKFLOW AND USER INTERACTION

A. FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

The principal functional requirement is that the application accepts a supported visual input, processes valid frames, invokes the configured detector and presents the resulting annotations. Other functional requirements depend on the implemented build and may include selecting a source, stopping a stream, displaying system status and saving selected event details. Each requirement should be mapped to a test case before the project is evaluated.

Non-functional requirements include responsiveness, reliability, maintainability, usability and privacy. Responsiveness concerns the time needed to process and display frames. Reliability concerns stable handling of unavailable inputs and recoverable errors. Maintainability is supported by separating video capture, inference, interface and data-access modules. Usability requires readable labels and clear status messages. Privacy requires access control and limited retention of any captured or derived data.

A typical operator workflow begins by launching the application and selecting an available video source. The system

initializes the required libraries and loads the configured YOLOv8 weights. Once the source is available, frames are read and analysed in sequence. The annotated output is then presented through the application's display layer. The operator observes the detections and decides whether the scene requires further inspection. When processing is complete, the video source and other resources should be released cleanly.

The interface should communicate system state explicitly. Useful states include model loading, waiting for a video source, processing, input unavailable and stopped. If the application exposes a web interface through Flask, the browser view should not imply that a camera is active when the backend has lost access to it. Error messages should be informative but should not expose database passwords, filesystem secrets or other sensitive configuration.

The human operator remains part of the decision loop. A detected person, vehicle or other object is simply a visual category prediction. It does not, by itself, indicate suspicious behaviour or an emergency. The system should therefore present detections as review cues and avoid automatically assigning blame or triggering consequential action without independent verification.

B. RECOMMENDED EVALUATION METRICS

A formal evaluation should use a representative set of annotated images or video frames captured under the intended operating conditions. The following measures are recommended:

- Precision: proportion of predicted detections that are correct.
- Recall: proportion of ground-truth objects that are detected.
- F1-score: harmonic mean of precision and recall.
- Mean Average Precision (mAP): aggregate detection quality across classes and IoU thresholds.
- Inference latency and frames per second (FPS): processing responsiveness on the target hardware.
- Operational reliability: successful input handling, recovery from unavailable cameras and stable runtime.

The evaluation should report the number of videos and frames, class distribution, annotation procedure, confidence threshold, image size, model version, hardware, software versions and test conditions. Results should be calculated from the actual test run and not estimated from general YOLO benchmarks.

Testing should include more than one visual condition. A useful test matrix may cover daylight and low-light scenes, near and distant objects, partial occlusion, different camera viewpoints, moving objects and scenes containing multiple instances. Each condition should be represented by enough annotated frames to support meaningful comparisons. The same test split should be used when comparing model settings so that changes in the reported metrics can be attributed to the configuration rather than to a different sample.

For object detection, a prediction is commonly counted as a true positive when its class is correct and its intersection-over-union (IoU) with a ground-truth bounding box meets a specified threshold. IoU is the area of overlap divided by the area of union of the predicted and reference boxes. Precision and recall can then be calculated from true-positive, false-positive and false-negative counts. The evaluation report should state the IoU threshold and class-averaging procedure used.

Latency should be measured over a representative run after any model initialization period, and the report should distinguish model inference time from total application time. Frames per second should be calculated from the number of successfully processed frames and elapsed processing time. Hardware details, input resolution and whether a GPU was used must accompany these values, because performance figures are not directly comparable across different configurations.

VIII. DISCUSSION

The main advantage of the proposed design is the integration of video handling and object detection within a familiar Python ecosystem. OpenCV handles video input and visual output, while YOLOv8 supplies the detection predictions. The modular design allows the user interface and persistence layer to be modified without changing the basic detection concept.

The system also has important constraints. Detection quality depends on the selected model weights, the classes represented during training and the visual conditions of the camera feed. Small, distant, partially occluded or poorly illuminated objects may be missed or incorrectly classified. A bounding box and confidence score do not establish that an event is dangerous. Human review is therefore necessary before acting on a detection.

IX. LIMITATIONS AND FUTURE WORK

A. CURRENT LIMITATIONS

- No quantitative accuracy, precision, recall, mAP, latency or FPS results have been supplied.
- The pretrained model is limited to the object categories supported by its weights; custom classes require an appropriately annotated dataset and training or fine-tuning.
- Performance may degrade under low light, motion blur, occlusion, unusual camera angles and crowded scenes.
- The current paper does not establish the implementation or validation of face recognition, identity tracking, abnormal-behaviour detection or automatic emergency alerts.
- Database schema, data-retention rules, authentication and deployment security require verification against the final application.

B. FUTURE ENHANCEMENTS

- Evaluate the system on a labelled dataset representative of the intended surveillance environment.
- Measure detection quality and processing speed on the target computer and camera configuration.
- Add object tracking to maintain temporary identities of detections across consecutive frames, if required.
- Develop configurable event rules and operator notifications, followed by testing for false alarms.
- Improve access control, encrypted communication, audit logging and data-retention controls.
- Provide a clear operator interface with camera status, event review and exportable reports.

X. ETHICAL AND PRIVACY CONSIDERATIONS

Surveillance systems process visual information that may include identifiable people. Deployment should follow applicable institutional policies and local laws, use cameras only for clearly defined purposes, restrict access to authorized operators and retain recordings only for an approved period. The prototype described here performs object-category detection and does not require facial identity recognition. Users should avoid treating automated detections as proof of misconduct and should provide appropriate human review, transparency and safeguards against misuse.

XI. CONCLUSION

This paper presented an AI-assisted surveillance prototype that integrates Python, OpenCV and YOLOv8 for object detection

in camera or video input. OpenCV provides the frame-acquisition and image-handling stage, YOLOv8 predicts object classes and locations, and the application layer presents annotated frames to support operator monitoring. Flask and PyMySQL form part of the stated project stack for application and database integration. The work is an implementation-oriented integration of existing tools rather than a new detection algorithm. Since no controlled evaluation results were provided, quantitative claims about accuracy or speed are intentionally omitted. Future work should validate the system with annotated surveillance data, report standard detection and latency metrics, and strengthen privacy, security and operational safeguards.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[2] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," version 8.0.0, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] T.-Y. Lin et al., "Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, vol. 8693, 2014, pp. 740–755, doi: 10.1007/978-3-319-10602-1_48.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[6] W. Liu et al., "SSD: Single Shot MultiBox Detector," in *Proc. European Conference on Computer Vision (ECCV)*, 2016, pp. 21–37.

[7] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," in *Proc. IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3464–3468, doi: 10.1109/ICIP.2016.7533003.

[8] F. W. Fazal Wahab et al., "Design and Implementation of Real-Time Object Detection System Based on Single-Shot Detector and OpenCV," *Frontiers in Psychology*, vol. 13, 2022, doi: 10.3389/fpsyg.2022.1039645.

[9] H.-T. Duong, V.-T. Le, and V. T. Hoang, "Deep Learning-Based Anomaly Detection in Video Surveillance: A Survey," *Sensors*, vol. 23, no. 11, Art. no. 5024, 2023, doi: 10.3390/s23115024.

[10] K. Doshi and Y. Yilmaz, "Continual Learning for Anomaly Detection in Surveillance Videos," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020.

[11] A. Huang et al., "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.