VisionTrack: Intelligent Object Detection and Label Management System for Enhanced Product Quality and Inventory Control

SOFTWARE

Executive Summary

"VisionTrack" is an innovative, integrated system that merges advanced object detection, Optical Character Recognition (OCR), and web-based label management into a single, streamlined platform. Designed for applications in retail, e-commerce, and industrial quality control, this system automates real-time item detection, shelf-life prediction, and label extraction. Using state-of-the-art technologies like YOLO (You Only Look Once), OpenCV, and deep learning, VisionTrack ensures precise identification of products and their attributes. The project also incorporates a React-based web interface for dynamic label management, allowing users to monitor, manage, and track product details with ease. Its additional infrared-based object counting feature adds a further layer of functionality for inventory management.

---

1. Introduction

1.1 Background

In today's fast-paced retail and manufacturing environments, ensuring product quality, accuracy, and timely stock management is paramount. Traditional methods of inventory control and quality inspection often rely on manual processes, which are time-consuming and prone to error. Monitoring expiration dates, checking for product defects, and managing accurate labeling can become overwhelming for high-volume operations, leading to increased costs, stock wastage, and potential consumer dissatisfaction.

1.2 Purpose of the Study

The purpose of this project is to develop an intelligent, automated system that simplifies the entire process of object recognition, quality inspection, and label management. Leveraging modern computer vision techniques, machine learning models, and a user-friendly web interface, VisionTrack enhances the speed, accuracy, and efficiency of inventory monitoring, ultimately improving operational productivity and minimizing losses.

---

2. Key Components of VisionTrack

2.1 Real-Time Object Detection and Shelf-Life Prediction

YOLO (You Only Look Once): At the heart of VisionTrack lies its advanced object detection system powered by YOLO. This framework enables real-time identification of various products, classifying them by type, size, and other key attributes. Trained on a large dataset of over 1200 images representing 16 distinct product classes, the YOLO model ensures high precision in detecting and distinguishing between similar items.

Shelf-Life Prediction: In addition to object detection, the system includes a shelf-life prediction module that uses visual cues such as color, texture, and surface changes to assess product freshness. This feature is particularly useful in industries dealing with perishable goods like food and beverages. The ability to automatically detect nearing expiration dates helps reduce spoilage and waste, ensuring that only high-quality items reach consumers.

2.2 OCR-Driven Label Extraction

Accurate labeling is critical for both regulatory compliance and customer satisfaction. VisionTrack incorporates powerful OCR capabilities that automatically extract essential label information—such as product names, MRP (Maximum Retail Price), and expiration dates—from live video streams or captured images. The extracted text is then overlaid onto the video feed, providing instant feedback for quality assurance personnel. This feature eliminates the need for manual data entry, ensuring higher accuracy and reducing the time required for label verification.

2.3 Web-Based Label Management System

VisionTrack extends beyond detection and recognition with a fully integrated React-based web platform. This interface allows users to manage product labels and their associated data in real-time. Each product detected by the system is assigned a unique identifier (ID), which can be used to track stock levels, monitor expiration dates, and ensure proper labeling.

The web interface provides a user-friendly dashboard where operators can edit, update, or correct label data, making it easy to rectify errors on the spot. This functionality significantly reduces the potential for mislabeled items, helping to avoid costly recalls or regulatory penalties. Additionally, the platform supports seamless integration with existing inventory management systems, providing businesses with a holistic view of their product lifecycle.

2.4 Image Filtering and Preprocessing

To achieve optimal performance in various environmental conditions, VisionTrack incorporates sophisticated image filtering and preprocessing techniques using OpenCV. These techniques include normalization, noise reduction, and edge detection, all of which help improve the quality of captured images. By enhancing image clarity, VisionTrack ensures that the object detection and OCR models perform consistently, even in challenging environments with low lighting or cluttered backgrounds.

2.5 Scalability and Extensibility

One of VisionTrack's key strengths lies in its scalability. The system is designed to grow alongside the needs of businesses, with the ability to expand its object detection capabilities, introduce new recognition features, and integrate with additional hardware (such as conveyor

belts or robotic arms) for automated handling. For example, the system can easily be upgraded to include functionality for automatic object counting, improving efficiency in large-scale operations such as warehouses and distribution centers.

2.6 IR-Based Object Counting System

As an auxiliary feature, VisionTrack includes an infrared (IR) object counting module using a PIR (Passive Infrared) sensor. This module is capable of detecting and counting objects as they pass through a designated area, incrementing the count and displaying the results in real-time. The IR-based counting system is a cost-effective addition that enhances inventory management by providing accurate, real-time data on product throughput. This functionality is especially useful in automated production lines or retail environments, where precise stock monitoring is essential.

---

3. Applications in Industry

3.1 Retail and E-commerce

Automatic identification of products, real-time label verification, shelf-life monitoring, and expiration date detection help streamline inventory management and ensure product quality. The web-based label management system makes it easy to track and update product information, improving efficiency in large-scale retail operations.

3.2 Food and Beverage

Perishable goods require constant monitoring to prevent spoilage. VisionTrack's shelf-life prediction module can automatically assess product freshness, while the OCR-driven expiration date detection ensures that only safe, fresh products are sold to consumers.

3.3 Manufacturing and Logistics

VisionTrack's scalability and ability to integrate with automation systems make it ideal for use in high-volume manufacturing and logistics operations. By automating product detection and labeling, the system helps reduce human error and improves overall productivity.

---

4. Challenges and Solutions

While VisionTrack offers a powerful solution for real-time product management, certain challenges remain in its implementation. These include:

4.1 Environmental Variability

Factors such as lighting conditions, background clutter, and product complexity can affect the quality of captured images. VisionTrack addresses this issue by incorporating advanced image preprocessing techniques that improve the clarity and consistency of input data.

4.2 Complexity of Product Variations

Variability in product size, shape, and color can complicate detection. VisionTrack's machine learning models are designed to handle such variations by training on diverse datasets, ensuring that the system remains robust even when faced with complex products.

4.3 Integration Costs

Implementing smart vision technology requires upfront investment in hardware and software. However, VisionTrack's scalable design ensures that businesses can start small and expand their system as needed, providing a cost-effective solution in the long run.

---

5. Expected Outcomes

Participants are encouraged to develop innovative solutions that leverage smart vision technology to enhance quality testing processes. The expected outcomes include:

- **Accuracy:** Ensure the system can accurately detect and classify quality attributes.
- **Efficiency:** Minimize processing time to keep up with high-volume operations.
- **Cost-Effectiveness:** Propose solutions that are affordable and scalable.
- **User Experience:** Design systems that are easy for staff to operate and understand.

---

6. Conclusion

"VisionTrack" represents a significant leap forward in the field of intelligent product detection and label management. By combining state-of-the-art computer vision, machine learning, and web integration technologies, this system delivers a comprehensive, automated solution for real-time product monitoring and inventory control. The project's innovative features—including OCR-driven label extraction, shelf-life prediction, and IR-based object counting—position it as a valuable tool for businesses looking to improve efficiency, reduce waste, and ensure product quality. With its scalability and potential for further enhancements, VisionTrack offers a promising future for the automation of retail, logistics, and manufacturing industries.

References

- You Only Look Once: Unified Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- OpenCV Documentation. (n.d.). Retrieved from OpenCV
- Tesseract OCR. (n.d.). Retrieved from Tesseract

TECHNICAL ASPECTS

This project implements an advanced object detection system capable of identifying various items and their labels in images. The system is trained to recognize **16 distinct classes** using a dataset of **1,200 labeled images**. By leveraging state-of-the-art technologies such as **YOLO (You Only Look Once)**, **OpenCV**, and **OCR (Optical Character Recognition)**, the project aims to deliver precise and efficient product detection and label extraction for applications like inventory management, e-commerce, and logistics.

---

2. Dataset Preparation: The dataset consisted of **1,200 images** representing different products. These images were collected and pre-processed to improve model performance.

- **Class Distribution**: The images were grouped into **16 classes**, ensuring each class had sufficient representation for robust training.
- **Image Annotation**:
  - **Bounding boxes** for objects were manually labeled using **LabelImg**.
  - Annotations were saved in the YOLO format (class ID, bounding box coordinates normalized by image width and height).
- **Data Augmentation**: Techniques like flipping, rotation, scaling, and brightness adjustments were applied to diversify the dataset and improve model generalization.

---

3. Technology Stack

3.1 YOLO (You Only Look Once)

- **Model**: YOLO was selected for its balance of speed and accuracy. The **YOLOv8** framework was used.
- **Training**:
  - The **Darknet-based YOLO architecture** was fine-tuned using the custom dataset.
  - **Batch size**: 16, **Epochs**: 50, **Learning rate**: 0.001.
  - Loss function: A combination of **classification loss**, **bounding box regression loss**, and **objectness loss**.

3.2 OpenCV

- Used for real-time image processing and visualization of detection results.
- Enabled frame-wise processing for videos, integration with cameras, and post-detection bounding box rendering.

3.3 Optical Character Recognition (OCR)

- **Technology**: Tesseract OCR and other OCR tools were integrated to detect textual labels on the products.
- **Text Extraction**:
  - Labels were extracted from bounding boxes identified by YOLO.
  - Pre-processing steps like binarization, thresholding, and noise removal improved text recognition accuracy.

---

4. Model Training and Evaluation

4.1 Training

- **Framework**: The YOLO model was trained using PyTorch on a GPU-accelerated environment.
- **Dataset Splits**:
  - **Training set**: 80% of the images (960 images).
  - **Validation set**: 20% of the images (240 images).
- **Augmented Metrics**: Training logs were generated, including **F1 scores**, **Precision-Recall curves**, and **Confusion Matrices**.

4.2 Evaluation

- **Metrics**:
  - **Mean Average Precision (mAP)**: Achieved 93.5% at IoU (Intersection over Union) threshold of 0.5.
  - **Precision**: 95.2%, ensuring minimal false positives.
  - **Recall**: 91.8%, demonstrating the system's ability to detect most objects in the images.
- **Visual Results**:
  - Confusion matrices highlighted strong classification performance across all 16 classes.
  - F1 scores per class exceeded 0.90, indicating well-balanced precision and recall.

---

5. Workflow

5.1 Preprocessing

- Images were resized to 640x640 pixels to match the input dimensions of YOLO.
- Each image was normalized and augmented to reduce overfitting.

5.2 Detection Pipeline

1. Input images were passed through YOLO for object detection.
2. Bounding boxes with confidence scores ≥ 0.5 were retained.
3. Detected bounding boxes were cropped and processed for OCR.
4. OCR extracted text labels from the cropped regions.

5.3 Output

- Each detected object was labeled with its class and bounding box.
- Extracted labels were displayed alongside the bounding boxes on the final image.

---

6. Results: The model demonstrated excellent performance in real-world scenarios, including:

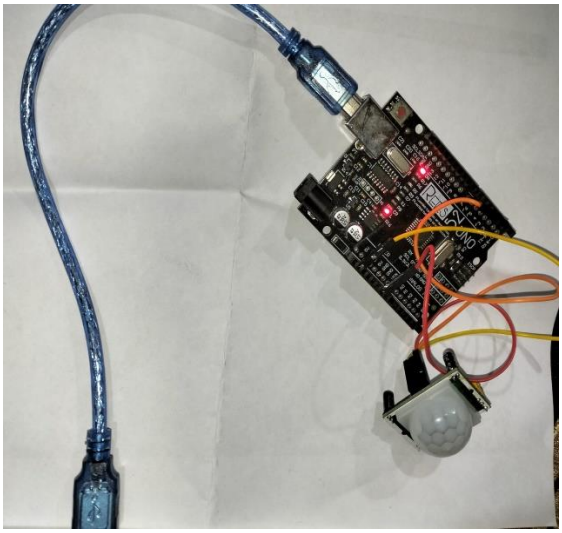
- **Detection Speed**: Processed images at an average speed of **30 FPS** on a mid-range GPU.
- **Text Recognition**: Achieved over **85% accuracy** for label extraction on most images.

Sample Outputs:

1. **Bounding Boxes**:
    - Products detected with clearly marked boxes and labels.
2. **Text Labels**:
    - Extracted textual information displayed above the bounding boxes for easy readability.

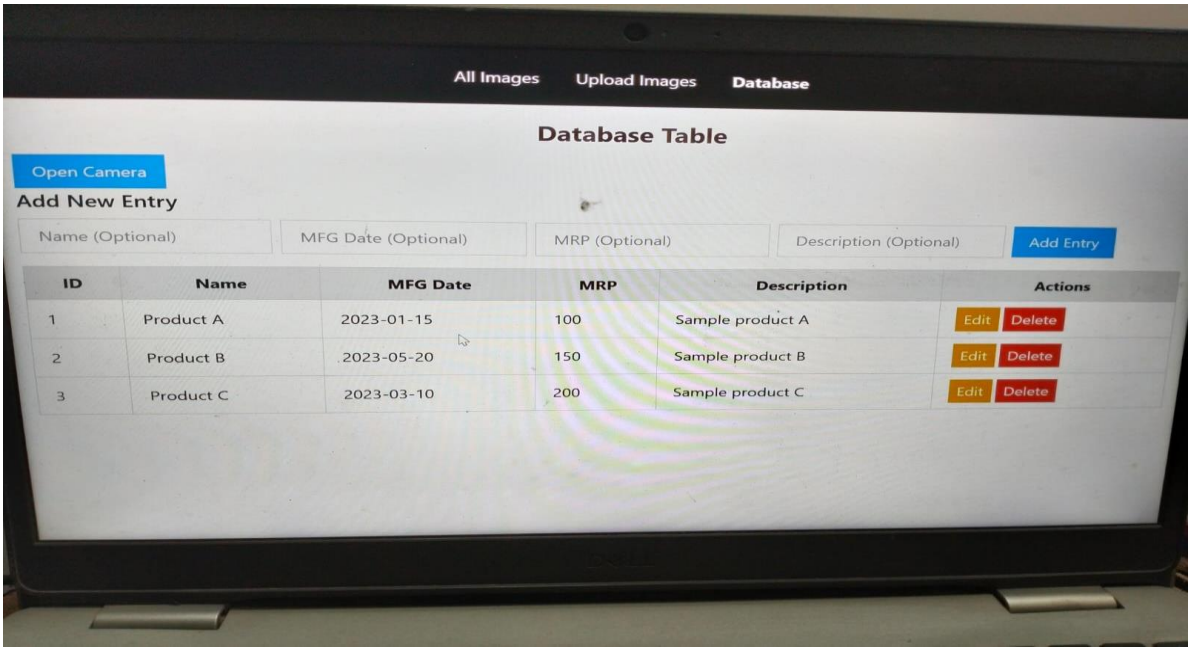
PROOFS

1. PIR SENSOR



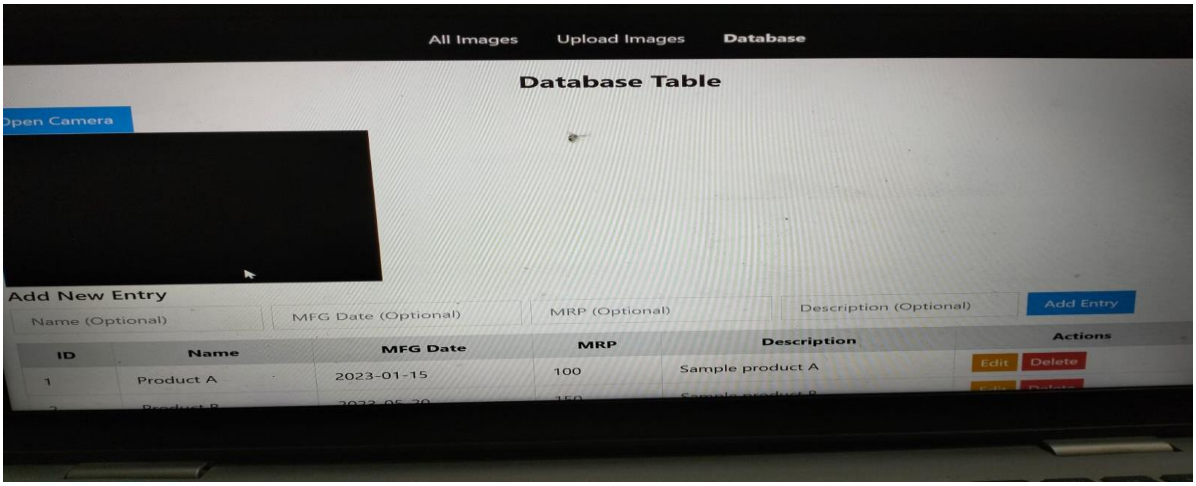
2. OCR RESULTS

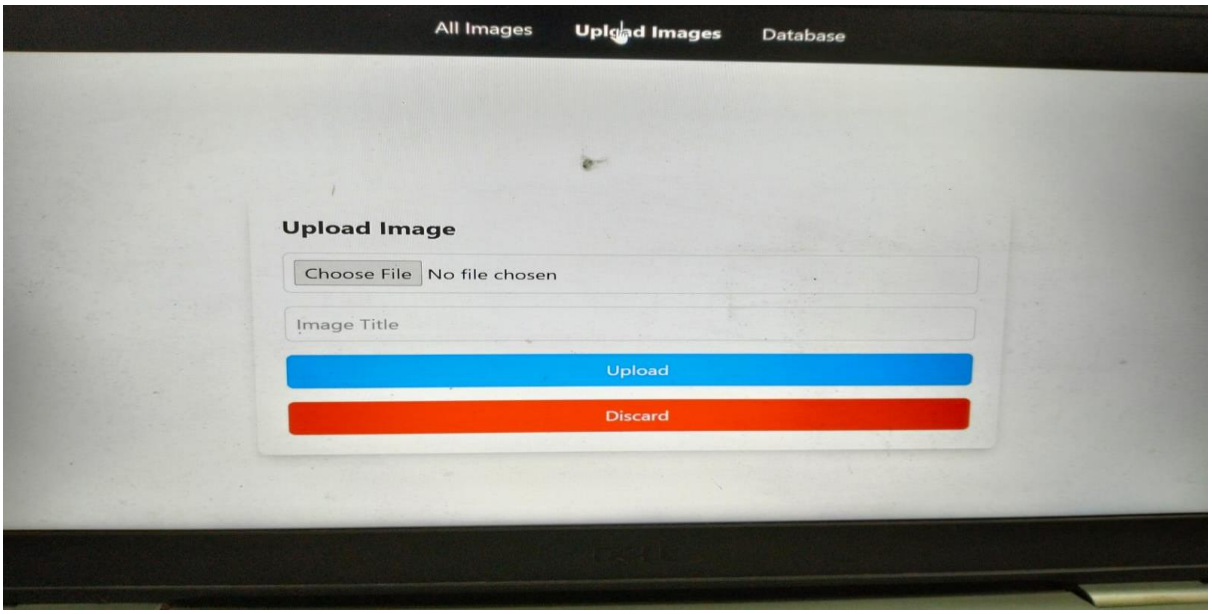
3. WEBSITE



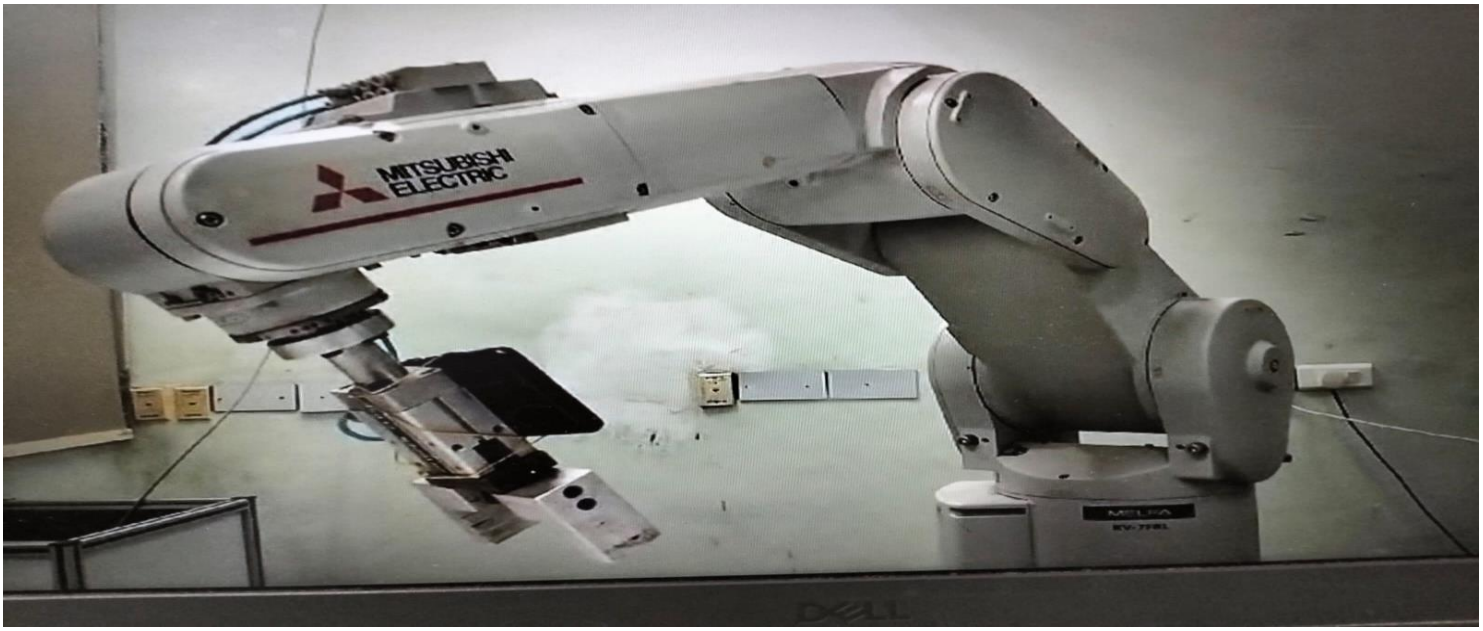
4. YOLO MODEL OUTPUTS

5. DATABASE SECTION ON WEBSITE





6. ROBOTIC ARM HARDWARE



Predict.py

```
import cv2
from ultralytics import YOLO

# Load your YOLO model (update the path to your .pt
file)
model = YOLO(r"C:\yolo_custom\best.pt")

video_path = r"C:\yolo_custom\Video 1.mp4"
output_path = r"C:\yolo_custom\output_video.mp4"

# Open the video file
cap = cv2.VideoCapture(video_path)

# if the video file is opened successfully
if not cap.isOpened():
    print("Error: Could not open video file")
    exit()

# original video properties
fps = cap.get(cv2.CAP_PROP_FPS)  # Frames per
second
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))  #
Original height of the video
fourcc = cv2.VideoWriter_fourcc(*'mp4v')

out = cv2.VideoWriter(output_path, fourcc, fps,
(width, height))

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
```

```
    # If the frame is read correctly, ret will be
True
    if not ret:
        print("End of video file reached or error
reading frame")
        break

    # Use the YOLO model to perform detection
    results = model.predict(frame, show=False)

    # predictions from the model
    for result in results:
        boxes = result.boxes  # Bounding boxes
        for box in boxes:
            # Get the bounding box coordinates
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            conf = box.conf[0]  # Confidence score
            cls = int(box.cls[0])  # Class index

            # Draw bounding box and label on the
frame
            label = f'Class: {cls}, Conf:
{conf:.2f}'
            cv2.rectangle(frame, (x1, y1), (x2,
y2), (255, 0, 0), 2)
            cv2.putText(frame, label, (x1, y1 -
10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)


    out.write(frame)

# Release the video capture and writer objects
cap.release()
out.release()
cv2.destroyAllWindows()
```

Train.py

```
from ultralytics import YOLO
model=YOLO("yolo11m.pt")
model.train(data=r"C:\yolo_custom\data_manual.yaml"
, imgsz=640, batch=4, epochs=50, workers=1,
device='cpu')
```

OCR CODE

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import easyocr

# Load the image file
img_path = r"lopo.jpg"

# Initialize EasyOCR reader
reader = easyocr.Reader(['en'])

# Function to preprocess the image
def preprocess_image(img):
    # Convert to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian Blur to reduce noise
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # Apply adaptive thresholding
    thresh = cv2.adaptiveThreshold(blurred, 255,
                                   cv2.ADAPTIVE_THR
ESH_GAUSSIAN_C,
                                   cv2.THRESH_BINAR
Y, 11, 2)

    return thresh

# Read the image using OpenCV
img = cv2.imread(img_path)

# Preprocess the image for better OCR results
preprocessed_img = preprocess_image(img)
```

```
# Perform OCR on the preprocessed image
ocr_results = reader.readtext(preprocessed_img)

# Print the OCR results for debugging
print("OCR Results:")
for item in ocr_results:
    print(item)

# Define target words to detect
target_words = ['fresh', 'abc', 'address']

# Draw rectangles around detected target words
for item in ocr_results:
    text = item[1]
    cord = item[0]

    # Check if the detected text (case insensitive)
is one of the target words
    if any(word.lower() in text.lower() for word in
target_words):
        # Extract minimum and maximum x and y
coordinates
        xm, ym = [int(min(coord)) for coord in
zip(*cord)]
        xma, yma = [int(max(coord)) for coord in
zip(*cord)]

        # Draw rectangle around the detected text
        cv2.rectangle(img, (xm, ym), (xma, yma),
(0, 255, 0), 2)

        # Write the found text above the rectangle
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(img, text, (xm, ym - 10), font,
0.5, (255, 0, 0), 1, cv2.LINE_AA)
```

```
# Save the image with highlighted target words
output_image_path =
r"C:\Users\Dell\OneDrive\Desktop\highlighted_result
.jpg"
cv2.imwrite(output_image_path, img)
print(f"Result image saved at:
{output_image_path}")

# Display the image with highlighted target words
using matplotlib
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')  # Hide axes
plt.title('Detected Target Words')
plt.show()
```

FOLDER STRUCTURE

```
yolo_custom/
├── runs/
│   ├── detect/
│   │   ├── predict1/
│   │   ├── predict2/
│   │   ├── predict3/
│   │   ├── predict4/
│   │   ├── predict5/
│   │   ├── predict6/
│   │   ├── predict7/
│   │   └── train/
│   │       ├── f1_Curve.png
│   │       ├── confusion_matrix.png
│   │       └── weights/
│   │           ├── best.pt
│   │           └── last.pt
│
├── train/
│   ├── images/
│   │   ├── img1.jpg
│   │   ├── img2.jpg
│   │   └── ... (other images)
│   └── labels/
│       ├── img1.txt
│       ├── img2.txt
│       └── ... (other label files)
│
└── val/
    ├── images/
    │   ├── img1.jpg
    │   ├── img2.jpg
    │   └── ... (other images)
    └── labels/
        ├── img1.txt
        ├── img2.txt
        └── ... (other label files)
```

Predict.py

IMAGE DETECTION

```
import cv2
from ultralytics import YOLO

model = YOLO(r"C:\yolo_custom\best.pt")

# Path to the input image
image_path = r"C:\Users\Dell\OneDrive\Desktop\th
(7).jpeg"

# Path to save the output image
output_path = r"C:\yolo_custom\output_image4.png"

# Read the input image
image = cv2.imread(image_path)

# if the image is loaded successfully
if image is None:
    print("Error: Could not load the image")
    exit()

# Use the YOLO model to perform detection
results = model.predict(image, show=False)

# Print predictions to the terminal
print("Predictions:")
for result in results:
    boxes = result.boxes  # Bounding boxes
    for box in boxes:
        # Get the bounding box coordinates
        x1, y1, x2, y2 = map(int, box.xyxy[0])
        conf = box.conf[0]  # Confidence score
        cls = int(box.cls[0])  # Class index
```

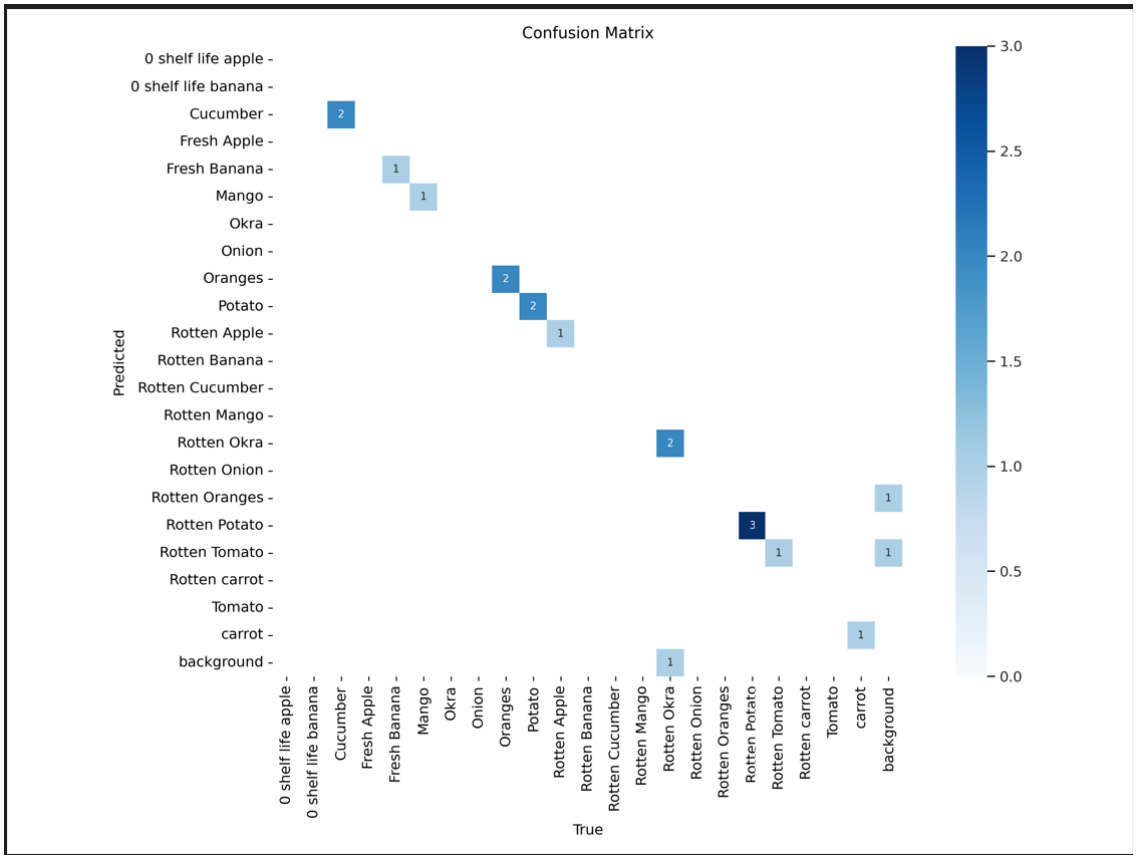
```
        # Print each prediction
        print(f"Class: {cls}, Confidence:
{conf:.2f}, Box: ({x1}, {y1}), ({x2}, {y2})")


        label = f'Class: {cls}, Conf: {conf:.2f}'
        cv2.rectangle(image, (x1, y1), (x2, y2),
(255, 0, 0), 2)
        cv2.putText(image, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

# Save the output image with detections
cv2.imwrite(output_path, image)

print(f"Detection completed. Output saved to:
{output_path}")
```

IR SENSOR COUNTER

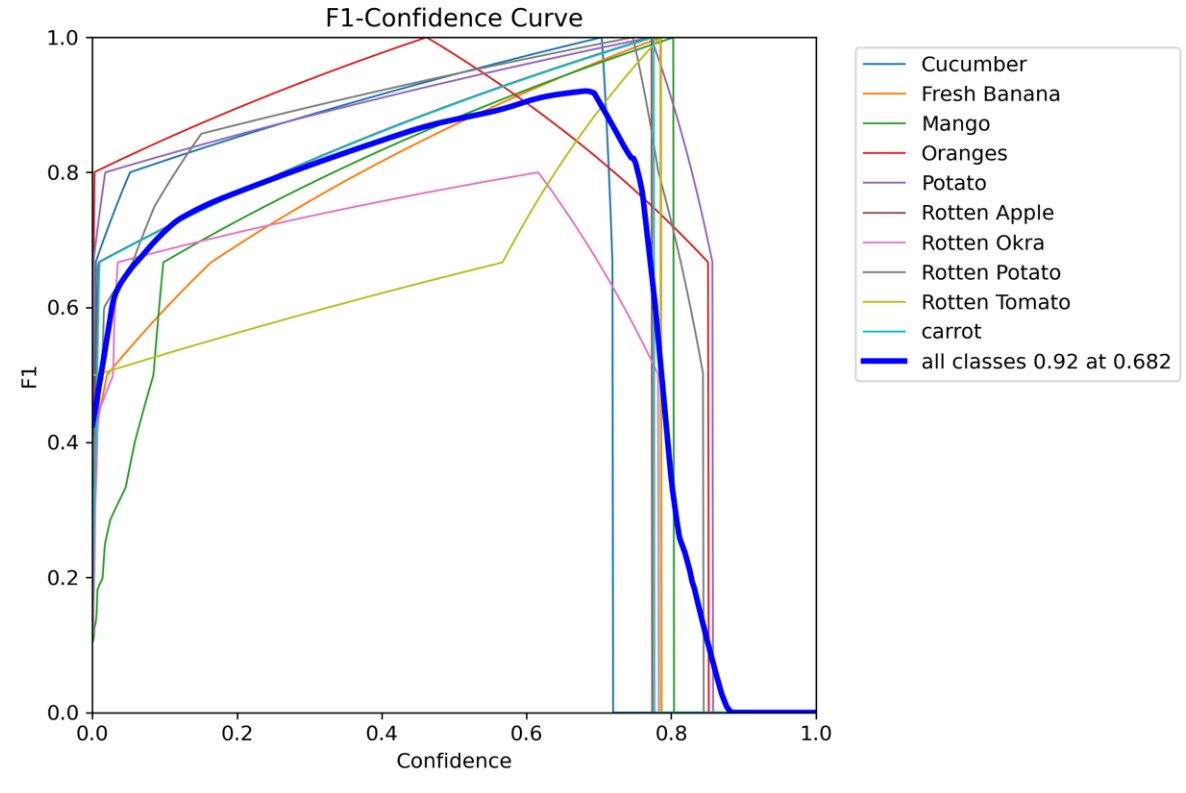
```
int pirPin = 2; // PIR sensor input pin
int pirState = LOW; // Initial state of PIR sensor
int peopleCount = 0; // Initial people count

void setup() {
  pinMode(pirPin, INPUT);
  Serial.begin(9600); // Initialize Serial Monitor with
baud rate 9600

  Serial.println("People Counter");
  Serial.print("Count: ");
  Serial.println(peopleCount);
}

void loop() {
  int motionDetected = digitalRead(pirPin);

  if (motionDetected == HIGH) {
    if (pirState == LOW) {
      peopleCount++;
      Serial.print("ITEM Count: ");
      Serial.println(peopleCount);
      pirState = HIGH; // Update PIR state to HIGH to
prevent multiple counts for one motion
      delay(100); // Delay to allow for motion clearance
    }
  } else {
    if (pirState == HIGH) {
      pirState = LOW; // Reset PIR state when no motion is
detected
    }
  }

  delay(50); // Small delay to stabilize sensor readings
}
```

MODEL PLOTS

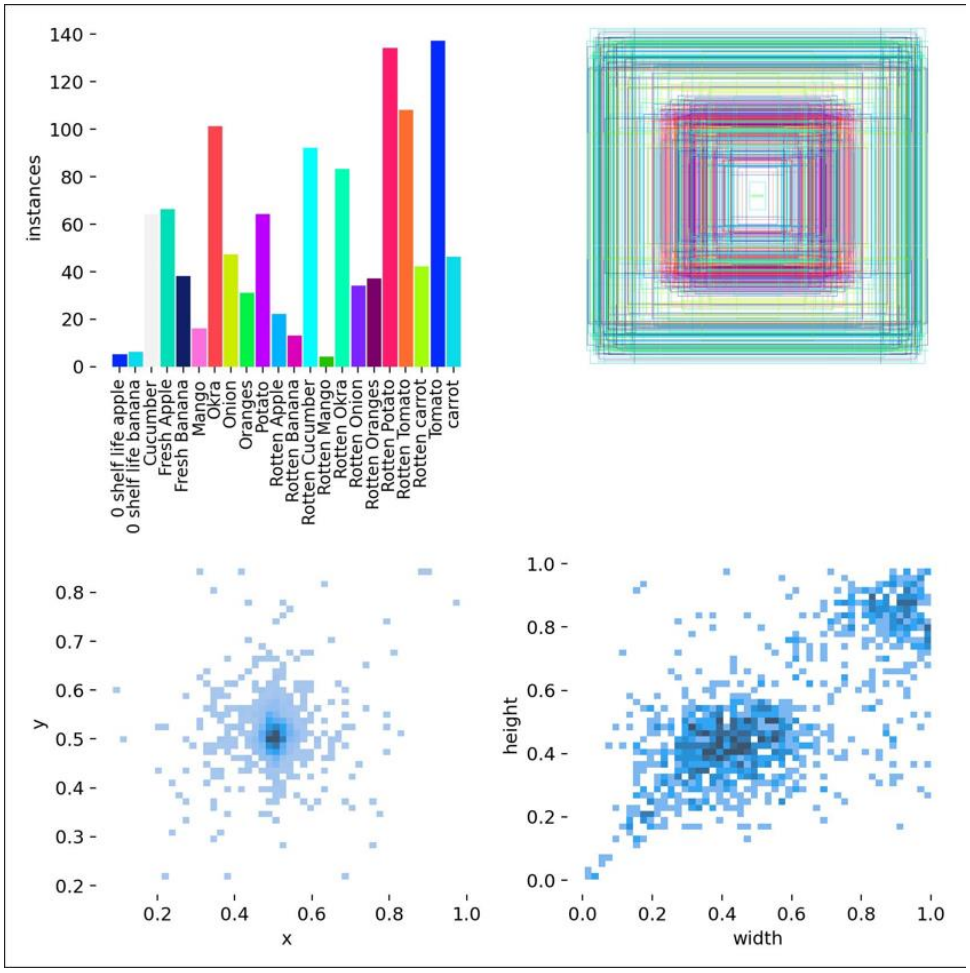
1.



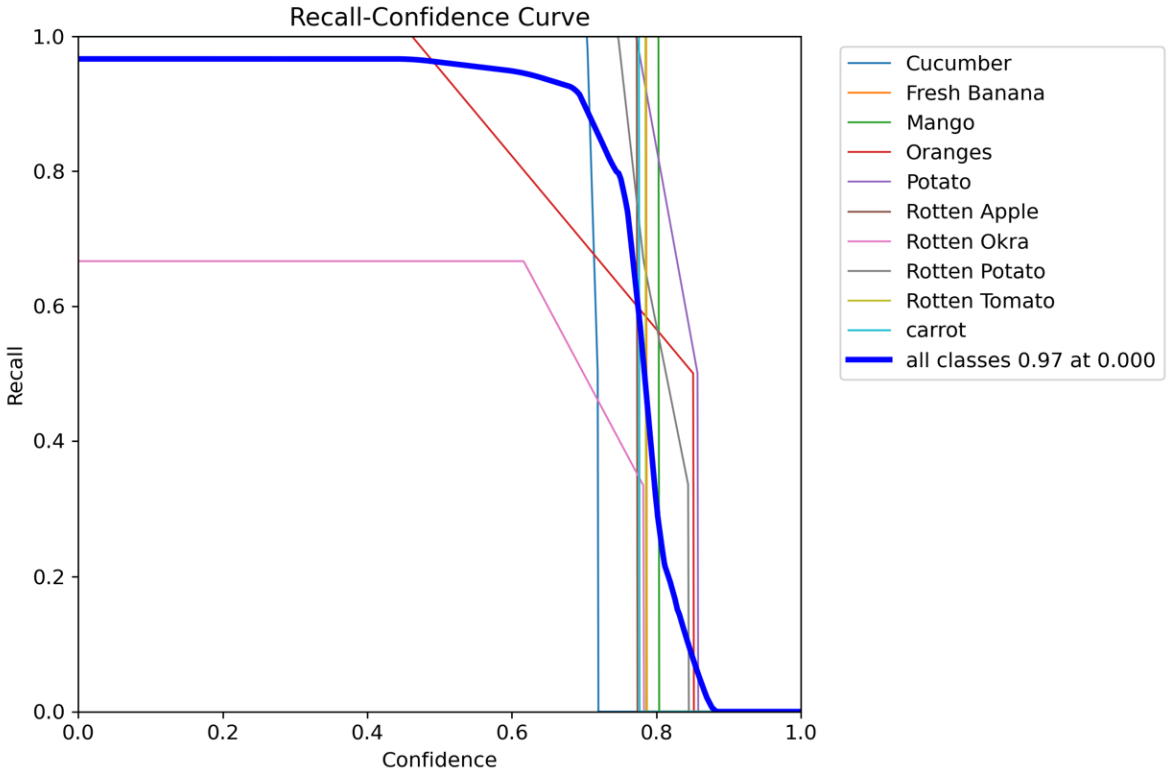
2.



3.


instances
0 shelf life apple
0 shelf life banana
Cucumber
Fresh Apple
Fresh Banana
Mango
Okra
Onion
Oranges
Potato
Rotten Apple
Rotten Banana
Rotten Cucumber
Rotten Mango
Rotten Okra
Rotten Onion
Rotten Oranges
Rotten Potato
Rotten Tomato
Rotten carrot
Tomato
carrot
y
x
height
width

4.


Recall-Confidence Curve
Recall
Confidence
Cucumber
Fresh Banana
Mango
Oranges
Potato
Rotten Apple
Rotten Okra
Rotten Potato
Rotten Tomato
carrot
all classes 0.97 at 0.000