

Retail Market Product Detection Portal

Project Documentation

AIM:

This project aims to develop a product detection system for grocery store shelves using **computer vision** techniques. The system uses a **YOLOv11 (You Only Look Once)** model trained on a small, manually annotated dataset of supermarket images. The goal is to detect various products in the images and display predictions with bounding boxes.

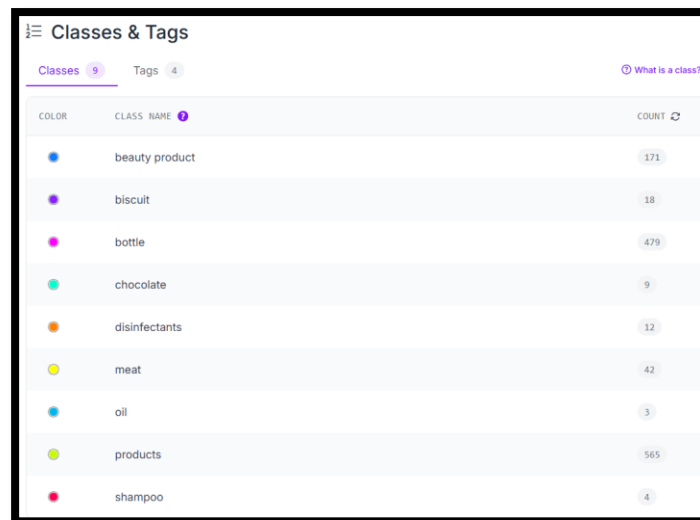
The system provides a user-friendly interface where users can upload images and visualize the detected products. Additionally, it offers predictions in **JSON format** to showcase the model's detection results.

Data Collection and Preprocessing:

The first step in the project involved the collection and preprocessing of images:

1. Data Collection:

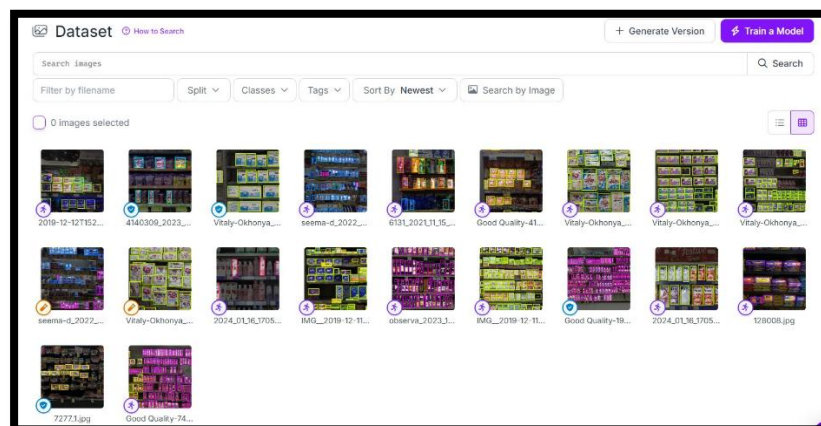
- A small dataset of supermarket shelf images was used. The dataset contained images with various grocery products. I've used roboflow software to manually annotate and label different products in different categories, including:
 - Beauty products
 - Biscuits
 - Bottles
 - Chocolates
 - Disinfectants
 - Meats
 - Oils
 - Products
 - Shampoos



COLOR	CLASS NAME	COUNT
	beauty product	171
	biscuit	18
	bottle	479
	chocolate	9
	disinfectants	12
	meat	42
	oil	3
	products	565
	shampoo	4

2. Data Annotation:

- The collected images were manually annotated using the Roboflow tool. Each object in the images was labeled according to one of the 9 predefined classes listed above.
- Roboflow allowed the annotations to be exported in various formats compatible with different model types.



3. Data Augmentation:

- Data augmentation techniques were applied to the dataset in Roboflow to improve model robustness. These techniques included:
 - Image flipping
 - Rotation
 - Scaling
 - Cropping
- Data augmentation is crucial in deep learning tasks when there is a limited amount of labelled data. It helps improve model generalization by simulating various real-world scenarios that the model might encounter.

Augmentations Outputs per training example: 3
Rotation: Between -15° and +15°
Grayscale: Apply to 15% of images
Saturation: Between -25% and +25%
Brightness: Between -15% and +15%
Exposure: Between -10% and +10%
Noise: Up to 0.1% of pixels

Model Training:

Once the dataset was ready, the model training phase began:

1. Model Selection:

- The YOLOv11 model was selected for the task of product detection. YOLO is known for its efficiency and speed in real-time object detection tasks, making it suitable for deployment in production environments.

2. Training Configuration:

- The model was trained using the Roboflow platform. Roboflow provides an easy-to-use interface for setting up models, training them on custom datasets, and evaluating performance.
- The training process involved adjusting hyperparameters such as:
 - Learning rate
 - Batch size
 - Number of epochs

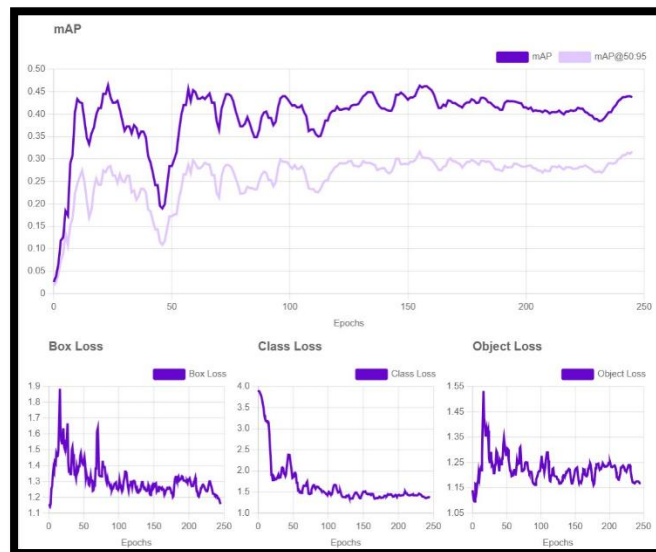
3. Model Evaluation:

- After training, the model was evaluated using standard object detection metrics, such as:
 - Precision: 87.7%
 - Recall: 42.6%
 - Mean Average Precision (mAP): 46.1%
- These metrics indicate the model's ability to correctly identify products in the images. Precision measures the percentage of correct positive predictions, recall measures the percentage of actual positive instances detected, and mAP is an overall measure of the model's accuracy.



4. Training Visualization:

- Graphs of training loss and accuracy were generated to track the model's performance over time. These graphs helped visualize how the model improved during training.



mAP (Mean Average Precision):

- The graph at the top shows the mAP for each epoch. mAP is a key evaluation metric for object detection models, representing the average of precision at different recall levels.
- The purple line represents the general mAP performance, while the light purple line (mAP@50:95) corresponds to a specific calculation of mAP considering different Intersection over Union (IoU) thresholds (from 0.5 to 0.95).
- As we see, mAP improves as the training progresses, though it flattens out a bit after around 150 epochs, indicating that the model is stabilizing in its detection accuracy.

Loss Curves:

- **Box Loss:** This represents the error in predicting the location of bounding boxes around detected objects. The graph shows the training loss over time, with the loss typically reducing as the model learns to make more accurate bounding box predictions. The sharp drop early on indicates that the model quickly learns how to improve box predictions in the initial training stages.
- **Class Loss:** This indicates the error in predicting the correct class for detected objects. As with box loss, the class loss decreases as training progresses, demonstrating that the model is improving in classifying objects correctly.
- **Object Loss:** This is the total loss that combines errors related to both the detection of the object and the classification of it. This curve shows the training progress for the model's ability to both detect and classify objects.

Training Dynamics:

- The sharp drops in losses early on (around epoch 50) suggest that the model quickly improves during the initial epochs.
- After around 100 epochs, the losses stabilize and progress in a more gradual manner, which is expected as the model converges toward an optimal state.

- This pattern is typical in deep learning, where the model learns quickly at first and then fine-tunes its weights in a more incremental way.

Model Deployment:

Once the model was trained, the next step was to deploy it to make predictions on new images:

1. *Deployment Setup:*

- The trained model was deployed using a Flask web server. Flask is a lightweight Python framework that allows easy creation of APIs and web applications.
- The model ID from Roboflow was used to access the trained model and make predictions.

2. *Flask API:*

- A RESTful API was created with two main endpoints:
 - `/predict_json`: Accepts an image, performs inference using the model, and returns the detection results in JSON format.
 - `/visualize`: Accepts an image, performs inference, and returns the image with bounding boxes drawn around the detected objects.

3. *Integration:*

- The Flask API was integrated with the trained model, allowing users to send images and get predictions either in JSON format or as visualizations with bounding boxes.

Frontend Development:

The final step involved developing a user-friendly interface for interaction with the model:

1. *Frontend Features:*

- The frontend was developed using HTML, CSS, and JavaScript.
- The page provides two options for the user:
 1. Visualize Image with Bounding Boxes: Upload an image and view the detected objects with bounding boxes drawn around them.
 2. Get Predictions (JSON): Upload an image and receive the detection results in a JSON format.

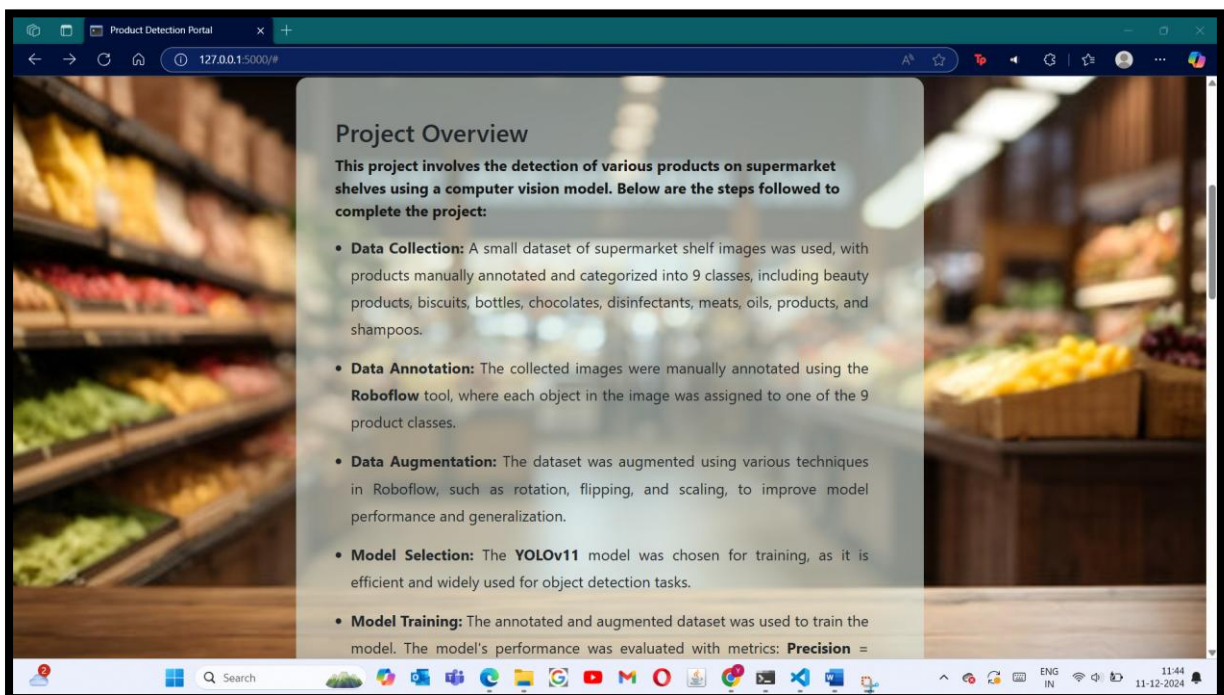
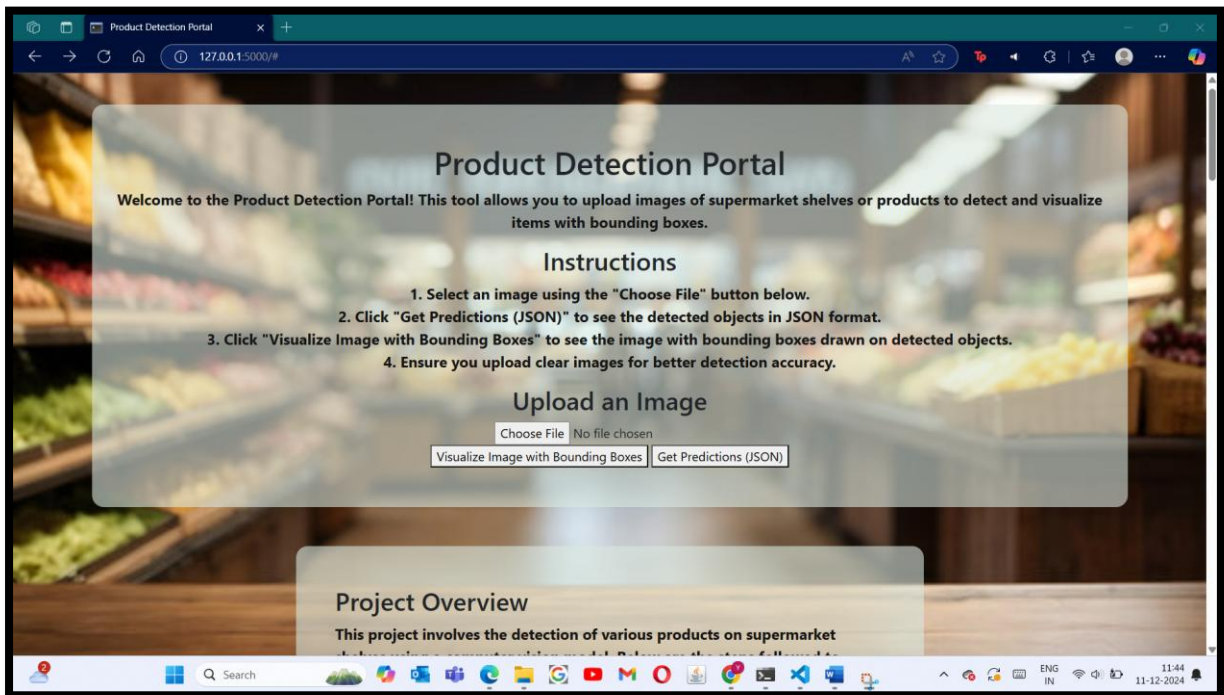
2. *Carousel Display:*

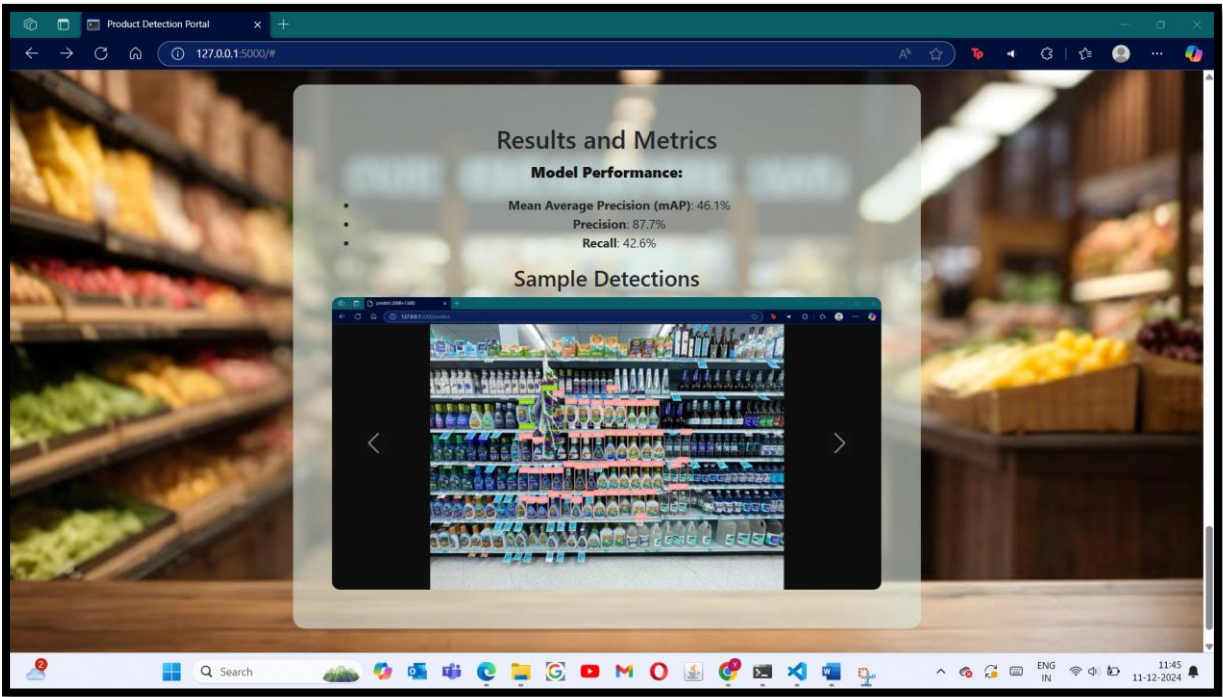
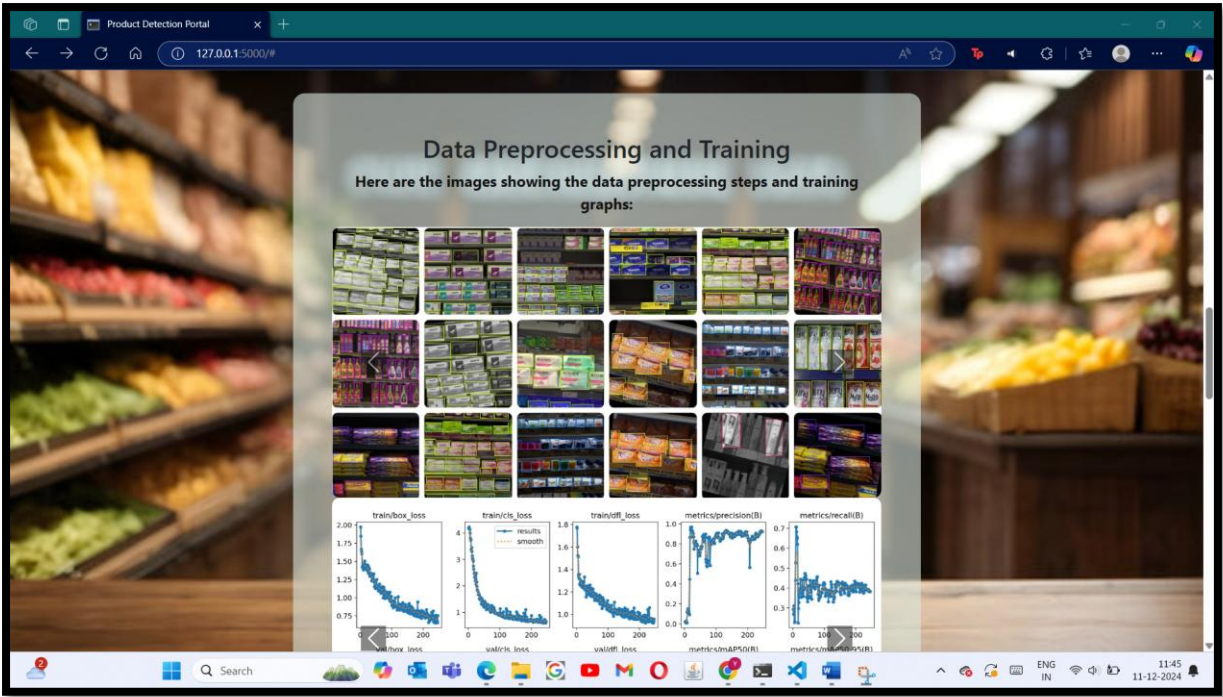
- A Bootstrap carousel was implemented to display images showing the data preprocessing, training graphs, and sample detection results. The carousel allows users to easily navigate through multiple images.

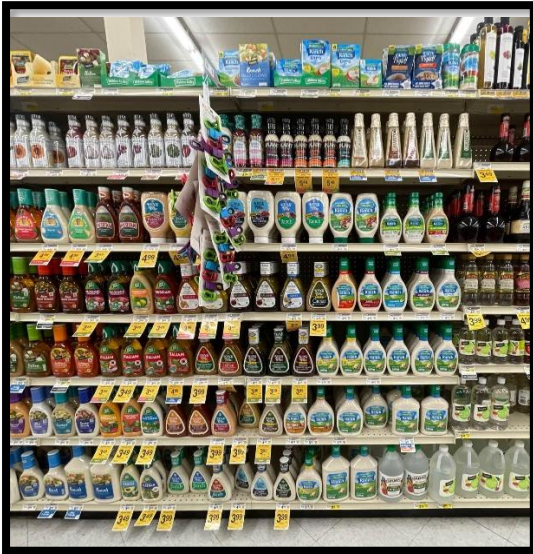
3. *Design:*

- The interface was made responsive, ensuring that it works well on different screen sizes (desktop, tablet, mobile).

This is how the interface looks like:





Results:**Input Image****Detected image****JSON Output for this particular image:**

```
1 {
2   "detections": [
3     {
4       "bbox": [
5         665,
6         975,
7         734,
8         1100
9       ],
10      "class": "bottle",
11      "confidence": 0.9819299578666687
12    },
13    {
14      "bbox": [
15        794,
16        1005,
17        858,
18        1100
19      ],
20      "class": "bottle",
21      "confidence": 0.9804309010505676
22    },
23    {
24      "bbox": [
25        688,
26        838,
27        750,
28        941
29      ],
30      "class": "bottle",
31      "confidence": 0.9789133667945862
32    },
33    {
34      "bbox": [
35        918,
36        637,
37        986,
38        771
39      ],
40      "class": "bottle",
41      "confidence": 0.9720029234886169
42    },
43    {
44      "bbox": [
45        625,
46        840,
47        687,
48        938
49      ],
50      "class": "bottle",
51      "confidence": 0.968684732913971
52    },
53    {
54      "bbox": [
55        882,
56        813,
```

Hence the detected products are grouped as “bottle.”

Conclusion and Future Work:

The Retail Market Product Detection Portal project successfully developed an object detection system for supermarket products. The system leverages a YOLO-based model, trained on a manually annotated dataset, and deployed via Flask. The project provides a functional web interface where users can upload images and receive real-time product detection predictions.

Future Work:

- **Dataset Expansion:** The current dataset is relatively small, and expanding it with more diverse images will improve model performance.
- **Real-time Detection:** The system can be enhanced to support real-time detection for video streams or live camera feeds.
- **Performance Tuning:** Further tuning of the model's hyperparameters could improve the precision and recall metrics.