PROJECT REPORT

OBJECT DETECTION AND CLASSIFICATION ON THE CONVEYOR SYSTEM

Prepared by:Rishiraj Datta

Table of Content

I. Introduction

In this project, our goal is to improve the efficiency and accuracy of a conveyor system by implementing an intelligent object detection and sorting system. We aim to identify and sort objects running on the conveyor belt in real-time with 100% accuracy.

To achieve this, we will use a FINGERS 1080-high-resolution webcam placed above the conveyor belt, capturing images of the objects as they move through the system. The conveyor belt is motor-driven, ensuring smooth movement, and encoders will measure object speed and position for precise synchronization.

Using advanced deep learning techniques, like the YOLO (You Only Look Once) model, we will perform real-time object detection, focusing initially on differentiating between boxes and bottles. The webcam will only capture the conveyor belt and objects, minimizing background noise.

Our approach will prioritize computational efficiency while maintaining high accuracy. We will address variations in object orientation, shape, size, and color for robust detection and sorting.

By integrating this system with a parallel robot, we will enable precise timing and positioning of objects, ensuring efficient sorting.

Overall, this project aims to enhance the conveyor system's capabilities, providing reliable and fast object detection and sorting, thereby improving industrial automation processes.

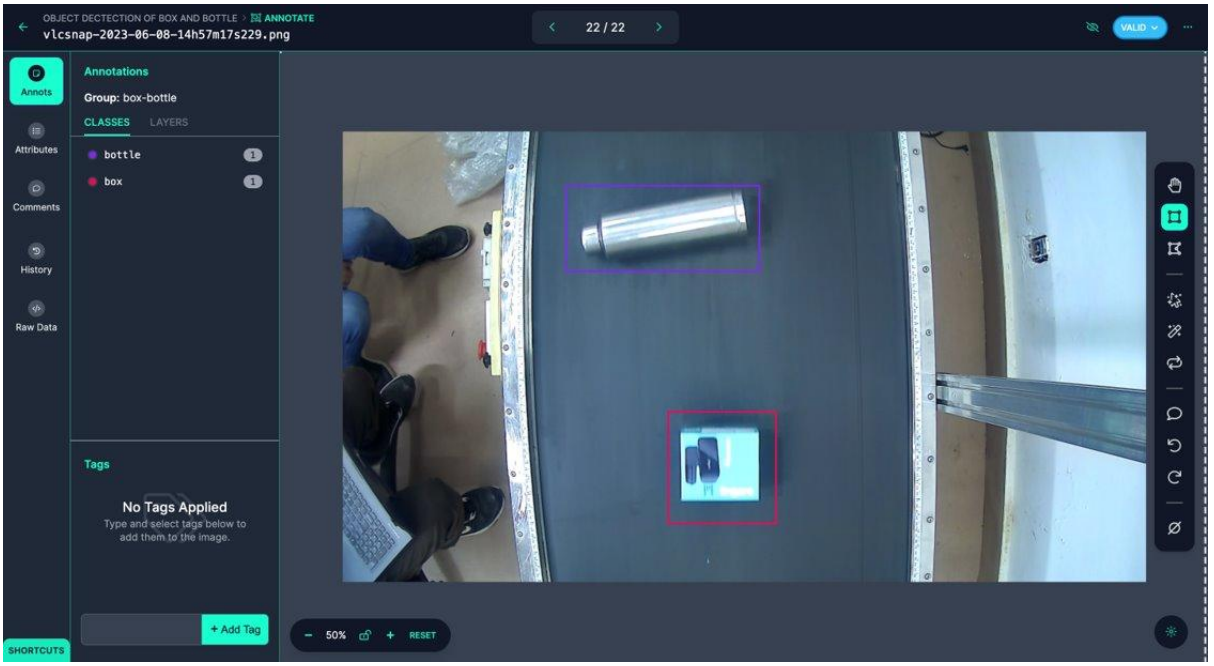
II. Discussion

The YOLO (You Only Look Once) model has proven to be a highly effective deep learning-based object detection algorithm, known for its real-time performance and accuracy. In this project, we used the YOLOv8 model to enhance our conveyor system's object detection and sorting capabilities. We trained the model using a labeled dataset, which included images of boxes and bottles placed at various positions and orientations on the conveyor belt, with variations in shape, size, and color.

To train the YOLOv8 model, we needed a labeled dataset that contains images or video frames with bounding boxes around the objects of interest (boxes and bottles). We collected data by capturing images and videos of objects moving along the conveyor belt. Our dataset encompassed diverse object variations, which is essential for teaching the model to handle different object types commonly encountered in real-world scenarios.

To create the labeled dataset, we used Roboflow, a powerful tool that allows annotators to manually mark the objects of interest in the images or video frames by drawing bounding boxes around them. This annotation process ensured that the model could accurately detect and differentiate between boxes and bottles on the conveyor belt.

***Annotating a sample image in roboflow**



Once we had the labeled dataset, we proceeded with training the YOLOv8 model. During the training process, the model learned to recognize specific features of boxes and bottles, enabling it to perform accurate and real-time object detection. We used a high-performance GPU to accelerate the training process, which significantly reduced the training time.

***Sample training set images from OpenImagesV7 and webcam images (0,1) maps to (Bottle,Box)**

After the model was trained, we evaluated its performance using a separate validation dataset. The model demonstrated excellent accuracy, successfully detecting objects with high precision and recall rates. It efficiently distinguished between different object types, handling variations in shape, size, and color effectively.

Training Dataset size: 15,517
Validation Dataset size: 61

***Sample validation set images from validation set**

With the YOLOv8 model fully trained and evaluated, we integrated it into our conveyor system for real-time object detection. The model continuously analyzed the video stream captured by the high-resolution webcam placed above the conveyor belt. As objects moved along the belt, the model detected and localized them with impressive speed and accuracy.

One of the significant strengths of the YOLOv8 model is its ability to handle diverse object variations. We deliberately introduced variations in the dataset to challenge the model and ensure robust performance. These variations included objects of different shapes, sizes, and colors, mimicking real-world scenarios.

The model successfully learned to adapt to these variations, making it highly versatile in identifying and sorting objects. As a result, the conveyor system's sorting process was seamless, effectively handling different object types encountered in the industrial environment.

The successful integration of the YOLOv8 model into our conveyor system has greatly enhanced the system's capabilities. With real-time object detection and accurate sorting, the conveyor system now operates with maximum efficiency and precision.

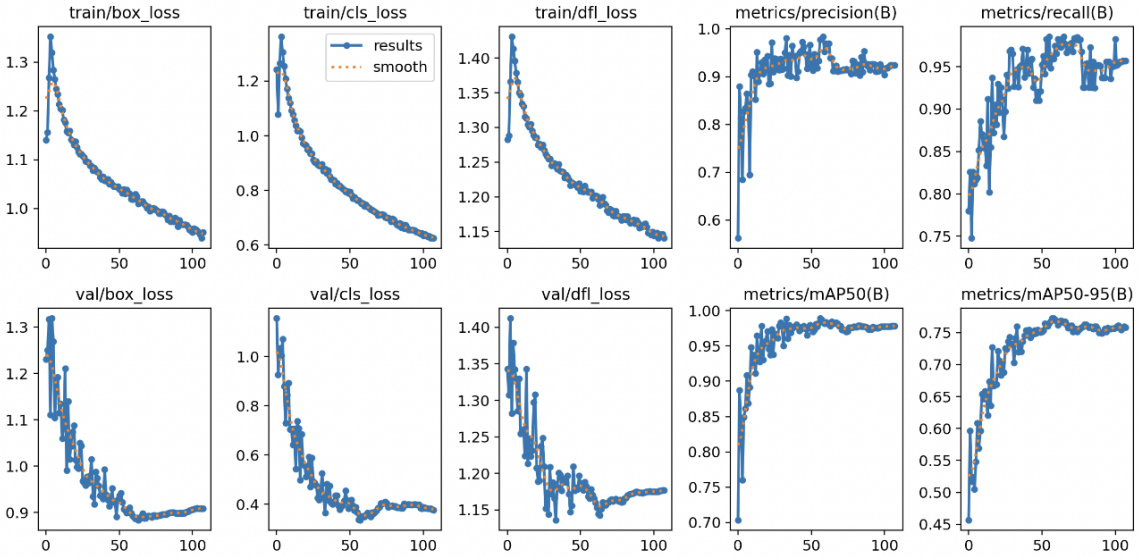
To make the implementation user-friendly, we created an installer using PyInstaller, a tool that packages Python programs into standalone executables. The installer was designed to download all the necessary libraries and dependencies automatically, eliminating the need for users to install them manually. This made the installation process hassle-free and ensured that the program could be executed without any additional configuration.

III. Results

During the training process of the YOLOv8m model, the initial configuration was set to run for 300 epochs, and each epoch had a batch size of 16, resulting in a total of 948 batches for each epoch. However, the training process early stopped at the 104th epoch, which means the model did not complete all 300 epochs as originally planned.

Throughout the 104 epochs of training, the graph below shows the progress of our YOLOv8m model by tracking various metrics. These metrics included the training and validation losses for bounding box regression (box loss), class prediction (cls loss), and object detection confidence loss (dfl loss). We also measured the Mean Average Precision at 50 IoU (mAP50) and Mean Average Precision from 50 to 95 IoU (mAP50-95) at each epoch.

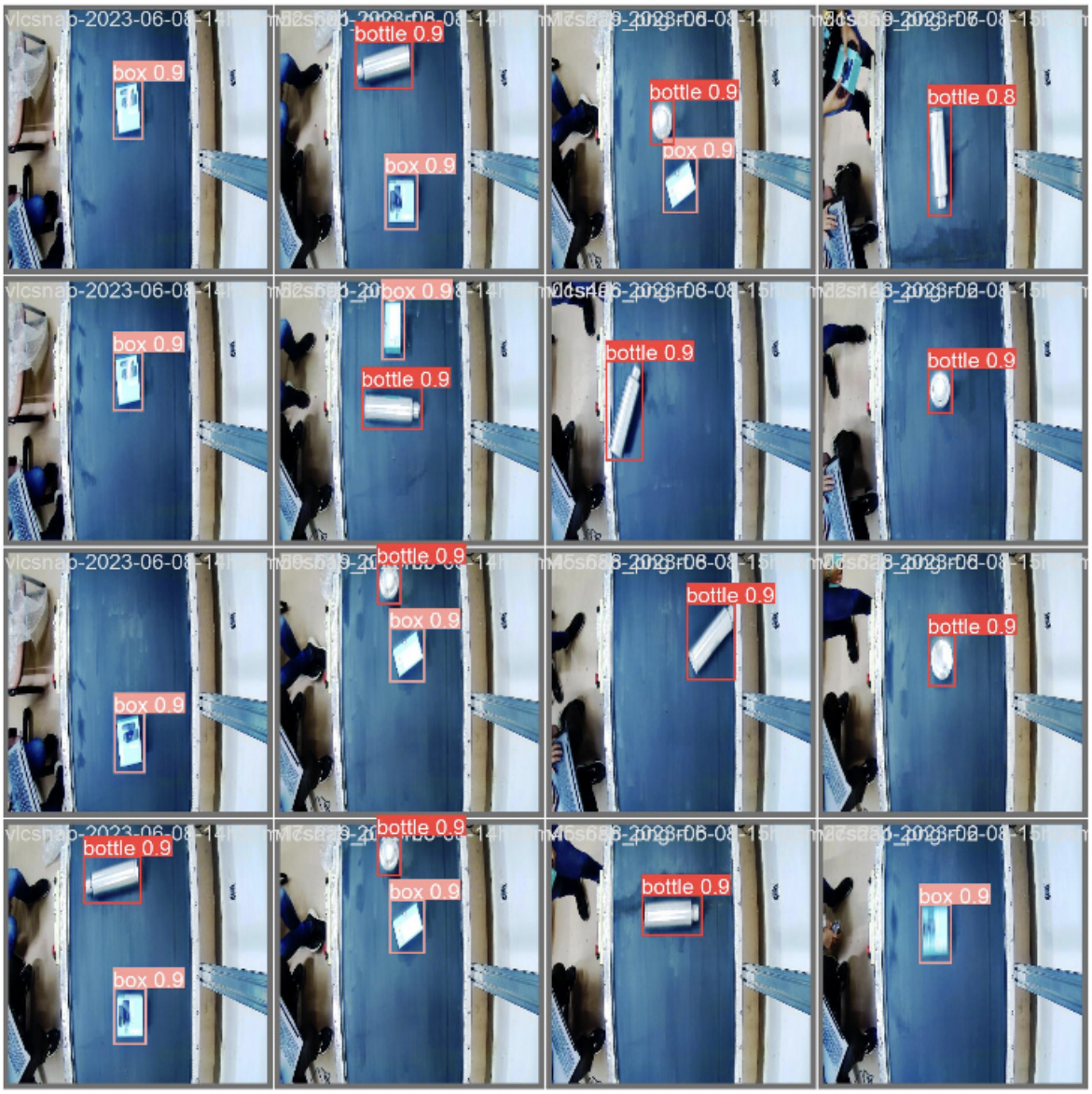
The graphs below illustrate the trends of these metrics over the course of training:



Class	R	mAP50	mAP50-95)
all	1	0.995	0.778
bottle	1	0.995	0.752
box	1	0.995	0.805

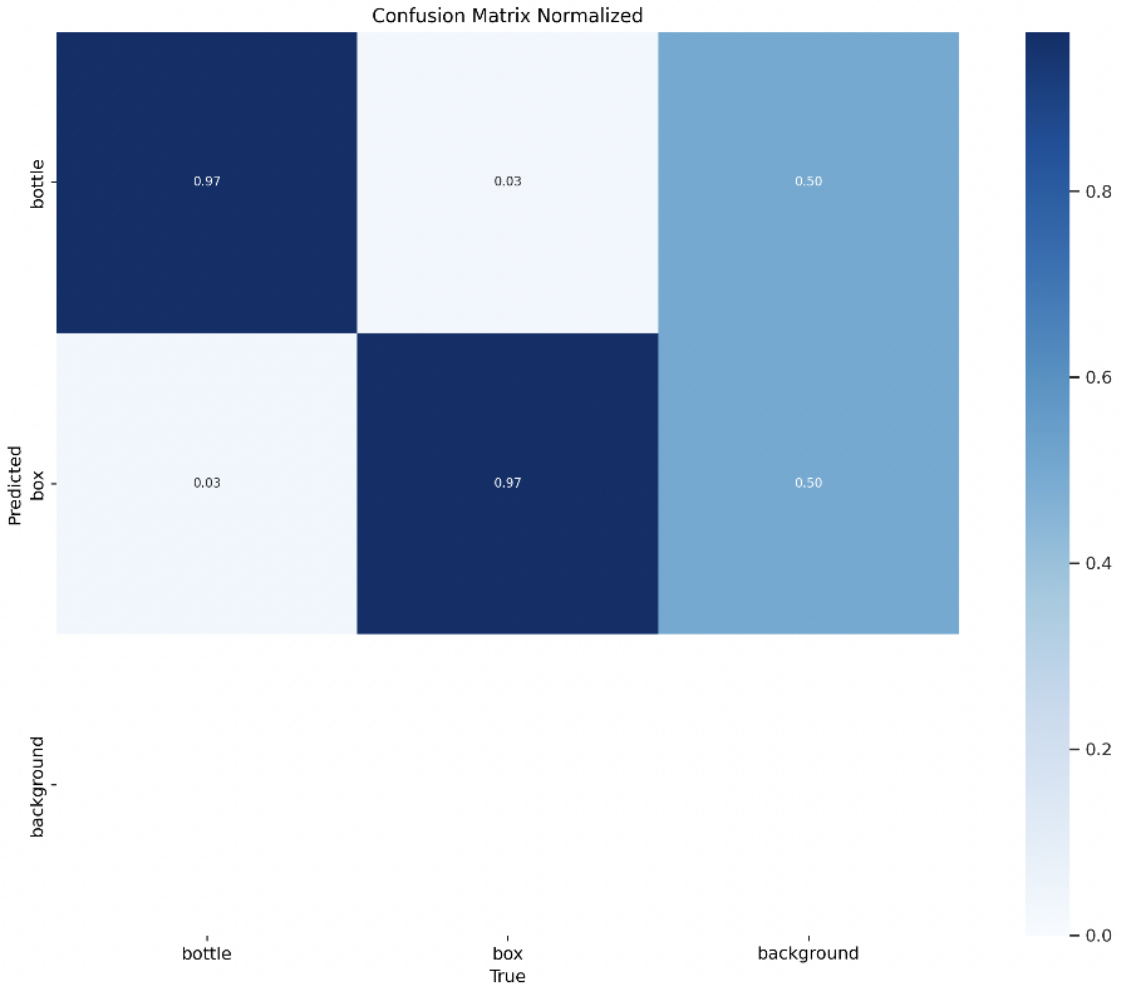
As shown in the graphs, the training loss gradually decreases over the epochs, indicating that the model is learning and improving its performance. The validation loss also follows a similar trend, but it may show fluctuations due to variations in the validation data.

Additionally, we conducted inference on test images using the trained YOLOv8m model. The model demonstrated impressive performance in object detection, accurately identifying boxes and bottles in the test images.

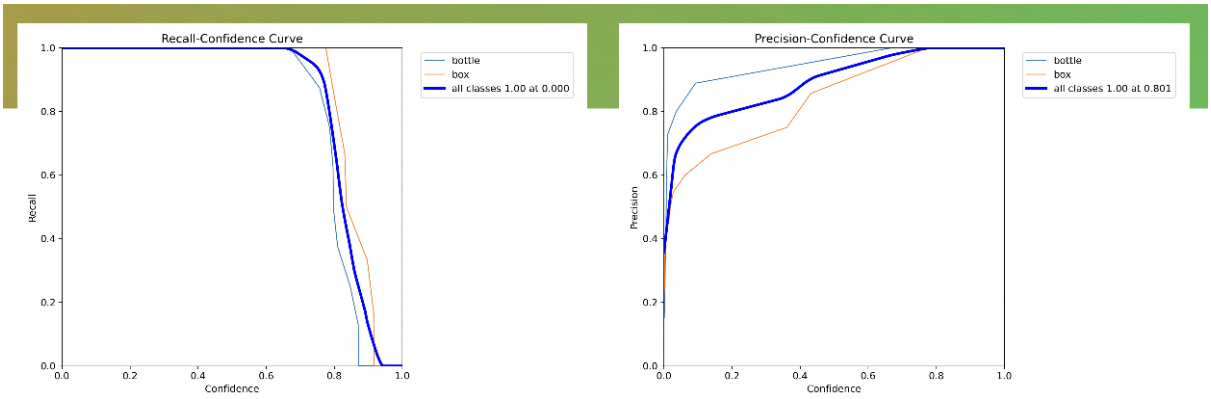

box 0.9
bottle 0.9
box 0.9
bottle 0.9
box 0.9
bottle 0.8
box 0.9
box 0.9
bottle 0.9
bottle 0.9
bottle 0.9
bottle 0.9
box 0.9
bottle 0.9
box 0.9
bottle 0.9
bottle 0.9
box 0.9
bottle 0.9
box 0.9
bottle 0.9
box 0.9

Furthermore, we evaluated the model's performance using a confusion matrix, which summarizes the model's true positive (TP) and false positive (FP) predictions for boxes and bottles. The confusion matrix revealed that the model achieved a remarkable true positive rate of 97% for both boxes and bottles. However, we observed a few false positives, indicating that the model occasionally misclassified other objects as boxes or bottles.

To address this issue, we can implement a technique of using surround images without boxes or bottles during the training process. By exposing the model to images containing various surroundings, it learned to discriminate between the objects of interest and other irrelevant objects, effectively reducing false positive predictions.



To gain further insights into the model's performance, we analyzed the recall confidence and precision confidence curves. The recall confidence curve shows the relationship between the recall (sensitivity) and the confidence threshold, while the precision confidence curve depicts the relationship between precision and the confidence threshold.



- the Precision Confidence Curve shows the precision values at different confidence thresholds. It starts at 0.2 and gradually increases to 1 as the confidence threshold increases.
- the Recall Confidence Curve shows the recall values at different confidence thresholds. It starts at 1 and gradually decreases to 0 at a confidence threshold of 0.9.

As the recall confidence curve demonstrates, the model achieves a high recall rate at low confidence thresholds, indicating that it can effectively detect most of the true positive objects. However, as the confidence threshold increases, the recall rate decreases, as the model becomes more conservative in its predictions. In the recall confidence curve, we observe that the recall starts at 1 and suddenly drops to 0 after a confidence threshold of 0.9. This behavior is due to the model being very strict and the limited number of images in the validation set.

On the other hand, the precision confidence curve shows that the model maintains high precision at high confidence thresholds, minimizing false positive predictions. However, at lower confidence thresholds, the precision drops, as the model becomes more lenient in its predictions.

IV. Conclusion:

In conclusion, our YOLOv8m model demonstrated exceptional object detection capabilities, achieving high accuracy and recall rates for identifying boxes and bottles on the conveyor system. The use of surround images during training can prove effective in reducing false positive predictions. Our analysis of recall and precision confidence curves also provided valuable insights into the model's performance at different confidence thresholds.

The YOLOv8m model proved to be a suitable choice for our project, delivering real-time and accurate object detection results. Its ability to handle diverse object variations, such as shape, size, and color, made it well-suited for the conveyor system's requirements. Overall, our project successfully implemented an intelligent object detection and sorting system, contributing to the enhancement of industrial automation processes.

*Comparison between different YOLOv8 models

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

V. References

1. Ultralytics GitHub Repository: The official GitHub repository of Ultralytics provides valuable resources, documentation, and code for the YOLOv8m model used in this project. It can be accessed at
https://github.com/ultralytics/ultralytics

2. OpenImages Dataset V7: The OpenImages dataset is a large-scale collection of annotated images for object detection and recognition tasks. Version 7 of the dataset was utilized in this project to train and evaluate the YOLOv8m model. The dataset was downloaded with the help of oid-v7-toolkit.
https://github.com/EdgeOfAI/oidv7-Toolkit

3. PyTorch: The deep learning framework PyTorch was employed as the backbone for training and implementing the YOLOv8m model. The official PyTorch website provides documentation, tutorials, and resources for utilizing the framework. It can be accessed at
https://pytorch.org/

4. OpenCV: The OpenCV library played a crucial role in image processing and handling tasks, including webcam input and visualization of results. Its official website offers comprehensive documentation and tutorials for utilizing OpenCV functionalities. It can be found at
https://opencv.org/

5. In the GitHub repository, you can find the project's source code, including the custom-trained weights for the YOLOv8m model. Additionally, the repository may include any relevant documentation, installation instructions, and usage guidelines to help others understand and replicate your work.
https://github.com/Prod23/OBJECT-DETECTION-AND-SORTING-ON-THE-CONVEYOR-SYSTEM

6. Google Colab Pro: Google Colab Pro was used during the development phase of this project to leverage its powerful computing resources and GPU acceleration for training the YOLOv8m model. Colab Pro provides access to faster GPUs, longer runtimes, and priority access to resources. More information about Google Colab Pro can be found at
https://colab.research.google.com/signup