Task: Object Detection with YOLO-based Fine-Tuning

Name: Sujan S
Roll No: 22PD35
Course: MSc Data Science

Video Explanation

https://drive.google.com/file/d/1GT3UNzcg7BkTRr5SejtTpRpy5XNP0Q9C/view?usp=sharing

1) Problem Statement

Build a real-time object detection system capable of identifying and classifying custom objects — including bottles, flowers, and tools — from live webcam input. The system should:

* Work in real-time
* Accurately classify across 9 defined object categories
* Be lightweight enough for edge deployment
* Offer an intuitive web interface for end-users

2) Project Architecture

1. Dataset Collection & Augmentation
   Images collected for 9 object classes; data augmented using Albumentations.

2. Annotation
   Used Roboflow to annotate and export in YOLOv5 format.

3. Model Training
   YOLOv5m was fine-tuned using custom data over 100 epochs.

```
val: data=/content/drive/MyDrive/dataset/data.yaml, weights=['runs/train/smartan_yolo_v5m_finalrun2/weights/bes
YOLOv5 🚀 v7.0-421-g79c4c31d Python-3.11.13 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)

Fusing layers...
Model summary: 212 layers, 20885262 parameters, 0 gradients, 48.0 GFLOPs
val: Scanning /content/drive/MyDrive/dataset/val/labels.cache... 278 images, 0 backgrounds, 0 corrupt: 100% 278
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 9/9 [00:09<00:00
                   all        278        278      0.901      0.941      0.956      0.726
         borosil_bottle        278         29      0.982          1      0.995      0.779
         plastic_bottle        278         32      0.959          1      0.992      0.814
      tupperware_bottle        278         29      0.784      0.897      0.866      0.684
           daisy_flower        278         35      0.977      0.886      0.982      0.697
        hibiscus_flower        278         29      0.898      0.914      0.944      0.536
            rose_flower        278         32      0.887      0.986      0.988       0.83
            screwdriver        278         32      0.764      0.909      0.901      0.694
                 hammer        278         33      0.879       0.88      0.936      0.715
                spanner        278         27       0.98          1      0.995      0.787
Speed: 0.3ms pre-process, 13.3ms inference, 3.0ms NMS per image at shape (32, 3, 640, 640)
Results saved to runs/val/exp4
```

5. Model Evaluation
   Precision, Recall, mAP50, and misclassifications tracked.

6. Deployment
   Live webcam inference using OpenCV + Flask + HTML frontend.

3) Classes

1) Water Bottles

Borosil Bottle

Plastic Bottle

Tupperware Bottle

2) Flowers

Daisy Flower

Hibiscus Flower

Rose Flower

3) Tools

Screwdriver

Hammer

Spanner

```
📊 Class-wise Summary:
borosil_bottle          | Labels: 109 | Images: 108
daisy_flower            | Labels: 113 | Images: 113
hammer                  | Labels: 114 | Images: 113
hibiscus_flower         | Labels: 115 | Images: 115
plastic_bottle          | Labels: 106 | Images: 106
rose_flower             | Labels: 109 | Images: 109
screwdriver             | Labels: 108 | Images: 108
spanner                 | Labels: 109 | Images: 109
tupperware_bottle       | Labels: 97  | Images: 97

🖼️ Total images: 978
```

4) Tools & Technologies Used

* YOLOv5: For training and detection
* Python & OpenCV: Real-time video processing
* Flask: Web interface for deployment
* Albumentations: Image augmentation
* Roboflow: Dataset annotation and export
* Google Colab / Local: For training environment

5) Methodology

1. Collect & annotate a balanced dataset
2. Apply data augmentation to boost performance
3. Train YOLOv5m model with custom data
4. Evaluate using mAP, Precision, Recall
5. Deploy using Flask with webcam support

6) Component Breakdown

* `train.py`: Used to train YOLOv5m model
* `val.py`: Used for evaluating model performance
* `detect.py`: Runs inference on test images or webcam
* `app.py`: Flask backend for live detection
* `templates/index.html`: Frontend interface
* `/runs/train/`: Stores training outputs
* `/runs/detect/`: Stores detection results

7) Input and Output Specifications

Input

* Image or webcam stream
* Format: JPG, PNG or live webcam frames

Output

* Detected object with bounding boxes
* Real-time class label and confidence score overlay
* Detection saved or shown via web interface

8) Result

Achieved significant improvement in mAP and real-time detection accuracy after augmentation and retraining with:

* Total images used: 978 (balanced across 9 classes)
* Final model: YOLOv5s, trained for 100 epochs

Successfully deployed via Flask with live webcam stream.

9) Output

Results got after testing on data

!python detect.py \
--weights /content/yolov5/runs/train/smartan_yolo_v5m_finalrun2/weights/best.pt \
--img 640 \
--conf 0.25 \
--source
/content/drive/MyDrive/dataset/train/images/bottle08_aug4_jpg.rf.ba097f3be8145bea5aa969f62
81c291a.jpg \
--name smartan_test


hammer

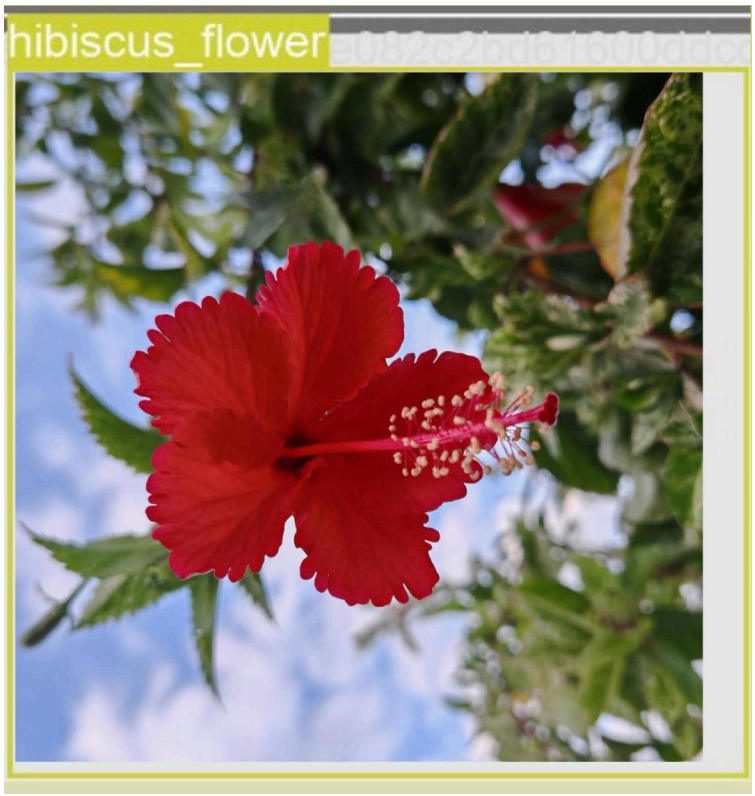

lastic_bottle 0.7
c12a7edb07b94f07


e_flower 0.88


screwdriver 0.76

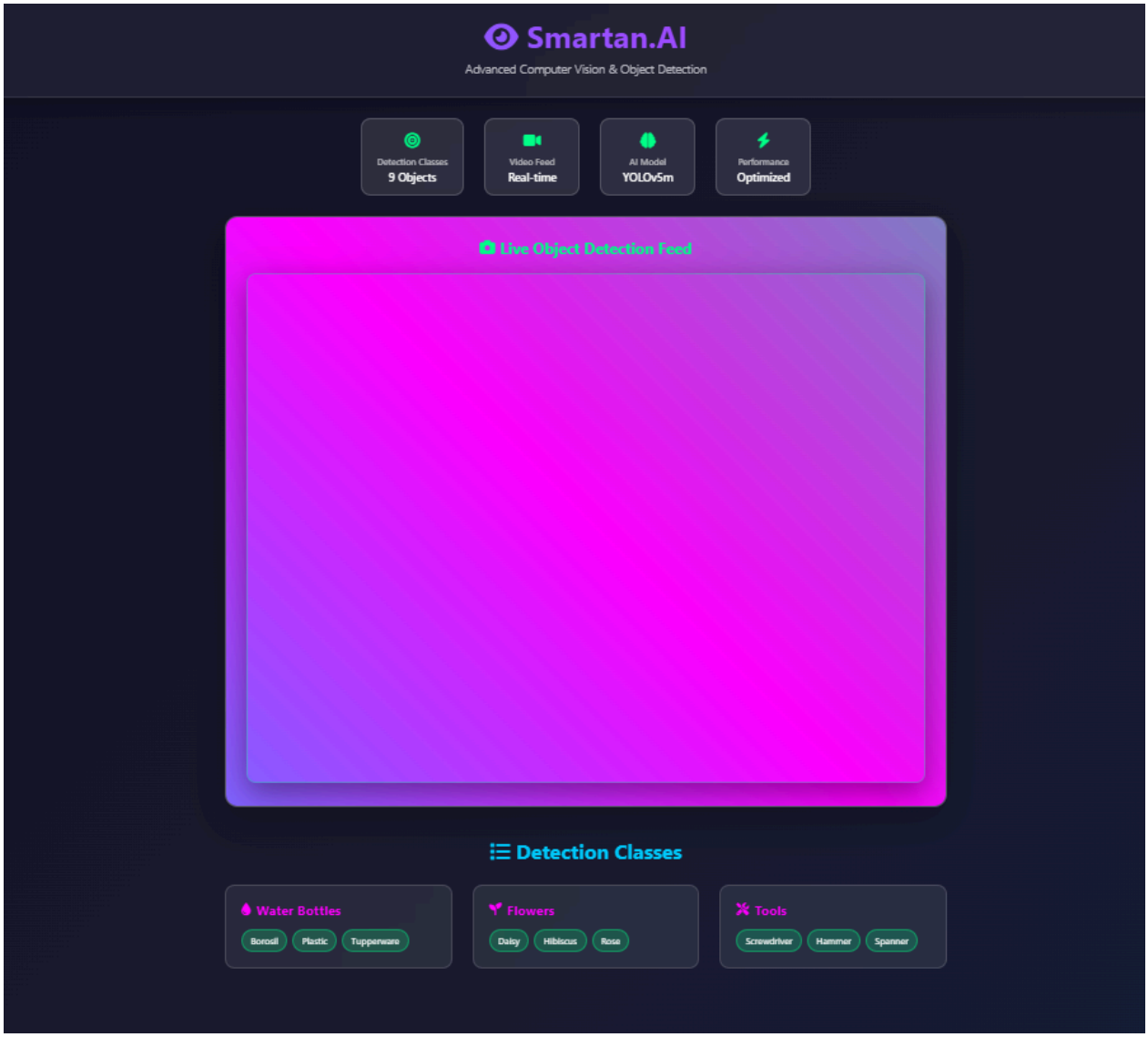

borosil_bottle 0.82
BOROSIL®


tupperware_bottle 0.84

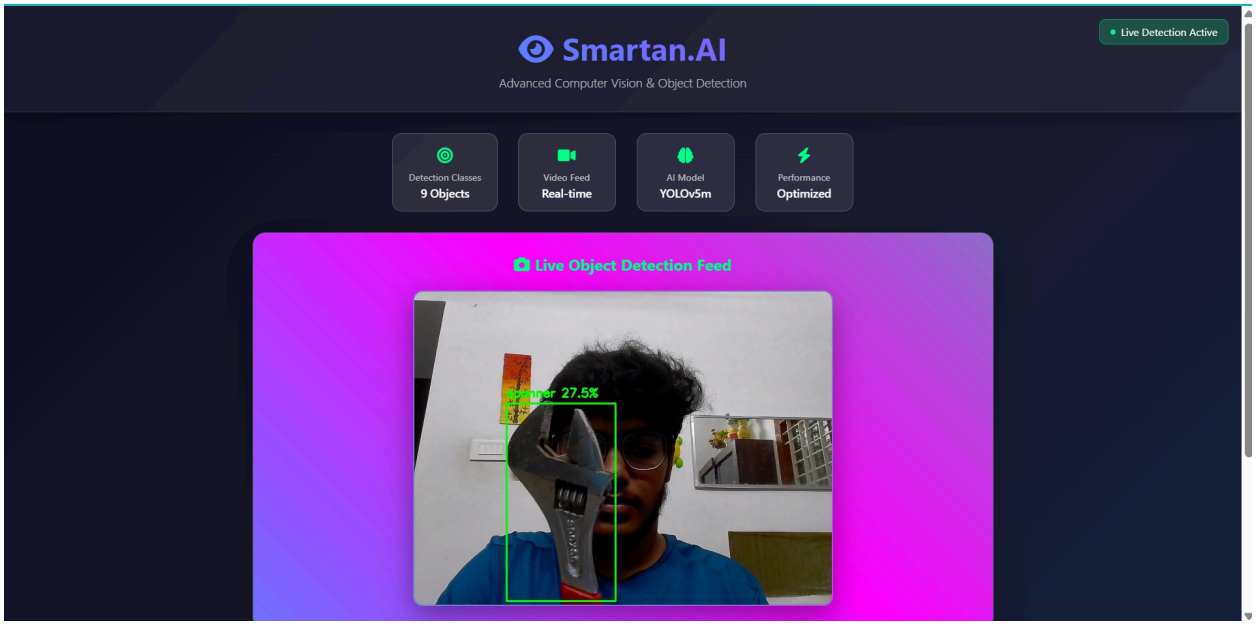

hibiscus_flower


daisy_flower 0.89

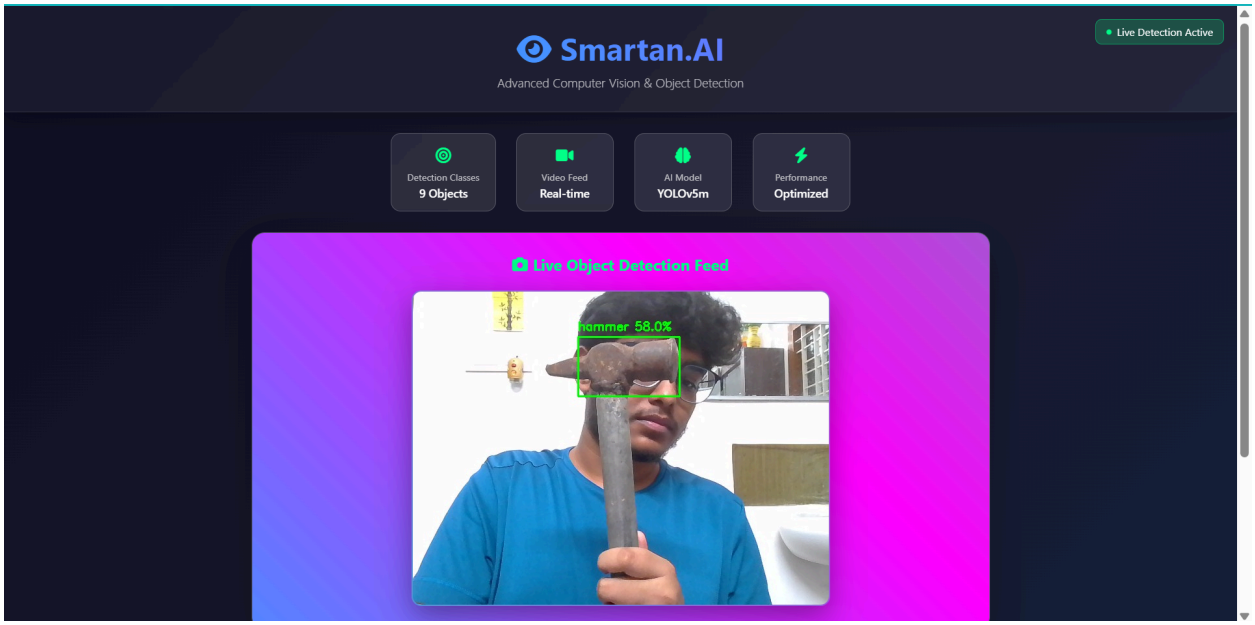
Frontend


Smartan.AI
Advanced Computer Vision & Object Detection
Detection Classes
9 Objects
Video Feed
Real-time
AI Model
YOLOv5m
Performance
Optimized
Live Object Detection Feed
Detection Classes
Water Bottles
Borosil
Plastic
Tupperware
Flowers
Daisy
Hibiscus
Rose
Tools
Screwdriver
Hammer
Spanner

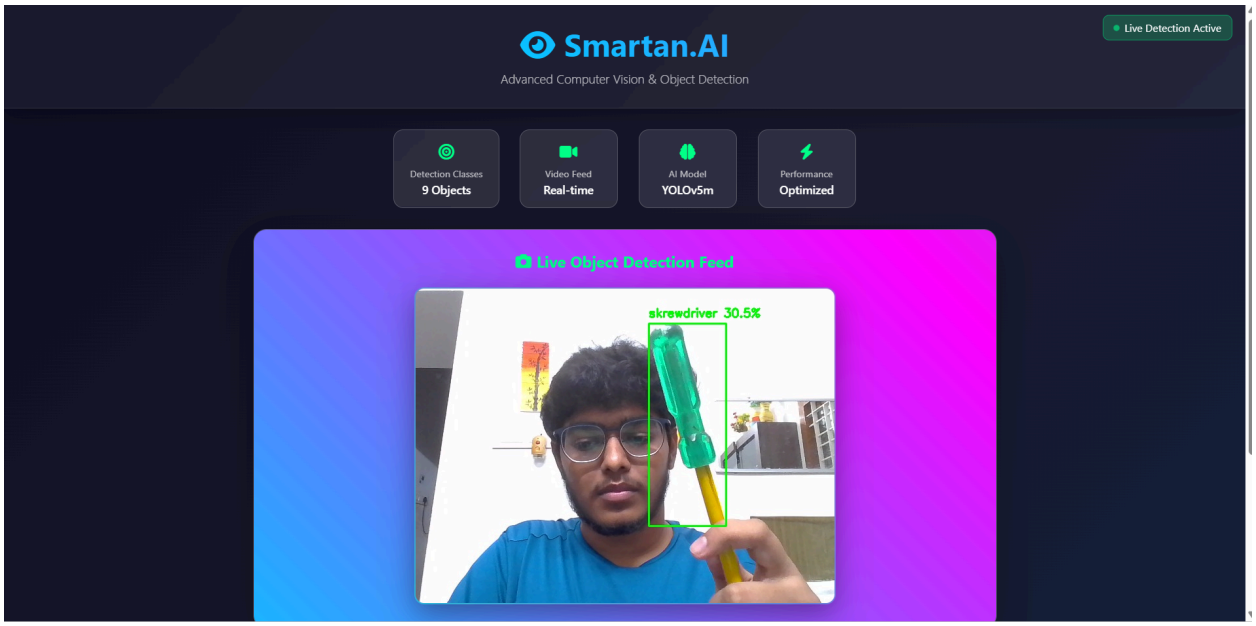
Spanner



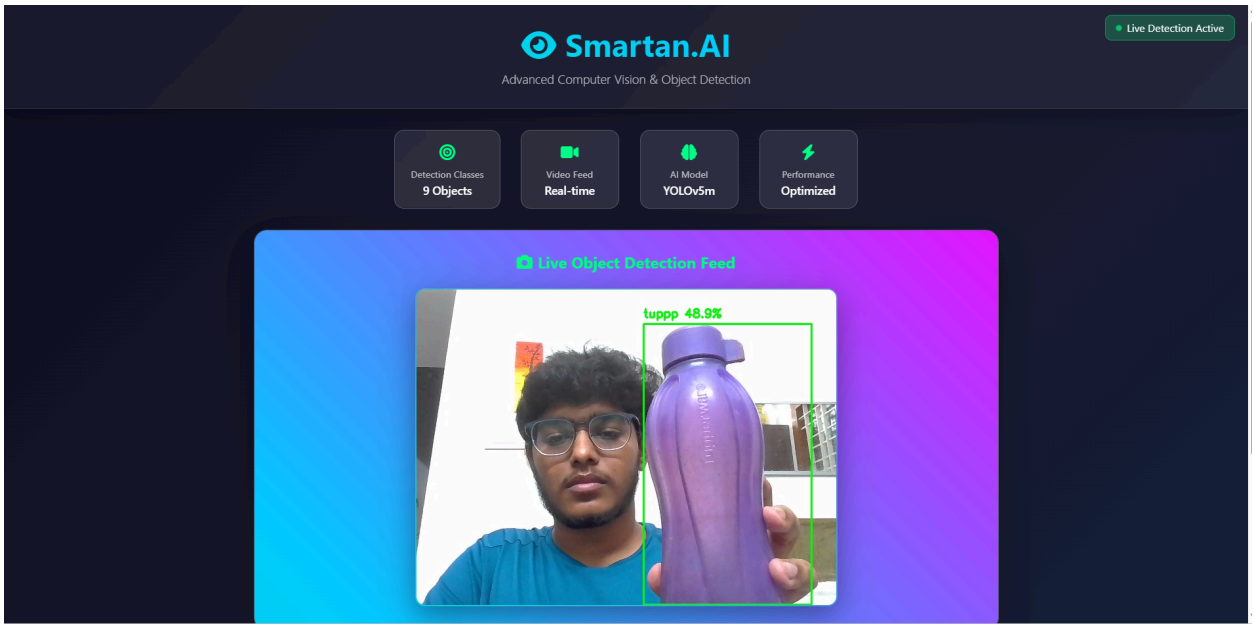
Hammer



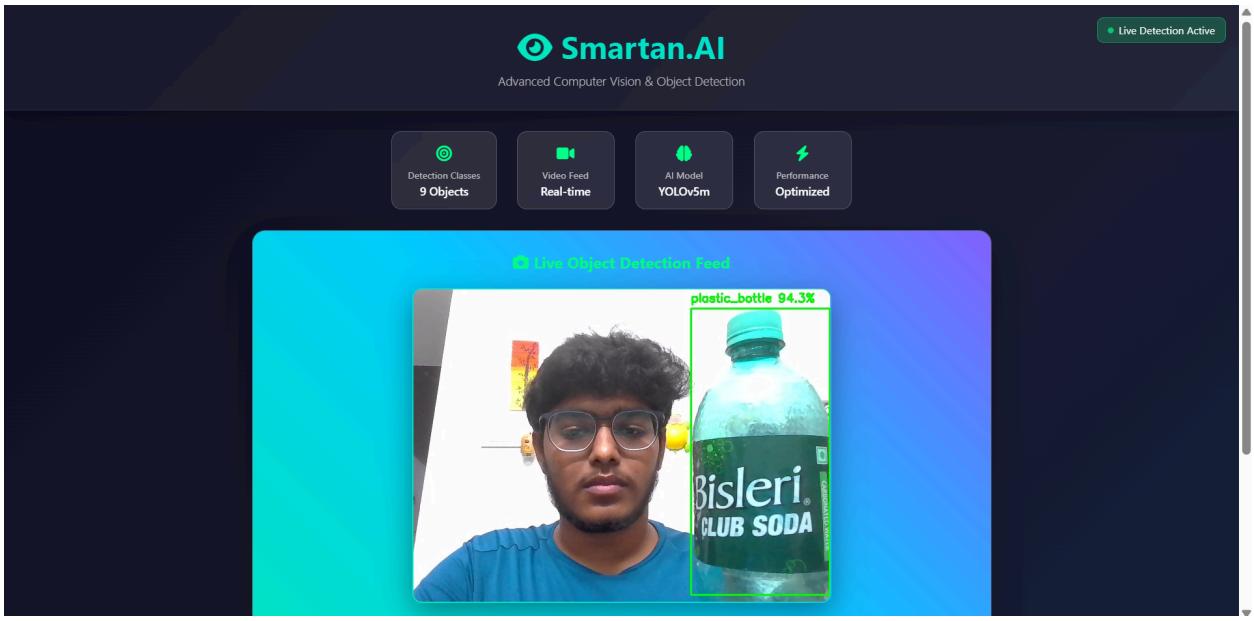
Skrewdriver



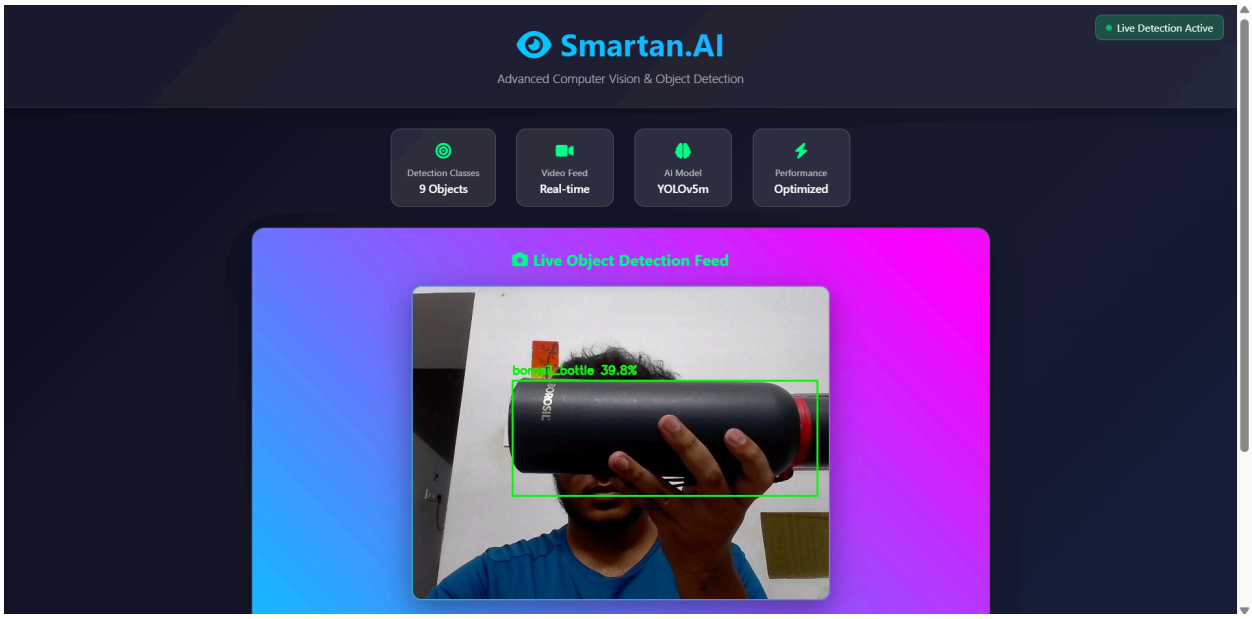
Tupperware Bottle



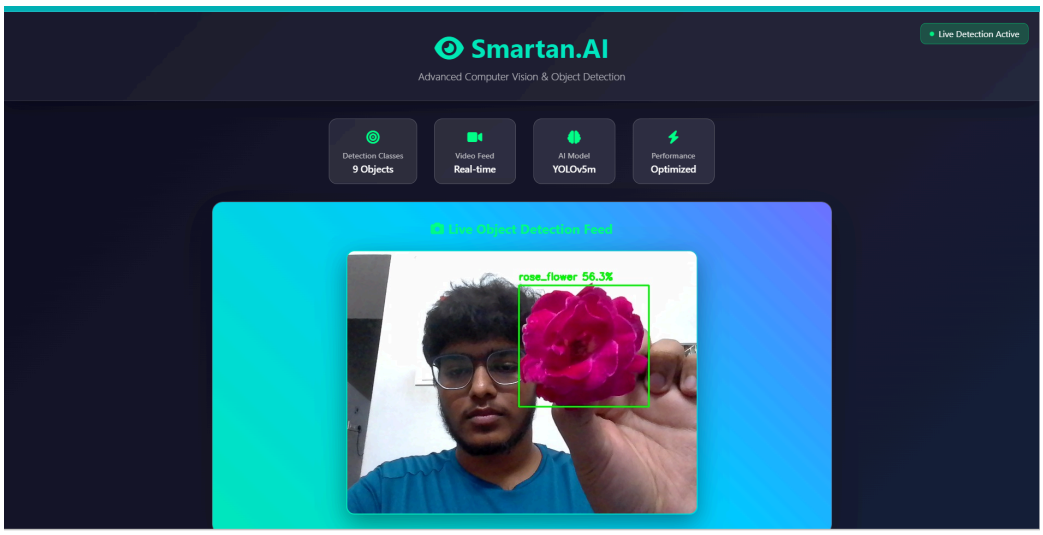
Plastic Bottle



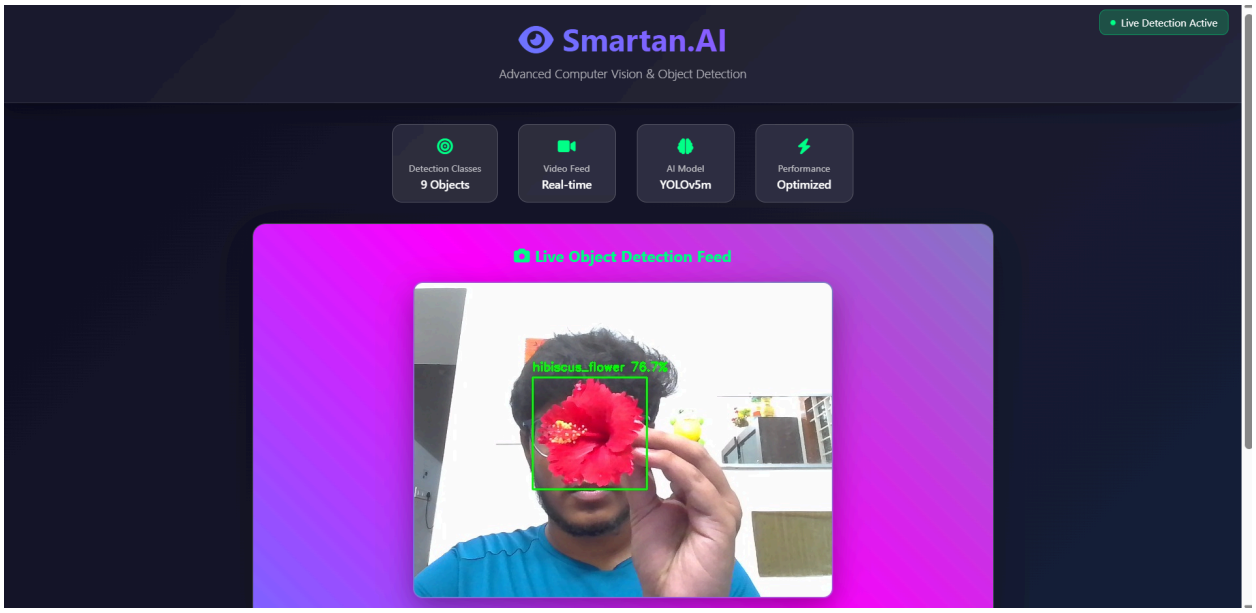
Borosil Bottle



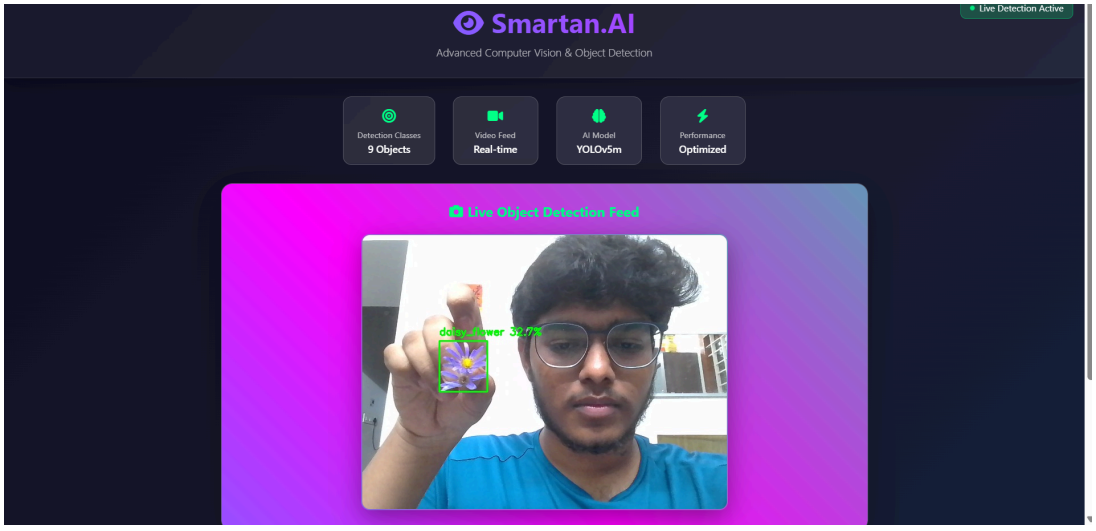
Rose



Hibiscus



Daisy



10) Limitations & Challenges

1) Small Dataset for Some Classes

Initially had fewer images (under 50) for tupperware_bottle, borosil_bottle, etc., which impacted early model performance.

2) Class Confusion

Visually similar items like hammer, spanner, and screwdriver were often misclassified due to overlapping features.

3) Long Training Time on CPU

Local training took over 9 hours for 100 epochs, delaying experiments—partially resolved using Colab GPU.

11) Future Improvements

* Fine-tune using YOLOv8 or other transformers
* Deploy on mobile/Jetson Nano for edge AI
* Add object count and audio feedback