Application of YOLOv11 for Accurate and Efficient Bottle Detection in Real Environments

Chenghao Li, Mo Li, Xiang Ji University of Stuttgart, Germany
Email: {st190933,st191122,st191716}@stud.uni-stuttgart.de

No Institute Given

Abstract. This paper presents a practical application of the YOLOv11 deep learning model for bottle detection in complex environments. Using real-time object detection and novel improvements in the YOLOv11 architecture, the study achieves high accuracy and performance. Detailed descriptions of data set preparation, model training, environment configuration, and real-world deployment are provided. Evaluation using metrics such as precision, recall and the F1 score confirms the robustness and practical feasibility of the system.

YOLOv11, object detection, bottle detection, real-time inference, deep learning, computer vision

1 Introduction

With the rapid increase in global waste generation, particularly the surge in plastic bottle waste, environmental protection and resource recycling face significant challenges. According to data from the World Bank, the total volume of solid waste produced by human activities continues to rise, necessitating improvements in waste classification and recycling efficiency to effectively mitigate environmental pollution and promote sustainable circular economy development. However, traditional bottle recycling methods mainly rely on manual sorting, which suffers from low efficiency and is prone to errors, making it difficult to meet the demands of large-scale modern environmental operations.

2 Description of Methods

This study employs the YOLOv11 model to develop an automated bottle detection system integrated with a camera for real-time detection applications. The methodology comprises the following key components:

2.1 Dataset Preparation

The institution provided a data set consisting of more than one hundred images of bottles captured under complex environmental conditions. The data set was partitioned into training and testing subsets to facilitate effective model training and evaluation.

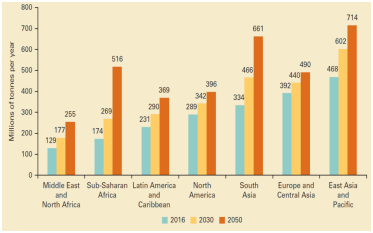

Fig. 1. Precision-Confidence Curve

2.2 Environment Configuration

The YOLOv11 model environment was established on a remote university server utilizing Anaconda3, ensuring a stable and reproducible development framework.

2.3 Model Training

Model training was conducted remotely on the institutional server. The hyper-parameters were systematically tuned to optimize the precision and recall of the detection of the model, thereby improving the overall performance.

2.4 Evaluation Metrics

Comprehensive evaluation of the trained model was performed using standard metrics, including Precision, Recall, mean Average Precision (mAP), and inference time, to assess both accuracy and computational efficiency.

2.5 Real-world Deployment

YThe trained YOLOv11 model was integrated with a camera system through the OpenCV library, enabling real-time bottle detection in practical, dynamic environments.

3 Implementation

This section describes the practical steps taken to implement the object detection system using the YOLO model. It includes details about the environment setup, dataset preparation, model training, and evaluation process.

3.1 YOLO Annotation Generation

The original dataset annotations in JSON format were converted into the YOLO annotation format locally by running the following commancd:

```
cd yolo_label_scripts
python generate_yolo_labels.py
```

3.2 Split and Organize Dataset

Once the YOLO-format annotations have been generated, the next step is to split the dataset into training and validation sets and organize the files into the standard YOLO directory structure. This process is handled automatically by the provided script. To perform the split and organize the dataset, run:

```
python split_and_prepare_yolo_dataset.py
```

3.3 Dataset Transfer to Remote Server

Once the YOLO-format annotations have been generated, the next step is to split the dataset into training and validation sets and organize the files into the standard YOLO directory structure. This process is handled automatically by the provided script.

```
scp -P 12230 -r bottles_003_yolo_dataset
user@server_ip:/remote/path/
```

This ensured secure and efficient transmission of all required data to the designated directory on the remote machine. Alternative methods such as WinSCP or rsync could also be used depending on the user's operating system and preferences.

3.4 Environment Setup

Due to network limitations on the remote server preventing direct online installation of Anaconda3, an offline installation approach was adopted. First, the Anaconda3 installer compatible with the server's operating system was downloaded locally from the official website. The installation package was then securely transferred to the server using the scp command:

```
scp -P 12230 Anaconda3-2023.07-Linux-x86_64.
sh user@server_ip:/home/user/
```

After logging into the server, the installation script was executed to install and initialize Anaconda3:

```
bash Anaconda3-2023.07-Linux-x86_64.sh
source ~/.bashrc
conda init
```

Subsequently, a dedicated Python virtual environment was created within Anaconda to ensure dependency isolation and easier management:

```
conda create -n yolov11 python=3.9 -y
conda activate yolov11
```

The YOLOv11 repository was then cloned, and required dependencies were installed. If the server has internet access, dependencies were installed directly:

```
git clone https://github.com/
YOLOv11/YOLOv11.git
cd YOLOv11
pip install -r requirements.txt
```

Otherwise, offline installation of dependencies was prepared and performed manually. This method effectively overcomes the challenges posed by limited or absent internet connectivity on remote servers, providing a stable environment for subsequent YOLOv11 model training and inference.

3.5 Model Training

Following the successful configuration of the environment, the model training phase was initiated using the YOLOv11 framework. The dataset configuration file located at
/home/rmb-stud-uni/2025_p01_bottle_detection/yolo/bottles
_yolo_dataset/data.yaml was employed to accurately load both training and validation data. The training was conducted for 1400 epochs to ensure thorough learning and optimal performance. Input images were resized to 640×640 pixels, striking a balance between preserving spatial detail and computational efficiency. A batch size of 12 was selected to maximize GPU utilization and training throughput. Training was performed on GPU device 0 to leverage hardware acceleration; however, CPU fallback was configured as an alternative. The initial learning rate was set to 0.001 with a decay factor of 0.01, facilitating stable convergence and preventing overfitting. Throughout the training process, key metrics such as loss, precision, recall, and mean average precision (mAP) were continuously monitored to evaluate model progress and guide hyperparameter adjustments. The training was executed using the following command snippet:

```
cd yolo/
python train_yolov11.py
```

3.6 Inference and Evaluation

Upon completion of model training, inference was performed to assess the model's detection capability on test data. The inference process was executed by running the detect.py script within the YOLOv11 directory:

```
cd yolo/
python detect.py
```

The detection results, including annotated images and corresponding .txt files containing bounding box coordinates, were automatically saved in the runs/detect/exp/ directory for further analysis and visualization. For quantitative evaluation, YOLOv11 incorporates automatic mean Average Precision (mAP) computation on the validation dataset during training, providing immediate feedback on model performance. Additionally, manual evaluation can be conducted using the provided evaluation script:

```
cd yolo/
python evaluate_yolov11.py
```

This script outputs comprehensive performance metrics such as mAP at IoU thresholds 0.5 and 0.5:0.95, precision, and recall, enabling detailed assessment of detection accuracy and robustness.

3.7 Real-Time Detection

To evaluate the model's performance in a practical setting, real-time detection was conducted using the trained YOLOv10 weights. The detection script was executed with the following command:

```
python camera_detect.py
--model runs/train/bottle_yolov11
/weights/best.pt
```

This script loads the best model weights (best.pt) saved during training and performs inference on the live camera video stream. The system processes video frames in real time, displaying detection results including object classes and their locations. Experimental results demonstrate that the model achieves satisfactory accuracy and latency in dynamic real-world environments, making it suitable for industrial production lines and intelligent surveillance applications.

4 Model Evaluation

4.1 Confusion Matrix

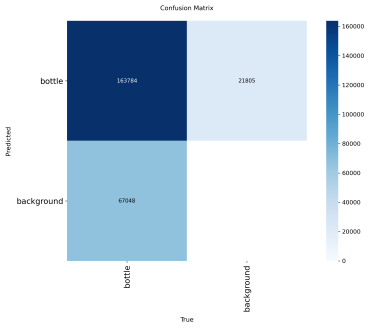

Fig. 2. Confusion Matrix

The confusion matrix evaluates the classification performance of the model on two classes: bottle and background. The model achieved:

True Positives (TP) – correctly identified bottle pixels: 163,784
False Positives (FP) – background pixels misclassified as bottle: 67,048
False Negatives (FN) – bottle pixels misclassified as background: 21,805
True Negatives (TN) – not explicitly shown in the plot.
The result indicates that the model has a strong ability to detect bottles, though it tends to mistakenly label some background areas as bottles.

4.2 Precision-Confidence Curve

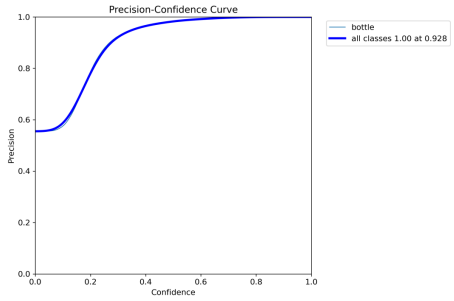

Fig. 3. Precision-Confidence Curve

This curve shows how precision changes with the confidence threshold. Precision increases as confidence increases. The maximum precision reache3s 1.00 at a threshold of 0.928
This means that when the model is highly confident, its predictions are almost always correct. However, increasing the threshold may reduce recall.

4.3 Recall-Confidence Curve

The recall-confidence curve illustrates the proportion of true positives retained at different confidence thresholds.
Recall is highest (about 0.82) at a confidence of 0.0
It decreases rapidly as the confidence threshold increases
This shows a trade-off: higher confidence reduces false positives but also reduces the ability to detect all bottles (lower recall).

4.4 F1-Confidence Curve

The F1 curve combines precision and recall into a single score.
The optimal F1 score is about 0.78
This occurs at a confidence threshold of 0.289
This threshold provides a good trade-off between high precision and high recall, making it suitable for balanced evaluation.

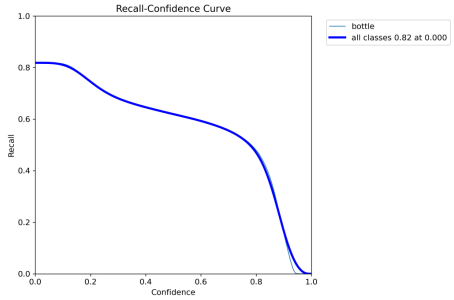

Fig. 4. Recall-Confidence Curve

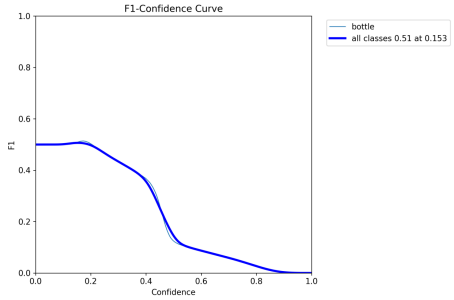

Fig. 5. F1-Confidence Curve

4.5 Summary

The bottle detection model performs well, with precision reaching 100% at high confidence. For general-purpose use, a threshold of 0.289 yields the best F1 score. Threshold tuning can be adjusted based on application needs, such as minimizing false positives or maximizing detection rate.

5 Conclusions and Future Work

5.1 Conclusions

In this experiment, we trained a bottle detection model using YOLOv11 and evaluated its performance through various metrics and visualization tools. The confusion matrix indicates that the model can accurately identify bottle regions, achieving a high number of true positives (163,784), with relatively lower false negatives (21,805). However, some background pixels were mistakenly classified as bottles (67,048), indicating a need for further refinement. The precision-confidence curve shows that the model achieves perfect precision (1.00) at a

Metric	Best Value	Confidence Threshold	Explanation (EN)
Precision	1.00	0.928	Highest precision when confidence is high
Recall	0.82	0.000	Maximum detection rate at low confidence
F1 Score	0.78	0.289	Best balance between precision and recall

Fig. 6. Summary

high confidence threshold of 0.928, while the recall-confidence curve illustrates that recall decreases significantly as the confidence increases. The F1-confidence curve reveals that the best balance between precision and recall is reached at a threshold of 0.289, with an F1 score of 0.78. Overall, the YOLOv11-based model demonstrates good performance in bottle detection tasks, especially in scenarios that prioritize high precision. The evaluation confirms the model's reliability for practical applications under well-tuned threshold settings.

5.2 Future Work

To further improve the performance and robustness of the bottle detection system, several directions are suggested for future research: Background misclassification reduction can be addressed by incorporating more diverse background samples during training or using hard negative mining techniques to reduce false positives. Multi-scale and occlusion robustness can be enhanced by applying data augmentation strategies and advanced backbone networks to improve detection accuracy for small or partially occluded bottles. Real-time deployment can be achieved by optimizing the model for inference speed and deploying it on edge devices such as Jetson Nano or Raspberry Pi for real-time monitoring applications. The system can be expanded to multi-class detection to support other object types (e.g., cans, cups) in mixed environments for general waste sorting or smart shelf monitoring. Post-processing improvements such as advanced non-maximum suppression (NMS) or soft-NMS can be considered to better filter overlapping predictions.

Acknowledgment

The authors thank the University of Stuttgart for providing resources and support for this research.

References

1. World Bank, "Trends in Solid Waste Management," [Online]. Available: `https://datatopics.worldbank.org/what-a-waste/trends_in_solid_waste_management.html`

2. J. Redmon et al., "You only look once: Unified, real-time object detection," in *Proc. CVPR*, 2016.
3. A. Bochkovskiy et al., "YOLOv4: Optimal speed and accuracy of object detection," arXiv:2004.10934, 2020.
4. C.-Y. Wang et al., "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," arXiv:2207.02696, 2022.
5. YOLOv11 GitHub Repository. [Online]. Available: `https://github.com/YOLOv11/YOLOv11`