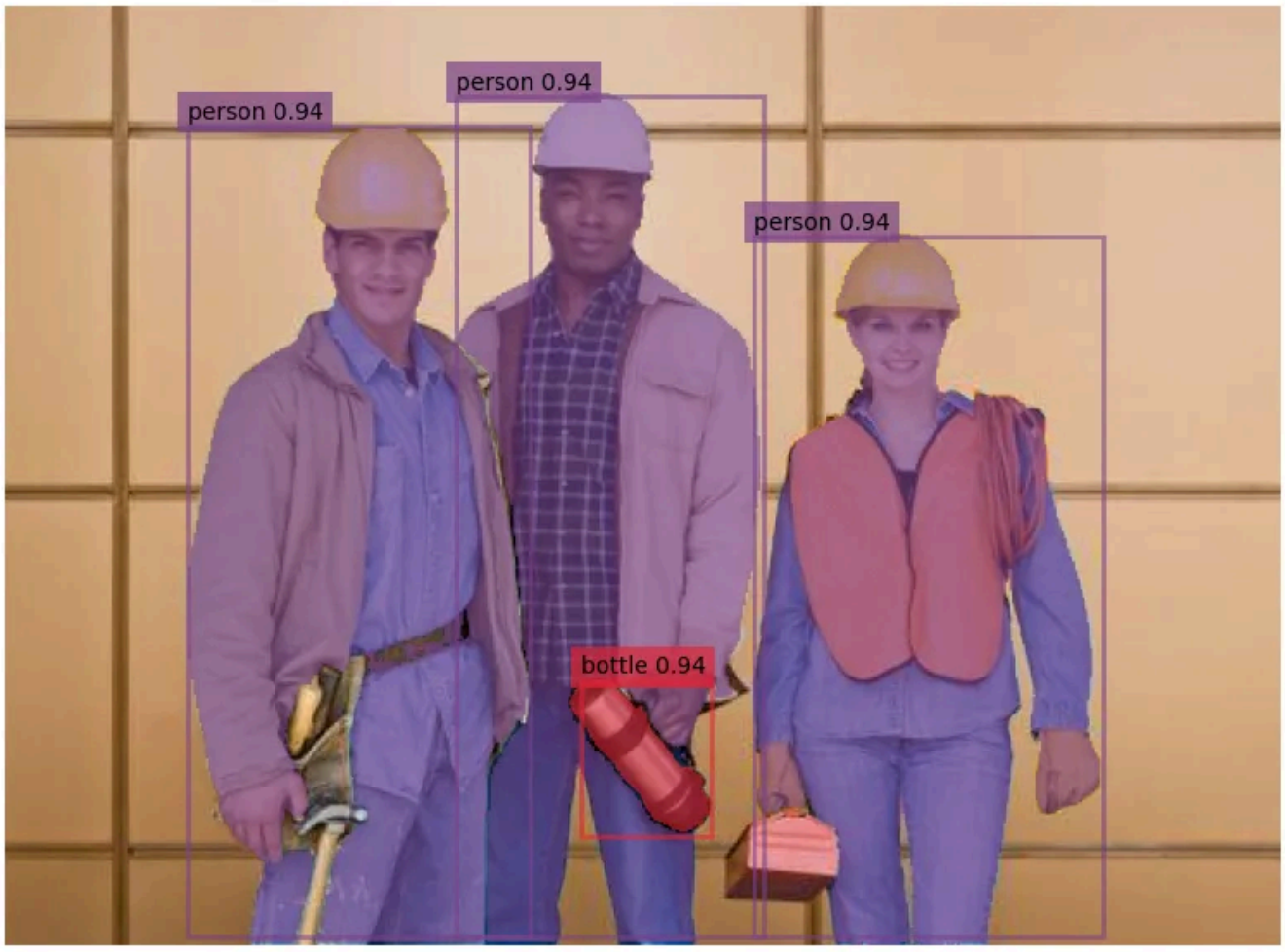
Dynamic Object Detection and Segmentation with YOLOv9+SAM

Sunidhi Ashtekar

9 min read · Mar 14, 2024



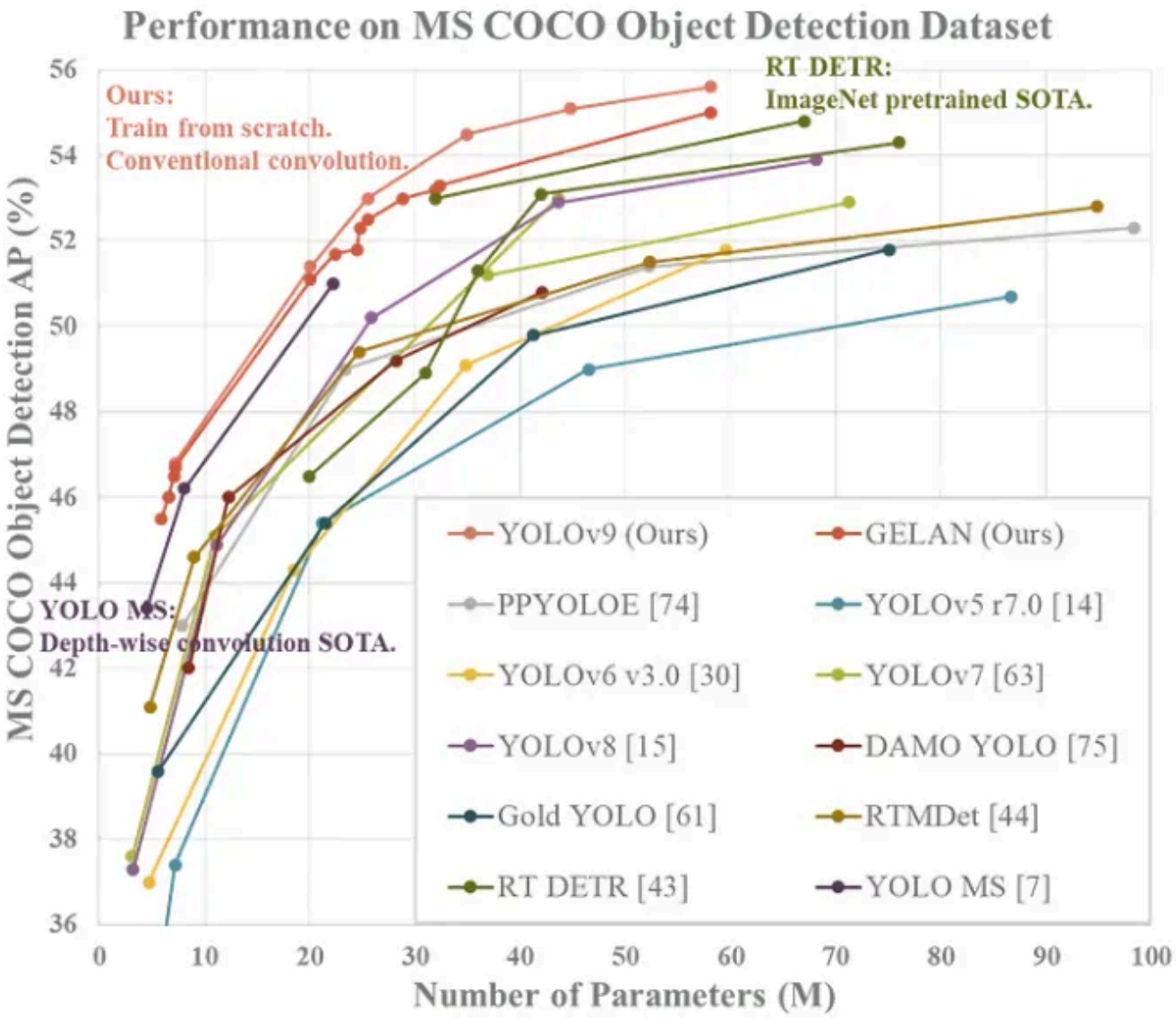
Detection and Segmentation with YOLOv9+SAM

In this article, I have examined a custom object detection model on the RF100 Construction-Safety-2 dataset with YOLOv9+SAM.

This integration not only elevates the accuracy and granularity of detecting and segmenting objects across varied images but also broadens the scope of applications — from enhancing autonomous driving systems to refining diagnostic processes in medical imaging.

By leveraging YOLOv9 for its efficient detection capabilities and SAM for its ability to segment objects in a zero-shot manner, this powerful combination minimizes the need for extensive retraining or data annotation, making it a versatile and scalable solution.

Introduction to YOLOv9 (You Only Look Once)



YOLOv9 Performance from the official YOLOv9 repository.

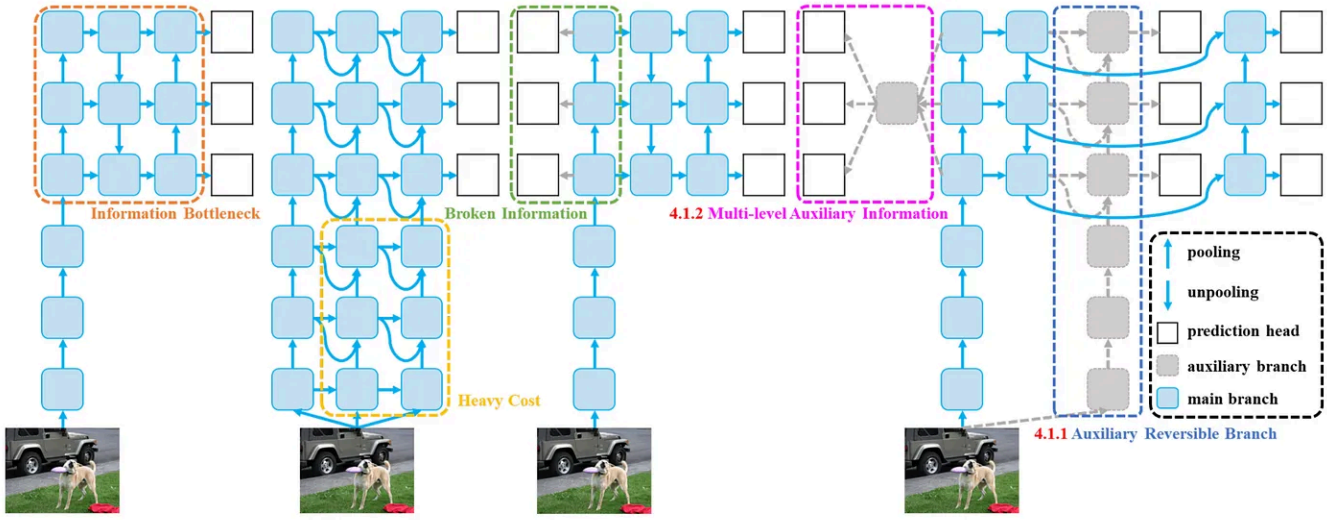


Figure 3. PGI and related network architectures and methods. (a) Path Aggregation Network (PAN)) [37], (b) Reversible Columns (RevCol) [3], (c) conventional deep supervision, and (d) our proposed Programmable Gradient Information (PGI). PGI is mainly composed of three components: (1) main branch: architecture used for inference, (2) auxiliary reversible branch: generate reliable gradients to supply main branch for backward transmission, and (3) multi-level auxiliary information: control main branch learning plannable multi-level of semantic information.

YOLOv9 Model Diagram from official YOLOv9 repository.

YOLOv9 introduces significant advancements in real-time object detection, incorporating Programmable Gradient Information (PGI) and the Generalized Efficient Layer Aggregation Network (GELAN) to enhance efficiency, accuracy, and adaptability, as evidenced by its performance on the MS COCO dataset.

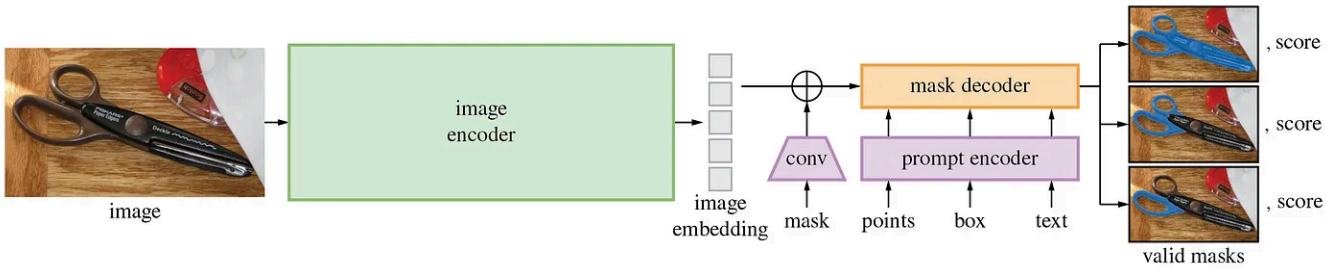
Leveraging the collaborative work of the open-source community and building upon Ultralytics YOLOv5, YOLOv9 addresses the challenge of information loss in deep learning with the Information Bottleneck Principle and Reversible Functions, preserving essential data across layers.

These innovative strategies boost the model's structural efficiency and ensure precise detection capabilities without compromising detail, even in lightweight models.

YOLOv9's architecture reduces unnecessary parameters and computational demands, allowing it to perform optimally across various model sizes, from the compact YOLOv9-S to the more extensive YOLOv9-E, showcasing a harmonious balance between speed and detection accuracy.

As a milestone in computer vision, YOLOv9 not only establishes new benchmarks but also broadens the horizons for AI applications in object detection and segmentation, highlighting the impact of strategic innovation and collaborative efforts in the field.

Introduction to Segment-Anything-Model (SAM)



SAM Model Diagram from official META repository

The Segment Anything Model (SAM) propels computer vision forward by simplifying image segmentation, crucial for a range of uses from scientific studies to creative endeavours.

SAM leverages the Segment Anything 1-Billion (SA-1B) mask dataset, the largest to date, to democratize segmentation by reducing the reliance on specialized expertise, heavy computational power, and extensive dataset annotations.

Under the Apache 2.0 license, SAM introduces a foundational model framework, allowing for easy task adaptation via simple prompts, mirroring advancements seen in natural language processing.

With training on over 1 billion diverse masks, SAM understands a generalized notion of objects, facilitating zero-shot transfer across unfamiliar domains and enhancing its utility across various fields, including AR/VR, creative arts, and scientific research.

The model's prompt-driven flexibility and wide-ranging task adaptability signal a significant leap in segmentation technology, positioning SAM as a versatile and accessible tool for both the research community and application developers.

About the Dataset

The project utilizes the RF100 construction dataset from Roboflow, specifically the Construction-Safety-2 subset, for demonstrating the capabilities of the integrated

models.

RF100 is an initiative sponsored by Intel, aimed at establishing an open-sourced benchmark for object detection models. It focuses on generality across datasets available on Roboflow Universe, promoting accessibility and speeding up the AI and deep learning development process.

The RF100 construction dataset, like other projects in Roboflow, is open-source and can be freely used.

Getting Started

To leverage the combined strengths of YOLOv9 and SAM in your project, these are the steps to follow

- Environment Setup
- Download Pre-trained Model Weights for YOLOv9 and SAM
- Inference on Images
- Visualization and Analysis
- Get Detection Results
- Segmentation with SAM

Environment Setup

A Google account is required to access Google Colab, a free cloud service providing necessary computational resources for deep learning tasks, including access to GPUs and TPUs up to 16 GB.

GPU Status Check

Firstly we ensure the availability and readiness of the GPU for processing and running the YOLOv9+SAM model.

```
[ ]  !nvidia-smi

Thu Mar 14 18:17:39 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA A100-SXM4-40GB          Off | 00000000:00:04.0 Off |                    0 |
| N/A   34C    P0              44W / 400W |      2MiB / 40960MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Mounting Google Drive

Next, we need to navigate to folder where the dataset is stored if you have downloaded the dataset or else we can directly use Roboflow to load the dataset.

```
from google.colab import drive
drive.mount('/content/drive')

or

%cd {HOME}/
!pip install -q roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="YOUR API KEY")
project = rf.workspace("roboflow-100").project("construction-safety-gsnvb")
dataset = project.version(2).download("yolov7")
```

Setting Up YOLOv9

Once the dataset is ready clone the YOLOv9 repository, then switch to the YOLOv9 directory and install the required dependencies to prepare for object detection and segmentation tasks.

```
!git clone https://github.com/SkalskiP/yolov9.git
%cd yolov9
!pip3 install -r requirements.txt -q
```

Display Current Directory

Store current working directory's path in the `HOME` variable for reference.

```
import os
HOME = os.getcwd()
print(HOME)
```

Download Model Weights

Let's create a directory for model weights and download specific YOLOv9 and GELAN model weights from their release pages on GitHub, crucial for initializing the models with pre-trained parameters

```
!mkdir -p {HOME}/weights
!wget -P {HOME}/weights -q https://github.com/WongKinYiu/yolov9/releases/downlc
!wget -P {HOME}/weights -q https://github.com/WongKinYiu/yolov9/releases/downlc
!wget -P {HOME}/weights -q https://github.com/WongKinYiu/yolov9/releases/downlc
!wget -P {HOME}/weights -q https://github.com/WongKinYiu/yolov9/releases/downlc
```

Download an Image for Inference

To obtain inference with YOLOv9 weights we have to set up a data directory and download a sample image for processing and set the path to this image in the `SOURCE_IMAGE_PATH` variable.

```
!mkdir -p {HOME}/data
!wget -P {HOME}/data -q /content/drive/MyDrive/data/image9.jpeg
SOURCE_IMAGE_PATH = f"{HOME}/image9.jpeg"
```

Run Detection with Custom Data

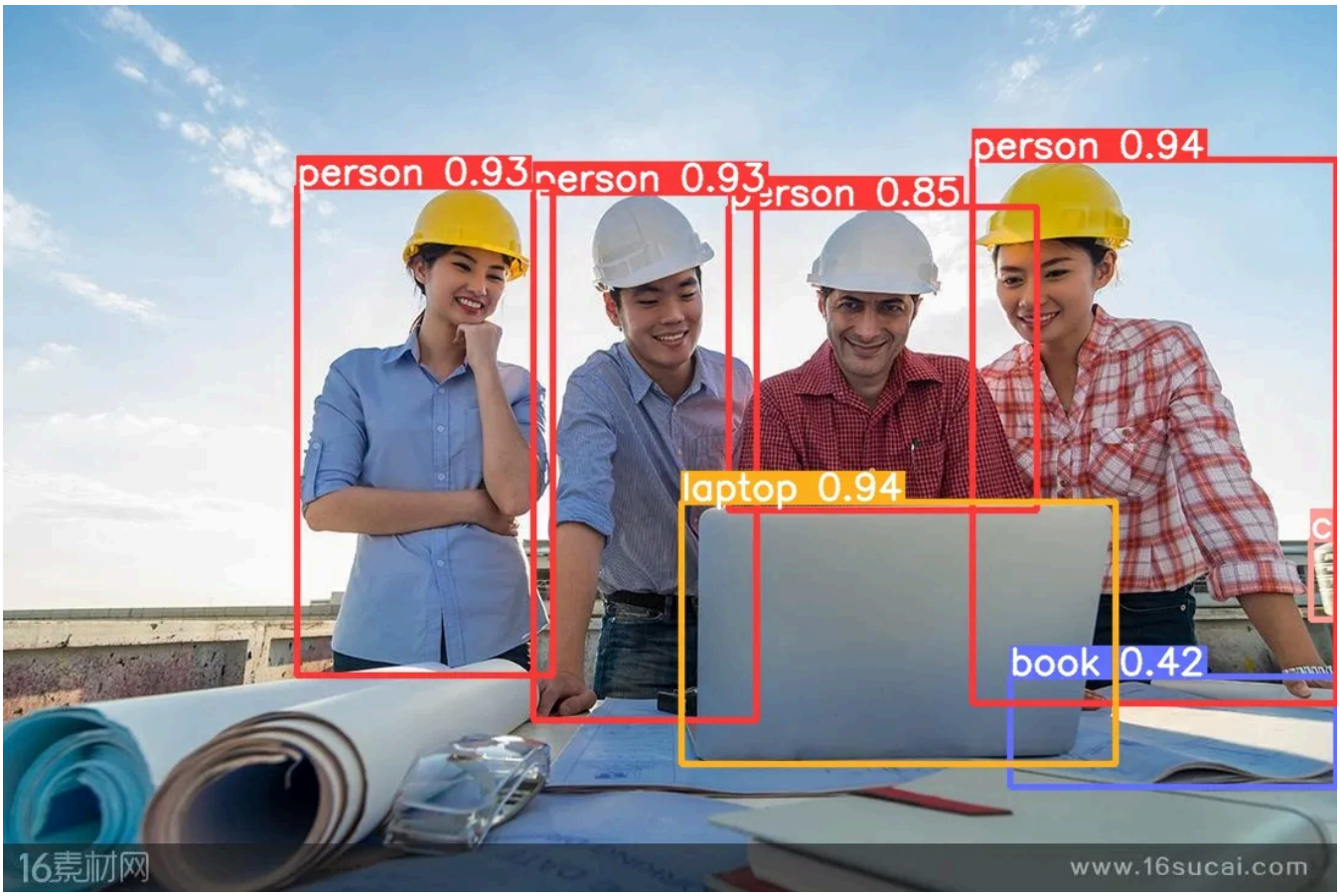
After this, execute the `detect.py` with specified parameters to perform object detection on an image, set a confidence threshold and save the detection results. This will create a text file with class_ids, bounding box co-ordinates and confidence scores that we will utilize later.

```
!python detect.py --weights {HOME}/weights/gelan-c.pt --conf 0.1 --source /cont
```

Display Detection Results

Then we utilize IPython's display and Image functions to showcase the detected objects within the specified path image, adjusting for optimal viewing.

```
from IPython.display import Image, display
Image(filename=f"/content/yolov9/runs/detect/exp/image9.jpeg", width=600)
```



Install Ultralytics

Install the Ultralytics package to access YOLO object detection model implementations and utilities, don't forget to import the YOLO class for object detection tasks.

```
!pip install ultralytics
from ultralytics import YOLO
```

Install Segment-Anything Model

Now let's install the Segment-Anything library and download the weights file for the SAM model and prepare for high-quality image segmentation tasks.

```
!pip install 'git+https://github.com/facebookresearch/segment-anything.git'

!wget https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth
```

Extract Detection Results and Confidence Scores

We will need the YOLOv9 detection results saved in text file above to extract class IDs, confidence scores, and bounding box coordinates. The co-ordinates here are normalized so let's convert them to image scale first and then print them to verify.

```
import cv2

# Specify the path to your image
image_path = '/content/drive/MyDrive/data/image9.jpeg'

# Read the image to get its dimensions
image = cv2.imread(image_path)
image_height, image_width, _ = image.shape

detections_path = '/content/yolov9/runs/detect/exp/labels/image9.txt'

bboxes = []
class_ids = []
conf_scores = []

with open(detections_path, 'r') as file:
    for line in file:
        components = line.split()
        class_id = int(components[0])
        confidence = float(components[5])
        cx, cy, w, h = [float(x) for x in components[1:5]]

        # Convert from normalized [0, 1] to image scale
        cx *= image_width
        cy *= image_height
        w *= image_width
        h *= image_height

        # Convert the center x, y, width, and height to xmin, ymin, xmax, ymax
        xmin = cx - w / 2
        ymin = cy - h / 2
        xmax = cx + w / 2
        ymax = cy + h / 2
```

```
            class_ids.append(class_id)
            bboxes.append((xmin, ymin, xmax, ymax))
            conf_scores.append(confidence)

    # Display the results
    for class_id, bbox, conf in zip(class_ids, bboxes, conf_scores):
        print(f'Class ID: {class_id}, Confidence: {conf:.2f}, BBox coordinates: {bb
```

Initialize SAM for Image Segmentation

Once SAM is initialized with specified pre-trained weights we proceed to select the model type from the SAM model registry for generating segmentation masks.

```
from segment_anything import sam_model_registry, SamAutomaticMaskGenerator, Sam
sam_checkpoint = "/content/yolov9/sam_vit_h_4b8939.pth"
model_type = "vit_h"
sam = sam_model_registry[model_type](checkpoint=sam_checkpoint)
predictor = SamPredictor(sam)
```

Load Image for Segmentation

With OpenCV library we load the image for processing with SAM, prepare it for segmentation.

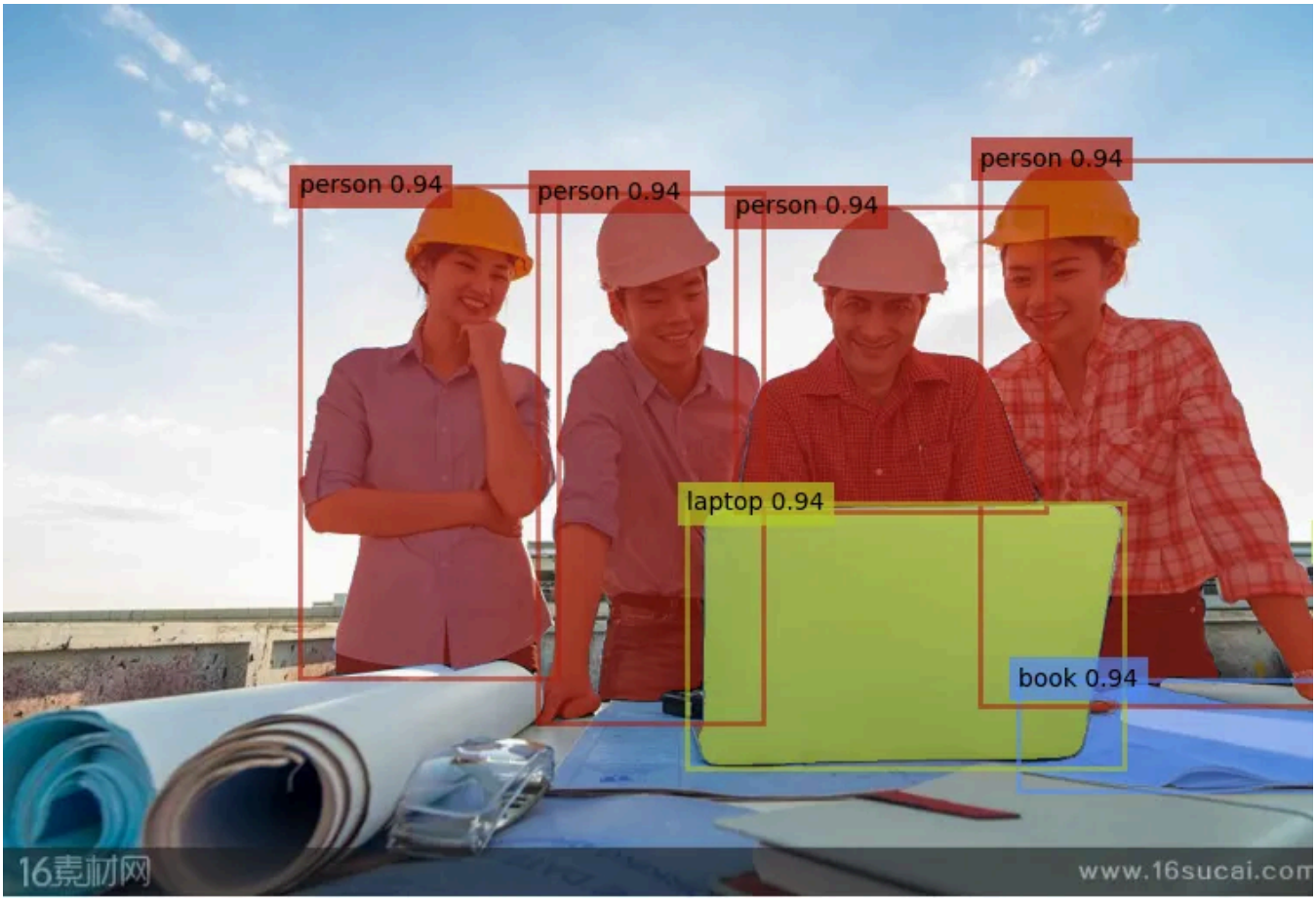
```
import cv2
image = cv2.cvtColor(cv2.imread('/content/drive/MyDrive/data/image9.jpeg'), cv2
predictor.set_image(image)
```

Visualization of the Results

To visualize the detection and segmentation results, we have to convert bounding boxes to segmentation mask using SAM. We assign unique colors to class IDs randomly then define helper functions for displaying masks, confidence scores and bounding boxes. The coco.yaml file used for mapping the class_ids to class names.

```
import matplotlib.patches as patches
from matplotlib import pyplot as plt
import numpy as np
import yaml

with open('/content/yolov9/data/coco.yaml', 'r') as file:
    coco_data = yaml.safe_load(file)
    class_names = coco_data['names']

for class_id, bbox, conf in zip(class_ids, bboxes, conf_scores):
    class_name = class_names[class_id]
    # print(f'Class ID: {class_id}, Class Name: {class_name}, BBox coordinates:

color_map = {}
for class_id in class_ids:
    color_map[class_id] = np.concatenate([np.random.random(3), np.array([0.6])]

def show_mask(mask, ax, color):
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * np.array(color).reshape(1, 1, -1)
    ax.imshow(mask_image)

def show_box(box, label, conf_score, color, ax):
    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    rect = plt.Rectangle((x0, y0), w, h, edgecolor=color, facecolor='none', lw=
    ax.add_patch(rect)

    label_offset = 10

    # Construct the label with the class name and confidence score
    label_text = f'{label} {conf_score:.2f}'

    ax.text(x0, y0 - label_offset, label_text, color='black', fontsize=10, va='
            bbox=dict(facecolor=color, alpha=0.7, edgecolor='none', boxstyle='s

plt.figure(figsize=(10, 10))
ax = plt.gca()
plt.imshow(image)

# Display and process each bounding box with the corresponding mask
for class_id, bbox in zip(class_ids, bboxes):
    class_name = class_names[class_id]
    # print(f'Class ID: {class_id}, Class Name: {class_name}, BBox coordinates:
```

```
        color = color_map[class_id]
        input_box = np.array(bbox)

        # Generate the mask for the current bounding box
        masks, _, _ = predictor.predict(
            point_coords=None,
            point_labels=None,
            box=input_box,
            multimask_output=False,
        )

        show_mask(masks[0], ax, color=color)
        show_box(bbox, class_name, conf, color, ax)

    # Show the final plot
    plt.axis('off')
    plt.show()
```



Extract the Masks

Lastly we generate a composite image that highlights detected objects against a white background, creating an aggregate mask from segmentation masks and applying it to blend the original image with a white background for enhanced visualization.

```
aggregate_mask = np.zeros(image.shape[:2], dtype=np.uint8)

# Generate and accumulate masks for all bounding boxes
for bbox in bboxes:
    input_box = np.array(bbox).reshape(1, 4)
    masks, _, _ = predictor.predict(
        point_coords=None,
        point_labels=None,
        box=input_box,
        multimask_output=False,
    )
    aggregate_mask = np.where(masks[0] > 0.5, 1, aggregate_mask)

# Convert the aggregate segmentation mask to a binary mask
binary_mask = np.where(aggregate_mask == 1, 1, 0)

# Create a white background with the same size as the image
white_background = np.ones_like(image) * 255

# Apply the binary mask to the original image
# Where the binary mask is 0 (background), use white_background; otherwise, use
new_image = white_background * (1 - binary_mask[..., np.newaxis]) + image * bir

# Display the new image with the detections and white background
plt.figure(figsize=(10, 10))
plt.imshow(new_image.astype(np.uint8))
plt.axis('off')
plt.show()
```

Conclusion

I hope this guide has been clear and that you've successfully managed to train your model. By leveraging the RF100 construction dataset from Roboflow, we saw how to use the models to recognize and segment objects with the custom dataset, demonstrating the practical applications of combining YOLOv9 and SAM for advanced computer vision tasks.

If you have any questions, recommendations, or critiques, please don't hesitate to reach out on LinkedIn. I'm open to discussions and eager to hear your feedback or assist with any challenges you might encounter.

References

To further explore the concepts and tools used in this project, I recommend visiting the following resources:

- DynamicDetect YOLOv9+SAM Git Repository
- YOLOv9 Official Repository: https://github.com/WongKinYiu/yolov9
- SAM Official Repository: https://github.com/facebookresearch/segment-anything
- Roboflow Blog: A great resource for the latest updates and tutorials on using Roboflow for computer vision projects. Check out their blog for insightful articles.
- https://blog.roboflow.com/how-to-use-yolov8-with-sam/
- https://blog.roboflow.com/train-yolov9-model/

This project stands as a testament to the potential of combining cutting-edge detection and segmentation technologies to push the boundaries of what's achievable in computer vision. Whether for academic research, industry applications, or personal projects, the integration of YOLOv9 and SAM, supported by the comprehensive RF100 datasets from Roboflow, offers a robust framework for exploring and innovating within the vast domain of computer vision.

Computer Vision | Object Detection | Deep Learning | Machine Learning | Yolov9

Edit profile

Written by Sunidhi Ashtekar

101 Followers

A Passionate Computer Vision & Machine Learning Engineer