Project Title: SkyVision - Aerial Object Detection

Project Report

Submitted By: Anjori Sarabhai, Apoorva Kashyap
Internship Project

Submitted To: Mr. Prashant Tiwari
SAG Lab, DRDO

GitHub Repository: https://github.com/anjorisarabhai/aerial-object-detection

---

1. Introduction

This project presents a deep learning-based aerial object detection system called **SkyVision**, designed to detect and classify objects from high-resolution aerial imagery. Utilizing the **DOTA dataset** and a **YOLOv8 model.**

Object detection in aerial images is vital for urban planning, surveillance, disaster management, and military applications. Unlike conventional images, aerial images pose challenges such as:

- High-resolution inputs
- Small and densely packed objects
- Varying orientations

---

2. Dataset Description

- **Source**: DOTA Official Dataset Website
- **Format**: JPG/PNG image format, High-resolution images (up to 4000×4000) [RGB and Grayscale]
- **Classes**: 15 object categories including ship, plane, vehicle, bridge, etc.
- **Annotations**: Polygon-based, later converted to YOLO format
- **Split**: Used training set for model fine-tuning and testing on sample images
- **Structure Screenshot Placeholder**:

```
/kaggle/input/dota-data/DOTA/val/images/P2595.png
/kaggle/input/dota-data/DOTA/val/images/P0524.png
/kaggle/input/dota-data/DOTA/val/images/P1651.png
/kaggle/input/dota-data/DOTA/val/images/P2242.png
/kaggle/input/dota-data/DOTA/val/images/P2789.png
/kaggle/input/dota-data/DOTA/val/images/P1128.png
/kaggle/input/dota-data/DOTA/val/images/P1179.png
/kaggle/input/dota-data/DOTA/val/images/P1742.png
/kaggle/input/dota-data/DOTA/val/images/P2135.png
/kaggle/input/dota-data/DOTA/val/images/P2420.png
/kaggle/input/dota-data/DOTA/val/images/P1666.png
/kaggle/input/dota-data/DOTA/val/images/P2617.png
/kaggle/input/dota-data/DOTA/val/images/P2011.png
/kaggle/input/dota-data/DOTA/val/images/P2044.png
/kaggle/input/dota-data/DOTA/val/images/P2231.png
/kaggle/input/dota-data/DOTA/val/images/P0801.png
/kaggle/input/dota-data/DOTA/val/images/P1880.png
/kaggle/input/dota-data/DOTA/val/images/P0841.png
/kaggle/input/dota-data/DOTA/val/images/P2766.png
/kaggle/input/dota-data/DOTA/val/images/P2541.png
/kaggle/input/dota-data/DOTA/val/images/P2721.png
/kaggle/input/dota-data/DOTA/val/images/P1825.png
/kaggle/input/dota-data/DOTA/val/images/P0989.png
/kaggle/input/dota-data/DOTA/val/images/P2608.png
/kaggle/input/dota-data/DOTA/val/images/P2255.png
...
/kaggle/input/dota-data/DOTA/train/images/P0760.png
/kaggle/input/dota-data/DOTA/train/images/P1900.png
/kaggle/input/dota-data/DOTA/train/images/P0276.png
/kaggle/input/dota-data/DOTA/train/images/P1556.png
```

```
/kaggle/input/annotations/labelTxt/P2387.txt
/kaggle/input/annotations/labelTxt/P2494.txt
/kaggle/input/annotations/labelTxt/P0358.txt
/kaggle/input/annotations/labelTxt/P2067.txt
/kaggle/input/annotations/labelTxt/P1164.txt
/kaggle/input/annotations/labelTxt/P0572.txt
/kaggle/input/annotations/labelTxt/P1142.txt
/kaggle/input/annotations/labelTxt/P1551.txt
/kaggle/input/annotations/labelTxt/P2257.txt
/kaggle/input/annotations/labelTxt/P0966.txt
/kaggle/input/annotations/labelTxt/P2503.txt
/kaggle/input/annotations/labelTxt/P0884.txt
/kaggle/input/annotations/labelTxt/P2729.txt
/kaggle/input/annotations/labelTxt/P1773.txt
/kaggle/input/annotations/labelTxt/P1344.txt
/kaggle/input/annotations/labelTxt/P1353.txt
/kaggle/input/annotations/labelTxt/P1955.txt
/kaggle/input/annotations/labelTxt/P0479.txt
/kaggle/input/annotations/labelTxt/P2321.txt
/kaggle/input/annotations/labelTxt/P1726.txt
/kaggle/input/annotations/labelTxt/P1306.txt
/kaggle/input/annotations/labelTxt/P1926.txt
/kaggle/input/annotations/labelTxt/P0140.txt
/kaggle/input/annotations/labelTxt/P0296.txt
/kaggle/input/annotations/labelTxt/P1908.txt
...
```

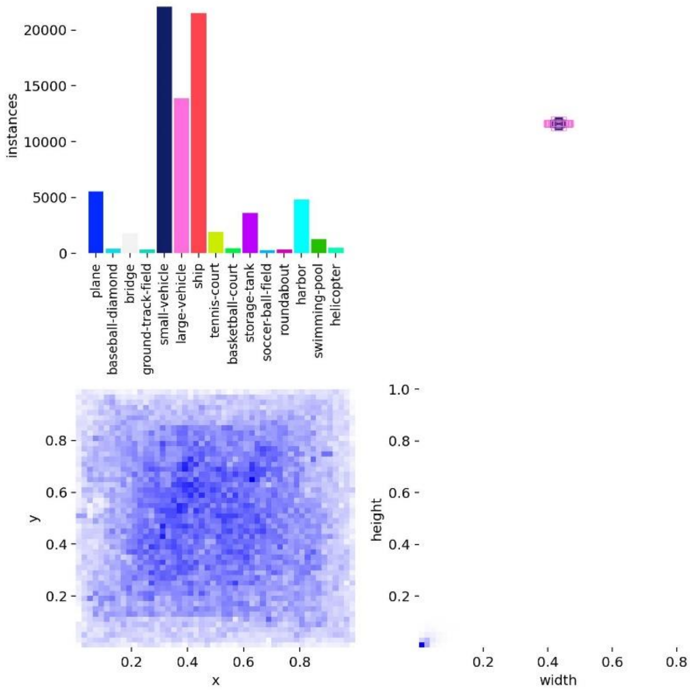
---

3. Exploratory Data Analysis (EDA)

Image-Level EDA

- Verified the DOTA dataset structure and the presence of annotation files in .txt (YOLO) or XML format.
- Loaded a random subset of aerial images to visually confirm object types like ships, vehicles, and buildings.
- Examined and visualized image dimensions — most images ranged between **800×800** to **4000×4000** pixels.
- Checked for corrupted or missing image-label pairs using automated scripts.

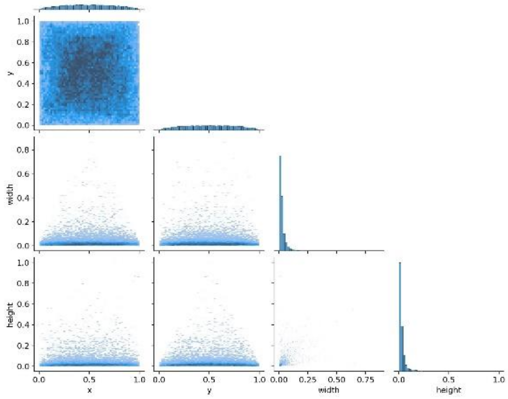
```
Class distribution:
ship: 28068
small-vehicle: 26126
large-vehicle: 16969
plane: 8055
harbor: 5983
storage-tank: 5029
tennis-court: 2367
bridge: 2047
swimming-pool: 1736
helicopter: 630
basketball-court: 515
baseball-diamond: 415
roundabout: 399
soccer-ball-field: 326
ground-track-field: 325
```



Annotation-Level EDA

- Verified that annotation files followed the YOLO format (.txt)
- Parsed and counted annotations across the dataset to analyze object density, most images contained 5 to 40 bounding boxes.
- Computed class distribution to identify the most and least frequent object categories; common classes included ship, vehicle, and building.
- Validated label consistency by checking that every image had a corresponding .txt label file and no malformed entries were present.
- Visualized bounding boxes for a small set of images using OpenCV to ensure correct alignment and accuracy of annotations.

```
 1 (baseball-diamond    ): 415 annotations
 2 (bridge              ): 2047 annotations
10 (soccer-ball-field   ): 326 annotations
 3 (ground-track-field  ): 325 annotations
 8 (basketball-court    ): 515 annotations
 7 (tennis-court        ): 2367 annotations
 4 (small-vehicle       ): 26126 annotations
 9 (storage-tank        ): 5029 annotations
 6 (ship                ): 28068 annotations
 5 (large-vehicle       ): 16969 annotations
12 (harbor              ): 5983 annotations
13 (swimming-pool       ): 1736 annotations
11 (roundabout          ): 399 annotations
 0 (plane               ): 8055 annotations
14 (helicopter          ): 630 annotations
```



4. Preparing Data for YOLOv8 Training

Data Structuring & Splitting

- Created the required directory structure for YOLOv8 training, organizing images and labels into separate train and valfolders under a yolo_dataset root directory.
- Performed a random 80-20 split of the dataset — selecting 80% of .png images for training and 20% for validation.
- Matched each selected image with its corresponding .txt label from the original annotations folder, ensuring one-to-one alignment.
- Used Python scripts to copy the images and labels into their appropriate subdirectories (images/train, images/val, labels/train, labels/val).
- Verified the completeness of the split by confirming that every image in each split had a corresponding label file.

Configuration & Validation

Created the data.yaml file required by YOLOv8, specifying:

- Relative paths to the images/train and images/val directories
- Total number of classes (nc: 15)
- A list of all class names used in the dataset

```
Train images: 1128
Val images: 283
```

```
TRAIN SET
Total Images : 1128
Total Labels : 1128
Missing Labels: 0
Missing Images: 0

VAL SET
Total Images : 283
Total Labels : 283
Missing Labels: 0
Missing Images: 0
```

---

5. Model Training & Evaluation

Imported the YOLO class from the ultralytics library and initialized the base model using YOLO('yolov8n.pt'), which provides a lightweight architecture suitable for fast training and inference. Trained the model using the prepared dataset by calling the .train() method with the following parameters:

- data='path/to/data.yaml'
- epochs=50 ● imgsz=1024
- project='runs/train', name='yolov8_base'
- During training, performance metrics including **loss**, **mean Average Precision (mAP)**, **precision**, and **recall** were automatically logged and plotted by the Ultralytics framework.
- Visualizations such as loss curves and mAP graphs were reviewed to track training progress and detect signs of overfitting or underfitting.
- The best-performing model checkpoint was saved as best.pt, which was later used for inference and evaluation.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      45/50      8.25G      1.351      1.049     0.9659        146        896: 100%|██████████| 282/282 [01:15<00:00,  3.72it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 36/36 [00:14<00:00,  2.53it/s]
                   all        283      20793      0.727      0.425      0.456      0.276

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      46/50      8.48G      1.338      1.011     0.9573        110        896: 100%|██████████| 282/282 [01:17<00:00,  3.62it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 36/36 [00:13<00:00,  2.75it/s]
                   all        283      20793      0.739      0.417      0.453      0.272

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      47/50      8.16G      1.349      1.061     0.9627        117        896: 100%|██████████| 282/282 [01:16<00:00,  3.69it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 36/36 [00:14<00:00,  2.50it/s]
                   all        283      20793      0.722      0.428      0.455      0.275

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      48/50      8.29G      1.351      1.046     0.9652        216        896: 100%|██████████| 282/282 [01:17<00:00,  3.62it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 36/36 [00:13<00:00,  2.70it/s]
                   all        283      20793      0.722       0.43      0.455      0.276

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      49/50      7.79G      1.336      1.028     0.9589         44        896: 100%|██████████| 282/282 [01:19<00:00,  3.55it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 36/36 [00:13<00:00,  2.62it/s]
                   all        283      20793      0.741      0.419      0.453      0.278


      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      50/50      9.05G      1.346      1.014     0.9497        175        896: 100%|██████████| 282/282 [01:19<00:00,  3.54it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 36/36 [00:13<00:00,  2.65it/s]
                   all        283      20793      0.735      0.422      0.457      0.278
```

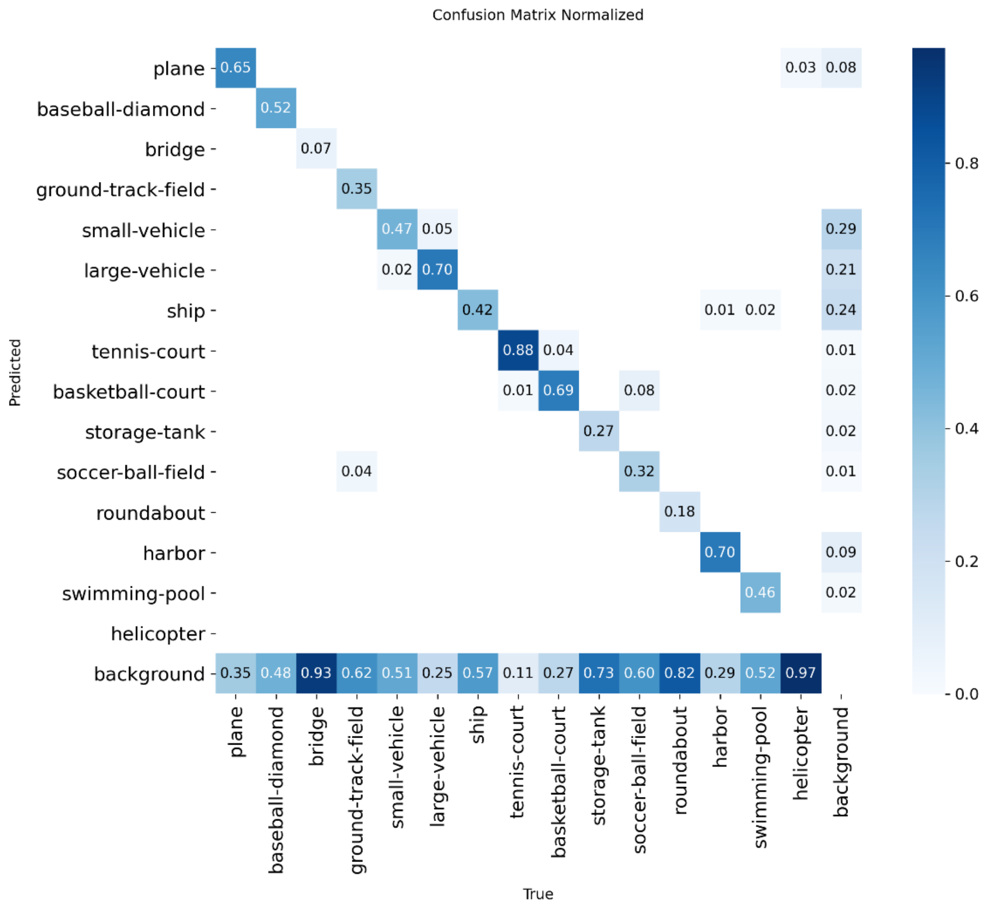
Evaluation & Inference

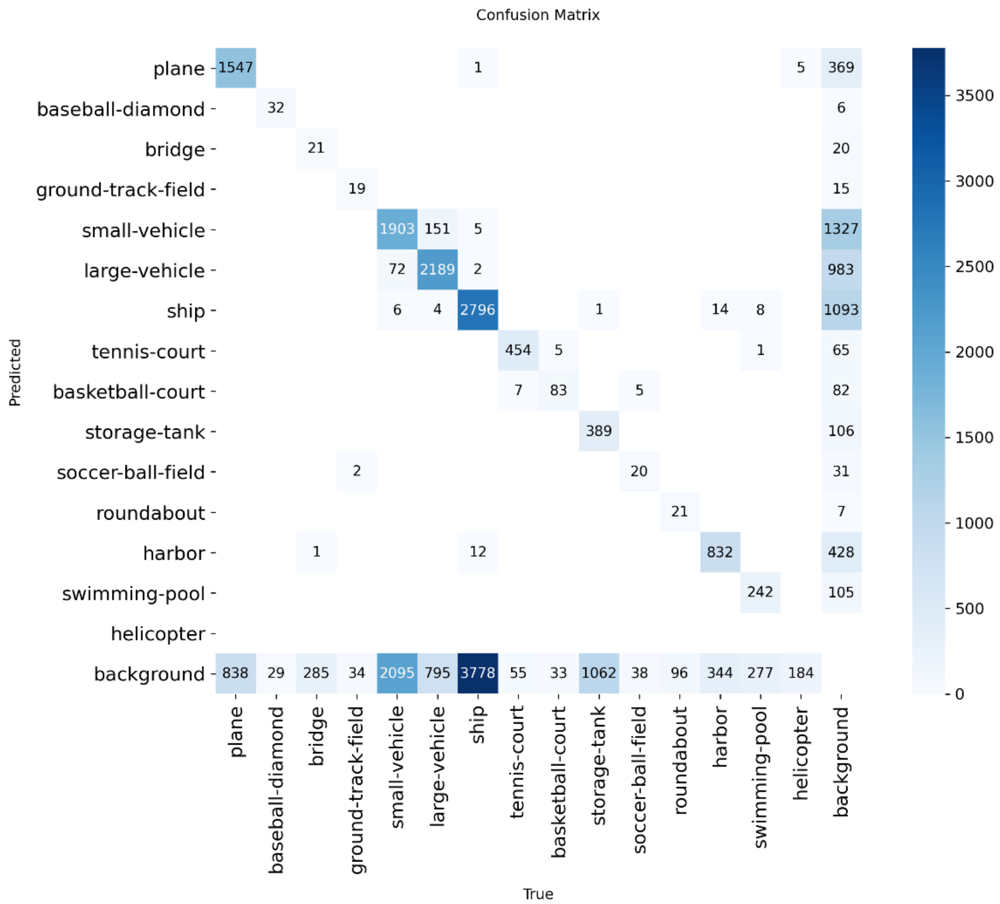
Loaded the trained model (`best.pt`) and performed inference on a random selection of 10 validation images from the `images/val/` directory.

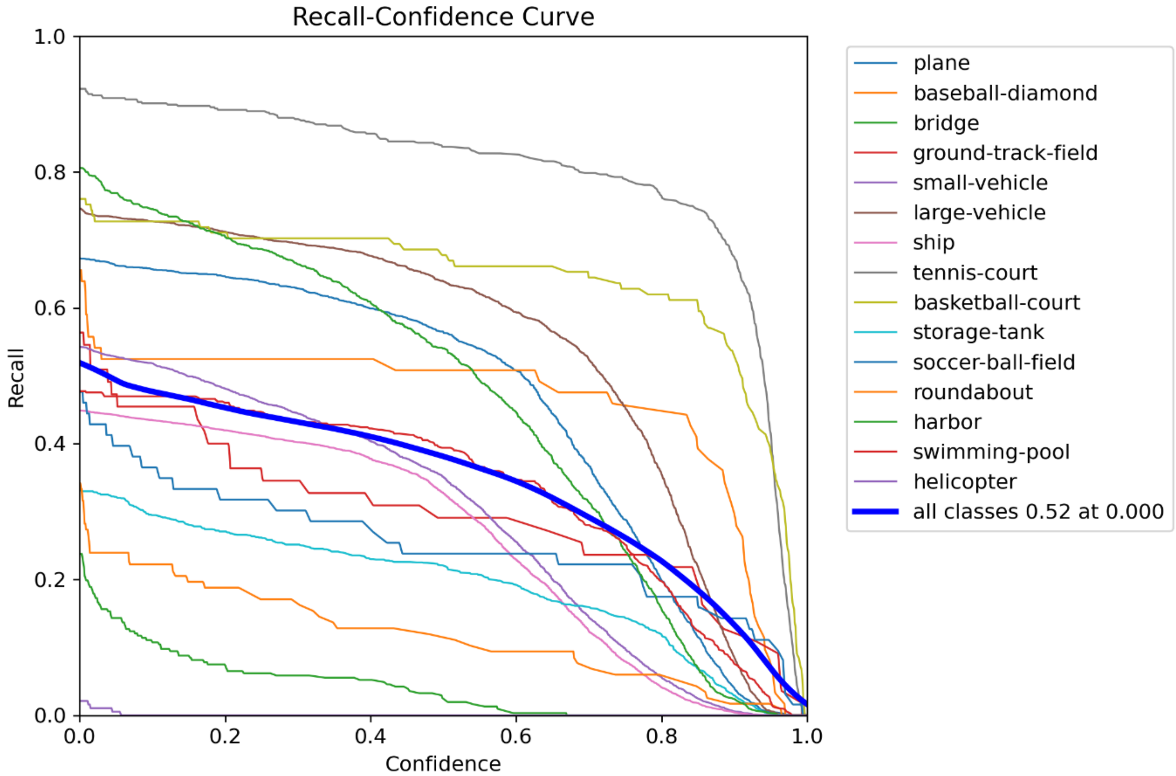
The `.predict()` method was used to generate annotated outputs with bounding boxes, which were then visualized using OpenCV and Matplotlib to confirm detection accuracy. Evaluated the model's performance on the full validation set using the `.val()` method, which returned key metrics including:

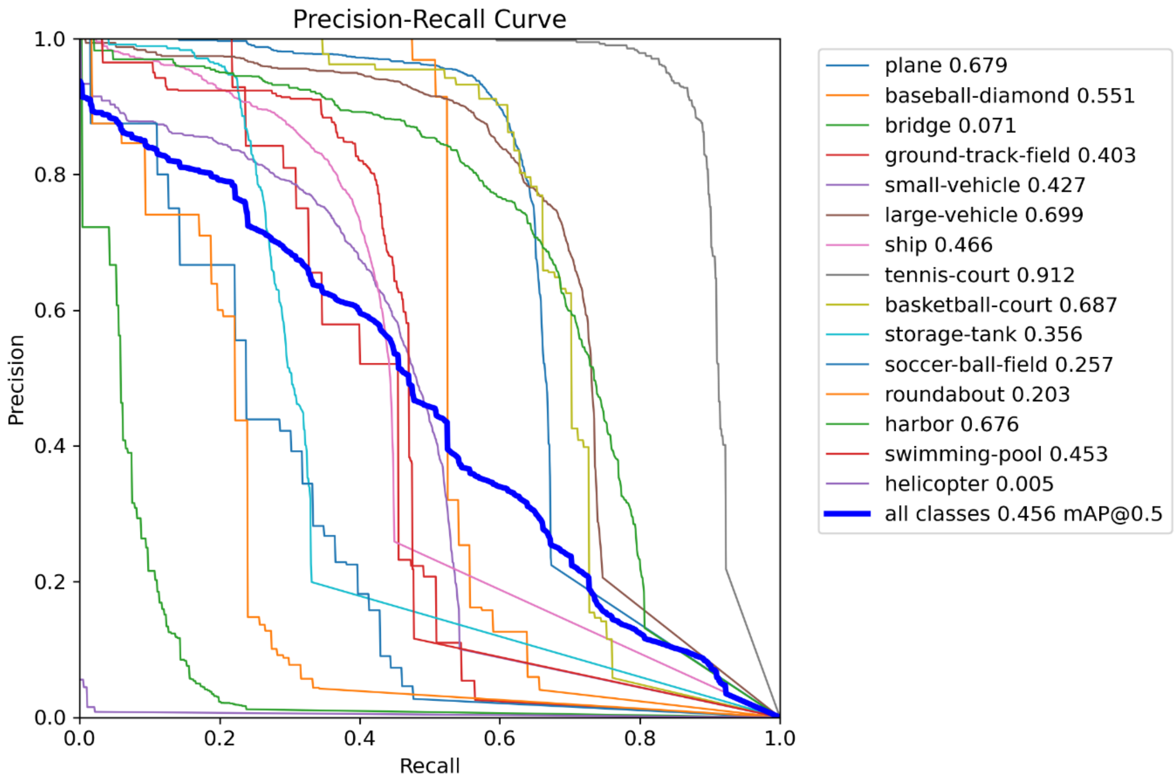
- mAP@0.5
- Precision
- Recall
- Confusion Matrix

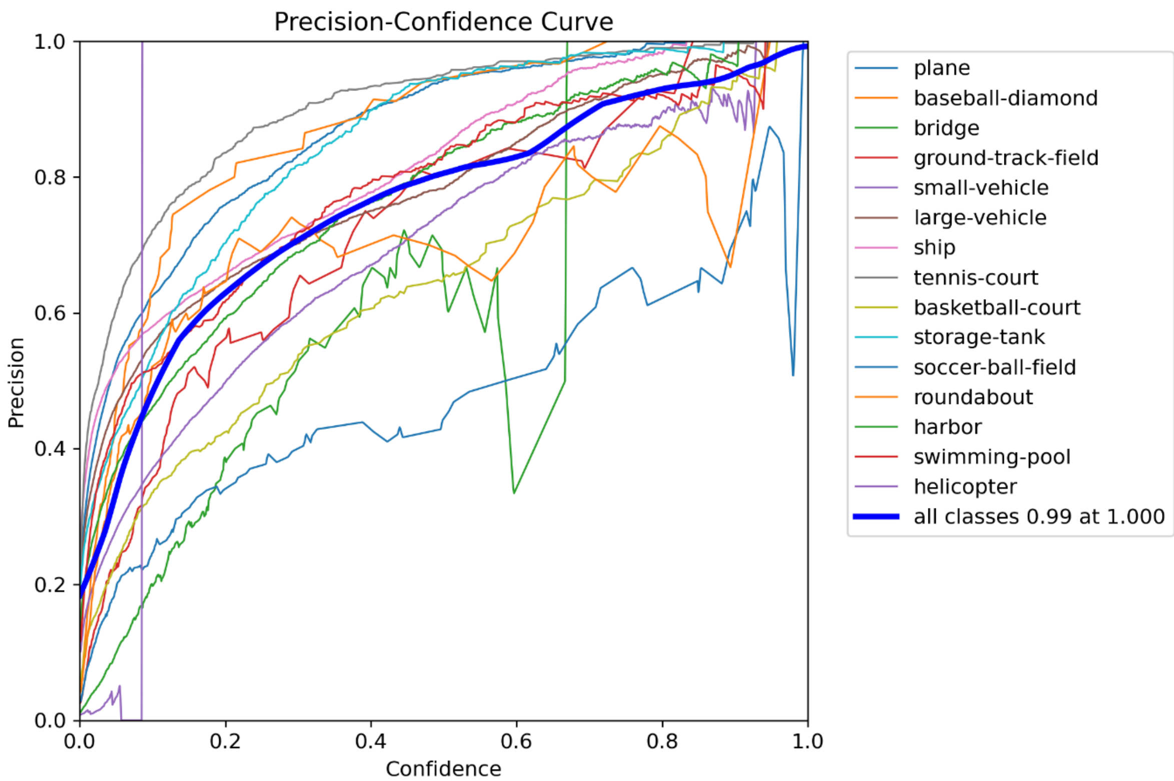
epoch	time	train/box	train/cls_	train/dfl_	metrics/p	metrics/r	metrics/n	metrics/n	val/box_l	val/cls_lo	val/dfl_lo	lr/pg0	lr/pg1	lr/pg2
1	100.23	2.0929	3.7342	1.2656	0.5722	0.1384	0.1134	0.0688	1.9209	2.433	1.1288	0.0002	0.0002	0.0002
2	193.59	1.9564	2.4	1.141	0.5857	0.1889	0.1869	0.1055	1.8499	2.0323	1.0972	0.0003	0.0003	0.0003
3	287.37	1.8773	2.0949	1.1114	0.6362	0.2267	0.2257	0.1238	1.7968	1.861	1.086	0.0005	0.0005	0.0005
4	380.89	1.7903	1.9247	1.0977	0.5721	0.2343	0.237	0.1287	1.7868	1.7614	1.0797	0.0005	0.0005	0.0005
5	472.23	1.7748	1.8146	1.0852	0.6063	0.2467	0.2592	0.1438	1.7311	1.6148	1.0479	0.0005	0.0005	0.0005
6	564.61	1.7596	1.7106	1.0863	0.6302	0.3022	0.3043	0.1684	1.6798	1.5126	1.0501	0.0005	0.0005	0.0005
7	654.56	1.7025	1.6391	1.0698	0.6423	0.3051	0.3158	0.1749	1.6877	1.4948	1.0455	0.0005	0.0005	0.0005
8	744.82	1.6771	1.5598	1.0605	0.6203	0.3179	0.3145	0.1802	1.6478	1.4471	1.0319	0.0005	0.0005	0.0005
9	835.51	1.639	1.484	1.0426	0.6924	0.3402	0.3543	0.1987	1.6326	1.443	1.0248	0.0004	0.0004	0.0004
10	927.47	1.6037	1.4738	1.0472	0.6411	0.3301	0.3486	0.1899	1.6352	1.3979	1.0255	0.0004	0.0004	0.0004
11	1018.1	1.6283	1.4068	1.0374	0.6852	0.351	0.3661	0.2066	1.6229	1.33	1.0182	0.0004	0.0004	0.0004
12	1109.5	1.5848	1.3581	1.0241	0.718	0.3551	0.3828	0.2145	1.5998	1.3028	1.0148	0.0004	0.0004	0.0004
13	1199.9	1.6027	1.3734	1.0287	0.6742	0.3462	0.3654	0.2042	1.6437	1.3788	1.0231	0.0004	0.0004	0.0004
14	1290.4	1.5775	1.3196	1.0191	0.6491	0.3556	0.3773	0.2072	1.6202	1.3009	1.0114	0.0004	0.0004	0.0004
15	1380.9	1.586	1.3592	1.0197	0.6477	0.3614	0.3729	0.2124	1.5853	1.2617	1.0053	0.0004	0.0004	0.0004
16	1472.6	1.5555	1.2866	1.0146	0.7078	0.3617	0.3918	0.22	1.6056	1.2677	1.016	0.0004	0.0004	0.0004
17	1564.1	1.5647	1.2904	1.0215	0.6778	0.3706	0.3914	0.2267	1.5648	1.226	1.002	0.0004	0.0004	0.0004
18	1659.6	1.4927	1.2042	1.0065	0.6786	0.3791	0.3974	0.2302	1.5389	1.1947	0.995	0.0003	0.0003	0.0003
19	1751.3	1.5244	1.2399	1.0044	0.7068	0.3818	0.4058	0.2336	1.5573	1.2229	1.0024	0.0003	0.0003	0.0003
20	1842.8	1.5331	1.2474	1.0022	0.6807	0.3903	0.4143	0.2432	1.5527	1.1867	0.9984	0.0003	0.0003	0.0003
21	1934.6	1.5393	1.2371	1.003	0.7369	0.3793	0.4089	0.2384	1.5337	1.205	0.9977	0.0003	0.0003	0.0003
22	2026.2	1.5188	1.2026	0.9997	0.659	0.3885	0.4084	0.2361	1.5488	1.1725	0.996	0.0003	0.0003	0.0003
23	2119.4	1.4892	1.1803	0.9945	0.6744	0.3807	0.4183	0.2458	1.5283	1.1497	0.9914	0.0003	0.0003	0.0003
24	2210.2	1.5119	1.1673	0.9917	0.6699	0.3931	0.4221	0.2432	1.5506	1.1528	0.996	0.0003	0.0003	0.0003
25	2303.2	1.493	1.1794	0.9981	0.6971	0.3913	0.4229	0.2494	1.5054	1.1345	0.9924	0.0003	0.0003	0.0003
26	2395.6	1.4807	1.1356	0.9899	0.677	0.3907	0.4201	0.2497	1.5371	1.1372	0.9901	0.0003	0.0003	0.0003
27	2487.7	1.4774	1.124	0.9871	0.6992	0.395	0.4253	0.2504	1.524	1.1321	0.9926	0.0003	0.0003	0.0003
28	2578.4	1.4783	1.1241	0.9901	0.6869	0.4046	0.4289	0.2506	1.5458	1.111	0.9942	0.0002	0.0002	0.0002
29	2670.8	1.4758	1.1082	0.9833	0.7125	0.4061	0.4317	0.2537	1.5447	1.1138	0.9929	0.0002	0.0002	0.0002
30	2760	1.4367	1.0953	0.9797	0.7379	0.4048	0.4405	0.2587	1.517	1.1154	0.9868	0.0002	0.0002	0.0002
31	2851	1.4205	1.0979	0.9777	0.6891	0.411	0.4346	0.2576	1.4997	1.0905	0.981	0.0002	0.0002	0.0002
32	2942.2	1.441	1.0711	0.9741	0.6989	0.4113	0.4453	0.2617	1.5151	1.0694	0.986	0.0002	0.0002	0.0002
33	3032.2	1.4155	1.0608	0.9711	0.6784	0.42	0.4364	0.2579	1.5067	1.0917	0.9848	0.0002	0.0002	0.0002
34	3123	1.4174	1.0614	0.9712	0.7078	0.4211	0.4474	0.2653	1.4996	1.0909	0.9862	0.0002	0.0002	0.0002
35	3214.7	1.4028	1.0432	0.9665	0.6944	0.4206	0.4432	0.2652	1.4803	1.0676	0.9779	0.0002	0.0002	0.0002
36	3305	1.4241	1.0573	0.9788	0.7058	0.4232	0.4475	0.2668	1.4897	1.0663	0.9823	0.0002	0.0002	0.0002
37	3397.5	1.3811	1.027	0.9709	0.7225	0.4088	0.4446	0.2618	1.5004	1.0573	0.9842	0.0002	0.0002	0.0002
38	3491	1.396	1.0254	0.9669	0.7418	0.4076	0.4484	0.2661	1.4903	1.0631	0.9804	0.0001	0.0001	0.0001
39	3582	1.3998	1.0194	0.9672	0.7165	0.4191	0.4463	0.2676	1.5031	1.0568	0.9822	0.0001	0.0001	0.0001
40	3672.5	1.4068	1.0408	0.9654	0.731	0.4261	0.4525	0.2709	1.4714	1.0485	0.9726	0.0001	0.0001	0.0001
41	3767.4	1.3954	1.1184	0.974	0.7013	0.4256	0.4474	0.2672	1.4859	1.0596	0.9811	0.0001	0.0001	0.0001
42	3859	1.3544	1.0922	0.9703	0.7437	0.4125	0.4506	0.2684	1.5093	1.0632	0.9824	9.89932	9.89932	9.89932e-05
43	3951.1	1.3589	1.0693	0.9638	0.7518	0.4169	0.4531	0.2739	1.4791	1.0374	0.9768	8.85784	8.85784	8.85784e-05
44	4043.2	1.3464	1.0648	0.9654	0.7282	0.4276	0.4506	0.2711	1.4867	1.0407	0.9779	7.81636e	7.81636e	7.81636e-05
45	4134.8	1.3511	1.0486	0.9659	0.7265	0.4247	0.4555	0.2756	1.4722	1.0345	0.9745	6.77488	6.77488	6.77488e-05
46	4227.3	1.3384	1.0113	0.9573	0.7395	0.4173	0.453	0.2724	1.485	1.023	0.9766	5.7334e-	5.7334e-	5.7334e-05
47	4320	1.349	1.0607	0.9627	0.722	0.4279	0.4549	0.275	1.4705	1.0305	0.9737	4.69192e	4.69192e	4.69192e-05
48	4412.9	1.3509	1.0455	0.9652	0.722	0.4297	0.4549	0.276	1.4778	1.027	0.9738	3.65044	3.65044	3.65044e-05
49	4507.1	1.3358	1.0277	0.9589	0.7413	0.4192	0.4534	0.2777	1.4713	1.0261	0.9721	2.60896	2.60896	2.60896e-05
50	4602	1.3457	1.0137	0.9497	0.7347	0.4225	0.4571	0.2783	1.4675	1.0204	0.9712	1.56748e	1.56748e	1.56748e-05

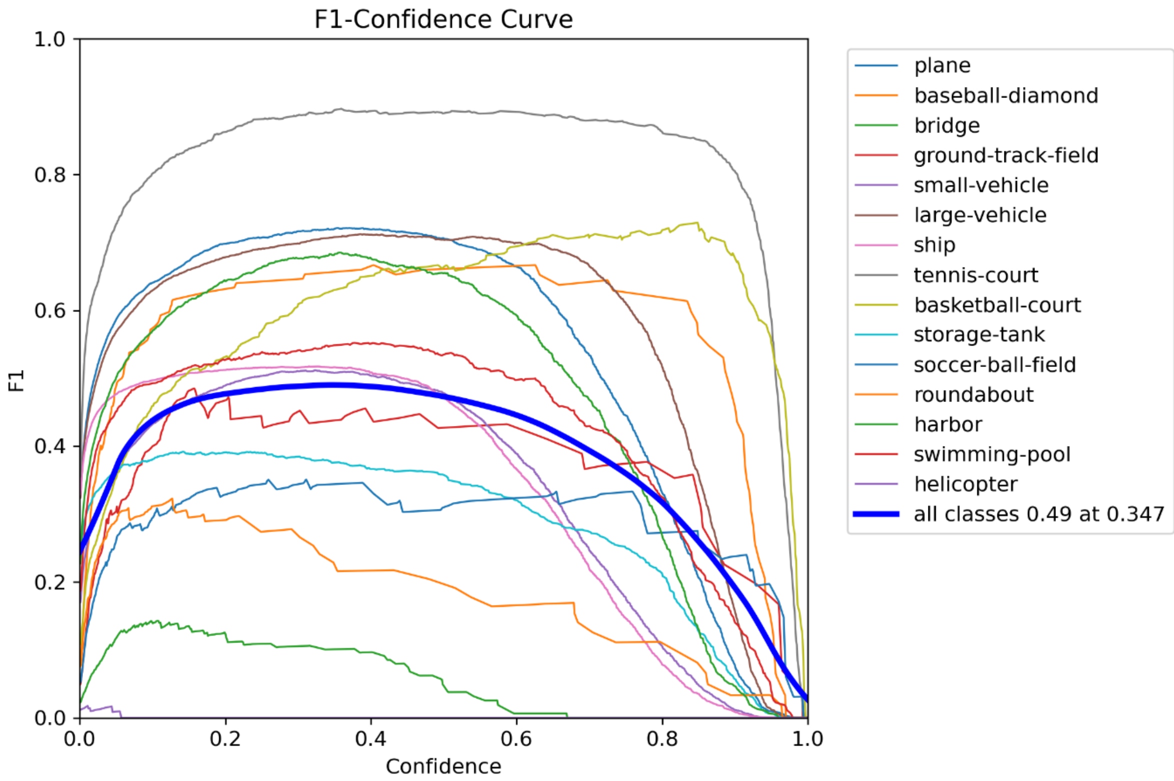

Confusion Matrix Normalized
Predicted
True
plane
baseball-diamond
bridge
ground-track-field
small-vehicle
large-vehicle
ship
tennis-court
basketball-court
storage-tank
soccer-ball-field
roundabout
harbor
swimming-pool
helicopter
background

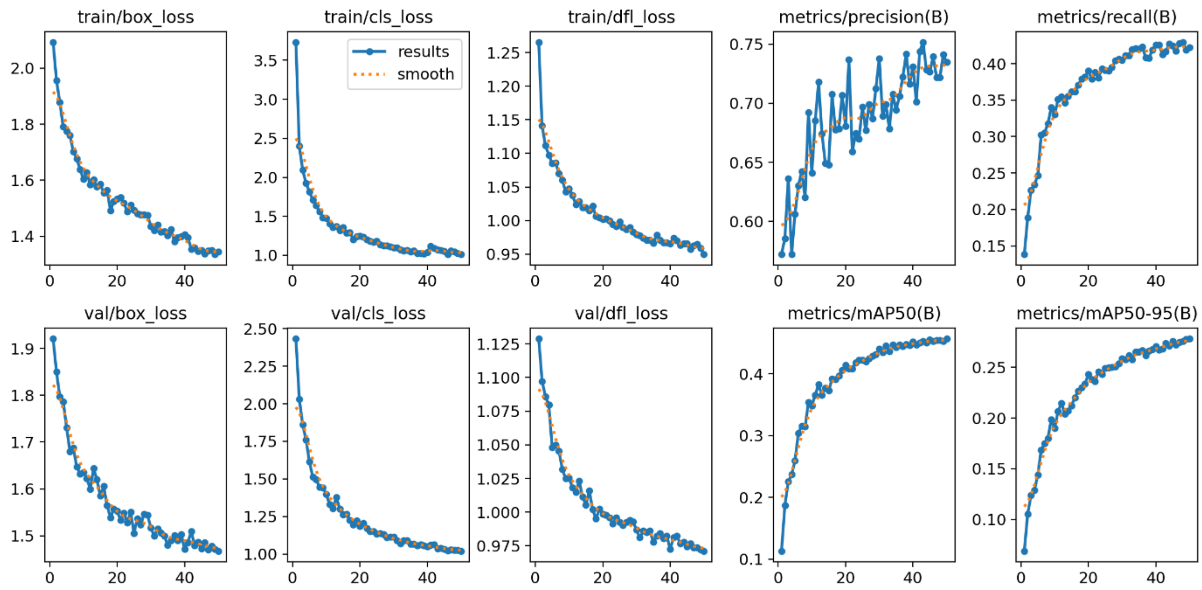

Confusion Matrix
Predicted
True
plane
baseball-diamond
bridge
ground-track-field
small-vehicle
large-vehicle
ship
tennis-court
basketball-court
storage-tank
soccer-ball-field
roundabout
harbor
swimming-pool
helicopter
background


Recall-Confidence Curve
Recall
Confidence
plane
baseball-diamond
bridge
ground-track-field
small-vehicle
large-vehicle
ship
tennis-court
basketball-court
storage-tank
soccer-ball-field
roundabout
harbor
swimming-pool
helicopter
all classes 0.52 at 0.000


Precision-Recall Curve
Precision
Recall
plane 0.679
baseball-diamond 0.551
bridge 0.071
ground-track-field 0.403
small-vehicle 0.427
large-vehicle 0.699
ship 0.466
tennis-court 0.912
basketball-court 0.687
storage-tank 0.356
soccer-ball-field 0.257
roundabout 0.203
harbor 0.676
swimming-pool 0.453
helicopter 0.005
all classes 0.456 mAP@0.5


Precision-Confidence Curve
Precision
Confidence
plane
baseball-diamond
bridge
ground-track-field
small-vehicle
large-vehicle
ship
tennis-court
basketball-court
storage-tank
soccer-ball-field
roundabout
harbor
swimming-pool
helicopter
all classes 0.99 at 1.000


F1-Confidence Curve
F1
Confidence
plane
baseball-diamond
bridge
ground-track-field
small-vehicle
large-vehicle
ship
tennis-court
basketball-court
storage-tank
soccer-ball-field
roundabout
harbor
swimming-pool
helicopter
all classes 0.49 at 0.347



---

6. Web Application (Flask + Bootstrap)

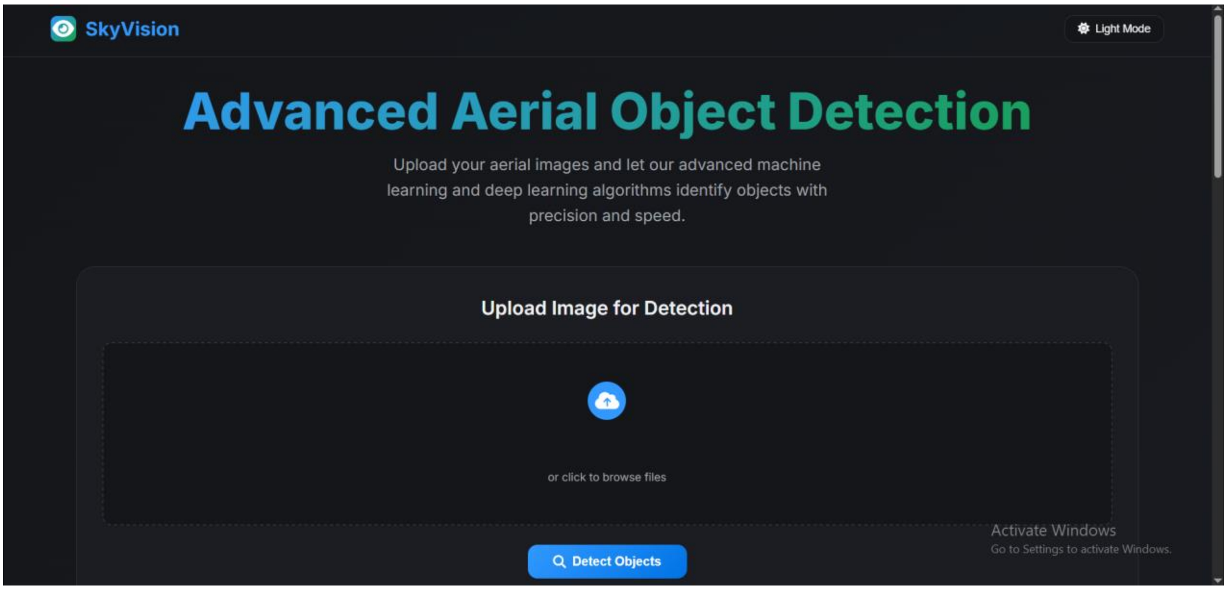
Backend (localapp.py)

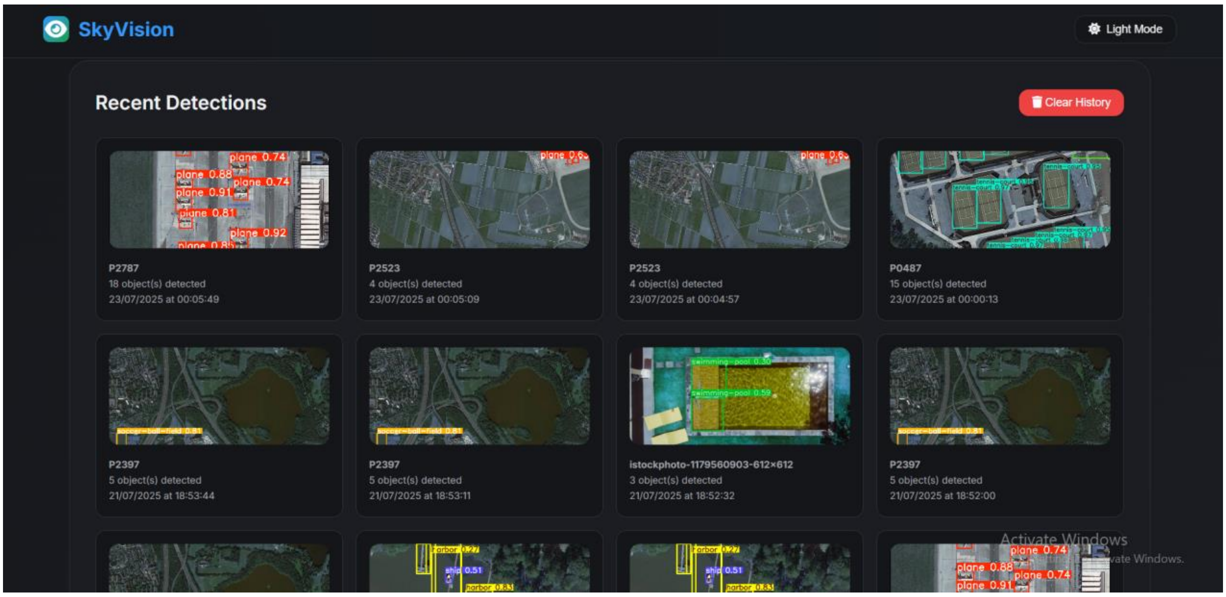
- Loaded the trained YOLOv8 model (best.pt) once during server startup to optimize performance.
- /: Serves the main HTML interface.
- /predict: Accepts uploaded images, runs object detection, saves results, and returns JSON response.
- /health: Returns a simple health check status for frontend readiness.
- Uploaded images are stored in a designated static/uploads/ directory. Detection results (with bounding boxes) are saved in the same directory for client display.
- Upon receiving an image, the model generates bounding boxes and class labels, and detection statistics are compiled and sent back in the response.

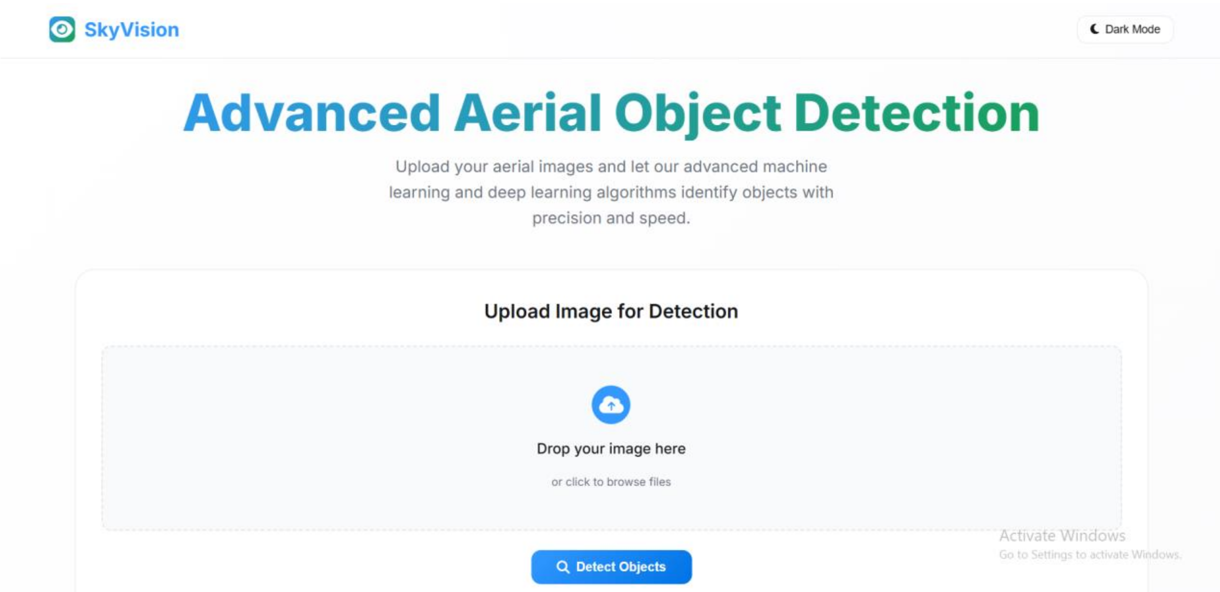
Frontend (HTML + Bootstrap)

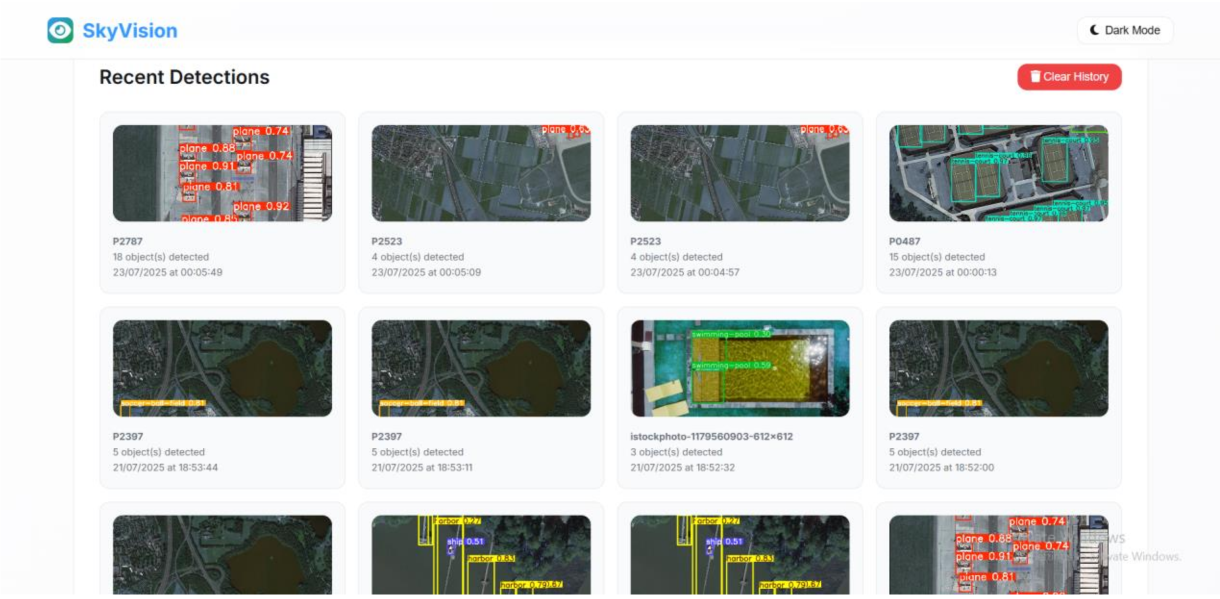
- User-friendly, responsive interface.
- Features:
    - drag-and-drop and file picker functionality for uploading images.
    - displayed both the uploaded image and the annotated result side by side using <img> tags.

.

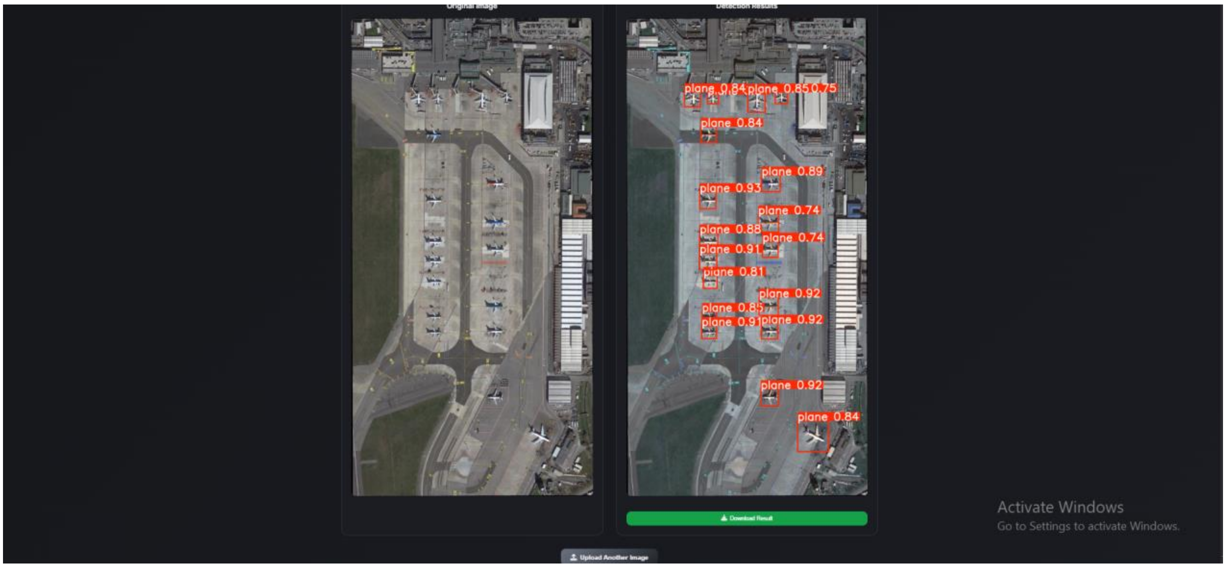

SkyVision
Light Mode
Advanced Aerial Object Detection
Upload your aerial images and let our advanced machine learning and deep learning algorithms identify objects with precision and speed.
Upload Image for Detection
or click to browse files
Detect Objects
Activate Windows
Go to Settings to activate Windows.

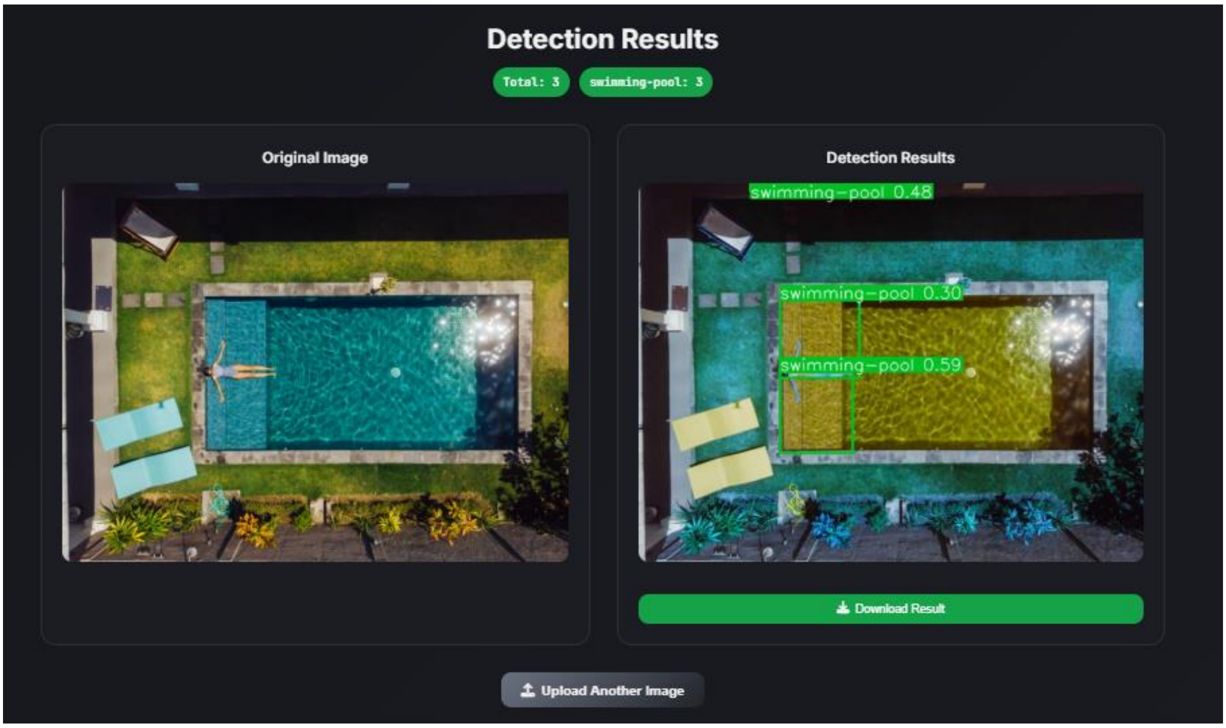

SkyVision
Light Mode
Recent Detections
Clear History
P2787
18 object(s) detected
23/07/2025 at 00:05:49
P2523
4 object(s) detected
23/07/2025 at 00:05:09
P2523
4 object(s) detected
23/07/2025 at 00:04:57
P0487
15 object(s) detected
23/07/2025 at 00:00:13
P2397
5 object(s) detected
21/07/2025 at 18:53:44
P2397
5 object(s) detected
21/07/2025 at 18:53:11
istockphoto-1179560903-612×612
3 object(s) detected
21/07/2025 at 18:52:32
P2397
5 object(s) detected
21/07/2025 at 18:52:00
Activate Windows

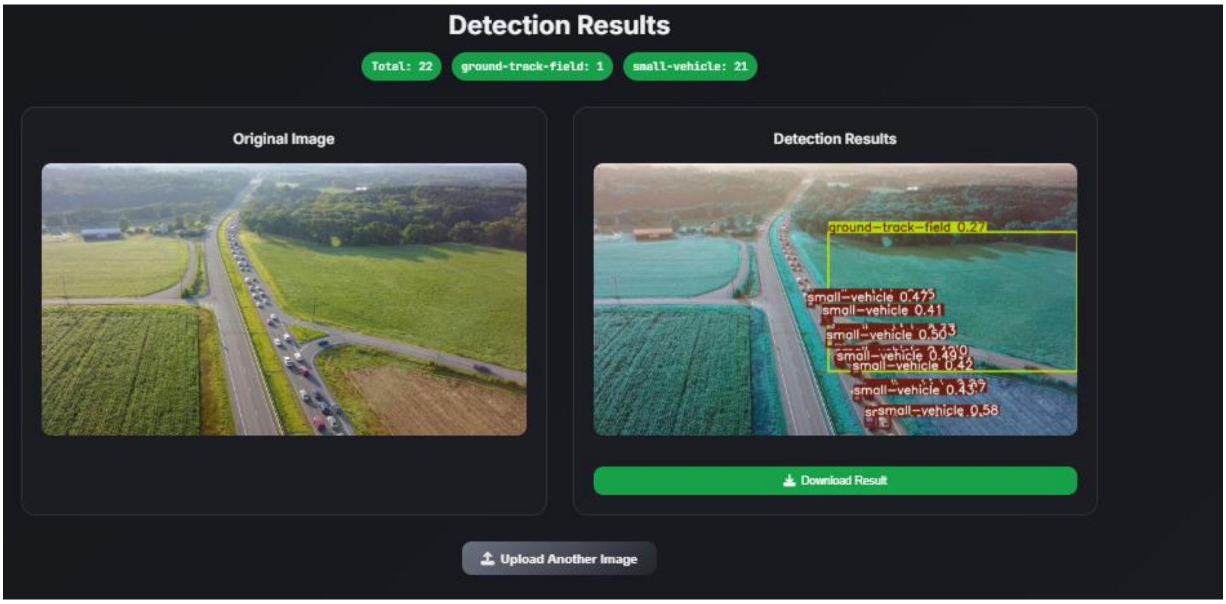

SkyVision
Dark Mode
Advanced Aerial Object Detection
Upload your aerial images and let our advanced machine learning and deep learning algorithms identify objects with precision and speed.
Upload Image for Detection
Drop your image here
or click to browse files
Detect Objects
Activate Windows
Go to Settings to activate Windows.

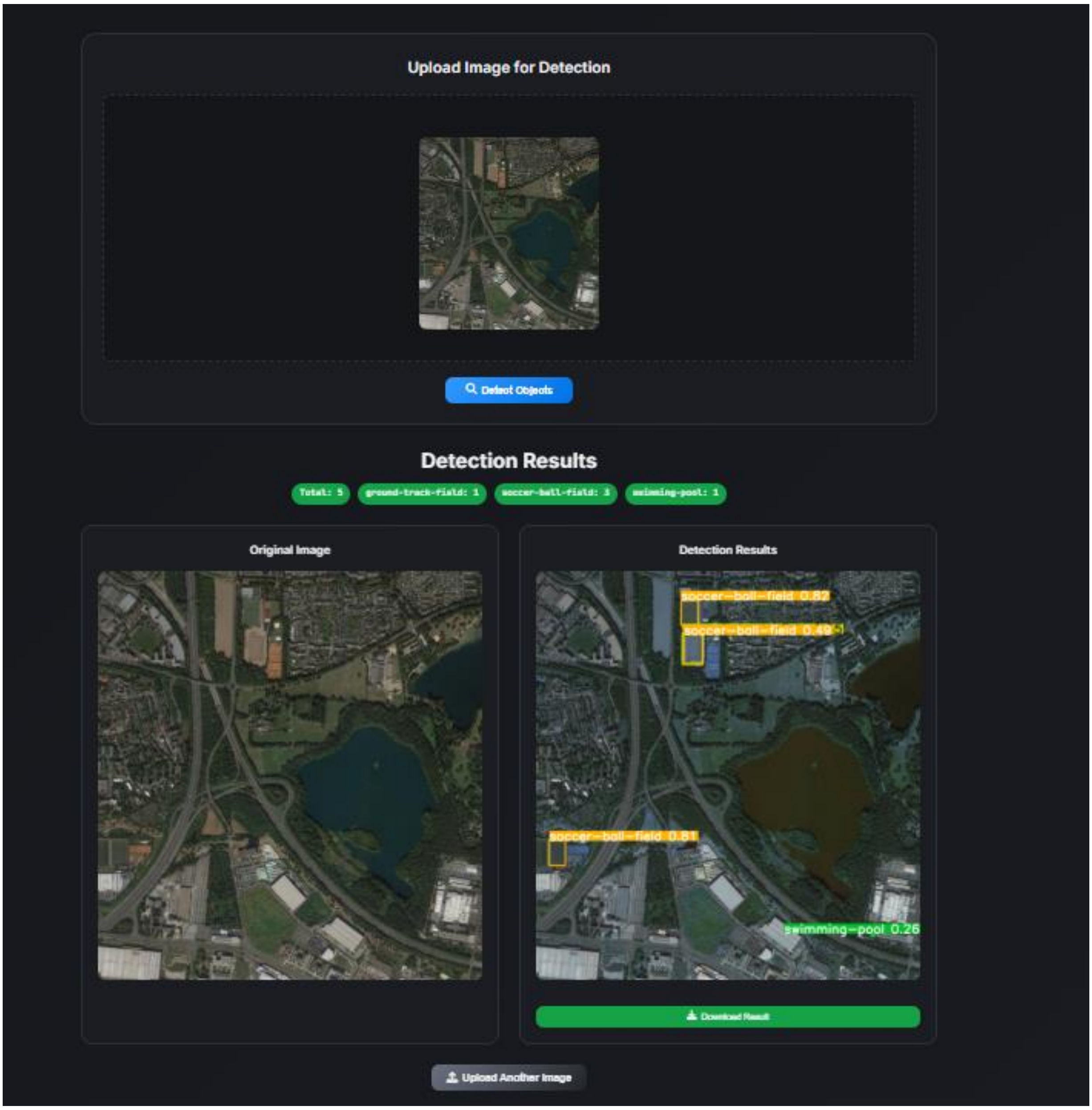

SkyVision
Dark Mode
Recent Detections
Clear History
P2787
18 object(s) detected
23/07/2025 at 00:05:49
P2523
4 object(s) detected
23/07/2025 at 00:05:09
P2523
4 object(s) detected
23/07/2025 at 00:04:57
P0487
15 object(s) detected
23/07/2025 at 00:00:13
P2397
5 object(s) detected
21/07/2025 at 18:53:44
P2397
5 object(s) detected
21/07/2025 at 18:53:11
istockphoto-1179560903-612×612
3 object(s) detected
21/07/2025 at 18:52:32
P2397
5 object(s) detected
21/07/2025 at 18:52:00

Results Display






Detection Results
Total: 22
ground-track-field: 1
small-vehicle: 21
Original Image
Detection Results
ground-track-field 0.27
small-vehicle 0.41
small-vehicle 0.42
small-vehicle 0.58
Download Result
Upload Another Image


Upload Image for Detection
Detect Objects
Detection Results
Total: 5
ground-track-field: 1
soccer-ball-field: 3
swimming-pool: 1
Original Image
Detection Results
soccer-ball-field 0.82
soccer-ball-field 0.49
soccer-ball-field 0.81
swimming-pool 0.26
Download Result
Upload Another Image

7. Technologies Used

Component	Tech Stack
Model Training	YOLOv8 (Ultralytics),PyTorch
Image Handling	OpenCV, PIL, NumPy
Data Analysis	Pandas, seaborn, matplotlib
Web App Backend	Flask, Python
Frontend UI	HTML, CSS, Bootstrap
Deployment	Localhost Testing
Logging	Python Logging, CSV Logger

8. Limitations & Future Work

- Add support for DOTA-v2 and rotated bounding boxes
- Deploy on cloud (Render, AWS, etc.)
- Add batch processing and performance charts
- Extend UI to allow video detection or heatmap overlays

9. Literature Review

- YOLO9000: Better, Faster, Stronger

Joseph Redmon and Ali Farhadi, 2017
Introduced the YOLOv2 architecture, improving detection speed and accuracy. This work was foundational in real-time object detection and laid the groundwork for modern YOLO variants like YOLOv8.

- **DOTA: A Large-Scale Dataset for Object Detection in Aerial Images**
  Gui-Song Xia et al., 2018
  Presented the DOTA dataset, which includes oriented bounding boxes across 15 categories in aerial images. This dataset became a benchmark for aerial detection research and was used in our project.
- **YOLOv5 by Ultralytics: Performance and Usability for Object Detection**
  Ultralytics, 2020–2023
  YOLOv5 was a major evolution in the YOLO series, introducing modular PyTorch training pipelines. It significantly simplified deployment and training, influencing YOLOv8's development.
- **YOLOv8: Cutting-Edge Object Detection with Ultralytics**
  Ultralytics, 2023
  YOLOv8 integrates improvements such as anchor-free detection and a streamlined CLI for training, validation, and export. This version was used in our project due to its ease of use and high performance on custom datasets.

---

10. References

- DOTA Dataset: https://captain-whu.github.io/DOTA/
- Ultralytics YOLOv8: https://docs.ultralytics.com
- Flask Framework: https://flask.palletsprojects.com