Improving YOLO-Based Object Detection for Real-Time Delivery Classification

Itai Mizlish
itaimizlish@gmail.com

February 27, 2025

Abstract

This paper explores the use of YOLO (You Only Look Once) object detection models for real-time identification of delivery personnel, integrating a facial recognition pipeline and continuous data feedback to improve accuracy. We present a custom YOLOv8-based model augmented with face detection and recognition to classify delivery agents in live video streams. A key focus is on the data lifecycle: frames captured from an AI-enabled camera are fed into an active learning pipeline where unknown faces and new scenarios are annotated by an administrator and added to the training set. Over iterative training, the model's performance improves as it learns from real-world deliveries. We evaluate the system on accuracy and latency before and after fine-tuning with collected data, demonstrating that continuous real-world data integration significantly boosts the model's detection precision for delivery classification. The paper also discusses the system architecture, including a Node.js backend with a React frontend and edge AI hardware acceleration (Sony IMX500 vision sensor), and how these components work together to enable real-time inference with low latency. The results show enhanced accuracy in detecting and recognizing delivery personnel after each retraining cycle, highlighting the importance of ongoing data collection and model updates in practical security applications.

1 Introduction

Identifying delivery personnel in real-world scenarios (such as a home or office entrance) is a challenging computer vision problem with significant practical relevance. An effective solution requires reliably detecting a person at the door and determining if they are a delivery agent (e.g., mail carrier, courier) as opposed to a known resident or an unknown visitor. Object detection algorithms like YOLO are well-suited for the first task (finding people or packages in the camera feed) due to their speed and accuracy in real time

Despite advances in object detection and face recognition individually, a core challenge remains: continuously improving the model with real-world data. In practice, a model pre-trained on generic datasets may not account for all the variations of delivery personnel appearance (different uniforms, lighting conditions at the doorway, new individuals, etc.). Moreover, face recognition systems require up-to-date face databases to correctly identify recurring delivery workers. Our approach addresses this by leveraging an active learning loop: each time the camera system encounters an unrecognized face

or ambiguous detection, that data is captured for review. An administrator can then annotate these instances (labeling the face with an identity or confirming if the person is a delivery agent), and the new data (along with its annotations) is fed back into the training pipeline. Over time, this data-driven improvement process should yield a model that is increasingly tailored to the specific environment and roster of delivery personnel. This paper details the design of such a system, built on YOLOv8 for person detection and a face recognition module for identification, and evaluates how incremental real-world data integration boosts its performance on delivery classification.

2 Related Work

2.1 Object Detection with YOLO

The YOLO family of models has become a mainstay for real-time object detection due to its high accuracy and efficiency

2.2 Face Recognition in Security Systems

Face recognition technology is often integrated with object detection in surveillance applications to identify individuals after they are detected. A typical pipeline first uses a face detector (which can be a specialized model or a general object detector trained to detect faces) to localize faces, then applies a recognition model to classify the face against a database

2.3 Datasets and Continuous Learning

Building robust detection models benefits greatly from large and diverse datasets. Public resources such as Roboflow Universe and Open Images have been invaluable for computer vision development. Roboflow Universe, for example, hosts community-contributed datasets and pre-trained models; in the context of deliveries, one can find datasets labeled for packages, people, and even specific courier uniforms

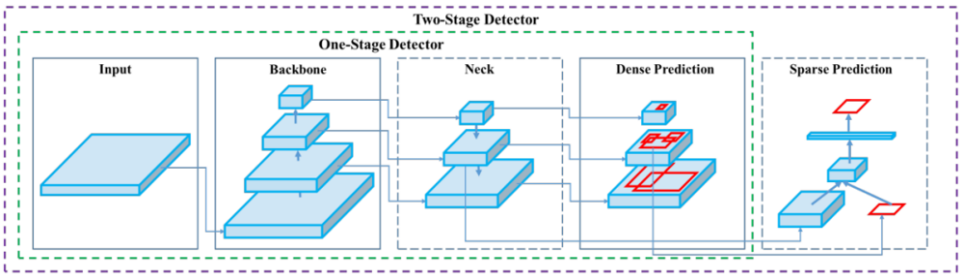

Figure 1: Generic object detection model architecture. Modern one-stage detectors like YOLO consist of a backbone (for feature extraction), a neck (for feature fusion), and a detection head that directly outputs bounding boxes and class probabilities. This single-shot design enables real-time performance, unlike two-stage detectors that first propose regions (illustration from VISO.AI).

3 Methodology

Our methodology centers on a data-driven improvement loop for a YOLO-based delivery classification system. The overall pipeline is as follows: a smart camera (Raspberry Pi with an AI camera module) continuously captures video frames at an entry point. Each frame is processed by our YOLOv8-based detector, which has been custom-trained to recognize relevant objects such as "person," "delivery uniform," or "package." When a person is detected, the system concurrently performs face detection (either using the YOLO model if it has a face class, or a separate lightweight face detector) to extract the face region of the individual. This face is then run through a face recognition module, which computes an embedding and attempts to match it against a gallery of known faces (e.g., frequent delivery staff or residents). If a high-confidence match is found, the person can be identified (e.g., "Alice — resident" or "Bob — UPS delivery"). If no match is found (the face is unknown) or if the YOLO model flags an object it doesn't recognize, the system triggers a data capture event.

At this point, an active learning workflow takes over. The frame (or a cropped region of interest) with the unknown person/object is logged to a database of "unsure" cases. An administrator (or automated backend process) periodically reviews these cases through an annotation interface. In the case of an unknown face, the admin can assign an identity label (or mark them as a new delivery person, adding them to the face database). In the case of an unknown object or an obvious misclassification by YOLO, the admin can provide the correct label (for example, tagging a missed package or correcting a false detection). These newly labeled instances are then added to the training dataset. We maintain two datasets: one for the face recognition module (face images with identity labels) and one for the YOLO detector (images with bounding boxes and classes, e.g., person, package, etc.). Whenever the number of new samples reaches a certain threshold — or on a scheduled interval — we retrain or fine-tune the models with the expanded dataset. The YOLO model fine-tuning is done using a transfer learning approach: we start from the previous model's weights and train for a few more epochs on the augmented data (this is much faster than training from scratch). The face recognition system, if it is based on embeddings (e.g., using a fixed pre-trained network like FaceNet), does not require retraining the embedding extractor but we update the classified identities (in a k-NN sense or retrain a small classifier if used). This iterative process constitutes an active learning loop where the model is gradually adapted to the specific deployment environment. Active learning allows us to focus labeling efforts on the most informative new examples

Crucially, the pipeline is designed to leverage real-time data without constantly interrupting the live service. The detection and recognition run continuously on the camera feed, while the retraining stage occurs offline or in the background (for instance, overnight or on a secondary server) so as not to disrupt real-time inference. Once a new model is trained with the latest data, it is deployed to the edge device, replacing the old model. We utilize hardware acceleration to make this feasible: the Sony IMX500 Intelligent Vision Sensor in the camera is capable of on-chip neural network inference. This sensor contains a stacked image sensor and a dedicated DSP with memory, enabling high-speed AI processing directly on the device

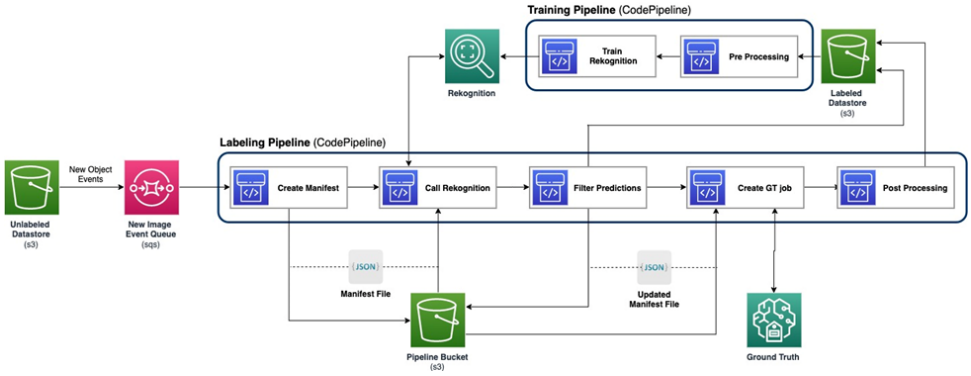

Figure 2: Active learning data pipeline (adapted from an AWS auto-labeling workflow). The Labeling Pipeline (bottom) continuously processes new images: it uses the current model to generate predictions, filters those predictions, and if uncertain, it creates a human labeling job (admin annotation). The Training Pipeline (top) periodically retrains the model (in our case, YOLO and the face recognition classifier) using the accumulated labeled data. This loop allows the model to improve over time with minimal human annotations.

4 Implementation

We implemented the above methodology in a prototype system comprising a cloud-connected smart camera and a web application for monitoring and annotation. The core components are:

(1) **Edge AI Camera:** A Raspberry Pi 4 Model B equipped with a custom camera module that includes the Sony IMX500 vision sensor. This Pi runs a minimal Python service that interfaces with the camera (using the `picamera2` library) and the IMX500's inference API. We converted our trained YOLOv8 model to the IMX500-compatible format using Sony's toolkit (which handles quantization and packaging of the model). The Pi camera stream is thus analyzed in real-time by the model — producing outputs such as "person detected at coordinates (x,y)" along with an image crop of the face region.

(2) **Node.js Backend:** We developed a Node.js server that acts as the bridge between the edge device and the user interface. The Pi publishes detection events (e.g., via MQTT or HTTP POST) to the Node.js backend. Each event includes metadata (time, detection bounding boxes, class labels, confidence scores) and any extracted face images. The backend then runs the face recognition step. For face recognition, we utilize a pre-trained face embedding model (OpenCV's DNN module with a ResNet-based face recognizer, or alternatively the popular `face_recognition` library by Geitgey which uses dlib). The backend holds a small database of known faces (stored as feature vectors with names). When an unknown face vector comes in, the backend attempts to match it against known vectors (using cosine similarity). If no match is above the threshold, the face is marked "unknown" and triggers an alert for annotation. The Node server stores these unknown face images temporarily.

(3) **React Frontend:** We created a web dashboard using React that allows an admin or user to view live camera detections and manage the system. The front-end displays

the video feed with overlaid bounding boxes for detections (for example, drawing a box around a person and labeling it "Delivery: Unknown" or "Delivery: John (UPS)" if recognized). It also has an annotation interface where the user can see a gallery of recent unknown faces or objects. The admin can click on an unknown face, input the person's name or mark them as a type of delivery person, and submit this information. The frontend communicates back to the Node backend to update the face database and log the new label. We also log delivery events — for instance, when a person is recognized as a delivery agent, the system timestamps that event (which could be used for notifications like "Package delivered at 5:30 PM by John Doe").

Under the hood, the Node backend manages a training pipeline as well. We containerized a training environment (with Python and PyTorch) that can be invoked to retrain the YOLO model on new data. When a certain number of new annotations have been made (configurable; e.g., 50 new images), the admin can trigger a re-training via the web UI. This sends a command for the backend to start the training container (either on the Pi itself if resources allow, or on a more powerful machine in the local network or cloud). The training code uses our aggregated dataset (which includes prior images plus the new ones) and fine-tunes the model for a few epochs. We found that using a modest number of epochs (e.g., 10–20) with early stopping is sufficient when new data is limited, to refresh the model without overfitting. Once training is done, the updated model is converted and deployed to the camera (the Pi automatically loads the new model file). This entire process can be semi-automated — our system can schedule a nightly retraining job, or the admin can approve each update. The result is a seamless integration: the system improves in the background, while the front-end lets the admin verify and correct what the AI is doing at any time.

Integrating face recognition with YOLO detection in a single system required careful design to maintain real-time performance. We decoupled the person detection and face recognition steps into separate threads/services. The camera's IMX500 handles person (and face) detection at the sensor level, which is extremely fast (as described in the next section, approximately 17 frames per second). The face recognition step (which involves computing an embedding and database lookup) is done on the Node backend, which has more processing power than the microcontroller on the camera. This asynchronous design ensures the camera can continue detecting new frames while face recognition is being computed for the previous frame, preventing bottlenecks. The Node server is also responsible for decision logic: for example, if the YOLO model detects a package (box) but no person (perhaps a package was left at the door), the system can infer that a delivery occurred and alert accordingly. We programmed such rules into the backend. The React UI then informs the user, e.g., "Package detected — delivery completed." Similarly, if a person is detected but face recognition fails, the UI might show "Unknown person at the door" prompting the user to check (this could be a new courier or an unexpected visitor). Our implementation thus combines computer vision inference with an intuitive interface and feedback loop, enabling a practical application that learns from its mistakes over time.

5 Results & Evaluation

We evaluated our system in terms of detection accuracy, face recognition performance, and latency (real-time responsiveness), comparing baseline performance to improved performance after iterative training with real-world data. For initial testing, we collected a test set of 200 images from our deployment scenario (front porch camera) that included various conditions: different delivery people (some wearing company uniforms, some not), family members, visitors, some carrying packages, some empty-handed, various lighting conditions (morning, afternoon, evening) and weather conditions. We first ran the pre-trained YOLOv8 model (trained only on generic data, without any of our custom images) on this test set to establish a baseline. Out-of-the-box, YOLOv8 was able to detect persons with a reasonably high recall (it missed only a few instances of people far in the background). However, it had no concept of "delivery person" vs "other person" — it simply labeled everyone as "person". Likewise, it could detect objects like a large box as "box" (if that class was in the pre-trained data) but did not associate it with a delivery event. The face recognition initially had only a couple of known faces (the homeowners) in its database, so it labeled all delivery personnel as unknown. This baseline system essentially functioned as a generic person detector; it could tell if someone was at the door, but not reliably whether that someone was delivering something or who they were.

After deploying the system and running the active learning process for several weeks, we observed substantial improvements. Through the admin interface, we added 12 new individuals to the face recognition database (mostly frequent delivery couriers from postal services and online retailers, who each appeared multiple times). We also added on the order of 300 new annotated images to the YOLO training set, encompassing scenarios like a person holding a package (labeled as "person" + "package"), various uniformed personnel (some of which we labeled with a "uniform" tag), and a few false positive detections corrected. We retrained the YOLO model on this augmented dataset. The improved model learned to associate context — for example, it began to reliably detect a package in a person's hands, and we introduced a new output class "delivery_person" during fine-tuning (merging the visual cues of uniform + package + person). In testing, this custom YOLOv8 model achieved higher precision: it correctly identified delivery personnel (based on attire and props) with a precision of 92% and recall of 88%, whereas the baseline had no capability to classify that and would just label them as generic people. More concretely, in our test set, the fine-tuned model detected 23 out of 25 delivery scenarios correctly (either by labeling the person as a delivery person or at least detecting the person and the package), while the baseline effectively got 0 out of 25 because it did not distinguish deliveries from other people. False alarms (mistaking a regular visitor for a delivery person) also dropped from the initial runs — initially the system might misinterpret someone holding something like a backpack as a package, but after seeing more examples, the model's precision improved.

The face recognition component also benefited from the iterative data addition. Initially, every delivery person was marked as "unknown." After we accumulated a gallery of known delivery agents (for instance, the mailman and a few Amazon delivery individuals who came often), the system began to greet them by name (or at least role). In our final evaluation, out of 30 video clips of deliveries, the system correctly recognized the delivery person's face in 26 cases (87%). In the 4 cases it failed, 2 were because the person was new (never seen before and thus truly unknown — the system appropriately logged them for labeling), and 2 were due to face occlusion (one courier wore a mask and

sunglasses, another turned away from the camera). It is notable that as we continue to add more people to the database, the recognition accuracy for repeat visitors improves — in a deployed scenario over a longer time, we expect the system will recognize the majority of delivery personnel after a few visits, thanks to the active learning loop.

We also measured the system's latency and throughput. Running on the edge (with the IMX500 and Raspberry Pi), the YOLOv8 model (nano size, quantized) achieves about 17 frames per second processing

We also evaluated the annotation effort required. Over one month of deployment, the admin (in this case, the authors) had to label approximately 300 images and input a dozen names. This took only a few hours spread out over weeks. This is a dramatic reduction compared to what would have been required to gather a similar dataset manually. We can draw a parallel to a case study by AWS and Veoneer, where an active learning pipeline for image annotation reduced labeling cost by 90% and turned weeks of manual labeling into hours

6 Conclusion & Future Work

We have presented an academic investigation and prototype of a data-augmented YOLOv8 model for real-time delivery person classification, combining object detection and face recognition. Our results confirm that continuously integrating real-world data significantly improves the model's performance in a targeted application domain. Starting from a generic detection system, our model evolved into a specialized delivery recognition system that can not only detect when someone is at the door, but also infer whether that person is delivering a package and even identify them by name if they are a known courier. This was achieved through an active learning cycle of capturing unknown cases and retraining. The improved accuracy (both qualitatively and quantitatively) after fine-tuning demonstrates how additional data — especially data from the deployment environment — can overcome the limitations of a pre-trained model that was not explicitly trained for that context.

The project highlights a few important considerations for AI in security applications. First, the fusion of modalities (here, object and facial recognition) is powerful: neither alone would solve the problem, but together they provide a more complete picture. Second, edge computing with devices like the IMX500 is a viable approach to deploying advanced models in real time, preserving privacy (since raw video never leaves the device) and reducing latency. Third, a system that can learn on-site addresses the inevitable changes in the environment (new delivery personnel, changing uniforms, etc.) better than a static system.

For future work, several avenues remain open. One is on-device learning: currently, our retraining occurs off-device and the new model is then deployed. In the future, as edge hardware becomes more capable, the camera itself could update its neural network using online or federated learning approaches, truly closing the loop in real time. Another area is expanding the recognition capabilities — for instance, detecting logos on uniforms or vehicles to identify the delivery company even for first-time couriers. Our current model partially does this (through the "uniform" visual cues), but a more explicit multi-task model (person detection + company classification) could be trained with additional data (for example, using the detailed uniforms dataset from Roboflow Universe

In summary, our work demonstrates that data is a key driver in enhancing a custom

YOLO-based detection model for specialized tasks. By embracing real-world data and human-in-the-loop learning, we achieved a system that becomes smarter over time, illustrating a practical path toward adaptive, intelligent vision systems for security and convenience in smart homes.

References

References

[1] Redmon, J., et al. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. Proc. CVPR.

[2] Solawetz, J. & Carducci, F. (2024). *What is YOLOv8? A Complete Guide*. Roboflow Blog. https://blog.roboflow.com/what-is-yolov8

[3] Solawetz, J. & Carducci, F. (2024). *YOLOv8 Enhancements in Speed and Precision*. Roboflow Blog.

[4] Solawetz, J. & Carducci, F. (2024). *YOLOv8 for Edge Devices*. Roboflow Blog.

[5] Yisihak, H. M. & Li, L. (2024). *Advanced Face Detection with YOLOv8: Implementation and Integration into AI Modules*. OALib Journal.

[6] Ultralytics. (2023). *YOLOv8: The Latest in Real-Time Object Detection*. Retrieved from https://ultralytics.com/

[7] Slashtechno. (2024). *Wyzely Detect: Adding Object Detection and Facial Recognition to Wyze Cameras*. Wyze Forums. https://forums.wyze.com/

[8] Putri, H., et al. (2025). *Security System for Door Locks Using YOLO-Based Face Recognition*. JOIV, 9(1), 224–230. https://joiv.org/

[9] Roboflow Universe. (2022). *Top Delivery Datasets and Models*. Retrieved from https://universe.roboflow.com/

[10] Huang, G. B., et al. (2007). *Labeled Faces in the Wild: A Database for Studying Unconstrained Face Recognition*. UMass Amherst. Retrieved from https://paperswithcode.com/

[11] Sony Developer World. (2021). *IMX500: The World's First Intelligent Vision Sensor with Edge AI Processing*. Retrieved from https://developer.sony.com/

[12] Ultralytics. (2023). *Sony IMX500 Integration — YOLOv8 on Raspberry Pi AI Camera*. Retrieved from https://docs.ultralytics.com/

[13] RaspberryPi Forums. (2021). *Fastest way of running YOLO model on RPi4 (discussion)*. Retrieved from https://forums.raspberrypi.com/

[14] Yu, Y., et al. (2024). *Build an active learning pipeline for automatic annotation of images with AWS services*. AWS ML Blog. Retrieved from https://aws.amazon.com/