OBJECT DETECTION FOR HOME SECURITY

1.Project Overview:

The project aims to develop a real-time home security system that detects and classifies objects such as people, pets, packages, or anomalies using live camera feeds. The system uses advanced object detection models (YOLOv3, YOLOv5, YOLOv8) to distinguish between normal and suspicious activity with high accuracy and minimal false alerts. This system is intended to be reliable for homeowners, renters, and small businesses seeking enhanced automated surveillance.

Inputs and outputs:

- Inputs - Real-time webcam video or prerecorded footage
- COCO dataset (80 object classes) for training/testing

Intermediate Outputs:

- Detected bounding boxes and confidence scores
- Real-time annotated video stream

Final Outputs:

- Labeled objects in video frames
- Alert notifications (via email) upon detecting specific "danger" classes like humans

2.Approach:

Model Implementation

- Utilized **YOLOv3, YOLOv5, YOLOv8** through the Ultralytics API.
- Each model was evaluated on the **mini-COCO dataset** and webcam stream.

Real-Time Detection (via webcam.py)

- Captures live frames from webcam using OpenCV.
- Applies YOLOv5 (yolov5m.pt) for prediction on each frame.
- Displays predictions on-screen with bounding boxes and class labels.

Dangerous Object Alerting

- List of monitored classes: "knife", "fire", "gun", "person" (we used these classes).
- When a dangerous object is detected, an email alert is sent using the Gmail SMTP server.
- Alert is triggered only once per session to avoid spamming (already_alerted flag).
  Email Configuration (configuration.py)
- Centralized config for **sender/receiver emails and app password**.
- Used smtplib and EmailMessage for secure email delivery (SMTP_SSL on port 465).

3.Experimental Protocol:

Dataset Setup:

- Trained and tested on **COCO-mini dataset** (subset of COCO with 80 classes).
- Frame resizing to 640x640, batch size of 8.
- Pixel normalization and filtering of low-quality images.

Model Training/Evaluation

- Used Ultralytics' YOLO() for model loading and training.
- Compared models based on:
- **mAP (mean Average Precision)**
- **Precision & Recall**
- **FPS (real-time performance)**
- **Per-class classification report**

Real-Time Inference

- Used cv2.VideoCapture(0) for webcam.
- Frame-by-frame object detection using YOLOv5.
- Used confidence threshold (conf=0.4) for filtering predictions.

Alerting Experiment

- Trigger tested with various objects under different lighting conditions.
- Email sent only once to avoid flooding inboxes.
- Verified delivery success using sample "person" detection.

Tools & Infrastructure

- Python, OpenCV, PyTorch, Ultralytics
- Gmail SMTP for email alerts
- Kaggle/Colab with **GPU (P100)** for training

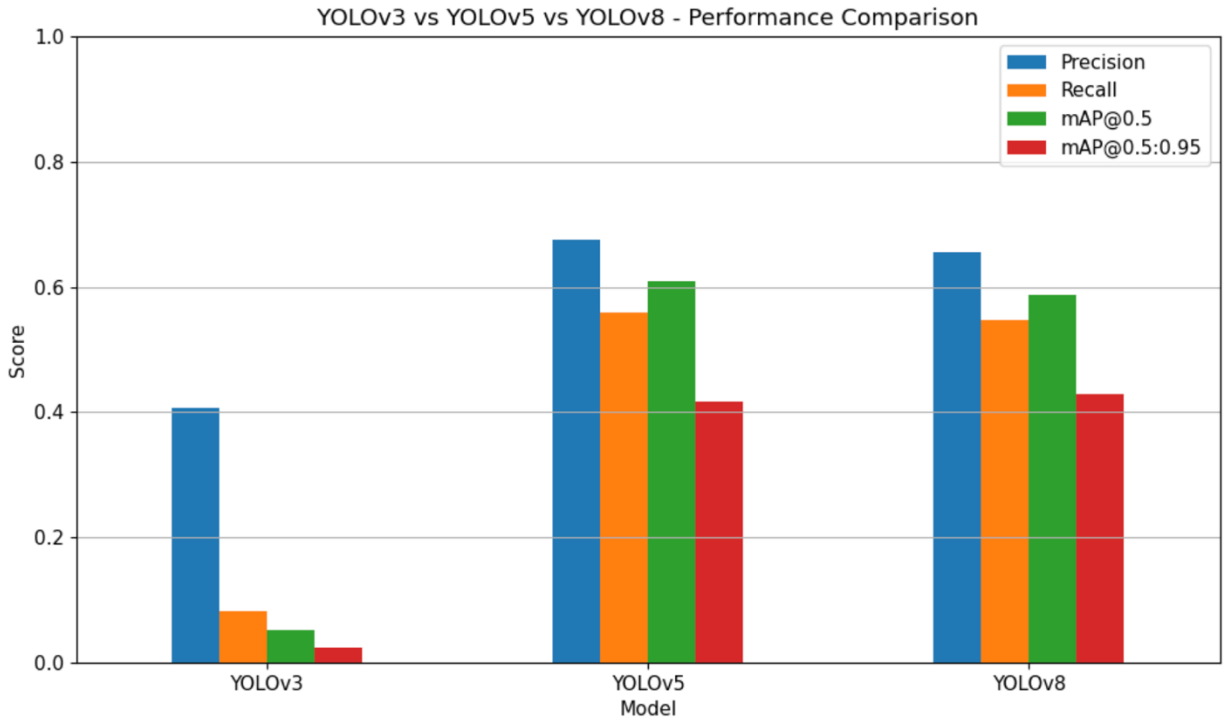
4.Results:

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95
YOLOv3	0.406	0.083	0.0532	0.0251
YOLOv5	0.675	0.56	0.609	0.416
YOLOv8	0.6551	0.5472	0.5871	0.4282

In addition to numerical evaluation, we visualized the detection outputs from each model. While YOLOv3 failed to reliably detect small or occluded objects, YOLOv5 and YOLOv8 consistently localized humans, animals, and packages with high confidence.

We also implemented **real-time webcam detection** using OpenCV. The best-performing model (YOLOv5) was deployed in a live scenario where the webcam stream was processed frame-by-frame, annotated in real-time, and alerts were sent via email when predefined dangerous classes like person were detected.

Below is a snapshot from the performance comparison:

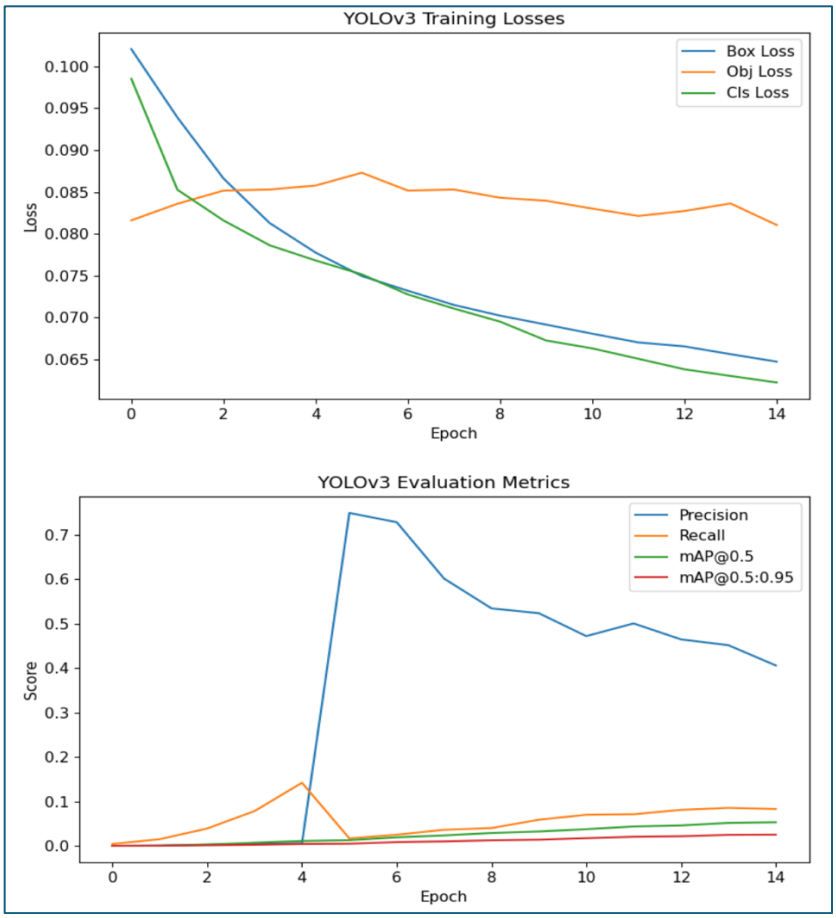


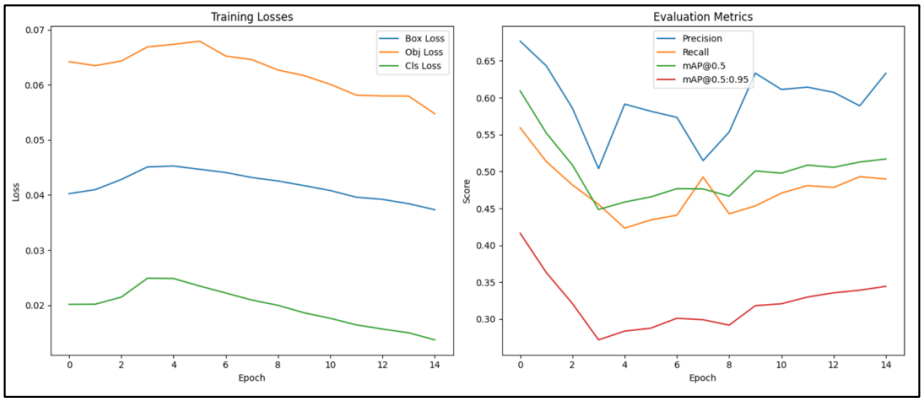
5. Analysis of Training Behavior and Model Outcomes:

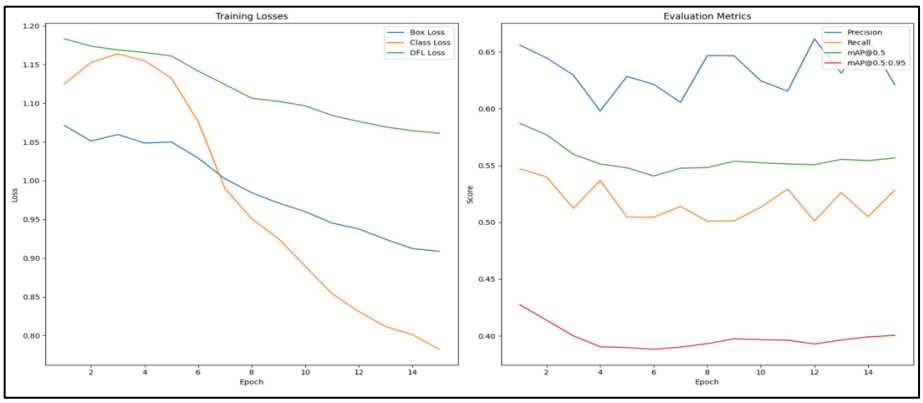
All three models—YOLOv3, YOLOv5, and YOLOv8—showed a consistent decrease in training losses, indicating proper learning. However, the evaluation metrics revealed stark differences in generalization. YOLOv3 underperformed significantly, with both precision and recall remaining low throughout training. Despite its loss decreasing, the model failed to detect relevant objects in validation, likely due to its older architecture lacking modern enhancements like anchor-free detection and advanced loss functions.

YOLOv8 achieved the highest mAP@0.5:0.95, showing better localization precision under stricter IoU thresholds. However, its recall fluctuated, and real-time performance lagged slightly due to its heavier architecture. YOLOv5, in contrast, maintained the highest precision and recall while delivering stable mAP values and smoother evaluation curves. It also performed better in real-time conditions, making it the most reliable choice for deployment.

In conclusion, YOLOv5 provided the best balance of accuracy, speed, and consistency. It generalized well, performed reliably in real-time scenarios, and outperformed both YOLOv3 and YOLOv8 in overall practical effectiveness for home security applications.





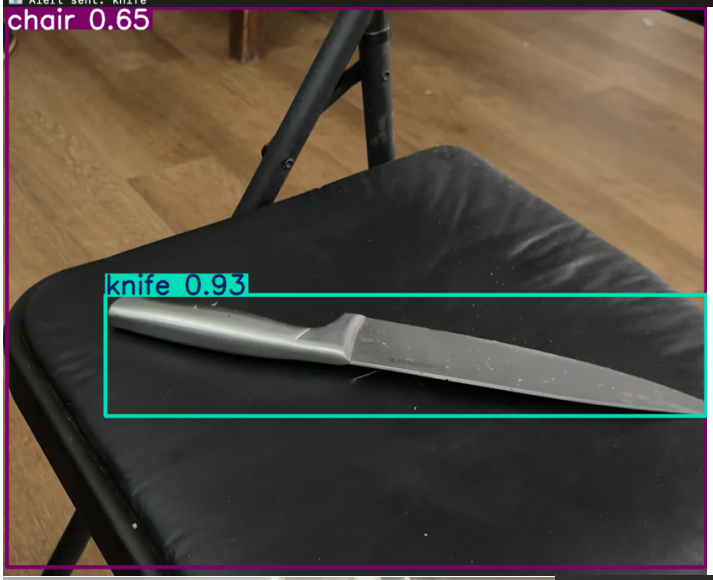


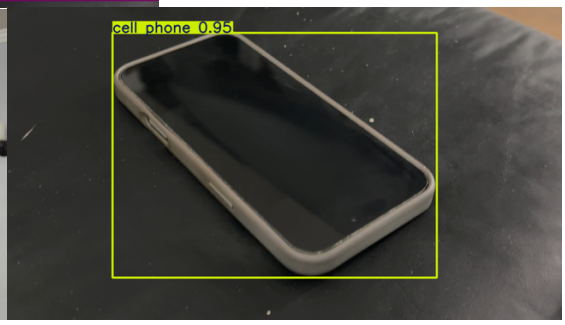
Output:

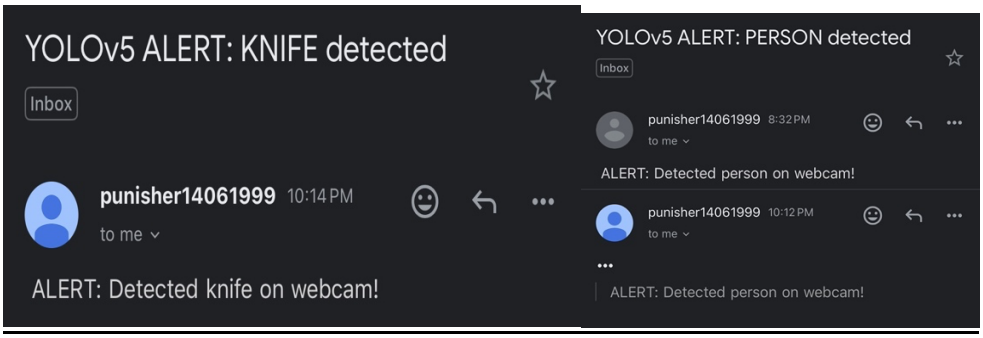
```
sai praneethDownloads%python webcam.py
PRO TIP 💡 Replace 'model=/Users/saipraneethch/Downloads/yolov5m.pt' with new 'model=/Users/saipraneethch/Downloads/yolov5mu.pt'.
YOLOv5 'u' models are trained with https://github.com/ultralytics/ultralytics and feature improved performance vs standard YOLOv5 models trained with https://github.com/ultralytics/yolov5.

 Webcam running. Press 'q' to quit.
📧 Alert sent: person
sai praneethDownloads%python webcam.py
PRO TIP 💡 Replace 'model=/Users/saipraneethch/Downloads/yolov5m.pt' with new 'model=/Users/saipraneethch/Downloads/yolov5mu.pt'.
YOLOv5 'u' models are trained with https://github.com/ultralytics/ultralytics and feature improved performance vs standard YOLOv5 models trained with https://github.com/ultralytics/yolov5.

 Webcam running. Press 'q' to quit.
📧 Alert sent: knife
```







The final system used the YOLOv5 model integrated with OpenCV to perform real-time object detection through a webcam. Upon running the script, the camera initialized, and the model began identifying objects in each frame with a set confidence threshold. During testing, it successfully detected classes like person, knife, and chair.

To focus on home security, we defined a set of "danger" classes—such as person, knife, and gun. From the above outputs we can notice When these were detected, the system automatically sent email alerts using Python's smtplib. For instance, upon detecting a person or knife, the user received alerts with messages like:
"ALERT: Detected person on webcam."

Objects not considered threats, like chair, were ignored by the alert system to prevent unnecessary notifications. This selective alerting ensured that users were notified only during potential danger, demonstrating both the model's effectiveness and the system's practicality for real-world security use.

6. Discussion and Lessons Learned:

This project provided hands-on experience in evaluating and deploying deep learning models for real-world object detection. Through implementing YOLOv3, YOLOv5, and YOLOv8, we observed how architecture, optimization, and hardware constraints influence model performance. YOLOv5 stood out as the most balanced model—combining speed and accuracy—which reinforced the importance of considering both metrics when selecting models for real-time applications like home surveillance.

One of the key takeaways was the trade-off between detection accuracy and runtime efficiency. While YOLOv8 was theoretically more accurate in localization (mAP@0.5:0.95), its slight performance overhead made it less suitable for live webcam streaming on standard hardware. We also learned the importance of careful metric interpretation, alert filtering, and threshold tuning to reduce false alarms and ensure reliable alert generation.

Looking forward, we aim to explore model quantization for edge deployment and implement behavior-based alert logic to enhance security decision-making. Overall, this project strengthened our understanding of practical model deployment and the nuances of balancing precision, performance, and usability.

7.Bibliography:

1. Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767.*
2. Lin, T. Y., Maire, M., Belongie, S., et al. (2014). Microsoft COCO: Common Objects in Context. *European Conference on Computer Vision (ECCV).*
3. Ultralytics YOLOv5 Documentation. *https://docs.ultralytics.com/models/yolov5/*
4. Ultralytics YOLOv8 Documentation. *https://docs.ultralytics.com/*