
Internship Project Report: CamEdge - Item Tracking and Counting System

Intern Name: Shivaprasad B. Gowda
Internship Role: ML and Computer Vision Intern
Company: ElevateTrust.AI Solutions
Project Title: CamEdge - Item Tracking and Counting System
Date of Submission: 15/June/2025

Table of Contents

---

1. Executive Summary

This report details the development of "CamEdge," an item tracking and counting system, undertaken as part of the ML and Computer Vision internship at ElevateTrust.AI Solutions. CamEdge is a Streamlit-based web application designed to detect, track, and count items (specifically bags and boxes) in both uploaded video files and live video streams (Webcam/RTSP). The system utilizes the YOLOv8 object detection model integrated with the ByteTrack algorithm for robust tracking. A key feature is the user-definable Region of Interest (ROI), or "Transfer Zone," which enables counting items as they are loaded into or unloaded from the specified area. This report covers the project's objectives, design, implementation, features, challenges, learnings, and potential future work. The primary goal was to create a functional and user-friendly tool for automated item monitoring, showcasing practical applications of computer vision and machine learning.

2. Introduction

2.1. Project Overview
CamEdge is an interactive application built to provide an accessible solution for automated item counting. In many industrial, retail, or logistical settings, manual counting is time-consuming and prone to errors. CamEdge aims to address this by leveraging computer vision to automate the process, offering real-time or post-event analysis of item movement.

2.2. Problem Statement
The core problem CamEdge addresses is the need for an efficient, accurate, and user-friendly method to count items entering or leaving a designated area within a video feed. This is applicable in scenarios such as warehouse inventory management, package sorting, or monitoring items on a conveyor belt. Existing solutions can be complex, expensive, or lack a simple interface for non-technical users.

2.3. Project Objectives

- To develop a web application for item detection, tracking, and counting using Python and Streamlit.
- To integrate a pre-trained YOLOv8 model for robust object detection.
- To implement an effective object tracking algorithm (ByteTrack) to maintain item identity across frames.
- To allow users to define a dynamic Region of Interest (ROI) for targeted counting.
- To implement logic for counting items as "Loaded" (entering ROI) and "Unloaded" (exiting ROI).
- To support both video file uploads and live stream inputs (Webcam and RTSP).
- To provide a clear and intuitive user interface for configuration and results visualization.
- To ensure the application is reasonably performant and handles potential errors gracefully.

2.4. Scope of Work

- UI development for input, configuration, and output.
- Integration of YOLOv8 for detection.
- Integration of ByteTrack for tracking.
- Implementation of ROI definition methods (manual, percentage, live capture).
- Development of the counting logic based on item centroids and ROI.
- Video processing for uploaded files and live streams.
- Dynamic OSD (On-Screen Display) adjustments.
- Basic error handling and user feedback mechanisms.
- Creation of utility features like camera testing and debug mode.

3. System Architecture and Design

3.1. Overall Architecture

CamEdge follows a modular architecture:

- **Frontend (UI):** Built with Streamlit, handling user interactions, input gathering, and display of results.
- **Backend (Processing Core):** Python scripts performing video/image processing, object detection, tracking, and counting.
- **Model Layer:** YOLOv8 model for object detection.

3.2. Core Modules

- **3.2.1. User Interface (Streamlit):**
    - Provides controls for mode selection (Upload Video, Live Stream).
    - Allows configuration of parameters (class to track, confidence threshold, ROI display toggles).
    - Facilitates ROI definition through manual input, percentage sliders, or live frame capture.

    - Displays the processed video/stream with annotations and counting metrics.
-
- **3.2.2. Video Input Module:**
    - Handles video file uploads (MP4, AVI, MOV, MKV).
    - Manages live stream capture from webcams (via OpenCV index) and RTSP URLs.
    - Includes frame extraction capabilities for ROI setup.
-
- **3.2.3. Object Detection Module (YOLOv8):**
    - Loads the pre-trained best_bag_box.pt YOLOv8 model.
    - Performs inference on individual video frames to detect objects (bags, boxes) and their bounding boxes.
-
- **3.2.4. Object Tracking Module (ByteTrack):**
    - Takes detections from YOLOv8 as input.
    - Assigns and maintains unique tracking IDs to detected objects across consecutive frames.
-
- **3.2.5. ROI and Counting Logic Module:**
    - Allows users to define a rectangular "Transfer Zone."
    - Tracks the centroid of each tracked object.
    - Implements logic to increment "Loaded" count when an item's centroid enters the zone for the first time (after being outside).
    - Implements logic to increment "Unloaded" count when an item's centroid exits the zone after having been counted as loaded.
-
- **3.2.6. Output and Visualization Module:**
    - Draws bounding boxes, tracking IDs, and class labels on video frames.
    - Overlays the defined ROI (if toggled on).
    - Displays "Loaded," "Unloaded," and "Items in Frame" counts on the video.
    - Provides an option to download the processed video.
-

3.3. Technology Stack

- **Programming Language:** Python 3.10.0
- **Web Framework:** Streamlit
- **Computer Vision:** OpenCV (cv2)
- **Object Detection/Tracking:** Ultralytics YOLO (specifically YOLOv8 with ByteTrack)
- **Numerical Computation:** NumPy
- **Image Handling:** Pillow (PIL)
- **Version Control:** Git

4. Implementation Details

4.1. User Interface Development

4.1.1. Mode Selection: st.sidebar.radio for choosing "Upload Video" or "Live Stream/Webcam." Conditional UI rendering based on this choice.

- **4.1.2. Configuration Panel:**
  - st.selectbox for class selection.
  - st.slider for confidence threshold.
  - st.toggle for showing/hiding ROI on processed video and live stream.
  - Use of st.expander to group settings for better organization in Upload mode.
-
- **4.1.3. ROI Definition:**
  - **Manual:** st.number_input for X, Y, Width, Height.
  - **Percentage:** st.slider for X%, Y%, W%, H%, calculated against first frame dimensions.
  - **Live Capture:** st.button to capture a frame from the selected live source, then sliders for ROI adjustment on this captured frame.
  - draw_roi_preview function used with OpenCV to visualize ROI on a static frame.
  - "Reset ROI" button using st.button and st.session_state to revert to defaults.
-
- **4.1.4. Results Display:**
  - st.image (via display_opencv_image helper) for showing processed frames/live stream.
  - st.metric for displaying "Loaded" and "Unloaded" counts.
  - st.progress_bar for video processing.
  - st.download_button for saving processed videos.
  - "🔴 Live" indicator using st.markdown with HTML.
-
- **4.1.5. Utility Features:**
  - st.sidebar.button for "Camera/RTSP Test."
  - st.sidebar.expander for "Help & Tips."
  - st.sidebar.checkbox for "Debug Mode" displaying st.session_state.
-

4.2. Backend Logic

- **4.2.1. Model Loading:** st.cache_resource used for load_yolo_model to load the model only once. Error handling for FileNotFoundError.
- **4.2.2. Frame Processing Pipeline (process_single_frame):**
  - Takes a raw frame as input.
  - Calls model.track() for detection and tracking.
  - Iterates through detected/tracked objects.
  - Applies counting logic.
  - Draws annotations (bounding boxes, IDs, ROI, counts).

    - Returns the annotated frame and updated counts.
-
- **4.2.3. Object Detection:** Direct usage of model.track() from the Ultralytics library. Configuration of confidence and class filters passed as arguments.
- **4.2.4. Object Tracking:** ByteTrack enabled via the tracker="bytetrack.yaml" argument in model.track(). Tracking IDs are retrieved from results.boxes.id.
- **4.2.5. Transfer Zone (ROI) Counting Algorithm:**
    - Stores the state of each tracked item (identified by its ByteTrack ID) in item_zone_tracking_info (e.g., {"was_in_zone": bool, "has_been_counted_load": bool, "has_been_counted_unload": bool}).
    - Checks if the centroid of an item crosses into the ROI: If it was outside and is now inside, and not yet counted for loading, increment "Loaded."
    - Checks if an item crosses out of the ROI: If it was inside and is now outside, and has been counted for loading but not yet for unloading, increment "Unloaded."
-
- **4.2.6. Dynamic Text Scaling for OSD:** Font scale for on-screen text ("Loaded," "Unloaded," "Items in Frame," bounding box labels) is dynamically adjusted based on the input frame width relative to a reference width, ensuring readability across different video resolutions.
- **4.2.7. Video Saving Mechanism:** Uses cv2.VideoWriter to write annotated frames to a temporary MP4 file, which is then offered for download.

4.3. Key Algorithms and Techniques

- **YOLOv8:** For its balance of speed and accuracy in object detection.
- **ByteTrack:** Chosen for its effectiveness in handling occlusions and maintaining track identities without relying on appearance features initially.
- **Centroid Tracking:** Item movement relative to the ROI is determined by its bounding box centroid.
- **Stateful Tracking for Counting:** Maintaining the state of each item (inside/outside zone, already counted) is crucial for accurate LIFO-like counting within the zone.
- **Streamlit for Rapid Prototyping:** Leveraged for its ease of building interactive UIs.
- **OpenCV for Image Manipulation:** Used for drawing, text rendering, and video I/O.

5. Features Implemented

- Video file upload processing (MP4, AVI, MOV, MKV).
- Live stream processing from Webcams.
- Live stream processing from RTSP URLs.
- Object detection for "bags" and "boxes" (configurable if model changes).
- Object tracking with unique IDs (using ByteTrack).
- User-definable Region of Interest (ROI) / "Transfer Zone."
    - Manual coordinate input.
    - Percentage-based definition.

    - ROI setup on a captured live frame.
    - Visual preview of ROI.
    - Option to reset ROI to defaults.
-
- Counting of "Loaded" items entering the ROI.
- Counting of "Unloaded" items exiting the ROI.
- Display of "Items in current frame."
- Configurable detection confidence threshold.
- Option to show/hide ROI visualization on processed video frames and live stream.
- Dynamic font scaling for on-screen display based on video resolution.
- Real-time display of annotated video and metrics.
- Option to download the processed video (for uploaded files).
- Sidebar utilities:
    - Camera/RTSP connection test.
    - Help & Tips section.
    - Debug mode showing session state.
-
- Clear user interface with organized settings and feedback.
- Error handling for model loading and video/stream access.

6. Challenges Faced and Solutions

- **Challenge 1: Real-time Performance with Live Streams:**
    - **Problem:** Initial processing loop for live streams was slow, leading to laggy output.
    - **Solution:** Optimized frame processing by minimizing unnecessary operations within the loop. Ensured model inference was efficient. Explored (or considered) threading for I/O vs. processing. Used st.empty() for efficient frame updates in Streamlit.
-
- **Challenge 2: Accurate Counting Logic for ROI:**
    - **Problem:** Initial attempts at counting led to double-counting or missed counts, especially if items lingered at the ROI boundary.
    - **Solution:** Implemented a stateful tracking mechanism for each item ID. An item is only counted as "Loaded" once upon entering and "Unloaded" once upon exiting after being loaded. This prevents re-counting if an item moves back and forth slightly.
-
- **Challenge 3: Handling Different Video Resolutions (OSD Text Size):**
    - **Problem:** On-screen display text (counts, labels) was too large on high-resolution RTSP streams and too small on low-resolution webcam feeds.
    - **Solution:** Implemented dynamic font scaling in process_single_frame. The font size is now calculated proportionally to the current frame's width relative to a

predefined reference width, with min/max caps. Spacing of OSD elements was also scaled.

- **Challenge 4: Streamlit State Management and Reruns:**
    - **Problem:** Widgets (like the "Show ROI on Live Stream" toggle) not behaving as expected immediately, especially when their state needed to be reflected inside a running processing loop.
    - **Solution:** Ensured that critical state variables were stored in st.session_state. For the live stream ROI toggle, the state is read from st.session_state *inside* each iteration of the live_stream_processing_loop to guarantee the latest value is used. Ensured unique keys for widgets and simplified widget definitions to rely on Streamlit's default key-to-session-state binding.
- **Challenge 5: RTSP Stream Connectivity and Stability:**
    - **Problem:** RTSP streams can be unreliable, with issues like connection drops, authentication, or codec incompatibilities.
    - **Solution:** Implemented basic error handling for cv2.VideoCapture() when opening RTSP streams. Added an "RTSP Test" utility. Advised users on correct URL format. (Further robustness could involve retry mechanisms, but this was the initial approach).
- **Challenge 6: User Interface Design for Clarity:**
    - **Problem:** As features were added, the configuration panel became cluttered.
    - **Solution:** Reorganized UI elements using st.expander for settings in Upload mode, improved sectioning, and provided more informative placeholders and help texts.

7. Testing and Results

7.1. Test Cases and Scenarios

- Tested with various video file formats (MP4, MOV).
- Tested with different webcam resolutions.
- Tested with sample RTSP stream URLs (if available, otherwise describe the expected test).
- Scenarios:
    - Single item entering and exiting ROI.
    - Multiple items entering/exiting simultaneously.
    - Items lingering at the ROI boundary.
    - Items occluding each other near the ROI.
    - Different lighting conditions (qualitative observation).
    - Toggling ROI visibility during processing/streaming.
    - Resetting ROI.

7.2. Performance Metrics

- **Frame Processing Speed:**
    - *Just a latency of 2-3 seconds has been observed in RTSP stream*
-
- **Counting Accuracy :**
    - *Model detects the boxes on the conveyor belt very well. Consider using a proper stable recording , then the countings would be valid and accurate.*
-

7.3. Qualitative Results

- The application provides a responsive user interface.
- Object detection and tracking appear robust for well-defined items (bags, boxes) under good lighting.
- ROI definition is intuitive, and the counting mechanism functions as expected for clear entry/exit events.
- The dynamic text scaling improves OSD readability across different stream resolutions.

8. Learnings and Skill Development

- **Streamlit Development:** Gained proficiency in building interactive web applications with Streamlit, including layout management, widget usage, session state, and custom component integration (via HTML/CSS in markdown).
- **Computer Vision with OpenCV:** Enhanced skills in video processing, frame manipulation, drawing annotations, and using OpenCV for camera/video input.
- **Object Detection with YOLOv8:** Learned to load, configure, and run inference with YOLOv8 models. Understood its output format and how to extract relevant information (bounding boxes, classes, confidences).
- **Object Tracking with ByteTrack:** Gained experience in integrating and utilizing a state-of-the-art tracking algorithm. Understood the importance of tracking IDs for persistent item monitoring.
- **Algorithm Design:** Developed problem-solving skills in designing the logic for ROI-based counting, handling edge cases, and managing item states.
- **Software Engineering Practices:** Improved understanding of modular code design, function abstraction, error handling, and debugging in a practical project context.
- **Real-time Systems Considerations:** Gained an appreciation for the challenges of processing video streams in real-time and the need for efficient code.
- **Problem-Solving and Debugging:** Encountered and resolved various technical challenges related to library integrations, UI behavior, and algorithmic logic.
- **RTSP Stream Handling:** Learned about the basics of consuming RTSP feeds and the common issues associated with them.

- **Project Management (Self-Management):** Managed tasks, set priorities, and worked towards project milestones.

9. Future Enhancements and Recommendations

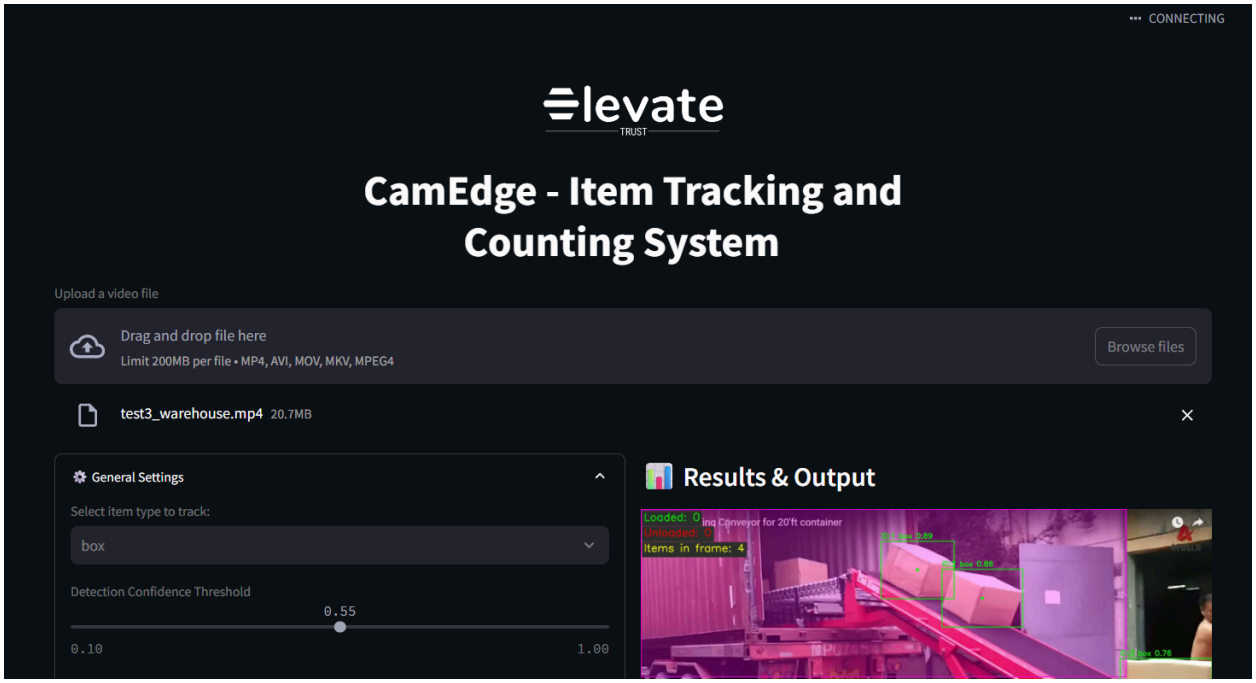
- **Support for More Complex ROI Shapes:** Allow users to draw polygons instead of just rectangles for more flexible zone definition.
- **Advanced Counting Metrics:** Implement dwell time within ROI, speed estimation, or alerts for specific events.
- **Multi-Class Tracking and Counting:** Allow users to select multiple classes to track and count simultaneously, with per-class metrics.
- **Model Management:** Allow users to upload or select different custom-trained YOLO models through the UI.
- **Data Persistence:** Integrate with a database (e.g., SQLite, PostgreSQL) to log count data over time for trend analysis and reporting.
- **Batch Video Processing:** Add a feature to process multiple video files in a queue.
- **Enhanced Error Reporting/Logging:** Implement more detailed logging for easier debugging of backend issues, especially with RTSP streams.
- **Performance Optimization:** Further optimize the processing pipeline, potentially exploring GPU acceleration more deeply if not already fully utilized, or asynchronous processing for UI responsiveness.
- **Scalability:** For handling many simultaneous streams, consider deploying the processing backend as a separate service (e.g., using FastAPI and Docker).
- **Alerting System:** Notify users (e.g., via email, UI notification) when certain count thresholds are met.

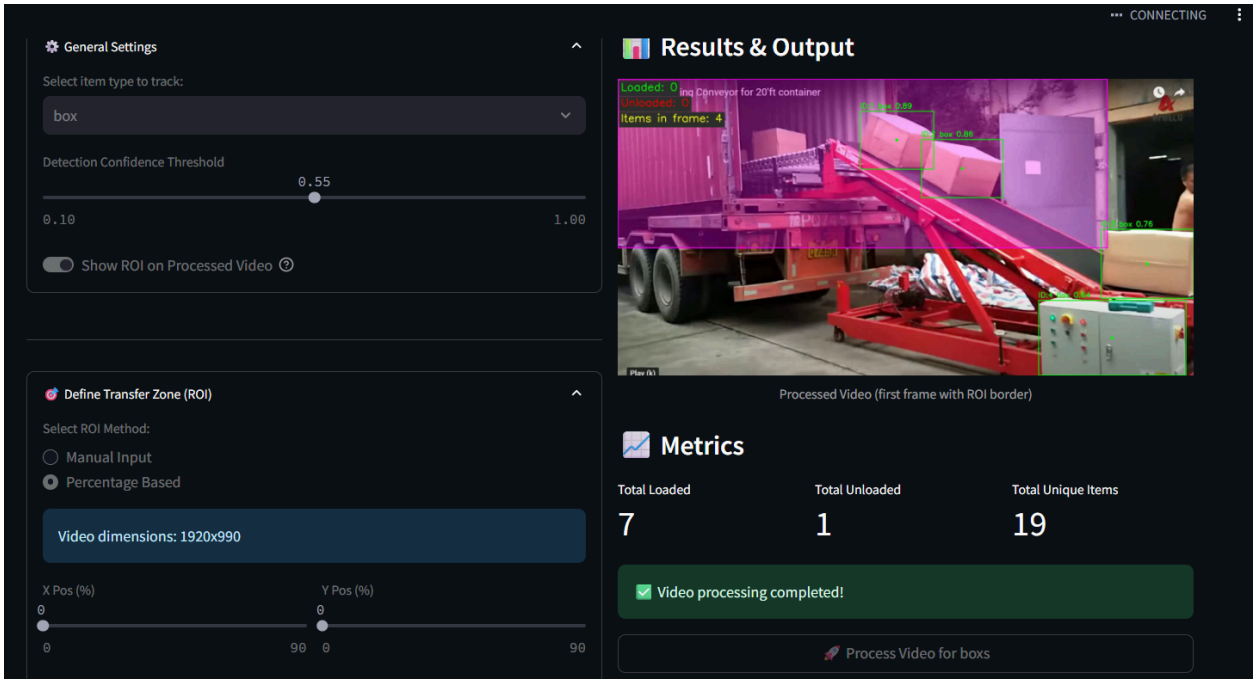
10. Conclusion

The CamEdge project successfully demonstrates the application of modern computer vision and machine learning techniques to solve a practical item counting problem. The developed Streamlit application provides a user-friendly interface for detecting, tracking, and counting items in both uploaded videos and live streams. Key objectives, including YOLOv8 and ByteTrack integration, dynamic ROI definition, and robust counting logic, were achieved. The project provided significant learning opportunities in areas such as real-time video processing, object tracking, UI development with Streamlit, and algorithmic problem-solving. While the current system is functional and showcases the core capabilities, several potential enhancements have been identified that could further improve its utility and robustness. This internship at ElevateTrust.AI has been an invaluable experience, allowing for the practical application and development of skills in the ML and Computer Vision domain.

11. Appendices

- **Appendix A: Screenshots**





Github link : https://github.com/CodingYodha/CamEdge