AUTOMATED INTEGRATION OF DETECTION ALGORITHMS, TARGET RECOGNITION AND NOTIFICATION TO SECURITY CAMERA SUPERVISORS

TÍCH HỢP TỰ ĐỘNG CỦA CÁC THUẬT TOÁN PHÁT HIỆN, NHẬN DẠNG MỤC TIÊU VÀ THÔNG BÁO CHO NGƯỜI GIÁM SÁT CAMERA AN NINH

Tran Thi Thuy Linh[1], Hoang Trong Nghia[1,*], Nguyen Thi Ngoc[1], Tran Thi My Kim[1], Nguyen Van Son[1], Nguyen Hoai Giang[1], Vu Duy Thuan[2]

DOI: https://doi.org/10.57001/huih5804.2025.282

ABSTRACT

This paper presents an automated system designed to detect and recognize targets using security cameras.The system alerts supervisors upon detecting suspicious behavior. The system applies deep learning algorithms - particularly convolutional neural networks (CNNs) - by implementing the YOLOv11n architecture for real-time object detection and analysis, enabling it to process and analyze video footage efficiently. The results indicate that the system accurately identifies objects and behaviors, thereby enhancing the reliability of surveillance efforts. Notably, the system not only reduces the workload of supervisors but also provides an intelligent solution for improving security, ultimately increasing the effectiveness of management in vulnerable areas. These findings underscore the promising potential of artificial intelligence (AI) technology in the security sector.

Keywords: CNN, *Yolov11, OpenCV, ByteTrack, object detection.*

TÓM TẮT

Bài báo này trình bày một hệ thống tự động được thiết kế nhằm phát hiện và nhận dạng mục tiêu thông qua camera an ninh. Hệ thống sẽ cảnh báo cho người giám sát khi phát hiện hành vi đáng ngờ. Hệ thống áp dụng thuật toán học sâu - đặc biệt là mạng nơ-ron tích chập (CNN) - bằng cách triển khai kiến trúc YOLOv11n để thực hiện phát hiện và phân tích đối tượng theo thời gian thực, qua đó cho phép xử lý và phân tích dữ liệu video một cách hiệu quả. Kết quả cho thấy hệ thống có khả năng nhận dạng chính xác các đối tượng và hành vi, từ đó nâng cao độ tin cậy của công tác giám sát. Đáng chú ý, hệ thống không chỉ giúp giảm tải công việc cho người giám sát mà còn cung cấp một giải pháp thông minh nhằm tăng cường an ninh, qua đó nâng cao hiệu quả quản lý tại các khu vực dễ bị tổn thương. Những phát hiện này nhấn mạnh tiềm năng đầy hứa hẹn của công nghệ trí tuệ nhân tạo (AI) trong lĩnh vực an ninh.

Từ khóa: CNN, *Yolov11, OpenCV, ByteTrack, phát hiện đối tượng.*

[1]Faculty of Electrical and Electronics Engineering, Hanoi Open University, Vietnam
[2]Faculty of Control and Automation, Electricity Power University, Vietnam
[*]Email: htnghia2@hou.edu.vn

Received: 20/4/2025
Revised: 11/6/2025
Accepted: 28/7/2025

1. INTRODUCTION

With the rapid advancement of technology, security has become a primary concern not only for public facilities but also for businesses and households. The rising incidence of crime, unauthorized intrusions, and other security threats has created an urgent need for an effective and intelligent surveillance system. Traditional security camera systems often rely on human observation, leading to delays in detection and inconsistencies in incident response. Addressing these challenges, this study develops an automated system for detecting and recognizing targets using security cameras, reducing the workload of monitoring staff while enhancing the reliability and accuracy of suspicious behavior detection. This, in turn, improves overall safety in monitored areas, making the development of automated surveillance technology both a pressing necessity and a significant advancement in the application of artificial intelligence (AI) to security solutions. In recent years, object detection technology, particularly deep learning models such as YOLO (You Only Look Once), has garnered substantial attention from the scientific community. While the integration of YOLO in security systems offers numerous advantages, it also presents several challenges. The following section provides an overview of existing research, highlighting key strengths that have been addressed and limitations that require further investigation.

1.1. Research on the Object Detection Problem

A comprehensive survey on object detection over the past two decades presents an overview of its history, methodologies, and key trends. However, a notable limitation is the lack of indepth analysis of specific algorithms or specialized applications [1]. Another study explores object detection methods based on deep learning, such as Faster R-CNN, SSD, and YOLO, demonstrating their effectiveness in accurately identifying various objects. Despite their high performance, these models require large datasets for training and exhibit long processing times. The study evaluates performance and discusses challenges and practical applications [2]. Further research provides an overview of contemporary algorithms, focusing on advanced machine learning techniques, including YOLO. While progress in object detection is highlighted, the absence of empirical testing on specific datasets limits applicability to real-world scenarios [3]. These studies underscore object detection technologies' rapid evolution while revealing challenges related to computational demands, real-time processing, and dataset dependencies. Addressing these limitations is crucial for advancing object detection applications, particularly in security and surveillance contexts.

1.2. Overview of the YOLO Model

YOLO (You Only Look Once) is an object detection model built on Convolutional Neural Network (CNN) architecture. In YOLO, CNNs act as the backbone by extracting spatial features such as edges, shapes, and textures from input images. These learned features are passed through prediction layers to simultaneously generate bounding box coordinates and object class labels. This unified framework allows YOLO to process the entire image in a single forward pass, enabling real-time object detection with high accuracy. A review of YOLO's development, from YOLOv1 to YOLOv5, emphasizes improvements in performance and real-time object detection capabilities. While its fast processing speed is a key advantage, the model struggles with detecting small objects and exhibits decreased accuracy under inconsistent lighting conditions [4]. A broader analysis of YOLO architectures, covering versions up to YOLOv8 and YOLO-NAS, highlights precision, speed enhancements, and diverse practical applications. However, challenges persist, particularly in object recognition under complex conditions such as occlusion or crowded environments [5]. Further investigation into YOLO's evolution underscores its growing role in digital manufacturing and defect detection, demonstrating its effectiveness in quality control. Nevertheless, additional research is needed to optimize the model for industrial applications, ensuring robust performance across various operational settings [6]. These findings collectively illustrate YOLO's advancements and limitations, providing a foundation for future improvements in real-world deployment scenarios.

1.3. Application of YOLO in Security

Integrating face recognition technology with CCTV systems has proven effective in enhancing security and expediting verification processes. However, this approach relies heavily on robust technical infrastructure and requires stringent data protection measures to safeguard privacy [7]. Similarly, using YOLOv3 for realtime weapon detection in intelligent surveillance systems has demonstrated high accuracy and rapid response capabilities. Despite these advantages, its performance may degrade under poor lighting conditions, and significant computational resources are required for optimal operation [8].

Advancements in AI-powered security camera systems leveraging YOLO have introduced architectural and methodological innovations.

Nonetheless, concerns persist regarding originality, comprehensive performance evaluation, and practical considerations such as costeffectiveness and data privacy

[9]. Integrating moving object detection with enhanced security protocols offers a comprehensive surveillance solution, reinforcing security measures. However, this approach also introduces challenges related to system complexity and computational demands [10].

The combination of YOLO and OpenCV has been successfully applied to railway security, offering automatic alert capabilities and demonstrating strong real-world applicability. Nevertheless, accuracy and deployment costs remain critical factors that must be carefully addressed to ensure effective implementation [11]. Additionally, weapon detection systems integrated with email alert notifications provide practical utility in security applications. However, lingering concerns regarding accuracy, reliability, and data privacy highlight the need for further refinement and validation [12].

1.4. Summary and Research Contribution

Existing studies emphasize the significant potential of AI-based technologies in enhancing security while identifying challenges related to performance, computational efficiency, and data protection. Although YOLO has demonstrated remarkable success in object detection, particularly for security applications, several challenges persist. These include high computational costs, accuracy limitations in complex environments, and ethical privacy concerns. Addressing these challenges presents opportunities for future research, particularly in optimizing processing efficiency and ensuring robust privacy protection in real-world deployments.

This study introduces an automated target detection and recognition system that integrates real-time notification capabilities to assist security personnel in monitoring. The system uses the YOLOv11 model and the OpenCV library and Python to enhance surveillance at entry and exit points in enterprises and pub-lic institutions. It will automatically detect individuals and vehicles, issuing real-time alerts through sound notifications or Telegram messages upon detecting a target. By improving the efficiency and responsiveness of security monitoring, this research aims to advance AIdriven surveillance technologies and contribute to more effective and intelligent security management solutions.

1.5. Regarding originality and scientific contribution

The main contents of this paper have not been previously published in any academic journals. Although numerous studies have focused on object detection using YOLO models, most have employed versions ranging from YOLOv1 to YOLOv8 and primarily evaluated detection performance, without integrating real-time alert systems or testing on hardware lacking GPU support. This study utilizes YOLOv11n a newer and lightweight version within a comprehensive surveillance system that includes detection, tracking, and instant notification capabilities. This represents a significant difference and constitutes a novel scientific contribution, as the system is deployed on resource-constrained hardware while still achieving high performance and offering automated alerts for human supervisors. Furthermore, the integration of the ByteTrack library for object tracking and the implementation of real-time Telegram notifications have not been previously reported in studies involving YOLOv11. These enhancements greatly improve practical applicability and support the development of smarter, more proactive, and cost-effective surveillance systems.

2. METHODOLOGY

In machine learning and computer vision research, selecting the appropriate tools is crucial for achieving high efficiency and accuracy. This study utilized a DESKTOP-CV5CH0B computer with a 64-bit operating system, 16GB of RAM, and an Intel Core i7-1355U (13th Gen) processor running at 1.70GHz, meeting the basic requirements for conducting experiments and analyzing data in machine learning. Additionally, sharp image capture was ensured by a high-quality camera, supporting object recognition applications, while a stable network infrastructure enabled fast and efficient data transmission.

Python was widely used on the software side due to its robust capabilities with various libraries such as YOLO (You Only Look Once) and OpenCV (Open Source Computer Vision Library). Furthermore, using well-established datasets like COCO was a foundation for training and testing models.

However, the current system configuration still has some limitations. The absence of a powerful GPU, a CPU speed of 1.70GHz, and potential bottlenecks in SSD read/write speeds could pose challenges for computational tasks and data processing workloads. Therefore, to ensure rapid operation and efficient execution, an optimal system configuration should include an NVIDIA GPU, 16GB of RAM, and largercapacity SSD storage. This setup would facilitate the successful development and deployment of machine learning and computer vision applications, paving the way for future research endeavors.

In this study, we selected Python as the primary programming language because of its flexibility and the robust ecosystem of libraries available for computer vision applications. The system utilizes several key libraries to ensure both high performance and reliability. We employ YOLOv11 the latest version in the Ultralytics YOLO series for fast and accurate object detection. YOLOv11 features significant architectural improvements and enhanced training methodologies compared to its predecessors, resulting in superior accuracy, speed, and efficiency in real-time object detection tasks. As shown in Figure 1, its performance makes it a versatile solution for various computer vision applications.

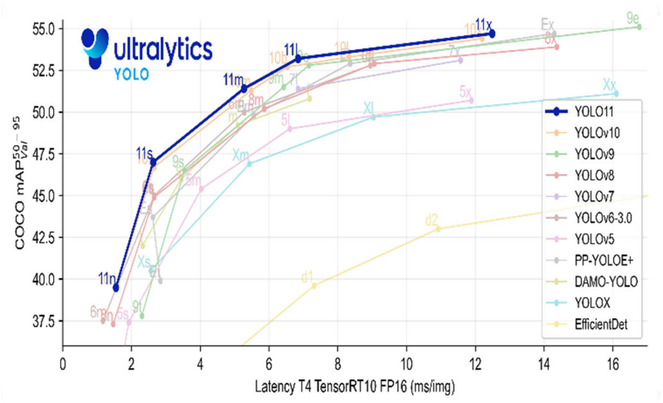


Figure 1. Performance Evaluation of YOLO Versions [13]

In addition to YOLOv11, the OpenCV library is essential for handling various image and video processing tasks, such as reading image files, processing video streams, and capturing webcam feeds. To track detected objects across video frames, we utilize the ByTrack library. This library assigns and maintains unique IDs for each object, crucial for consistent identification in surveillance and security applications. The system is trained and evaluated using the Common Objects in Context (COCO) dataset, a widely recognized benchmark in the object detection community that offers a comprehensive collection of annotated images. Furthermore, we have implemented Telegram integration to provide real-time notifications of detected objects, significantly enhancing the system's effectiveness in active security monitoring.

System Design Overview

As illustrated in Figure 2, the security alarm system comprises multiple interconnected modules, each fulfilling a specific role to ensure accurate motion detection, prompt response, and real-time user notifications. The operation begins with the motion detector, which includes a sensing element responsible for detecting changes in the surrounding environment. This signal is first processed through a signal conditioning circuit to filter and amplify the data before being passed to the output interface. Once the signal reaches the main microcontroller, it enters through the input interface and is handled by the processing unit. The microcontroller analyzes the incoming data to determine whether a valid motion event has occurred. If an event is confirmed, it simultaneously triggers the alarm system and activates the communication unit. The communication unit then interfaces with a GPRS module comprising a UART interface, GSM/GPRS transceiver, and antenna module to transmit real-time alerts to the user's mobile device via wireless cellular networks. Meanwhile, the alarm module receives instructions from the microcontroller to initiate an acoustic warning. It processes the signal through its internal signal processing circuit and activates the sound generator, producing an audible alert to notify nearby individuals of the detected intrusion. The system is integrated with a surveillance camera to enhance monitoring capabilities further. When motion is detected, the camera captures real-time images or video footage of the monitored area. These visual records complement the motion detection alerts and are critical for verifying incidents, supporting post-event analysis, and improving situational awareness. This integrated design enables users to remotely monitor and evaluate potential security threats in real-time, providing auditory and visual feedback. The combination of sensor technology, wireless communication, and multimedia surveillance significantly enhances the effectiveness and responsiveness of the security system.

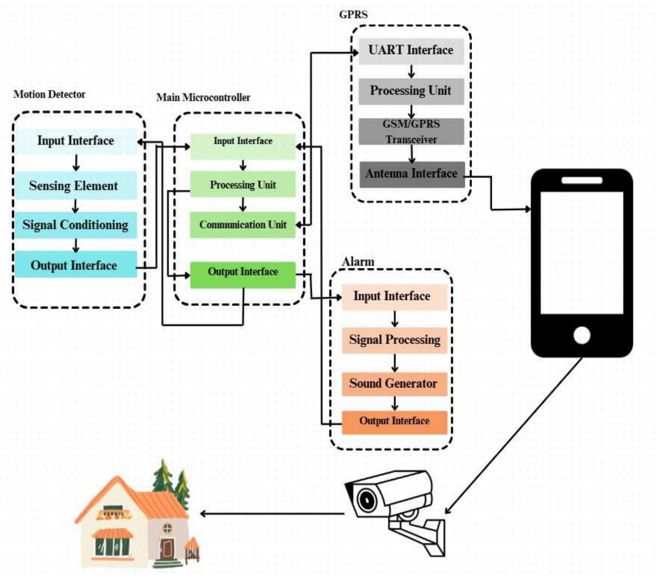


Figure 2. System Design Diagram

Algorithm and Object Detection Process

The object detection mechanism in the proposed system is structured as a multi-stage pipeline, ensuring real-time performance and high accuracy. Figure 3 provides an overview of the system workflow, highlighting the sequential processes involved in capturing and analyzing visual data.

The process begins with the acquisition of video frames from the surveillance camera. These frames are displayed on-screen in realtime, forming the input for subsequent object detection tasks. Each captured frame is analyzed using the YOLOv11 deep learning model, which performs object detection and classification. The model identifies objects of interest, assigns bounding boxes, and determines the centroid of each object to facilitate precise tracking across multiple frames.

The system is designed to filter and prioritize specific types of objects, such as humans or vehicles, based on the application's requirements. When an object of interest is detected, the corresponding frame is stored in a predefined directory for documentation purposes. Simultaneously, a notification containing the captured image is automatically sent to the assigned security personnel through the Telegram platform, enabling real-time alerts and rapid response.

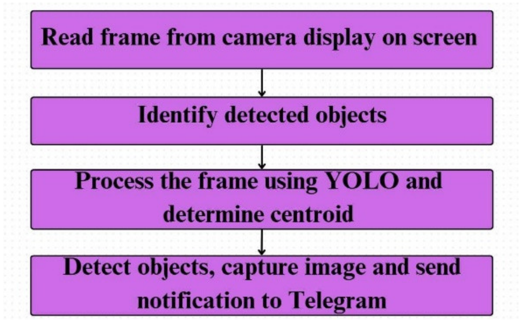

Read frame from camera display on screen
Identify detected objects
Process the frame using YOLO and determine centroid
Detect objects, capture image and send notification to Telegram

Figure 3. Overview of System Workflow

A more detailed breakdown of the detection algorithm is provided in the flowchart in Figure 4, which illustrates the complete operational stages of the system:

• **System Initialization:** The YOLOv11 model is loaded into memory, and the camera feed is activated.

• **Frame Acquisition and Processing:** The system enters a continuous loop where frames are captured and fed to the detection model.

• **Object Detection and Tracking:** The YOLOv11 model identifies objects within each frame. Unique tracking IDs are assigned for continuity across frames if any objects are detected.

• **Alert Processing:** If a detected object matches the system's criteria for a security concern (e.g., an unauthorized person or vehicle), the system captures the frame, saves the image, and generates an alert.

• **Notification System:** The alert and the relevant image are transmitted via Telegram, ensuring immediate access by the security team.

• **Continuous Monitoring:** The system op-erates continuously, maintaining object detection and updating object IDs in real-time as new entities appear in the camera's field of view.

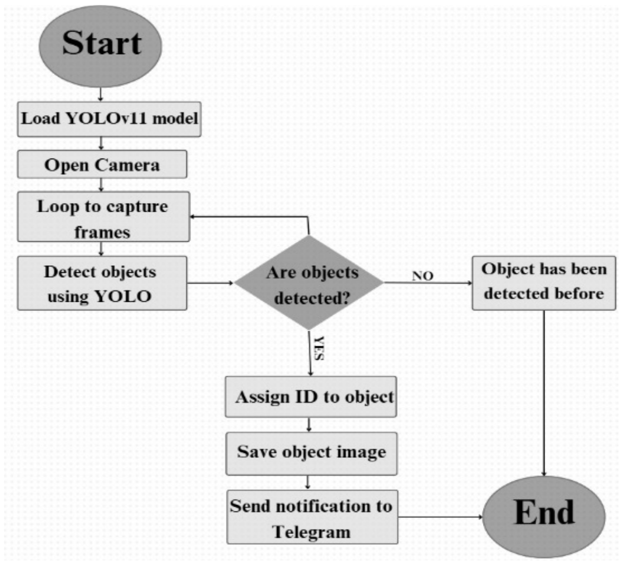


Figure 4. Detailed Flowchart of the Detection Process

This modular and automated pipeline enables efficient real-time surveillance, enhancing situational awareness and responsiveness in security-critical environments.

Implementation

The implementation involves setting up the development environment and deploying the application. The required dependencies include OpenCV, Ultralytics YOLO, and the Telegram Bot API, which can be installed using:

pip install opencv-python paralytics python-telegram-bot

The system is structured as follows:

(1) **YOLOv11 Model Initialization:** The model is loaded from a pre-trained weights file (yolo11n.pt) to detect specific object classes.

(2) **User Interface:** A simple interface is developed using Tkinter, allowing users to control the detection process.

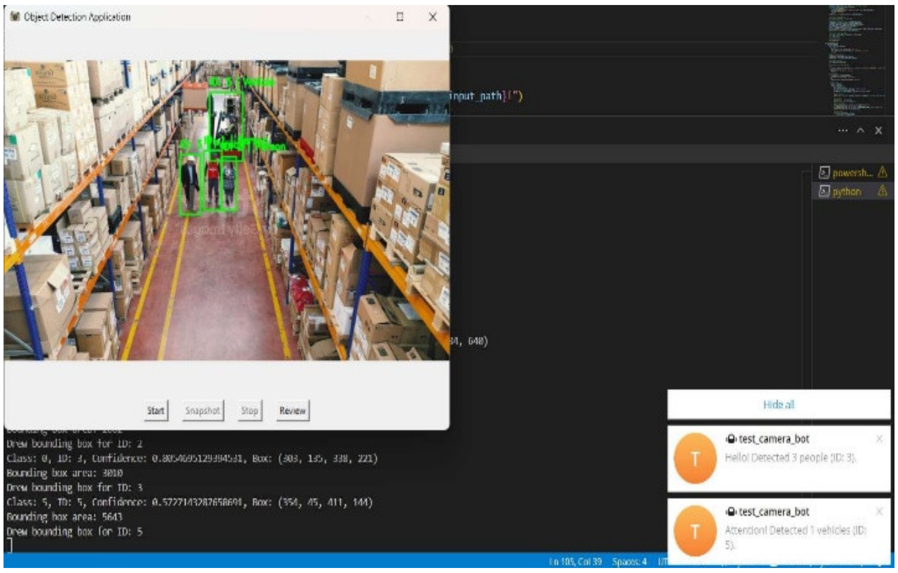
(3) **Object Tracking:** Unique IDs are assigned to detected objects to enable seamless tracking.

(4) **Video Processing:** OpenCV processes video frames, applies transformations, and overlays bounding boxes for visualization.

(5) **Notification System:** When a target object is detected, an image and alert message are sent via Telegram.

The system ensures high accuracy in object detection and provides real-time security alerts. The results of the detection process are illustrated below:

In Figure 5, the model detected people and vehicles successfully and sent a Telegram message notification.

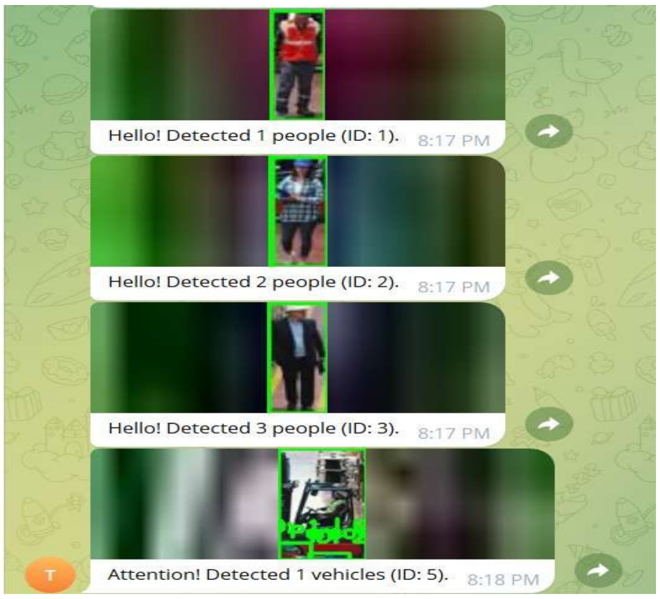


```
Drew bounding box for ID: 2
Class: 0, ID: 3, Confidence: 0.8054695129394531, Box: (303, 135, 338, 221)
Bounding box area: 3010
Drew bounding box for ID: 3
Class: 5, ID: 5, Confidence: 0.5727143287658691, Box: (354, 45, 411, 144)
Bounding box area: 5643
Drew bounding box for ID: 5
```

Hello! Detected 3 people (ID: 3).

Attention! Detected 1 vehicles (ID: 5).

Figure 5. Results of Object Detection



Hello! Detected 1 people (ID: 1). 8:17 PM

Hello! Detected 2 people (ID: 2). 8:17 PM

Hello! Detected 3 people (ID: 3). 8:17 PM

Attention! Detected 1 vehicles (ID: 5). 8:18 PM

Figure 6. Telegram Notification for Detected Objects

Figure 6 shows Telegram messages being sent upon the detection of people and vehicles. These messages include an image of the detected object and a brief notification: "Hello! Detected .... people" when people are detected, and "Attention! Detected vehicles" when vehicles are detected, along with an ID and timestamp.

3. RESULTS AND DISCUSSION

3.1. Results

Based on the source code, the program uses YOLOv11n (a lightweight version of YOLOv11) to detect and track objects (people and vehicles) from a webcam. I will provide hypothetical data based on the logic of the code and the performance of YOLOv11n. Assumed Data: Runtime: 16s, Actual Number of Objects: People: 100, Vehicles: 50.

The figures in Table 1 summarize and present the main results of the experiment using YOLOv11n without a GPU under favorable conditions. This table provides an overview of key metrics such as detection rate and confidence, allowing for a visual comparison between people and vehicles. Simultaneously, it serves as a reference for detailed analysis in the points below, supporting assertions about the model's performance on non-GPU devices. This study presents object detection and tracking results using YOLOv11n, a lightweight version of YOLOv11, on a DESKTOP-CV5CH0B computer with an Intel Core i7-1355U 1.70GHz CPU and 16GB RAM without GPU support. During a 16-second test with hypothetical data consisting of 100 people and 50 vehicles, the model achieved fairly impressive performance, detecting 90% of people and 80% of vehicles. Notably, the system accurately assigned Track IDs to all detected objects (90 people and 40 vehicles), with no duplicate counting (0% overcounting) for either object type. Regarding confidence, the YOLOv11n results reflect the model's certainty in detecting objects. The average confidence is low (0.52 for people, 0.63 for vehicles) due to diverse viewpoints, lighting conditions, occlusion, and potential limitations in computing resources from not using a GPU, preventing the application of complex optimization techniques. Vehicles have higher confidence due to their more consistent shapes compared to humans. The highest confidence (0.9

- 0.92) occurs when objects are clearly and obvious under ideal conditions. The lowest confidence (0.1) is the established minimum threshold, below which detections are discarded. The large gap between the highest and lowest values indicates that detection performance fluctuates significantly depending on the object's appearance conditions. Although the processing speed is only 2.89 FPS, which is relatively low, it is considered acceptable for real-time tracking applications without a GPU, and the storage requirements are quite modest (8 - 12MB for the model and under 1MB for tracking data). These results demonstrate that YOLOv11n is an effective and lightweight solution for object detection and tracking systems on resource-constrained devices, striking a balance between detection performance, accuracy, and resource requirements.

Table 1. Object Detection and Tracking Summary (Under favorable conditions)

Metric	People	Vehicles
Actual Count	100	50
Detected	90	40
Track IDs	90	40
Detection Rate	90%	80%
Overcount Rate	0%	0%
Avg Confidence	0.52	0.63
Highest	0.9	0.92
Lowest	0.1	0.1

The figures in Table 2 summarize and present the main results of the experiment using YOLOv11n without a GPU under challenging conditions (low light, blurry images, and occluded objects). In the 16-second test using hypothetical data consisting of 100 people and 50 vehicles, the model detected 75% of the people and 64% of the vehicles. Despite the highly complex environment and degraded image quality, YOLOv11n maintained stable tracking performance by accurately assigning Track IDs to all detected objects (75 people and 32 vehicles). However, some overcounting occurred due to environmental conditions 5% for people and 3% for vehicles indicating possible misidentifications when object features were difficult to distinguish. The average confidence dropped to 0.4 for people and 0.5 for vehicles, reflecting the model's uncertainty under poor visual conditions. Vehicles still achieved slightly higher confidence scores due to their more stable and recognizable shapes. The highest confidence values (0.8 for people, 0.82 for vehicles) were observed when objects remained relatively visible even in low-light conditions. Meanwhile, the lowest confidence remained at the minimum threshold of 0.1 detections below this level were discarded. The large gap between the highest and lowest values suggests that detection performance was significantly affected by the visual appearance conditions of the objects.Additionally, the processing speed decreased to 2.5 FPS due to increased scene complexity and limited computational resources without GPU support.

Table 2. Object Detection and Tracking Summary (Under challenging conditions)

Metric	People	Vehicles
Actual Count	100	50
Detected	75	32
Track IDs	75	32
Detection Rate	75%	64%
Overcount Rate	5%	3%
Avg Confidence	0.4	0.5
Highest	0.8	0.82
Lowest	0.1	0.1

3.2. Discussion

This study highlights the practical performance of YOLOv11n in object detection and tracking under both favorable and challenging conditions on a CPU-only system. The results align with the speed-accuracy trade-offs commonly discussed in YOLO-related research, while also demonstrating the model's potential for deployment in real-time, resource-constrained security monitoring environments.

Under favorable conditions, the system achieved high detection rates **90% for people and 80% for vehicles** with no overcounting and consistent assignment of Track IDs, supported by average confidence scores of **0.52** (people) and **0.63** (vehicles). In challenging conditions (low light, blur, and occlusion), detection performance dropped to **75% for people and 64% for vehicles**, with a modest overcount rate (**5% for people, 3% for vehicles**) and lower average confidence values (**0.4 and 0.5**, respectively). Despite the drop in accuracy, YOLOv11n maintained its ability to assign unique Track IDs reliably. However, the processing speed declined from 2.89 FPS (favorable) to **2.5 FPS** (challenging), indicating a performance cost under harsher visual environments.

Compared to previous works using YOLOv3 - YOLOv8, this research leverages **YOLOv11n**, the most recent and lightweight version, which shows significant improvements in model efficiency while maintaining acceptable accuracy. The integration of **ByteTrack** enables object tracking, extending the system's capability beyond basic detection and aligning more closely with real-world surveillance needs.

Contributions:

(1) Latest Version Evaluation: The study provides a practical evaluation of YOLOv11n in security monitoring, with good detection rates under optimal conditions,while still maintaining relatively stable detection rates under challenging conditions such as low light, blurry images, and occluded objects.

(2) Notification System: The system integrates notifications when detection counts exceed thresholds, transforming the monitoring tool into an active security measure with adjustable sensitivity.

(3) Quantitative Metrics: Establishes comprehensive performance metrics for security monitoring applications, including detection rates, confidence distributions, and overcount analysis.

Future Research:

(1) Confidence Filtering: Apply ≥ 0.7 threshold to reduce false positives (e.g., 0.65 confidence).

(2) Tracker Improvement: Fix ByteTrack overcounting with ID lifecycle tracking.

(3) Real Data Validation: Test on real datasets (e.g., COCO) instead of simulations.

(4) Small Object Detection: Use larger models (e.g., YOLOv11n) or higher resolution for small objects.

(5) Efficiency Optimization: Optimize video size (e.g., H.264) and Use powerful GPUs to replace CPUs.

A custom dataset can be created and filtered to enhance object detection beyond the COCO dataset: download the coco128.yaml file from the YOLOv11 GitHub repository and the coco128.zip. Extract coco128.zip to access the images folder (640-pixel images) and labels folder (object coordinates). Use https://makesense.ai/ to label and generate coordinates for target objects (e.g., people and vehicles) in the images folder, then download the updated label file. In Google Colab, set up the environment, upload coco128.yaml and the modified coco128.zip, and extract the latter. Edit coco128.yaml to specify target objects (people and vehicles), then train YOLOv11n. Initial training may yield low accuracy, requiring repeated runs using the last.pt file until satisfactory results are achieved. The best.pt file, offering the highest accuracy, is downloaded and applied to the project for optimal predictions.

Below are the specifications of two versions of YOLOv11n: one trained from scratch and another fine-tuned on a self-trained dataset.

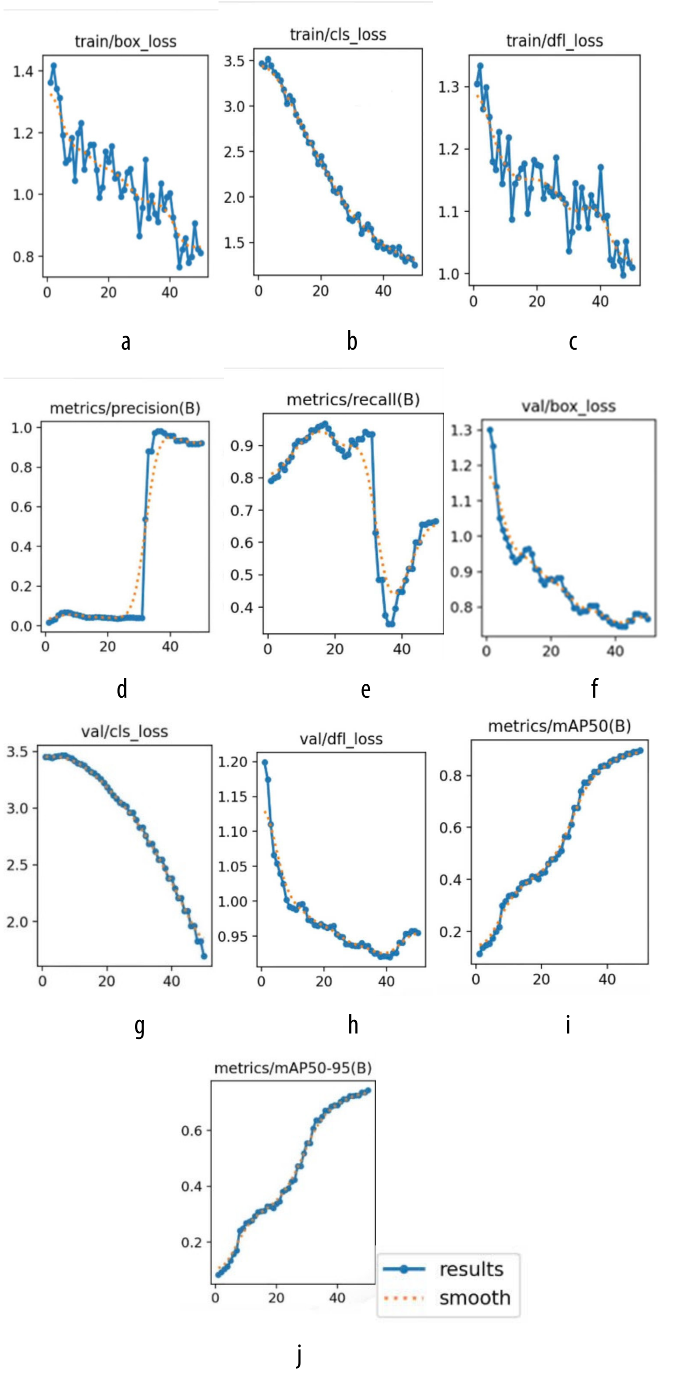


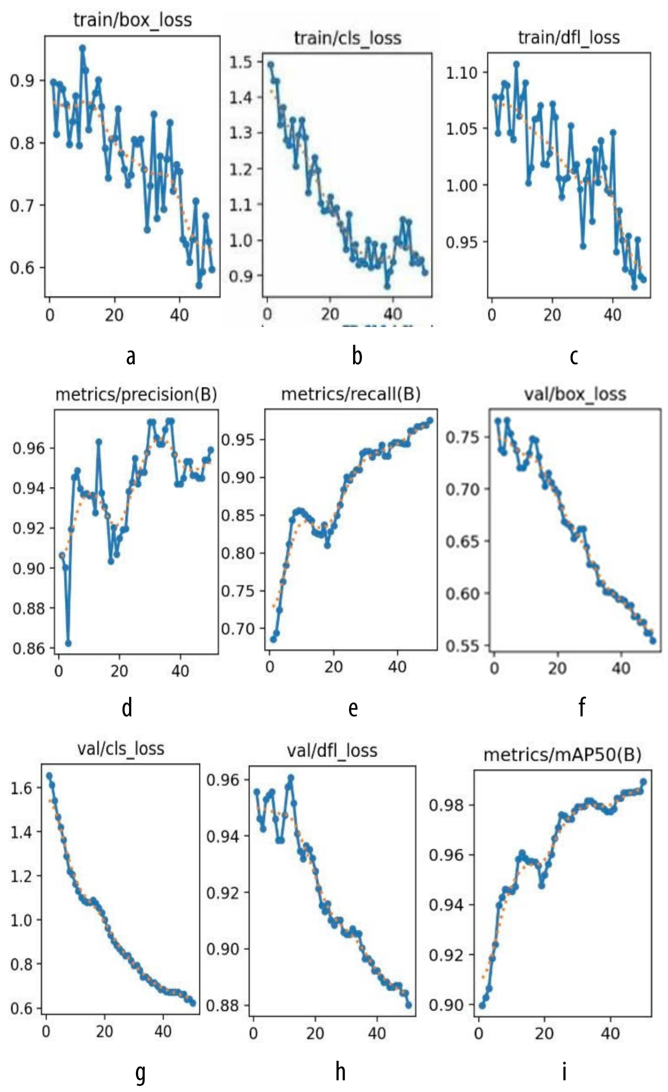
Figure 7. Trained from scratch and a self-trained dataset

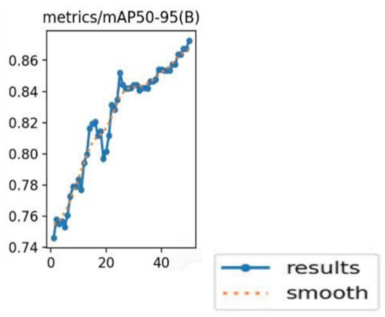
In Figure 7 and Figure 8, we conducted training and evaluation of object detection performance over 50 epochs, with loss and performance metrics recorded on both the training and validation sets. For Figure 7, the loss metrics on the training set showed significant improvement: train/box_loss (bounding box loss on the training set) in Figure 7(a) decreased from 1.4 to 0.8, showing enhanced ability to accurately locate objects; train/cls_loss (classification loss on the training set) in Figure 7(b) dropped sharply from 3.5 to 1.5, reflecting major progress in correctly classifying objects and train/dfl_loss (Distribution Focal Loss on the training set, a loss type for improving box prediction) in Figure 7(c) reduced from 1.3 to 1.0 after 50 epochs, improving the detailed accuracy of bounding boxes. On the validation set, val/box_loss (bounding box loss on the validation set) in Figure 7(f) decreased from 1.2 to 0.8, val/cls_loss (classification loss on the validation set) in Figure 7(g) from 3.5 to 2.0, and val/dfl_loss (Distribution Focal Loss on the validation set) in Figure 7(h) from 1.2 to 0.95, indicating good learning but slight overfitting due to the gap between train/cls_loss (1.5) and val/cls_loss (2.0). Regarding performance, Figure 7 recorded a precision of metrics/precision(B) (precision of bounding boxes) in Figure 7(d) at 0.8, a recall of metrics/recall(B) (recall of bounding boxes) in Figure 7(e) at 0.8, with significant fluctuation in the recall metric ranging from 0.7 to 0.9 across epochs. The metrics/mAP50(B) (mean Average Precision at IoU 0.5 for bounding boxes) in Figure 7(i) reached 0.7, while metrics/mAP5095(B) (mean Average Precision from IoU 0.5 to 0.95 for bounding boxes) in Figure 7(j) only reached 0.55, indicating limited overall performance at higher IoU thresholds.

In contrast, Figure 8 demonstrates superior performance across all evaluated metrics. On the training set, the bounding box loss (train/box_loss), which measures the accuracy of predicted object locations, decreased from 0.9 to 0.65, as shown in Figure 8(a). The classification loss (train/cls_loss), indicating the model's ability to classify objects correctly, decreased from 1.5 to 0.9 in Figure 8(b), while the Distribution Focal Loss (train/dfl_loss), which enhances bounding box precision, dropped from 1.1 to 0.9 in Figure 8(c). These reductions reflect improved learning efficiency compared to Figure 7.

On the validation set, the bounding box loss (val/box_loss) decreased from 0.75 to 0.55 in Figure 8(f), the classification loss (val/cls_loss) from 1.6 to 0.6 in Figure 8(g), and the Distribution Focal Loss (val/dfl_loss) from 0.96 to 0.88 in Figure 8(h). Notably, the small gap between training and validation losses (e.g., train/cls_loss at 0.9 vs. val/cls_loss at 0.6) indicates reduced overfitting and better generalization.

In terms of performance, Figure 8 achieved a precision of 0.94 for bounding boxes (metrics/ precision(B)) as shown in Figure 8(d), and a recall of 0.90(metrics/recall(B)) in Figure 8(e), both exhibiting stable upward trends with minimal fluctuation after 20 epochs. Furthermore, the mean Average Precision at an IoU threshold of 0.5 (metrics/mAP50(B)) reached 0.96 in Figure 8(i). In contrast, the mean Average Precision across IoU thresholds from 0.5 to 0.95 (metrics/mAP50-95(B)) reached 0.86 in Figure 8(j), both significantly outperforming the corresponding values in Figure 7. The training curves (blue lines) and smoothed curves (yellow dotted lines) in Figure 8 also show a consistent improvement with reduced fluctuations after 20 epochs, unlike the more erratic behavior observed in Figure 7, especially in metrics such as recall and mAP.





j

Figure 8. The version that was fine-tuned on a self-trained dataset

Overall, Figure 8 outperforms Figure 7 across nearly all dimensions. In terms of loss, Figure 8 shows markedly lower values on both the training and validation sets (e.g., val/box_loss of 0.55 vs. 0.8, val/cls_loss of 0.6 vs. 2.0), indicating enhanced learning, generalization, and reduced overfitting due to the smaller gap between the two datasets. Regarding performance metrics, Figure 8 achieved higher scores: precision at 0.94 vs. 0.8, recall at 0.90 vs. 0.8, mAP50 at 0.96 vs. 0.7, and mAP50-95 at 0.86 vs. 0.55, indicating more accurate and robust object detection, particularly at higher IoU thresholds. Additionally, Figure 8 exhibited greater training stability, with key performance metrics (precision, recall, mAP) displaying reduced variance after 20 epochs, whereas Figure 7 exhibited notable fluctuations, especially in recall (ranging from 0.7 to 0.9). These results suggest that Figure 8 not only delivers superior performance but also offers greater robustness and reliability, making it the preferred model for the object detection task in this study.

4. CONCLUSION

This research successfully developed an automated monitoring system integrating YOLOv11, OpenCV, and ByteTrack for realtime target detection, recognition, and notification in security camera applications. The system achieved a fairly high detection rate on constrained hardware, leveraging YOLOv11's advanced capabilities. Real-time Telegram notifications transformed passive monitoring into a proactive security solution. The YOLOv11n model, fine-tuned on a custom dataset, outperformed its scratch-trained counterpart, reducing overfitting and enhancing training stability. By automating detection and alerting, the system overcomes traditional limitations such as human error and delayed responses. The study contributes to the academic literature by evaluating YOLOv11n in real-world security contexts, integrating ByteTrack for object tracking, and establishing comprehensive performance metrics. However, challenges include low processing speed due to the absence of GPU support, inconsistent reliability in complex conditions, and the need for validation on real-world datasets. Future development should focus on GPU optimization, improving small object detection, applying stricter confidence thresholds to reduce false positives, testing on diverse datasets, and addressing privacy concerns. This research highlights the transformative potential of AI-driven surveillance technology in creating smarter, more efficient, and reliable security solutions.

REFERENCES

[1]. Z. Zou, K. Chen, Z. Shi, Y. Guo, J. Ye, "Object detection in 20 years: A survey," in *Proceedings of the IEEE,* 111, 3, 257-276, 2023. doi: 10.1109/JPROC.2023.3238524.

[2]. Q. Zhao, P. Zheng, S.-T. Xu, X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, 30, 11, 3212-3232, 2019. doi: 10.1109/TNNLS.2018.2876865.

[3]. Y. Amit, P. Felzenszwalb, R. Girshick, "Object detection," in *Computer Vision,* K. Ikeuchi, Ed. Cham: Springer, 660-670, 2021. doi: 10.1007/978-3-030-63416-2_660.

[4]. P. Jiang, D. Ergu, F. Liu, Y. Cai, B. Ma, "A review of YOLO algorithm developments," *Procedia Computer Science,* 199, 1066-1073, 2022. doi: 10.1016/j.procs.2022.01.135.

[5]. J. Terven, D. M. Córdova-Esparza, J. A. Romero-González, "A comprehensive review of YOLO architecture in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Machines, Learning, and Knowledge Engineering*, 5, 4, 1680-1716, 2023. doi: 10.3390/make5040083.

[6]. M. Hussain, "YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection," *Machines,* 11, 7, 677, 2023. doi: 10.3390/machines11070677.

[7]. H. J. Mun, M. H. Lee, "Design for visitor authentication based on face recognition technology using CCTV," *IEEE Access,* 10, 124604-124618, 2022. doi: 10.1109/ACCESS.2022.3223374.

[8]. S. Narejo, B. Pandey, D. E. Vargas, C. Rodriguez, M. R. Anjum, "Weapon detection using YOLO v3 for smart surveillance system," *Mathematical Problems in Engineering,* 1, 9975700, 2021. doi: 10.1155/2021/9975700.

[9]. P. Vijayakumar, G. Praveen Santhoshkumar, M. Pyingkodi, C. Shwetha, S. Sibitha, K. Vedha Shruthi, "Deep learning based smart security camera system using YOLO algorithm in security setting," in *2024 2nd International Conference on Self Sustainable Artificial Intelligence Systems* (ICSSAS), Erode, India, 703-709, 2024. doi: 10.1109/ICSSAS64001.2024.10761043.

[10]. S. Al-E'mari, Y. Sanjalawe, H. Alqudah, "Integrating enhanced security protocols with moving object detection: A YOLO-based approach for real-time surveillance," *in 2024 2nd International Conference on Cyber Resilience* (ICCR), Dubai, United Arab Emirates, 1-6, 2024. doi: 10.1109/ICCR61006.2024.10532863.

[11]. S. Jyothis, A. S. Nair, K. VR, R. M., D. Visakh, "Developing a SmartRail security system with YOLO and OpenCV," *Journal of Ubiquitous Computing and Communication Technologies*, 6, 1, 28-38, 2024.

[12]. S. M. Keerthana, R. Sujitha, P. Yazhini, "Weapon detection for security using the YOLO algorithm with email alert notification," in 2024 *International Conference on Innovations and Challenges in Emerging Technologies* (ICICET), Nagpur, India, 1-6, 2024. doi: 10.1109/ICICET59348.2024.10616365.

[13]. Ultralytics, *YOLOv11: Model overview*. 2025. [Online]. Available: https://docs.ultralytics.com/vi/models/yolo11/. [Accessed: 26 Mar. 2025].

[14]. Ultralytics, *Tracking: YOLOv11 model overview*, 2025. [Online]. Available: https://docs.ultralytics.com/vi/modes/track/. [Accessed: 26 Mar. 2025].

[15]. Ultralytics, *Dataset: Common objects in context*, 2025. [Online]. Available: https://docs.ultralytics.com/datasets/detect/coco/. [Accessed: 26 Mar. 2025].

THÔNG TIN TÁC GIẢ

Trần Thị Thùy Linh[1], Hoàng Trọng Nghĩa[1], Nguyễn Thị Ngọc[1], Trần Thị Mỹ Kim[1], Nguyễn Văn Sơn[1], Nguyễn Hoài Giang[1], Vũ Duy Thuận[2]

[1]Khoa Điện - Điện tử, Trường Đại học Mở Hà Nội

[2]Khoa Điều khiển và Tự động hoá, Trường Đại học Điện lực