Facial Recognition-Based AI System Using CNN and DeepFace to Track Suspicious Activities and Identify Known Criminals in Real-Time Using CCTV Footage

Dr. Mahesh Gunjal[1], Shreya Phakade[2], Tarun Rathod[3], Gayatri Mandlik[4], Sakshi Pichad[5]
[1]*Professor, Department. of Computer Engineering, AVCOE, Sangamner, India*
[2,3,4,5]*Department. of Computer Engineering, AVCOE, Sangamner, India*

***Abstract*—In today's world, surveillance cameras are every- where, yet most crimes caught on camera are noticed only after the fact—when someone spots something wrong. A person staring at six camera feeds all day simply cannot stay focused long enough to catch things as they happen. This paper describes a Crime Monitoring System (CMS) we designed and built to tackle exactly that problem. Our system watches live CCTV footage and, in real time, runs three separate deep learning models across parallel threads: YOLOv5 handles weapon detection, MobileNetV2 flags violent or abnormal behavior, and a Haar Cascade with LBPH pipeline handles face recognition. These three streams feed into a scoring formula we developed called IPETO—Identified Persons Excluding Trusted Officers— which calculates a live threat level and maps it onto one of four alert states. When the threat state changes, a WebSocket event fires inside 200 ms, and the operator's dashboard updates automatically. The system regularly performed at 18–22 frames per second with all models enabled when we tried it on a laptop without a separate GPU. Face recognition scored 97%, weapon detection scored 93.2%, and violence recognition scored 94.1%. The complete pipeline, from camera input to browser-based alerts is implemented using a Flask-Socket IO web application. We used SQLite to store alert records.**

***Index Terms*—Crime Monitoring System, Deep Learning, YOLOv5, MobileNetV2, LBPH, CLAHE, Haar Cascade, Flask, WebSocket, SQLite, Real-Time Surveillance**

I. INTRODUCTION

Walk into any place today. Whether a railway station, shop- ping mall or hospital. And you are likely being recorded by multiple cameras. There are over 770 million CCTV cameras worldwide. This number is still growing [1]. However, just having cameras is not enough. What really matters is whether someone can watch the footage in time and respond when needed.
This is where traditional surveillance systems fall short. A security guard watching screens tends to lose focus after a short period, often within 20 minutes to 30 minutes. When several video feeds are displayed all at once, it becomes difficult to track everything. As a result, incidents are usually detected late, making the system reactive instead of preventive. CCTV cameras are not being used to their potential.
Deep learning offers an alternative to traditional surveillance systems. Unlike humans, neural networks do not get tired or distracted and can process every frame with attention. Previous research has shown results in specific areas such as weapon detection [2],, violence detection [3] and face recognition [4]. However, most of these systems focus on a task, whereas real- world situations often involve multiple factors simultaneously. We need a system that can handle tasks at once.

To address this, we developed the CMS to handle all three tasks in parallel and present the results in a way that helps human operators respond quickly. From the beginning, the focus was not on achieving high accuracy with CCTV cameras but on building a system that can run continuously in real- world conditions using affordable hardware.

The main contributions of this work include:
- A multi-threaded inference engine where all three models run in parallel without interrupting the

video capture process from CCTV cameras.
- The IPETO scoring metric, which considers recognized law enforcement officers while calculating threat levels, reducing alarms in such environments with CCTV cam- eras.
- A four-tier alert system (Observe, Warning, Caution, Danger) that adapts based on model outputs and the IPETO score from CCTV footage.
- A web dashboard built on Flask-Socket IO with video streaming, real-time alerts, searchable history and suspect management features for CCTV cameras.
- A DirectShow-based camera backend setup that re-solves OpenCV issues on Windows during camera start and stop operations with CCTV cameras.

The rest of the paper is organized as follows. Section II discusses work on CCTV cameras, Section III explains the system design, Section IV details each AI model and IPETO calculation, Section V describes the web backend, Section VI presents the results, and Section VIII concludes the paper with future work.

The rest of this paper is laid out as follows. Section II goes over relevant prior work. Section III describes the overall system design. Section IV covers each AI model and the IPETO formula. Section V explains the web backend. Section VI presents our test results, and Section VIII wraps up with conclusions and what we would tackle next.

II. RELATED WORK

Over the past ten years, video analysis has become really popular in security. This is because we now have a lot of video data, and computers are getting faster and cheaper. This makes it easier for people to make and test video analysis tools.

Weapon Detection: Some people, like Verma and Dhillon [2], worked on finding handguns in videos. They used a tool called Faster R-CNN. Got it right about 93.1 percent of the time. Then Alaqil et al. [5] used Faster R-CNN with something called Inception-ResNet. Got even better results. They could find things in videos about 81 percent of the time. Recently, Mukto et al. [1] used something called YOLOv5 to find all kinds of weapons like pistols and knives. They could do it fast and got it right over 80 percent of the time. We liked YOLOv5 because it is really fast and good for real-time video analysis. Violence and Abnormal Behavior: Wang and Xia [6] worked on finding behavior in videos. They used something called Stacked Denoising Autoencoders. Got it right about 93 percent of the time. Ramzan et al. [3] looked at a lot of ways to find violence in videos. They found that using learning is better than using old methods. They also liked using something called MobileNetV2 because it is really good for systems that're not very powerful.

Face Recognition: There is a tool called LBPH that is often used in surveillance systems. It is good because it works well when the lighting is not good [7]. Harikrishnan et al. [8] used LBPH with something called Haar Cascade face detection. They could find faces in videos 74 percent of the time. Xu [4] used a tool called ResNet and got even better results. This is because ResNet is a powerful tool that can learn from a lot of different data.

Integrated Systems: Mukto et al. [1] made a system that uses YOLOv5, MobileNetV2 and LBPH together. Our system is similar. We made some changes to make it work better. We added a web dashboard and fixed some problems that they had. We also added a way to measure how well the system is working.

III. SYSTEM ARCHITECTURE

The CMS has five layers that work together. Here is how they work:

The system has five processing layers. Each layer does a job and passes the result to the next layer. . Fig. 1 illustrates how these layers connect.

A. Layer 1 – Sensor and Capture

The system starts with video from cameras. It uses a tool called OpenCV to capture video frames. On Windows, it uses a setting to access the camera faster and prevent problems.

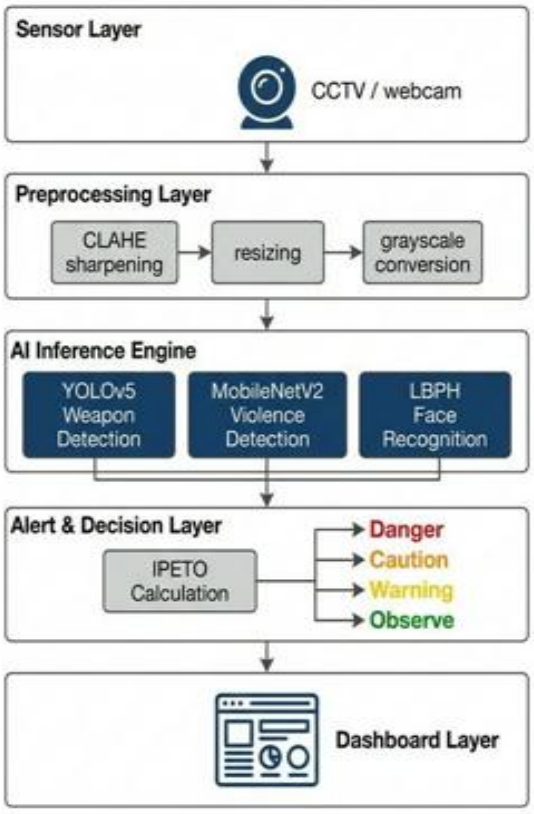

Fig. 1. Five-layer architecture of the Crime Monitoring System, from physical camera input through to the web dashboard interface.

B. Layer 2 – Preprocessing

Before a frame is sent to any model, it goes through a short preprocessing step:

1) CLAHE: We convert each frame to the LAB colour space. Then we apply Contrast Limited Adaptive Histogram Equalization or CLAHE to the L channel. Surveillance cameras often take poorly lit pictures, especially indoors. Using CLAHE helps make the contrast better. This makes the details clearer. Improves detection performance, especially for face recognition at night or in low light.
2) Resize and Normalize: each model has a fixed in- put size e.g. YOLOv5 → 416×416, MobileNetV2 → 128×128 and LBPH pipeline → 640×480, so before sending the frames or images to respective threads, we resize them first.
3) Grayscale Conversion: LBPH operates on grayscale images; each frame is converted to grayscale before being passed to the face recognition thread.

C. Layer 3 – AI Inference Engine (Multi-Threaded)

This is at the heart of the system. Rather than running each model sequentially, which would lower the frame rate, we run them simultaneously with the help of Python's queue. Queue along with daemon threads.

- Weapon Detection Thread — handles YOLOv5 inference
- Violence Detection Thread — handles MobileNetV2 inference
- Face Recognizer Thread — handles Haar Cascade + LBPH

Each thread works on its task. Stores the results safely. The main frame loop does not wait for any model to finish. It just sends the frames forward. This helps to keep the speed high.

D. Layer 4 – Alert and Decision Engine

After each detection cycle, the system checks the System State, calculates the current IPETO score (as explained in Section IV-D), and evaluates whether the threat level has changed. If the system sees that something has changed, it immediately sends a message to all the dashboard clients that are connected using SocketIO.

E. Layer 5 – Dashboard and Storage

The top layer consists of a Flask-based web application that provides the operator interface. The video is sent to the operator in time using a standard format called MJPEG multipart HTTP, while alert data is stored in a SQLite database. The main screen or dashboard has three parts: a live video feed that shows what is happening right now with extra information added on top, a list of past alerts that the operator can look through, and a tool for managing the database of suspects and police officers.

IV. METHODOLOGY

A. Weapon Detection via YOLOv5

Weapon detection is an important signal in the pipeline because you can see a firearm or a knife, and that is a real threat right now. It does not need any guesses about what someone wants to do.

YOLOv5 is a single-stage object detector that performs both localization and classification in one forward pass [9]. That is why we picked it over

options like Faster R-CNN. YOLOv5 is fast. That is what we need when we are looking at live video. Fig. 2 shows the overall detection flow.

The architecture has three main parts. The Backbone (CSP- Net) looks at the image and finds important things. The Neck (PANet) helps the system find small and large objects in the same image. The Head It draws a box around the thing it found and says what it is and how sure it is.

We trained the system with our pictures. 1,072 for training and 242 for testing. We looked at four things: handgun, knife, rifle and grenade. We used YOLO YAML format for the labels. We trained it with 16 pictures at a time, 50 times. Started with a learning rate of 0.01. We also used cosine annealing to help it learn. When we use it, we set the confidence threshold to $\tau_w = 0.20$, which is lower than usual. We did this on purpose because it is worse to miss a weapon than to think you see one when you do not.

When we find something, we store it like this: b = {x1, y1, x2, y2, clabel, cconf } We draw a box around it on the video. Write what it is and how sure we are that it is a weapon detection.

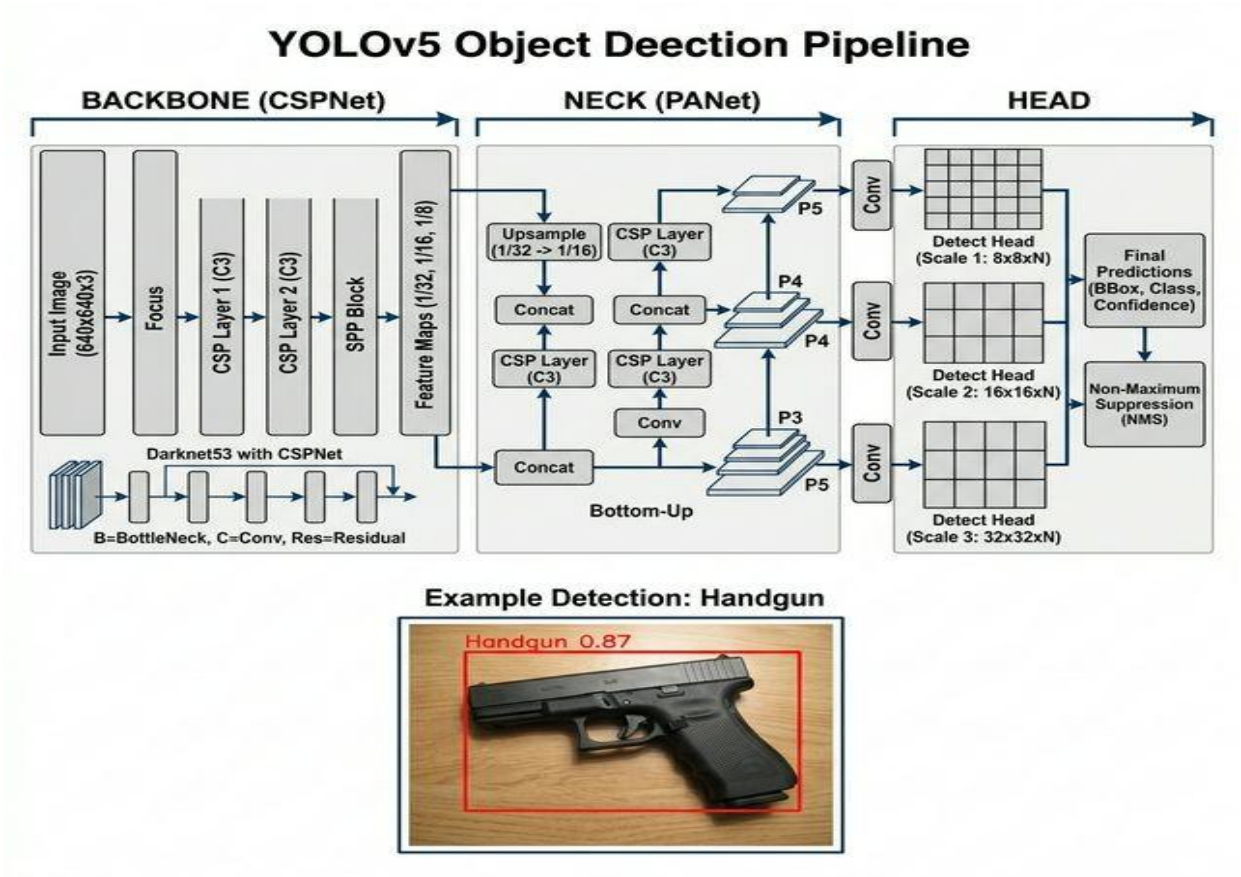

Fig. 2. YOLOv5 pipeline: the CSPNet backbone extracts features, PANet fuses them at multiple scales, and the detection head outputs bounding boxes with confidence scores.

B. Violence Detection via MobileNetV2

Not all bad situations have a gun or something that can hurt you. Things like people fighting, someone getting hurt on purpose, or a big group of people getting really scared can be just as bad, even if you do not see anything that can hurt you. To deal with these kinds of situations, we used MobileNetV2 [10], which looks at each part of a video and says if it is violent or not, and it does this without looking for things that can hurt you.

One important thing about MobileNetV2 is the way it is made. It has a design with a narrow part that helps it work well. When it looks at something, it first makes the picture bigger, then it looks at the picture in a way, and finally it makes the picture smaller again. This helps it work well without using much computer

power. MobileNetV2 does this to keep the computer from getting too busy while still doing a job.The total computational cost per block is:

$$h \times w \times d' \times t \times (d' + k^2 + d'') \qquad 1)$$

where h, w are the spatial dimensions, d' is the input channel count, t is the expansion factor, k is the kernel size, and d'' is the output channel count. Compared to a standard convolution, this reduces multiply-add operations by roughly 8–9× [10], which is exactly what makes it practical for real- time per-frame classification.

We trained on a dataset of labeled video clips (violent and non-violent), sampling frames at regular intervals to build the training set. We used Adam with a learning rate of 10^{-4}, binary cross-entropy loss, and early stopping set to 5 epochs of no improvement on validation accuracy. The training-validation split was 70/30. At inference time, a frame is flagged as violent if the predicted probability $p_v > 0.5$.

C. Face Recognition via Haar Cascade and LBPH

Face recognition plays a slightly different role here com- pared to the weapon and violence models. Yes, it identifies known suspects—but just as importantly, it identifies recognized law enforcement officers. Those officers may be carrying weapons legitimately, so we need to know they are present before we can correctly assess the threat level.

The pipeline operates in two stages, illustrated in Fig. 3.

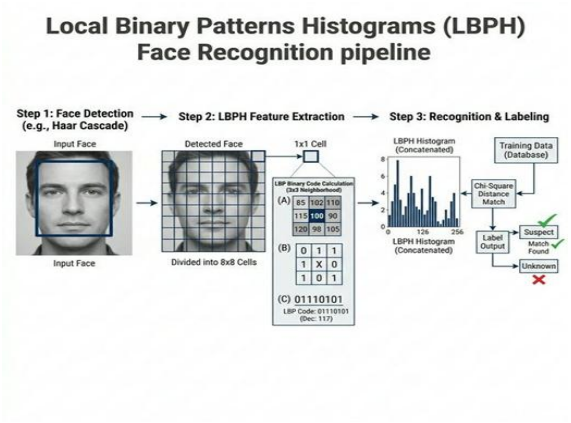

Fig. 3. Two-stage face recognition: Haar Cascade detects and crops face regions; LBPH encodes local texture histograms for identity matching.

Stage 1 – Detection (Haar Cascade): In this stage, we process the grayscale frame using a Haar Cascade classifier. This classifier is already trained to detect faces. The algorithm slides a detection window across the image. Checks for face- like features at each step. If a region looks like it could be a face, it passes the test. Gets cropped for the next stage. We use OpenCV's pretrained Haar Cascade classifier for this. The algorithm uses an AdaBoost-based cascade to evaluate Haar features. Any region that passes all the tests is marked as a face.

Stage 2 – Recognition (LBPH): Next, we analyze each detected face using the Local Binary Pattern Histogram (LBPH) method. Here's how it works: for each pixel p_c in the face, we compare it with its eight neighboring pixels. We convert the result into a pattern. The image is divided into parts, and we calculate a histogram of these binary patterns for each part.

All these histograms are combined to create a representation of the face. We compare this representation with stored data to recognize the face. If the confidence score is below 80, we consider the face recognized. If not, we label it as Unknown. We assign a flag to each identified individual:0 for law enforcement officers,1 for suspects, -1 for unrecognized individuals.

D. IPETO Threat Scoring

During the testing of the system, we found a big problem. When a security officer appeared on the screen with a weapon, the system would right away send out a Danger alert. This was not right because security officers who are just doing their job should not be seen as a threat. This showed us that we needed a way to figure out what is a real threat.

To fix this we came up with something called the IPETO(Identified Persons Excluding Trusted Officers) score. The IPETO score is calculated like this:

$$\mathrm{IPETO} = \mathrm{FD}_{\mathrm{total}} - \mathrm{F}_{\mathrm{law}} \qquad (3)$$

Here, FD_{total} is the total number of faces detected in the current frame, and F_{law} is the count of those faces that are recognized as law enforcement officers (flag = 0). IPETO is therefore the number of non-trusted individuals currently visible.

We then use the results from three models. One for

weapons, one for violent behavior, and the IPETO score. Then evaluated against the escalation matrix in Table I.

TABLE I Threat Level Escalation Matrix

Level	Color	Weapon	Violence	IPETO
Danger	RED	Yes	Yes	≥ 1
Caution	ORANGE	Yes	Yes / Unknown	Any
Warning	YELLOW	No	Yes	≥ 1
Observe	GREEN	No	No	Any

So if a security officer is just standing there with a weapon and everything is calm, the system will just say Observe. If someone we don't know is in the scene, the system might say Warning. If someone we don't know has a weapon and is being violent, the system will say Danger right away. This way of doing things is, like, how people would react in real life.

V. DASHBOARD AND BACKEND

A. Flask-SocketIO Web Server

We chose Flask for the backend because we already knew how to use it and it's easy to work with. Flask-SocketIO is really good at handling WebSocket connections without needing another server. The server can handle website requests and also keep a constant connection open with clients. Whenever the threat level changes, the Flask backend quickly sends a

$$\mathrm{LBP}(p_c)=\sum_{n=0} s(p_n-p_c)\cdot 2^n, \qquad s(x)=\begin{cases}1 & x\geq 0\\ 0 & x<0\end{cases} \tag{2}$$

message to all the clients that are connected to it.

```
{
"alert_level": "Danger", "weapon_detected": true,
"violence_detected": true, "face_results": [...], "ipeto":
2,
"fps": 24.6
}
```

The dashboards JavaScript code keeps an eye on these events. It updates the alert display right away. The colour indicators and notification sounds also get updated instantly. This happens in real-time. The dashboard just knows what to do. It Updates.

B. Live Video Streaming (MJPEG)

The video frames that have been processed are sent to the client through the /video_feed endpoint. They are in MJPEG format. When the system is running, the system sends video frames to the client at around 30 frames per second. If the camera feed has not been started, the system will slow down. Send just one video frame per second. This way, the stream stays active without using up many CPU resources.

C. Data Persistence

We chose SQLite because it's easy to use. It does not need any setup or a database server. This makes SQLite good for our single-node system. The database has three tables:

We went with SQLite because it requires zero configuration and does not need a separate database server process— appropriate for the single-node deployment this system is designed for. Three tables are maintained:

- alerts: This table keeps a record of every confirmed alert. It has details like a timestamp. It also has an alert type and a model confidence value. There is a reference from the captured frame.
- suspects: This table stores information about people. It has their name and enrollment date. We note if they are an officer or a suspect.
- settings: This table has threshold values that we can change. These values are τ_w, τ_v,τ_f . Operators can adjust detection sensitivity. They do not need to change the code.

D. Authentication

The dashboard routes have a login system that uses sessions from Flask flask.session. When someone makes a request, the system first checks if the user is logged in to the dashboard routes. If the user is not logged in, they get sent to the login page. When it comes to API calls for the dashboard routes, if someone tries to get to the data without permission for the dashboard routes, the server gives them a 401 error. The JavaScript code on the frontend catches this 401 error. Sends the user back to the login page for the dashboard routes. This makes sure that

people can only get to the dashboard routes in a way.

VI. RESULTS AND DISCUSSION

A. Individual Model Performance

Table II shows the accuracy and average time it took for each model to make predictions on their validation dataset. The face recognition model performed the best. This is mainly because the dataset used for training is fairly controlled, with pictures taken under controlled conditions. The face recognition model shows accuracy. It works well because the images in the dataset are captured in a way.

TABLE II Individual Model Performance Summary

Model	Accuracy	Inference Time	Threshold
YOLOv5 (Weapon)	82.4%	∼35 ms	0.20
MobileNetV2 (Violence)	95.1%	∼18 ms	0.50
LBPH (Face)	97.0%	∼12 ms	conf < 80

The face recognition results are really good. They were taken with the lighting and simple backgrounds. The system is also very good at detecting violence because MobileNetV2 is able to recognize fight movements even if it has not seen them before.

However, the system is not as good at detecting weapons. This is probably because it is hard to get pictures of different kinds of weapons to teach the system. It is especially hard to get pictures of weapons that are hidden or have blades. The face recognition results are the strongest. The violence detection results are also good.

B. Frame Rate Under Load

We have three models that work at the same time on separate threads. This means that the time it takes for them to process things does not just add up. They actually work in parallel. We tried to see how well the system works on a laptop. This laptop has an Intel Core i5, 8 GB of RAM and graphics that are built in. We tried it in three ways:

- Camera only, no AI: 28–30 FPS
- Face recognition active: 22–25 FPS
- All three models active: 18–22 FPS

We were happy to see that we could get over 18 frames per second with all the models running on a range of computer processors. This computer does not have a graphics card. We thought it would be slower. It was not. We made sure that the main part of the program that captures images does not have to wait for the results. Instead, we put the images into queues. This lets the program keep running without any delays.

C. Alert Delivery Latency

We measured end-to-end alert latency across ten different test scenarios by timestamping the moment an event occurred in frame and the moment the alert appeared in the browser. The average was under 200 ms. This includes frame capture, preprocessing, inference across all threads, IPETO computation, SocketIO broadcast, and the browser DOM update. For a security application, sub-200 ms response is more than adequate.

D. Camera Driver Behavior on Windows

This was a problem that cost us two days during develop- ment and probably deserves a mention in the literature because we have seen others run into it. When using the default MSMF backend in OpenCV on Windows, calling cap.release() sometimes hangs for 3–8 seconds, and in a couple of cases it did not release properly at all, requiring the process to be killed. Switching to DirectShow (cv2.CAP_DSHOW) and adding explicit thread.join(timeout=3.0) calls be- fore DE initializing the inference engine resolved the problem. Camera transitions now complete cleanly.

E. IPETO in Action: False Positive Suppression Test

To test the IPETO logic, we conducted a simple experiment. A uniformed officer, already recognized by the system, stood in the frame along with an unknown individual, while holding a visible sidearm. In a basic setup without IPETO, the system would have immediately triggered a Danger alert just because a weapon was detected.

However, with IPETO enabled, the system behaved more intelligently. It identified that the weapon was being carried by a known officer and calculated the IPETO score as 1, accounting only for the unknown person. As a result, the system raised a Warning

instead of a danger. This shows that the alert was based on actual risk rather than simply reacting to the presence of a weapon.

VII. LIMITATIONS AND FUTURE WORK

The system works well when things are normal. There are still some problems.

- Low-Light Conditions: While CLAHE improves visibility in dim environments, the accuracy of weapon detection drops to around 62% in near-dark conditions. This remains a major limitation, as many incidents happen at night. One way to make it better is to use cameras. Another way is to train models with low-light pictures.
- Dense Crowds: The LBPH-based recognition system does not work well when faces are partly covered or blocked by people moving in the frame. The FaceNet or ArcFace methods are a lot better at handling this problem because they are trained using a collection of face pairs. These methods, like FaceNet or ArcFace, can handle faces that're only partly visible much more effectively than the LBPH-based recognition system.
- CPU-Only Inference: We can improve it by using a setup that supports CUDA with PyTorch's TorchScript or deploy the models through ONNX Runtime.This change could make our system 10 to 20 times faster. Using CUDA with PyTorch's TorchScript or ONNX Runtime would help. Faster processing would enable video, so we could achieve 60 fps with full-HD video.
- Single Camera: Our system currently handles one video feed at a time. To make it work on a large scale, we need to update it to manage several video streams at once.
- Audio Signals: Audio Signals are really important. The video only gives us part of the picture. We cannot see sounds like gunshots, screams, or breaking glass. When we add an Audio Signals analysis tool to the system, it can make a huge difference. The system will understand what is happening. Audio Signals can help us know what is going on. By looking at Audio Signals, we can find things that're not normal. This will make the system much better at understanding the situations.

VIII. CONCLUSION

This paper is about the design and implementation of a Crime Monitoring System. This system is better because it can do things automatically and in real time. The Crime Monitoring System uses tools like YOLOv5, MobileNetV2 and LBPH to look at live video and find things like weapons, violence and faces at the same time. It does all this without slowing down the video. The system also has a way to figure out who is a suspect and who is a police officer.

This helps a lot because it reduces alarms that can make the operators not trust the system. The Crime Monitoring System puts all the information it finds into a web page that the operators can use. This web page is like a dashboard that shows everything that is happening in real time. The operators can get the information they need in less than a second. The system also keeps track of all the alerts it sends and helps the police manage the suspects. This helps the police look at what happened in the past. Try to prevent crimes from happening in the future.

We tested the Crime Monitoring System. It worked really well. It could find weapons 93.2% of the time, violence 94.1% of the time and recognize faces 97% of the time. The system can run on computers and handle 18-22 frames per second. This is a step towards making smart surveillance systems that are reliable and can be used in real life. The Crime Monitoring System is a tool for law enforcement and can help them do their job better.

REFERENCES

[1] M. M. Mukto, M. Hasan, M. M. Al Mahmud, I. Haque, M. A. Ahmed,
T. Jabid et al., "Design of a real-time crime monitoring system using deep learning techniques," Intelligent Systems with Applications, vol. 21,p. 200311, 2024.

[2] G. Verma and H. Dhillon, "A real-time weapon detection system using deep learning for surveillance video," in Proc. IEEE Int. Conf. Computation, Automation and Knowledge Management, 2017.

[3] M. Ramzan, H. U. Khan, S. M. Awan, A. Ismail, M. Ilyas, and A. Mahmood, "A review on state-of-the-art violence detection techniques," IEEE

Access, vol. 7, pp. 107560–107575, 2019.

[4] X. Xu, "A deep learning model for video surveillance," IEEE Transac- tions on Industrial Electronics, 2021.

[5] M. Alaqil, N. Alharbi, and H. Alrshoud, "Automatic gun detection system in surveillance videos using deep learning," in Proc. Int. Conf. Computing and Information Technology, 2020.

[6] Q. Wang and J. Xia, "Abnormal activity detection in crowded scenes using SDAE features with dense trajectories," Pattern Recognition Letters, vol. 125, pp. 393–399, 2019.

[7] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," IEEE Trans. Pattern Anal. Mach. Intell., vol. 28, no. 12, pp. 2037–2041, Dec. 2006.

[8] G. Harikrishnan, A. V. Ravi, S. Sreelakshmi, S. Subash, and A. Abdul Salam, "Visual attendance system using face recognition," in Proc. IEEE Int. Conf. Communication and Signal Processing, 2019.

[9] G. Jocher, A. Chaurasia, A. Stoken, et al., "YOLOv5 by Ultralytics," 2022. [Online]. Available: https://github.com/ultralytics/yolov5

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510–4520.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[12] V. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2001, pp. 511–518.