Brand Recognition and Item Couting + Frontend

Team : ajitesh.kumar

January 13, 2025

Introduction

In the GRID 6.0 competition, our team is leveraging smart vision technology to develop an innovative solution for automating quality testing processes in the e-commerce industry. Our approach combines advanced imaging systems and machine learning algorithms to address critical challenges such as item recognition, packaging integrity verification, expiration date detection, and freshness assessment of produce.

We have already worked on designing and training machine learning models for object detection and OCR, focusing on brand recognition, item counting, and extracting key details like manufacturing and expiration dates. Additionally, we are exploring methods to predict the freshness and shelf-life of perishable items by analyzing visual patterns and cues.

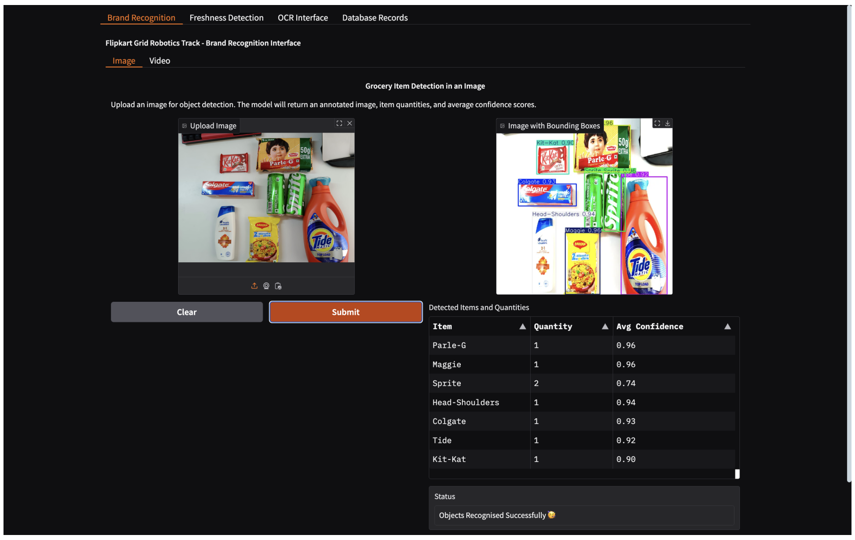
Our goal is to create a system that is accurate, efficient, and scalable while being easy to integrate with existing workflows. By addressing challenges such as environmental variability and computational efficiency, we aim to deliver a practical solution that improves quality control and enhances operational efficiency. Through this project, we strive to showcase our technical expertise, innovation, and ability to solve real-world problems.

Overview

This project implements a brand recognition and item counting system using YOLO (You Only Look Once) object detection models. The system processes input images, detects grocery items, and updates a database with the detection results. It includes features for image preprocessing, object detection, annotation, and database integration.

Modules and Functions

The following sections describe the primary modules and functions in the system.



Flipkart Grid Robotics Track - Brand Recognition Interface

Grocery Item Detection in an Image

Upload an image for object detection. The model will return an annotated image, item quantities, and average confidence scores.

Detected Items and Quantities

Item	Quantity	Avg Confidence
Parle-G	1	0.96
Maggie	1	0.96
Sprite	2	0.74
Head-Shoulders	1	0.94
Colgate	1	0.93
Tide	1	0.92
Kit-Kat	1	0.90

Status

Objects Recognised Successfully 🥳

Figure 1: Brand recognition and Item Counting

Brand Recognition and Item Counting in Videos

In addition to processing static images, the system is designed to handle brand recognition and item counting in video streams. By leveraging advanced algorithms, it ensures accurate detection and tracking of objects across video frames.

Key Features and Techniques

- **Intersection over Union (IoU):** IoU is employed to measure the overlap between predicted bounding boxes and ground truth, ensuring robust object tracking across frames. This technique helps in reducing redundant detections and improves accuracy in scenarios where multiple objects are closely positioned.
- **Frame Adjustment:** The system dynamically adjusts video frames to optimize detection performance. By skipping or selectively processing frames, it balances computational efficiency with detection accuracy, making it suitable for real-time applications.
- **Temporal Consistency:** Temporal information is utilized to track objects across frames, ensuring consistent detection results. This prevents the system from treating the same object in successive frames as separate detections.

- **Real-Time Processing:** The video processing pipeline is optimized for speed, enabling near real-time detection and annotation of video feeds. This is particularly useful in applications such as inventory management and quality control in production lines.

Workflow for Video Processing

1. **Video Frame Extraction:** Frames are extracted from the video feed at a predefined interval to ensure efficient processing.
2. **Object Detection on Frames:** Each extracted frame is passed through the YOLO model for brand recognition and item counting.
3. **Tracking and Aggregation:** Detected items are tracked across frames using IoU and temporal consistency techniques. Results are aggregated to provide a comprehensive summary of detected objects in the video.
4. **Output Generation:** Annotated video frames and a detailed summary of brand counts and confidence scores are produced as outputs.

By extending the system to video processing, this solution demonstrates its versatility and effectiveness in handling dynamic data sources. It is particularly valuable for applications requiring continuous monitoring, such as automated quality control in warehouses and retail environments.

1. `preprocess_image`

Purpose: Prepares input images for object detection by resizing and enhancing them.

Arguments:

- `image`: Input image in BGR format (`numpy.ndarray`).
- `input_size`: Target image dimensions (default is 640x640 pixels).
- `augment`: Boolean flag to apply data augmentation (default is `False`).

Returns: Preprocessed image in RGB format (`numpy.ndarray`).

Process:

- Converts the image from BGR to RGB.

Algorithm 1 Brand Recognition and Item Counting

```
 1: Input: Image/Video, YOLO Model, Confidence Threshold
 2: Output: Annotated Image/Video, Detection Summary
 3: Step 1: Preprocess Input
 4: if Image then
 5:     Resize and enhance the image
 6: else if Video then
 7:     Extract and adjust video frames
 8: end if
 9: Step 2: Detect Objects
10: if Image then
11:     Run YOLO on the image
12: else if Video then
13:     for each frame do
14:         Run YOLO and apply IoU for object tracking
15:     end for
16: end if
17: Step 3: Annotate and Aggregate Results
18: if Image then
19:     Annotate image and calculate item counts
20: else if Video then
21:     Annotate frames and aggregate counts across frames
22: end if
23: Step 4: Update Database and Generate Output
24: Update database with detected item counts and return annotated im-
    age/video with summary
```

- Resizes the image to the target dimensions using nearest-neighbor interpolation.
- Enhances the image by adjusting brightness and contrast.

2. `detect_grocery_items`

Purpose: Detects grocery items in a given image using a YOLO model.

Arguments:

- `image`: Input image in BGR format.
- `model_path`: Path to YOLO model weights (default is `Weights/kitkat_s.pt`).
- `threshold`: Confidence threshold for detection (default is 0.5).

Returns:

- Annotated image in RGB format (`numpy.ndarray`).
- Summary table containing detected item names, counts, and average confidences.
- Status message indicating success or errors.

Process:

1. Validates the input image.
2. Loads the YOLO model from the specified path.
3. Preprocesses the input image.
4. Runs object detection on the processed image.
5. Filters detected objects based on the confidence threshold.
6. Generates an annotated image with bounding boxes.
7. Aggregates detection results into a summary table.
8. Updates the database asynchronously with detected item counts.

3. `batch_detect_grocery_items`

Purpose: Processes multiple images for grocery item detection.

Arguments:

- `images`: List of input images in BGR format.
- `model_path`: Path to YOLO model weights (optional).
- `threshold`: Confidence threshold for detection (default is 0.5).

Returns: List of detection results for each input image.

Process:

- Iterates through the list of images.
- Calls `detect_grocery_items` for each image.

Key Features

- **Advanced Preprocessing:** Enhances image quality for better detection accuracy.
- **Asynchronous Database Updates:** Uses threading to handle database operations without affecting detection performance.
- **Dynamic Thresholding:** Filters detection results based on user-defined confidence thresholds.
- **Batch Processing:** Supports detection on multiple images in a single execution.

Applications

This system is designed for:

- Brand recognition in retail and e-commerce.
- Automated item counting in inventory management.
- Enhancing quality control through accurate object detection.

Conclusion

The project demonstrates an efficient method for detecting and counting grocery items using advanced object detection techniques. With its modular design, the system is highly adaptable for real-world applications in e-commerce, inventory management, and quality control.

Frontend Implementation with Gradio

The entire frontend of the project is developed using **Gradio**, an open-source Python library for building user-friendly interfaces for machine learning models and applications. This choice simplifies the interaction between users and the backend, providing an intuitive interface to upload images, view detection results, and monitor the system's performance.

Key Features of the Frontend

- **Interactive UI:** The Gradio interface allows users to upload images or batches of images directly, with real-time visualization of detection outputs and summaries.
- **Scalability:** By leveraging Gradio's `queue` feature, the system is designed to handle multiple user requests efficiently, ensuring smooth operation even under high traffic.
- **Ease of Integration:** Gradio seamlessly integrates with the backend YOLO detection pipeline, maintaining low latency and high accuracy.
- **Cross-Platform Accessibility:** The frontend is accessible via any web browser, making the system platform-independent.

Scalability with the queue Feature

Gradio app comes with a built-in queuing system that can scale to thousands of concurrent users. Because many of your event listeners may involve heavy processing, Gradio automatically creates a queue to handle every event listener in the backend. Every event listener in your app automatically has a queue to process incoming events.

How the Queue Works:

- When a user uploads an image, the request is added to the queue.
- Each request is processed sequentially, ensuring resource optimization and preventing server overload.
- Users are notified about the status of their requests and can view results as soon as processing is complete.

Frontend Workflow

1. **Image Upload:** Users upload one or more images through the Gradio interface. Supported formats include JPG, PNG, and BMP.
2. **Real-Time Detection:** The uploaded images are passed to the backend YOLO model for object detection.
3. **Visualization:** The annotated images, along with detection summaries (class names, counts, and confidences), are displayed in the frontend.
4. **Database Integration:** Detected items are logged into the database in real-time, ensuring data consistency and availability for future analysis.
5. **Error Handling:** Clear and descriptive messages are shown in case of invalid inputs or detection errors, enhancing user experience.

Advantages of Using Gradio

- **User-Friendly:** Requires minimal setup and coding effort, allowing for rapid deployment of interactive interfaces.
- **Scalable Design:** Handles large workloads effectively through the queue mechanism.
- **Customizability:** Offers extensive options for interface customization to suit specific application needs.
- **Open-Source:** Freely available, with active community support for enhancements and troubleshooting.

This Gradio-based frontend ensures an intuitive and scalable platform, meeting both usability and performance requirements of the project.