AI Based Embedded Vision System for Real-Time Intelligent Surveillance (IoT Enabled)

B. Navya Sree[1], B. Harshitha[2], R. Sowmya[3], V. Raghavendra[4]

[1]Undergraduate Student, Department of Electronics & Communication Engineering
[2]Undergraduate Student, Department of Electronics & Communication Engineering
[3]Undergraduate Student, Department of Electronics & Communication Engineering
[4]Assistant Professor, Department of Electronics & Communication Engineering
Sreenidhi Institute of Science & Technology, Ghatkesar, Hyderabad-501301

[1]bele.navyasree26@gmail.com, [2]harshithabandari21@gmail.com, [3]rallabandisowmya67@gmail.com

***Abstract*— There has been a big rise in the need for smart surveillance systems in the last few years because people are worried about security in both public and private places. Conventional surveillance systems depend significantly on ongoing human oversight, which is ineffective, labor-intensive, and susceptible to mistakes. This project offers an AI-based embedded vision system for smart intelligent surveillance, utilizing a Raspberry Pi 4 and OpenCV, to address these constraints. The proposed system will use a webcam to automatically find people and spot suspicious activities in real time. The system processes live video streams, takes frames out of them, and uses computer vision techniques to find people. Further, machine learning models are utilized to classify activities such as fighting, robbery, and normal behavior. The datasets used for training the model are obtained from Kaggle, ensuring improved accuracy and performance.**

***Index Terms*— Surveillance, Raspberry Pi, OpenCV, Activity Detection, Machine Learning.**

I. INTRODUCTION

In the modern era of smart cities and interconnected infrastructure, the demand for robust, intelligent, and automated surveillance systems has never been greater. Traditional surveillance systems, while effective as passive monitoring tools, fall critically short in providing automated threat detection, actionable alerts, and real-time decision-making capabilities. The limitations of human-monitored CCTV systems — including operator fatigue, high manpower costs, inconsistent attention, and delayed response — have driven researchers and engineers toward AI-integrated alternatives.

The rapid advancement of deep learning, computer vision, and embedded computing has created a unique convergence point where it is now feasible to deploy intelligent surveillance on low-cost, power-efficient hardware. This project exploits precisely this convergence, presenting a fully embedded, IoT-enabled intelligent surveillance system built around a Raspberry Pi 4 platform — a credit-card sized single-board computer — equipped with a standard USB webcam and powered by sophisticated AI algorithms.

The system goes beyond simple motion detection or person counting. It incorporates a layered AI pipeline that first identifies humans in the scene and then performs temporal activity analysis to understand what those humans are doing. If a suspicious or dangerous activity — such as fighting, theft, vandalism, or assault — is detected, the system immediately sends an automated alert to designated security personnel via a Telegram Bot notification, complete with a timestamped snapshot for immediate assessment.

This project's driving force is a major gap in current surveillance technology: the failure of traditional systems to interpret human behavior in real time. Thousands of events every day—ranging from minor theft in retail stores to vicious attacks in public areas and vandalism on institutional properties—go unnoticed or unreported until long after the event just because no human operator was watching at the exact time or because the sheer amount of camera feeds made continual monitoring impossible.

II. METHODOLOGY

The system is built on a layered, pipeline-based architecture with five functional layers: (1) Acquisition Layer, (2) Preprocessing Layer, (3) Detection Layer, (4) Recognition Layer, and (5) Alert Layer. Each layer gets processed data from the

layer below and sends its output up, making a clear separation of concerns that lets each part be tested and optimized on its own.

The architecture is meant to work like a pipeline that runs events one after the other. The Acquisition, Preprocessing, and Detection layers all run at full frame rate all the time. The Recognition Layer only turns on when the Detection Layer confirms that people are present. This saves computing power when no one is in view. The Alert Layer only goes off when there is proof of suspicious activity, which stops alert fatigue from false positives.

Hardware Architecture

The Raspberry Pi 4 Model B with 4GB LPDDR4 RAM is the main part of the hardware. It has a quad-core ARM Cortex-A72 processor running at 1.5 GHz and USB 3.0 connectivity for fast camera data transfer. The main sensor is the USB webcam, which can record in 1080p but only works at 720p/30FPS for the target application. For the operating system and model storage, the system needs a microSD card (64GB Class 10 recommended) and a stable 5V/3A USB-C power supply to work well when running AI workloads that use a lot of computing power.

The onboard 802.11ac Wi-Fi module (or Gigabit Ethernet for wired deployments) connects to the network and is only used for Telegram Bot API communication during alert dispatch. The core AI inference pipeline is very important because it doesn't need a network connection, which means it will keep working even in places where the internet is not always available.

Software Architecture

The software stack is made up entirely of Python 3.9 and uses the following important libraries and frameworks:

• OpenCV 4.7+: The main computer vision library for capturing video, preprocessing frames, and making DNN-based model inferences through the cv2.dnn module.

• TensorFlow Lite 2.x / Keras: A framework for training CNN-LSTM models on a development machine and running TF Lite inference on a Raspberry Pi.

• NumPy: Used for managing frame buffers and manipulating feature vectors.

• python-telegram-bot 20.x: An asynchronous Telegram Bot API client for sending alert messages and images.

• Imutils: functions that make it easier to thread video streams and resize frames.

• Pillow (PIL): Take pictures, add notes, and encode JPEGs for Telegram image attachments.

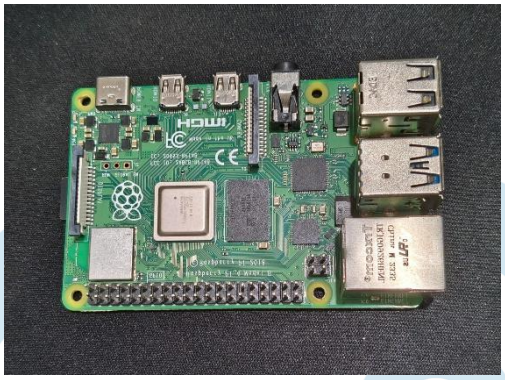
The operating system is Raspberry Pi OS (64-bit, Bullseye), which was chosen because it has the best ARM64 support and works with all the Python libraries that are needed. The 64-bit operating system is necessary for making full use of 4GB of RAM and for working with TensorFlow Lite's ARM64 optimized binaries.

Data Flow Design

The data flow through the system follows a strict order, with branching based on the detection state. The main processing loop works like this:

1. Frame Acquisition: The Video Capture thread gets a frame from the webcam at 30 frames per second and puts it in a queue that is safe for threads.
2. Preprocessing: The main thread takes the frame out of the queue, resizes it to 416x416 (for YOLO input) and 224x224 (for CNN input), normalises the pixel values, and applies histogram equalisation if it sees that the contrast is low.
3. Human Detection: The YOLOv4-Tiny DNN processes the pre-processed frame. If no people are found with a confidence level greater than 0.5, the frame is thrown away and the loop starts over.
4. Frame Buffering: If people are found, the 224x224 frame is added to a circular frame buffer that is 30 frames long.
5. Activity Recognition: The CNN-LSTM model classifies the activity once the buffer has 30 frames.
6. Alert Decision: The alert module is turned on if the predicted class is suspicious (not Normal) and the confidence level is higher than 0.75.

7. Alert Dispatch: The current frame is marked up and sent out through the Telegram Bot API. A cooldown timer stops alerts from going off for the same event more than once (default: 30seconds).



Real-Time Video Processing

1. Video Capture Strategy

Real-time performance on the Raspberry Pi 4 depends on being able to capture video quickly. A simple, single-threaded way to capture video is to have the main loop call cv2.VideoCapture.read() synchronously, which means that the CPU has to wait for the camera driver to send the next frame, which causes a lot of latency. This wait time is about 33ms per frame for our 30 FPS webcam, which makes the overall pipeline throughput very low.

The solution employed is a dedicated capture thread using Python's threading module. A producer thread runs independently, continuously reading frames from the camera and placing them into a thread-safe queue (queue.Queue with maxsize=2 to prevent memory buildup). The main processing thread consumes frames from the queue as fast as it can process them. This producer-consumer pattern decouples frame acquisition from frame processing, ensuring that the camera buffer never overflows and that the processing pipeline always has the most recent frame available.

2. Frame Preprocessing Pipeline

Each captured frame undergoes a standardized preprocessing pipeline before entering either the detection or classification model:

• Resolution Scaling: cv2 changes the size of the frames to 416x416 for YOLO detection. INTER_LINEAR interpolation (which balances quality and speed) and 224x224 for CNN-LSTM classification, which uses cv2.INTER_AREA for downscaling.

• Color Space Normalization: We divide the pixel values by 255.0 to change them from [0, 255] to [0.0, 1.0]. This normalization is necessary for stable inference in neural networks.

• Blob Creation: To do YOLO inference with OpenCV DNN, frames are turned into network blobs using cv2.dnn.blobFromImage() with mean subtraction (0, 0, 0) and swapRB=True to change BGR to RGB. Adaptive Contrast Enhancement: When the mean pixel value is less than 60, this means that the light is low. CLAHE (Contrast Limited Adaptive Histogram Equalization) is used to make features more visible without making the whole frame brighter.

• Managing the Frame Buffer: The circular frame buffer is a collections.deque with a maxlen of 30. When a new pre -processed frame is added, the oldest frame is automatically thrown away. This keeps a sliding window of the last 30 frames (1 second of video at 30 FPS or 3 seconds at 10 FPS).

3. Performance Optimization for Edge Deployment

The Raspberry Pi 4's ARM Cortex-A72 processor, while capable for a single-board computer, is significantly less powerful than the dedicated GPUs used in typical deep learning development environments. Several optimization strategies are employed to achieve acceptable inference throughput:

•Frame Skipping: The activity recognition model runs inference every 10th frame rather than every frame (i.e., at 3 FPS activity inference rate even if video is captured at 30 FPS). Human detection runs every 3rd frame. This dramatically reduces CPU load while maintaining sufficient temporal resolution for activity classification.

• TensorFlow Lite Quantization: INT8 quantization is used to transform the CNN-LSTM model into TensorFlow Lite format, which reduces the model's size by around 4 times and the inference time by 2–3 times on ARM processors with SIMD optimization.

• Multi-threading: Using Python's threading module with thread-safe queues for inter-thread communication, detection and recognition inference runs in different threads from the main capture loop.OpenCV Optimizations: OpenCV is compiled with NEON SIMD optimizations for ARM, and cv2.setNumThreads(4) utilizes all four RPi 4 CPU cores for parallelized convolution operations.

•Lazy Activation: The computationally expensive CNN-LSTM model is not loaded into memory until the first person is detected, reducing startup memory footprint.

System Workflow

The complete system workflow, from camera initialization to alert dispatch, is described step by step:

Step 1 — System Initialization

The system loads the YOLOv4-Tiny model weights and configuration files into OpenCV DNN, initializes the frame buffer as an empty deque, and sets up the Telegram Bot connection by checking the bot token and chat ID. The video capture thread is initialized on startup. A file records system status; a linked monitor may optionally show it.

Step 2 — Continuous Detection Loop

Frames are constantly dequeued from the capture thread by the primary loop. For human detection, YOLOv4-Tiny processes every third frame. At 150–200ms per frame, the YOLO forward pass on Raspberry Pi 4 provides effective detection rates of 5–7 FPS. Bounding boxes on the screen output draw attention to discovered people.

Step 3— Alert Decision and Cooldown

However, if the predicted class is one of the five suspicious classes (Fighting, Theft, Robbery, Accident, and Vandalism) and the greatest probability value surpasses the confidence threshold of 75%, and the alert cooldown timer has expired (i.e., the timer has reached 30 seconds after the last alert was sent), then the system sends an alert to dispatch. This cooldown ensures that Telegram messages are not sent repeatedly for the same ongoing incident.

Step 4— Telegram Alert Dispatch

The alert module captures the current frame, labels it with the activity label, percentage of confidence, timestamp, and a warning border, encodes it as JPEG, and sends it via the Telegram Bot API's send Photo method with a caption in a set format. A dispatch is usually completed within 1–2 seconds on a conventional WiFi connection. A text alert with complete information is sent in a separate send Message call.

III. Results and Discussion

In this chapter, we show the experimental results of the proposed system in terms of the human detection accuracy, activity recognition accuracy, inference latency, system resource usage, and alert delivering performance.

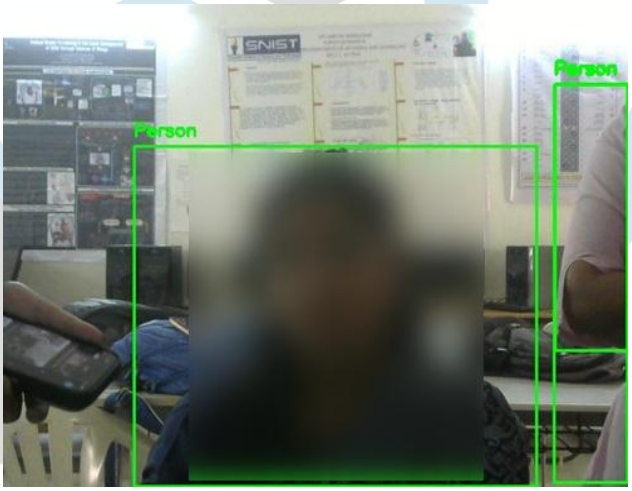
Human Detection Performance

YOLOv4-Tiny detection of persons is evaluated using 500 test images of an indoor surveillance scene with different lighting conditions, number of persons per frame (1-4 persons) and partial occlusion.

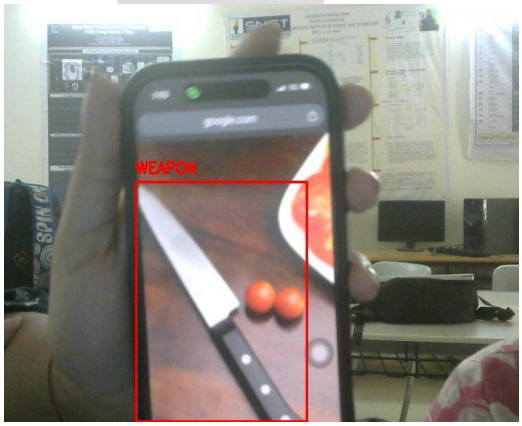
Condition	Precision (%)	Recall (%)	F1 Score (%)	Avg. Inference Time
Good Lighting	94.2	91.8	93.0	178ms
Low Light	87.6	84.3	85.9	183ms
Partial Occlusion	82.1	79.5	80.8	181ms
Multiple Persons (3-4)	89.4	86.7	88.0	192ms
Overall Average	**88.3**	**85.6**	**86.9**	**183ms**

Table: Human Detection Performance Under Various Conditions

1. **Person Detection**



2. **Weapon Detection**



3. Suspicious Activity Detection

System Performance Metrics

End-to-end system performance is measured over a 4-hour continuous operation test

Performance Metric	Measured Value
Video Capture Frame Rate	30 FPS (full resolution 720p)
Person Detection Rate	~7 FPS effective detection
Activity Classification Rate	~1 classification per 3-5 seconds
Average YOLO Inference Time	183ms per frame
Average CNN-LSTM Inference Time	2.1 seconds per 30-frame window
Alert Delivery Latency (Detection → Telegram)	2.4 seconds average
CPU Utilization (Detection active)	78–85% (across 4 cores)
RAM Usage	1.8GB of 4GB
Operating Temperature	58–63°C (with cooling fan)
False Positive Alert Rate	0.8 alerts/hour
Continuous Operation Test Duration	8 hours (no crashes/memory leaks)

Table: System Performance Summary

REFERENCES

[1] Sultani, W., Chen, C., & Shah, M. (2018). Real-world Anomaly Detection in Surveillance Videos. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6479-6488.

[2] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.

[3] Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934.

[4] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4510-4520.

[5] Simonyan, K., & Zisserman, A. (2014). Two-Stream Convolutional Networks for Action Recognition in Videos. Advances in Neural Information Processing Systems (NIPS), 568-576.

[6] Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2015). Long-term Recurrent Convolutional Networks for Visual Recognition and Description. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2625-2634.

[7] Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M. (2015). Learning Spatiotemporal Features with 3D Convolutional Networks. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 4489-4497.

[8] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861.

[9] Ji, S., Xu, W., Yang, M., & Yu, K. (2013). 3D Convolutional Neural Networks for Human Action Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(1), 221-231.

[10] Stauffer, C., & Grimson, W. E. L. (1999). Adaptive Background Mixture Models for Real-Time Tracking. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 246-252.

[11] Laptev, I. (2005). On Space-Time Interest Points. International Journal of Computer Vision, 64(2-3), 107-123.

[12] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems (NIPS), 1097-1105.