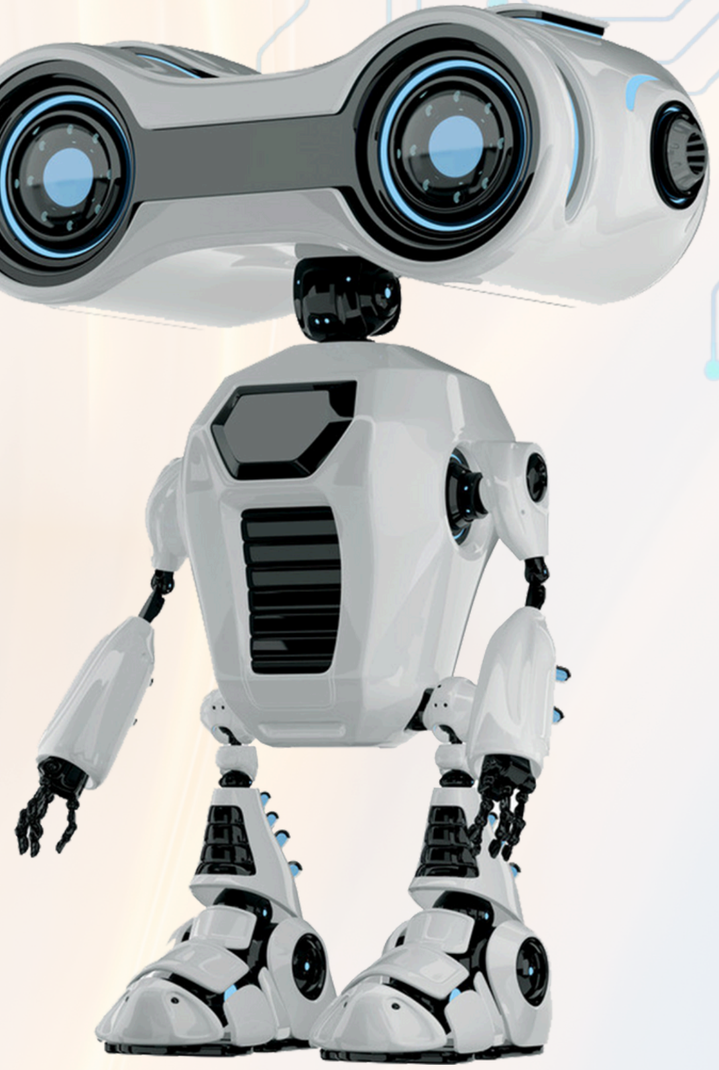
MID-TERM PROJECT

IMPLEMENTING OBJECT DETECTION ON DATASET



PRESENTORS

Flores, Edrian

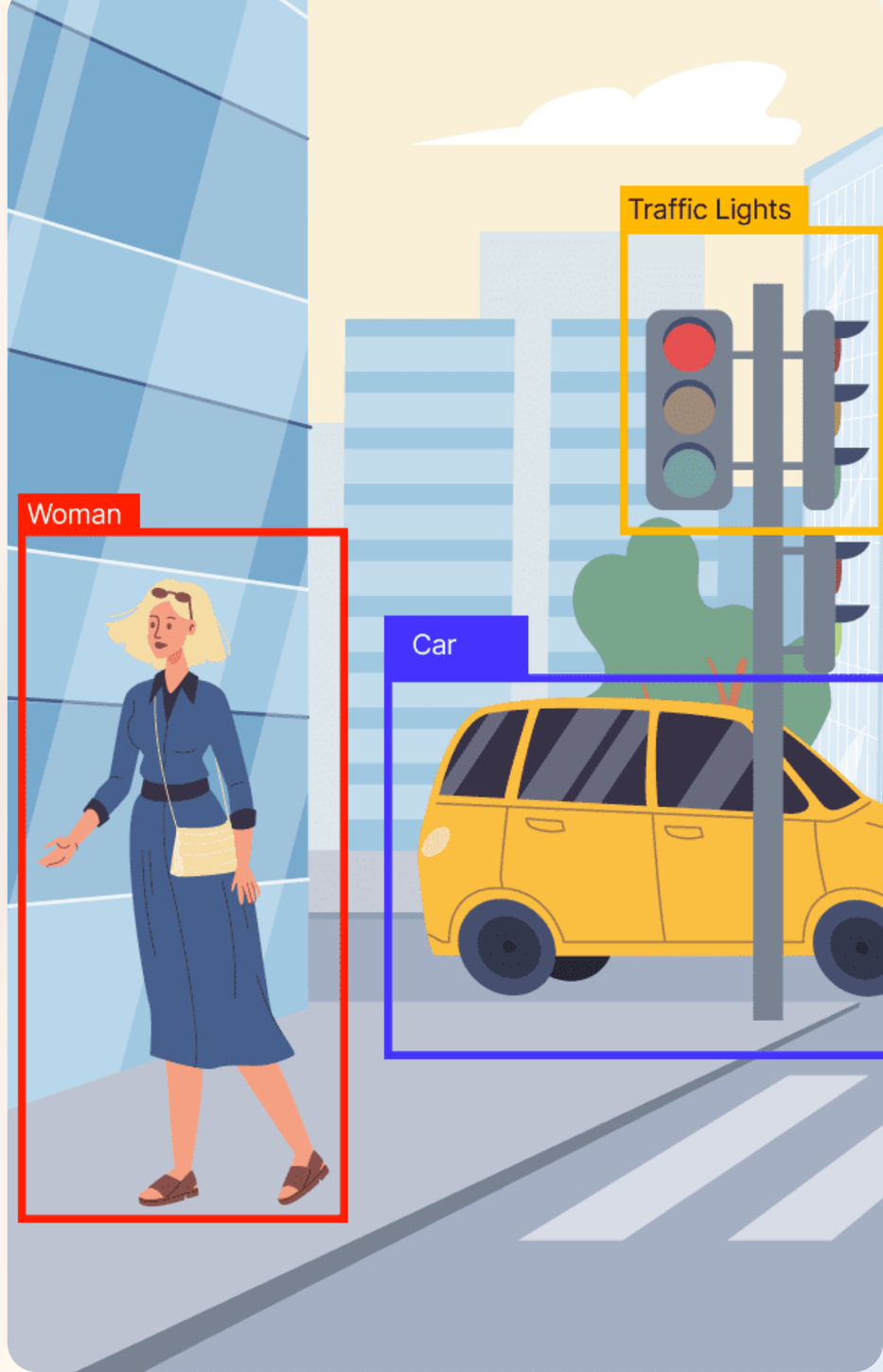
Villadiego, Trish

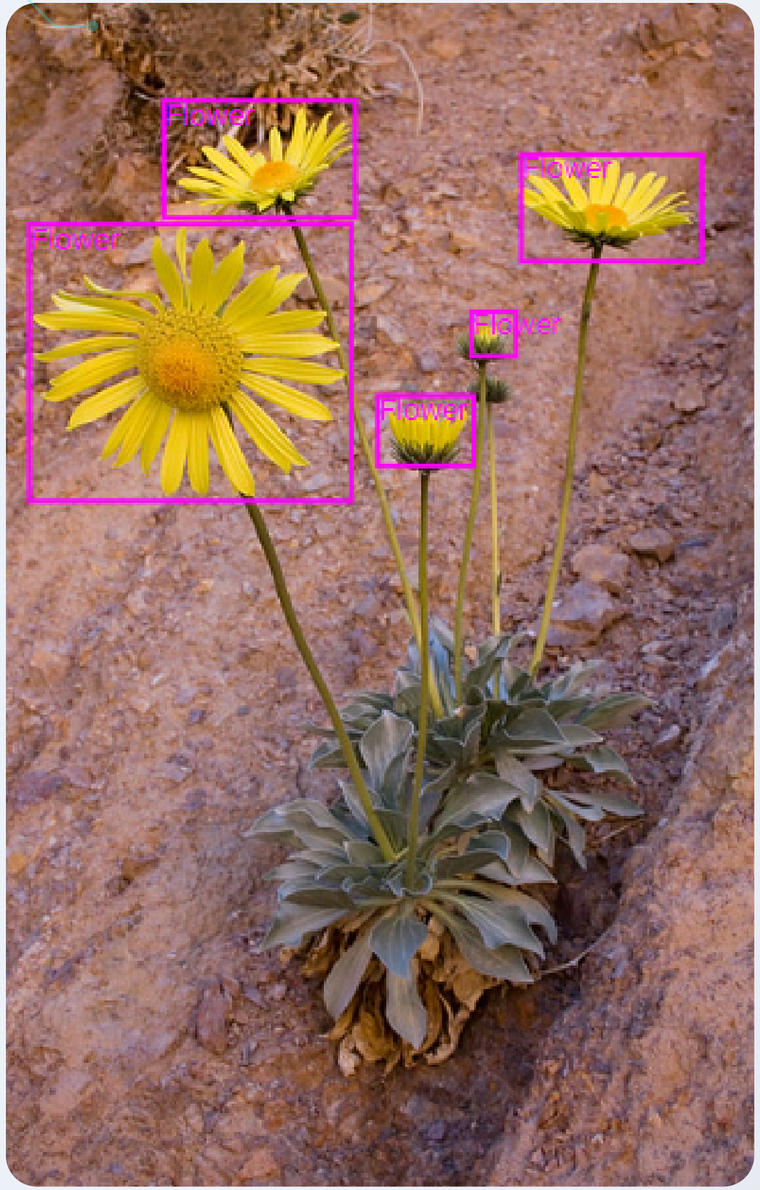
PROJECT OVERVIEW

Our project is focused on building an object detection model using machine learning. Object detection enables the identification and localization of objects within images, which has applications in fields like security, retail, and autonomous vehicles.

Topics to Cover:

- Data Preparation
- Model Implementation
- Training the Model
- Testing and Evaluation
- Discussion of Challenges
- Conclusion and Next Steps





DATA PREPARATION

Dataset Used: Flowers Dataset

Dataset Description: The dataset contains various images of flowers, providing a good basis for detecting distinct objects (flowers) within a complex background. This dataset also has labeled bounding boxes for flowers, which are necessary for supervised training in object detection.

Preprocessing Steps: The dataset underwent preprocessing steps, such as Auto Orient and Resize (Stretch to 640x640), within the Roboflow environment. Roboflow, a popular platform for managing computer vision datasets, can automate tasks like resizing, orienting images, and even labeling bounding boxes. These actions ensure all images in the dataset are consistently oriented and sized, which is crucial for deep learning models, as it standardizes input dimensions and helps improve model performance.

```python
import zipfile
import os

# Path to the zip file
dataset_zip_path = '/content/Flower(Multiclass).zip'

# Directory to extract to
extract_dir = '/content/Flower(Multiclass)/'

# Unzipping the file
with zipfile.ZipFile(dataset_zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

print(f"Dataset unzipped to {extract_dir}")
```

```python
# Path to the data.yaml file
yaml_file_path = '/content/Flower(Multiclass)/data.yaml'

# Reading and displaying the content of the data.yaml file
with open(yaml_file_path, 'r') as file:
    data = file.read()
    print(data)
```

MODEL IMPLEMENTATION

Object Detection Algorithm: We chose YOLO (You Only Look Once) due to its real-time processing capabilities and strong accuracy in object detection tasks.

Building Model:

```
!pip install ultralytics
```

Explanation: To implement YOLO, this command installs the Ultralytics library.

TRAINING THE MODEL

Dataset Splitting:

To evaluate the performance of our flower detection model effectively, we divided the dataset into three parts: Training, Validation, and Test sets. This method is known as the 70/20/10 split.

Training Process:

```
from ultralytics import YOLO

# Load the model
model = YOLO('yolov10m.pt')

model.train(
    data='/content/Flower(Multiclass)/data.yaml',
    epochs=100,
    imgsz=640,
    batch=16,
    workers=4,
    name='yolov10m-flowerst1',
    device=0
)
```

Using the ultralytics library, the model is first loaded with a pre-trained weight file named yolov10m.pt. This weight file acts as a starting point for training, as it contains parameters learned from previous training on a similar dataset, which helps speed up the learning process and improve model performance on the current flower dataset.

OUTPUT

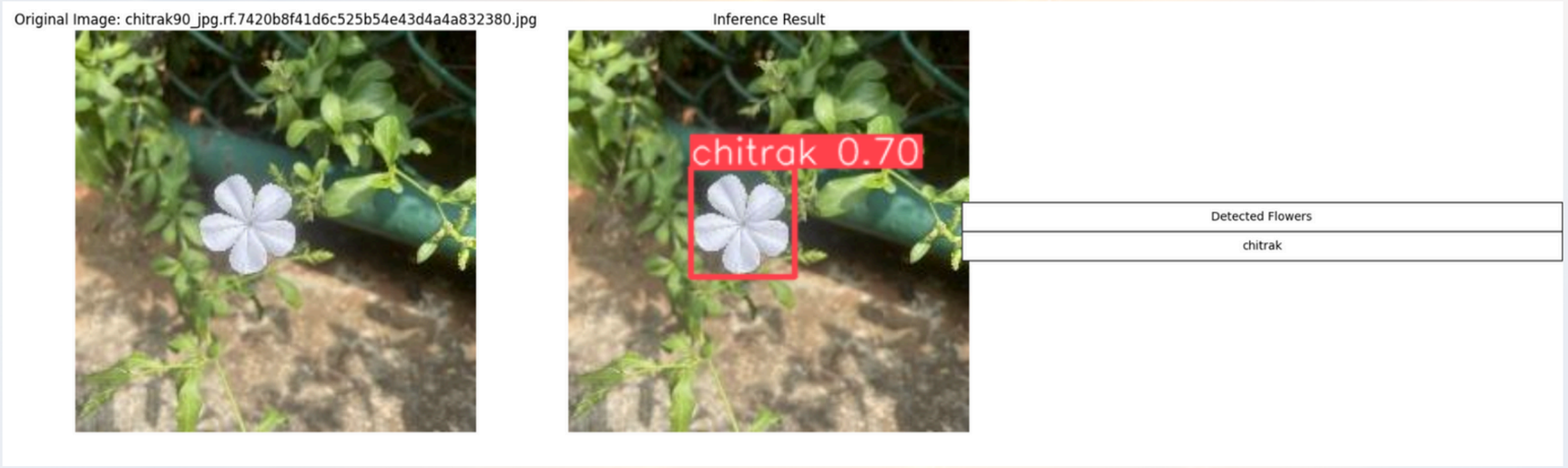
```
Final Training Metrics:
    Precision   Recall   mAP@50  mAP@50-95
99    0.68977  0.87253  0.74898    0.63364
```

```
Metrics for Best Checkpoint (Epoch 75):
Precision    0.69556
Recall       0.91492
mAP@50       0.76905
mAP@50-95    0.63957
Name: 75, dtype: float64
```

```python
    # Record speed metrics
    preprocess_time = (end_preprocess - start_preprocess) * 1000  # Convert to ms
    inference_time = (end_inference - start_inference) * 1000  # Convert to ms
    postprocess_time = (end_postprocess - start_postprocess) * 1000  # Convert to ms
    total_time = preprocess_time + inference_time + postprocess_time

    speed_metrics.append({
        'preprocess': preprocess_time,
        'inference': inference_time,
        'postprocess': postprocess_time,
        'total': total_time
    })

# Calculate average speed metrics
if speed_metrics:
    avg_preprocess = sum([x['preprocess'] for x in speed_metrics]) / len(speed_metrics)
    avg_inference = sum([x['inference'] for x in speed_metrics]) / len(speed_metrics)
    avg_postprocess = sum([x['postprocess'] for x in speed_metrics]) / len(speed_metrics)
    avg_total = sum([x['total'] for x in speed_metrics]) / len(speed_metrics)
    fps = 1000 / avg_total

    print(f"Average Speed Metrics (ms per image):")
    print(f"  Preprocessing: {avg_preprocess:.2f} ms")
    print(f"  Inference: {avg_inference:.2f} ms")
    print(f"  Postprocessing: {avg_postprocess:.2f} ms")
    print(f"  Total: {avg_total:.2f} ms")
    print(f"Estimated FPS: {fps:.2f} frames per second")

# Display 3 to 5 random original and annotated images side by side along with a table of detected flowers
num_images_to_display = min(5, len(original_images))  # Display up to 5 images or fewer if less available
if num_images_to_display >= 3:  # Ensure there are at least 3 images to display
```

Output: describe how the model is performing during the training process.

```
Average Speed Metrics (ms per image):
  Preprocessing: 0.57 ms
  Inference: 46.92 ms
  Postprocessing: 0.99 ms
  Total: 48.48 ms
Estimated FPS: 20.63 frames per second
```

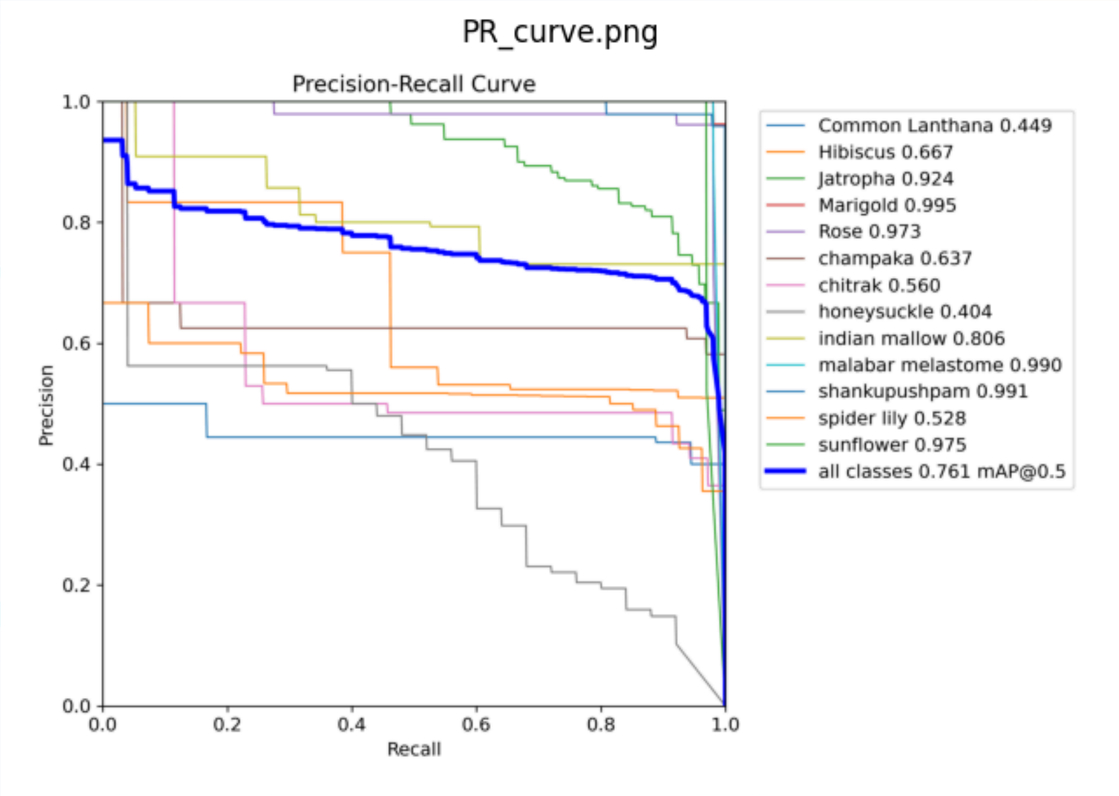


TESTING AND EVALUATION

Testing: We used unlabeled set of images to ensure validity of the object detection.

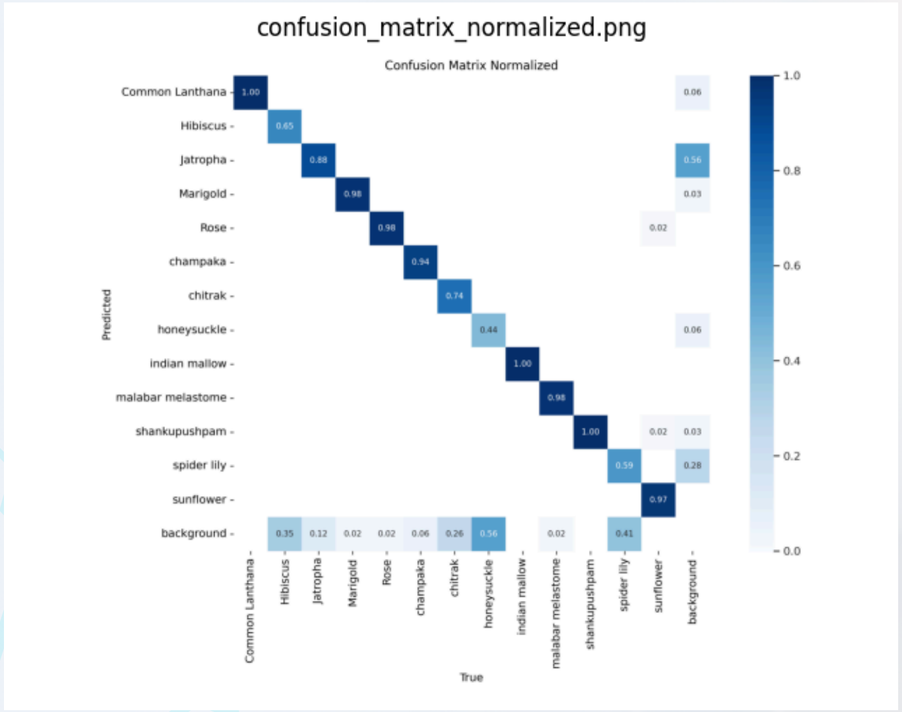
```python
import os
import random
import cv2
import matplotlib.pyplot as plt
from ultralytics import YOLO
import pandas as pd
import time

# Load the YOLO model (replace with your model path)
model = YOLO('/content/runs/detect/yolov10m-flowerst1/weights/best.pt')

# Path to your test image folder
image_folder = '/content/Testing set/Testing set'

# Get a list of all images in the folder
images = [img for img in os.listdir(image_folder) if img.endswith(('.jpg', '.png', '.jpeg'))]

# Prepare to store original and annotated images, detections, and speed metrics
original_images = []
annotated_images = []
detected_flowers_list = []
speed_metrics = []

# Process each image in the folder
for image_file in images:
    image_path = os.path.join(image_folder, image_file)

    # Preprocessing - Load the image
    start_preprocess = time.time()
    image = cv2.imread(image_path)
    if image is None:
        continue  # Skip if image cannot be read
    original_images.append(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for display with matplotlib
    end_preprocess = time.time()

    # Inference using YOLO model
    start_inference = time.time()
    results = model(image_path)
    end_inference = time.time()
```

Evaluation Metrics:

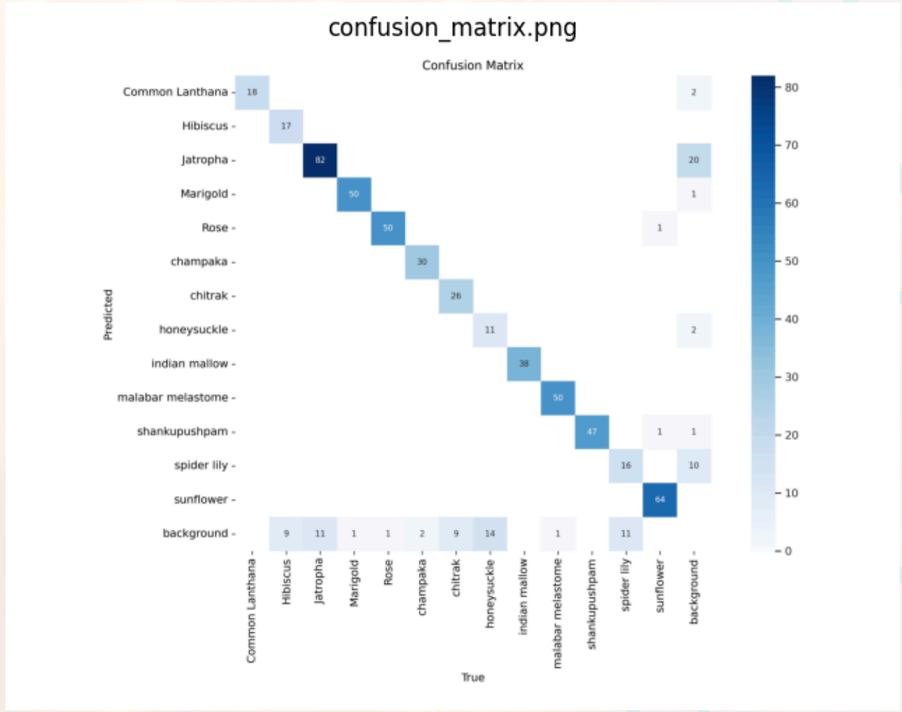


PR Curve: Precision-Recall curve shows the balance between precision and recall at different thresholds.

confusion_matrix_normalized.png



confusion_matrix.png



Confusion Matrices: Two versions—one standard and one normalized—show the distribution of correct and incorrect classifications, helping identify any significant class-wise errors.

Visual Results: After running detections, the code displays a set of original and annotated images side-by-side to visually assess model accuracy, along with tables listing detected objects. This approach provides insights into the model's detection capabilities, speed, and performance in complex cases.

Comparison: YOLOv10 vs HOG-SVM

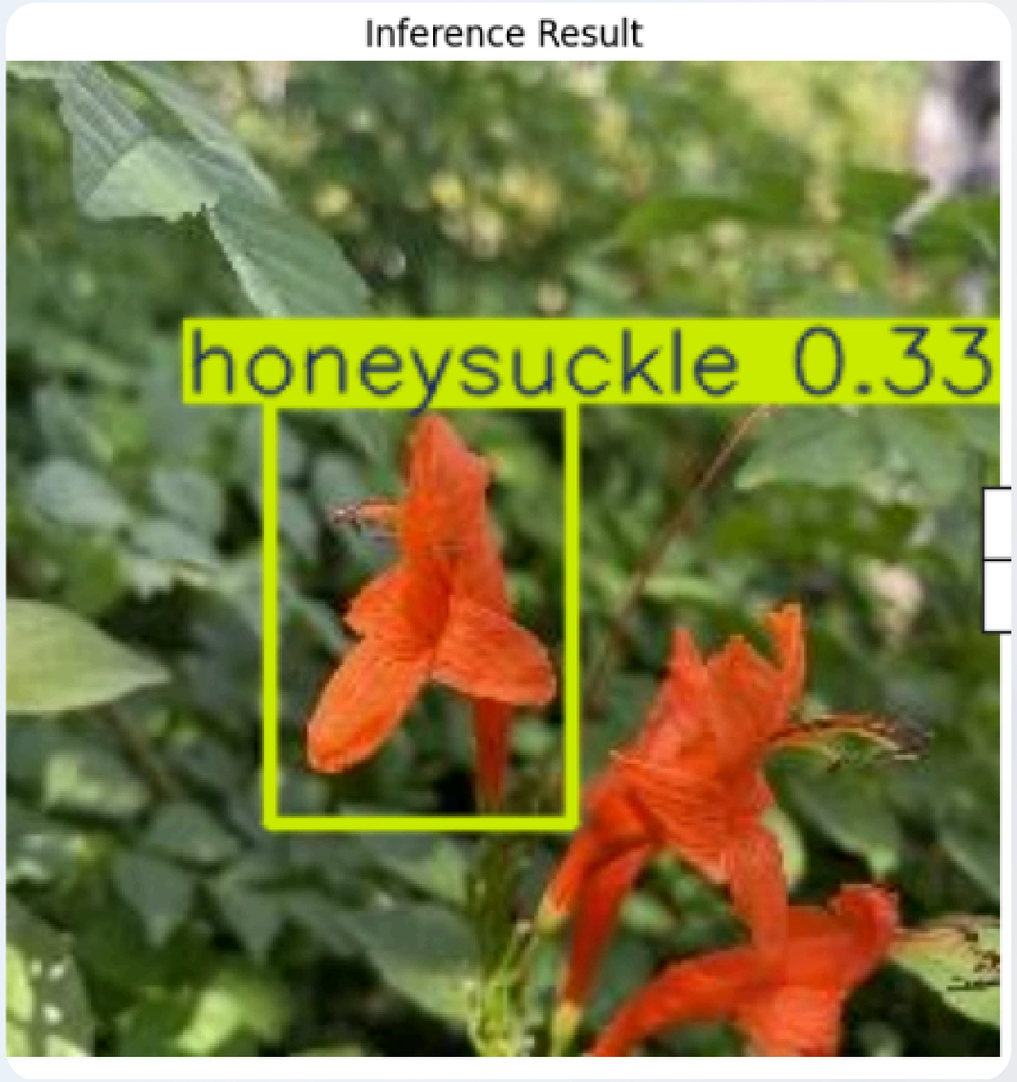
YOLOv10 after training for 100 Epochs:

```
Final Training Metrics:
    Precision   Recall   mAP@50  mAP@50-95
99    0.68977  0.87253  0.74898    0.63364
```

HOG-SVM Evaluation:

```
Validation Accuracy: 0.9938144329896907
Validation Precision: 0.9
Validation Recall: 0.9473684210526315
```

```
Testing Accuracy: 1.0
Test Precision: 1.0
Test Recall: 1.0
```



DISCUSSION OF CHALLENGES

Challenges: YOLOv10

- **Low Confidence Detections:** Some detections, like the "honeysuckle" example (confidence score of 0.33), have low confidence, indicating uncertainty in classification. This can result in missed or inaccurate detections, especially in complex backgrounds.
- **Class Confusion:** YOLOv10 sometimes misidentifies similar-looking flowers, particularly when different species share color or shape characteristics. This reflects the need for more distinct features or additional training data to improve separation between similar classes.
- **Variability in Lighting and Backgrounds:** High variability in lighting conditions and complex backgrounds can lead to lower model accuracy, as seen in natural environments where flowers are often occluded or blended with leaves.
- **Training Time:** YOLOv10 requires significant computational resources and time to train effectively (100 epochs in this case), which can be a limitation for faster model development cycles.

DISCUSSION OF CHALLENGES

Challenges: HOG-SVM

- Complex Backgrounds: HOG-SVM struggles with distinguishing flowers from complex backgrounds, leading to false positives.
- Redundant Detections: The model often detects the same flower multiple times due to overlapping features.
- Low Variability Handling: It performs poorly with changes in scale, orientation, and lighting.
- Feature Engineering Dependency: HOG-SVM requires extensive manual tuning for optimal performance.

CONCLUSION AND NEXT STEPS

Conclusion: Recap the main parts of the project: Suggest potential improvements or future work

Dataset: Always make sure that the dataset and model you'll be using are compatible/same version, always make sure that if you'll be using a synthetic dataset, it should be appropriate for the task you are trying to do (Object Detection vs Classification).

Model Selection: For easier workflow, choose which one you're comfortable with, and in the cases that you're unfamiliar with all of them, try to look for tutorials and pick the algorithm model with the most detailed tutorial you can find so you won't be stuck and clueless at the same time.

THANK YOU!