COURSE PROJECT

Title: Roar-to-Restore-A-Smart-System-for-Rural-Wild-Animal-Detection

18CSC312J - Artificial Intelligence and Applications in Cloud Computing

(2018 Regulation)

III Year/ VI Semester

Academic Year: 2022 -2023

By

Kundeti Naga Aravind - RA2011028010067

Shayan Hore – RA2011028010073

Under the guidance of

Dr. Vaishnavi Moorthy

Assistant Professor

Department of Networking and Communications

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Kancheepuram

MAY 2023

Table Of Contents:

Problem Statement:

The Real-Time Animal/Person Detection with Computer Vision project aims to solve the problem of accurately detecting and counting the number of animals and people in real-time video streams. This is an important problem in various applications such as wildlife monitoring, security surveillance, and crowd management. Traditional methods of object detection and counting are time-consuming and may not achieve high accuracy. This project uses deep learning and computer vision techniques to provide an accurate and real-time solution to this problem. The project also includes a web-based visualization dashboard to provide a user-friendly interface for monitoring the animal/person count over time.

Introduction:

The Real-Time Animal/Person Detection with Computer Vision project is a computer vision-based solution that uses deep learning to detect and count the number of animals and people in real-time video streams. The system detects and classifies two types of objects: "wild animals" (hands) and "domestic animals or people" (faces). The project is built using Python and Flask, and uses the YOLO object detection algorithm to achieve high accuracy. A visualization dashboard is also included to display the animal/person count over t

Methodology:

The project's implementation can be divided into two main parts: the first is the object detection and counting algorithm, and the second is the web-based visualization dashboard.

Object Detection and Counting Algorithm:

The YOLO (You Only Look Once) algorithm is used in this project to detect and classify the objects in real-time video streams. YOLO is a state-of-the-art object detection algorithm that can detect multiple objects in an image or video with high accuracy and speed. The algorithm works by dividing the image or video into a grid and predicting the object's class and location for each cell in the grid. The final output of the algorithm is a list of bounding boxes that enclose the detected objects and their respective class probabilities.

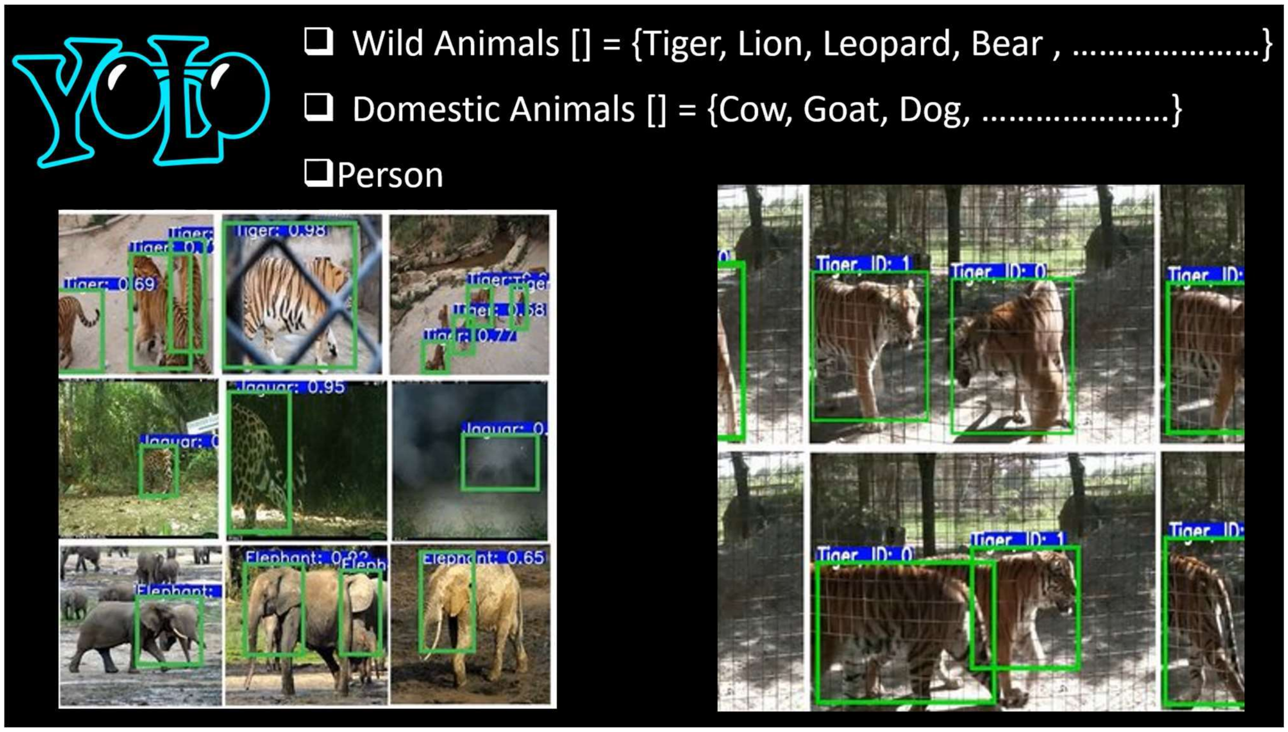
The YOLO algorithm is trained on a dataset of images that contain domestic animals, wild animals, and people. The training data is annotated with bounding boxes that enclose the objects of interest. The YOLO algorithm is trained to recognize these objects and predict their class probabilities accurately.

Web-Based Visualization Dashboard:

The web-based visualization dashboard is built using Flask, a Python web framework. The dashboard displays the animal/person count over time in real-time. The dashboard consists of two main components: the video player and the count graph.

The video player displays the real-time video stream captured by the camera. The object detection algorithm runs in real-time on the video stream, and the detected objects are displayed on the video player as bounding boxes. The count graph displays the animal/person count over time. The count graph updates in real-time as the object detection algorithm detects new objects.

Roar to Restore Protects Villages from Wild Animal Attacks

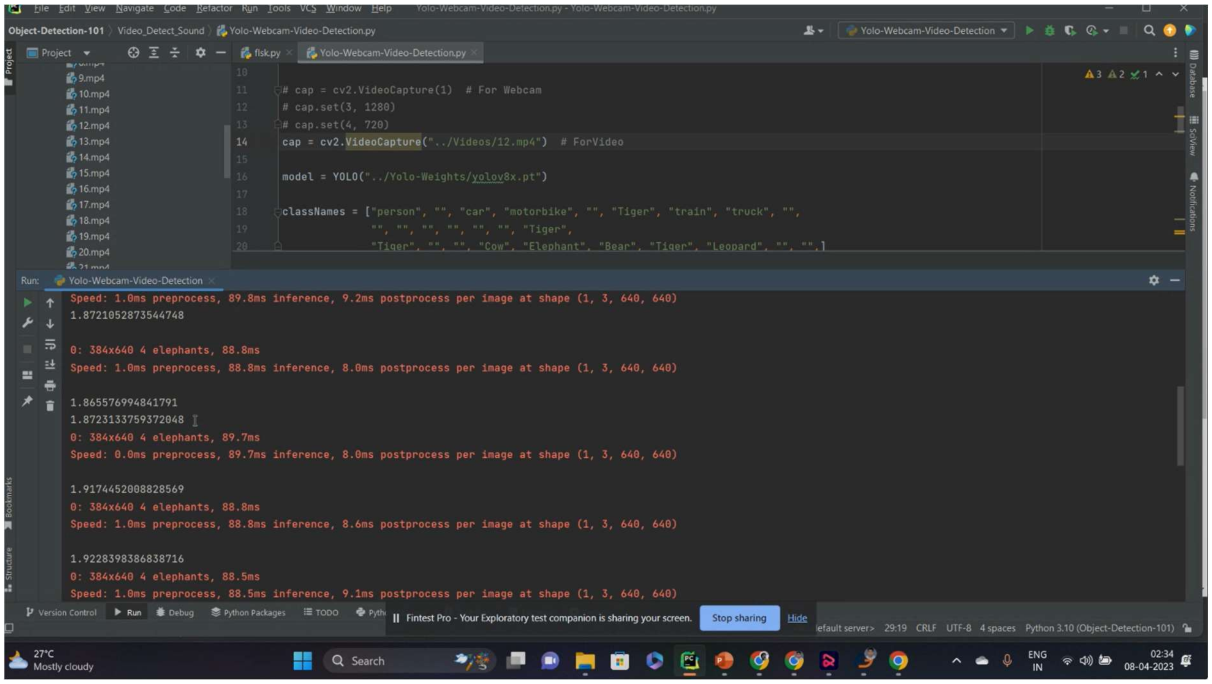

YOLO
❑ Wild Animals [] = {Tiger, Lion, Leopard, Bear , ...}
❑ Domestic Animals [] = {Cow, Goat, Dog, ...}
❑Person

The Roar to Restore system uses a combination of sensors, sound recognition algorithms, and a sound-producing device to detect and alert the presence of wild animals. The sensors detect the movement and heat signatures of animals, and the sound recognition algorithms analyze the sensor data to identify the species of the animal. Once an animal is identified as a potential threat, the sound-producing device emits a loud sound to alert the villagers and drive away the animals.

The system is designed to promote the co-existence of humans and wildlife in rural areas and contribute towards achieving Sustainable Development Goal 15 - Life on Land. By reducing the frequency and severity of wild animal attacks in rural areas, the system promotes human and animal safety, as well as reducing the economic losses associated with crop destruction.

The Roar to Restore project presents an innovative solution for the detection and protection of villages from wild animal incursions. The system can be easily deployed in rural areas and has the potential to be used as an effective tool for wildlife management and protection. By harnessing the power of sound, this system promotes peaceful co-existence and contributes towards achieving Sustainable Development Goal 15.

Roar to Restore in Action



```
# cap = cv2.VideoCapture(1)  # For Webcam
# cap.set(3, 1280)
# cap.set(4, 720)
cap = cv2.VideoCapture("../Videos/12.mp4")  # ForVideo

model = YOLO("../Yolo-Weights/yolov8x.pt")

classNames = ["person", "", "car", "motorbike", "", "Tiger", "train", "truck", "",
              "", "", "", "", "", "", "Tiger",
              "Tiger", "", "", "Cow", "Elephant", "Bear", "Tiger", "Leopard", "", "",]
```

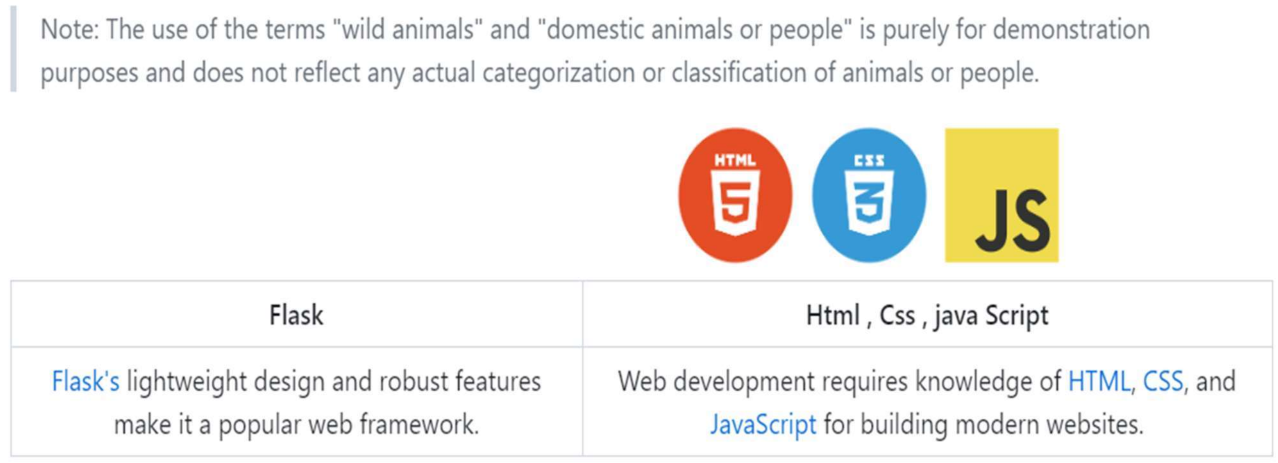
```
Speed: 1.0ms preprocess, 89.8ms inference, 9.2ms postprocess per image at shape (1, 3, 640, 640)
1.8721052873544748

0: 384x640 4 elephants, 88.8ms
Speed: 1.0ms preprocess, 88.8ms inference, 8.0ms postprocess per image at shape (1, 3, 640, 640)

1.865576994841791
1.8723133759372048
0: 384x640 4 elephants, 89.7ms
Speed: 0.0ms preprocess, 89.7ms inference, 8.0ms postprocess per image at shape (1, 3, 640, 640)

1.9174452008828569
0: 384x640 4 elephants, 88.8ms
Speed: 1.0ms preprocess, 88.8ms inference, 8.6ms postprocess per image at shape (1, 3, 640, 640)

1.9228398386838716
0: 384x640 4 elephants, 88.5ms
Speed: 1.0ms preprocess, 88.5ms inference, 9.1ms postprocess per image at shape (1, 3, 640, 640)
```

Object-Detection-101
Video_Detect_Sound
Yolo-Webcam-Video-Detection.py
flsk.py
Run: Yolo-Webcam-Video-Detection
Fintest Pro - Your Exploratory test companion is sharing your screen.
Stop sharing
Hide
27°C Mostly cloudy
02:34 08-04-2023

The Roar to Restore system uses sensors and sound recognition algorithms to detect wild animals and emit a loud sound to alert the villagers. Watch how the system detects and scares off wild animals, while remaining silent when detecting harmless animals such as cows. This innovative solution promotes co-existence of humans and wildlife in rural areas, reducing the frequency and severity of wild animal attacks, and contributing towards achieving Sustainable Development Goal 15 - Life on Land.

Real-Time Animal/Person Detection with Computer Vision

This project uses computer vision to detect and count the number of "wild animals" (hands) and "domestic animals or people" (faces) in real-time video. It's built with Python and Flask, and uses the YOLO object detection algorithm to achieve high accuracy. The project also includes a visualization dashboard to display the animal/person count over time.



> Note: The use of the terms "wild animals" and "domestic animals or people" is purely for demonstration purposes and does not reflect any actual categorization or classification of animals or people.

Flask	Html , Css , java Script
Flask's lightweight design and robust features make it a popular web framework.	Web development requires knowledge of HTML, CSS, and JavaScript for building modern websites.

Documentation

See below for a quickstart installation and usage example, and see the YOLOv8 Docs for full documentation on training, validation, prediction and deployment.

Install

Pip install the ultralytics package including all requirements in a **Python>=3.7** environment with **PyTorch>=1.7**.

```
pip install ultralytics
```

Usage

CLI

YOLOv8 may be used directly in the Command Line Interface (CLI) with a `yolo` command:

```
yolo predict model=yolov8n.pt source='https://ultralytics.com/images/bus.jpg'
```

`yolo` can be used for a variety of tasks and modes and accepts additional arguments, i.e. `imgsz=640`. See the YOLOv8 CLI Docs for examples.

Python

YOLOv8 may also be used directly in a Python environment, and accepts the same arguments as in the CLI example above:

```
from ultralytics import YOLO

# Load a model
```

```
model = YOLO("yolov8n.yaml")  # build a new model from scratch
model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)

# Use the model
model.train(data="coco128.yaml", epochs=3)  # train the model
metrics = model.val()  # evaluate model performance on the validation set
results = model("https://ultralytics.com/images/bus.jpg")  # predict on an image
success = model.export(format="onnx")  # export the model to ONNX format
```
Models download automatically from the latest Ultralytics release. See YOLOv8 Python Docs for more examples.

Models

All YOLOv8 pretrained models are available here. Detect, Segment and Pose models are pretrained on the COCO dataset, while Classify models are pretrained on the ImageNet dataset.

Models download automatically from the latest Ultralytics release on first use.

Detection

See Detection Docs for usage examples with these models.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset. Reproduce by `yolo val detect data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance. Reproduce by `yolo val detect data=coco128.yaml batch=1 device=0|cpu`

Segmentation

See Segmentation Docs for usage examples with these models.

Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-seg	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s-seg	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m-seg	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l-seg	640	52.3	42.6	572.4	2.79	46.0	220.5
YOLOv8x-seg	640	53.4	43.4	712.1	4.02	71.8	344.1

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset.
  Reproduce by `yolo val segment data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo val segment data=coco128-seg.yaml batch=1 device=0|cpu`

Classification

See Classification Docs for usage examples with these models.

Model	size (pixels)	acc top1	acc top5	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B) at 640
YOLOv8n-cls	224	66.6	87.0	12.9	0.31	2.7	4.3
YOLOv8s-cls	224	72.3	91.1	23.4	0.35	6.4	13.5
YOLOv8m-cls	224	76.4	93.2	85.4	0.62	17.0	42.7
YOLOv8l-cls	224	78.0	94.1	163.0	0.87	37.5	99.7
YOLOv8x-cls	224	78.4	94.3	232.0	1.01	57.4	154.8

- **acc** values are model accuracies on the ImageNet dataset validation set.
  Reproduce by `yolo val classify data=path/to/ImageNet device=0`
- **Speed** averaged over ImageNet val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo val classify data=path/to/ImageNet batch=1 device=0|cpu`

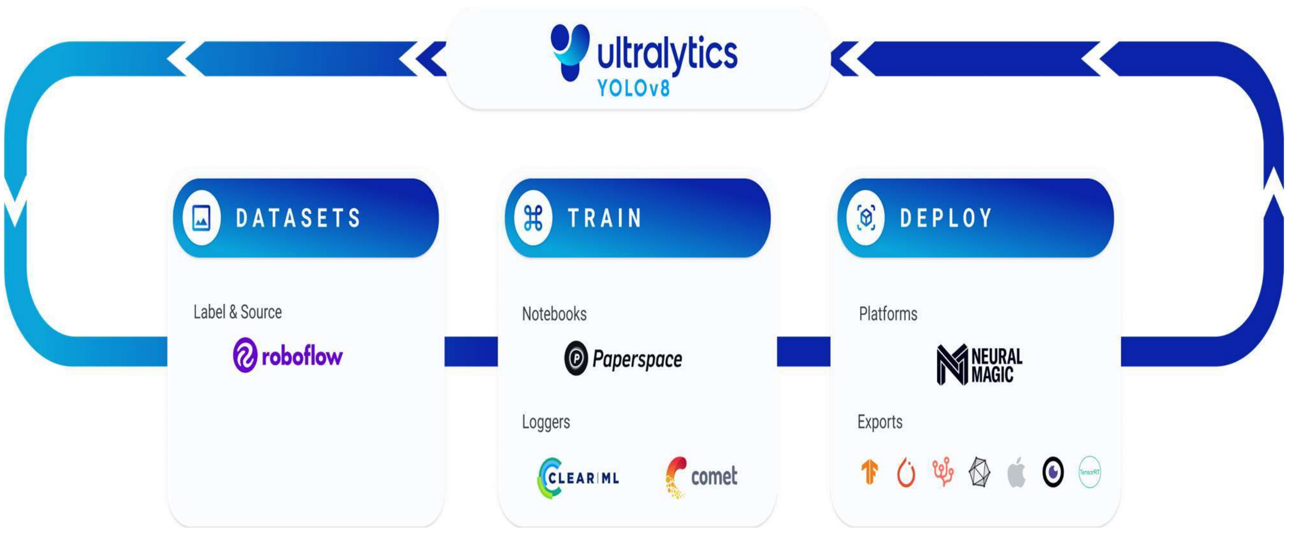
Pose

See Pose Docs for usage examples with these models.

Model	size (pixels)	mAP^box 50-95	mAP^pose 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-pose	640	-	49.7	131.8	1.18	3.3	9.2
YOLOv8s-pose	640	-	59.2	233.2	1.42	11.6	30.2
YOLOv8m-pose	640	-	63.6	456.3	2.00	26.4	81.0
YOLOv8l-pose	640	-	67.0	784.5	2.59	44.4	168.6
YOLOv8x-pose	640	-	68.9	1607.1	3.73	69.4	263.2
YOLOv8x-pose-p6	1280	-	71.5	4088.7	10.04	99.1	1066.4

- **mAP^val** values are for single-model single-scale on COCO Keypoints val2017 dataset.
  Reproduce by `yolo val pose data=coco-pose.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo val pose data=coco8-pose.yaml batch=1 device=0|cpu`

Integrations

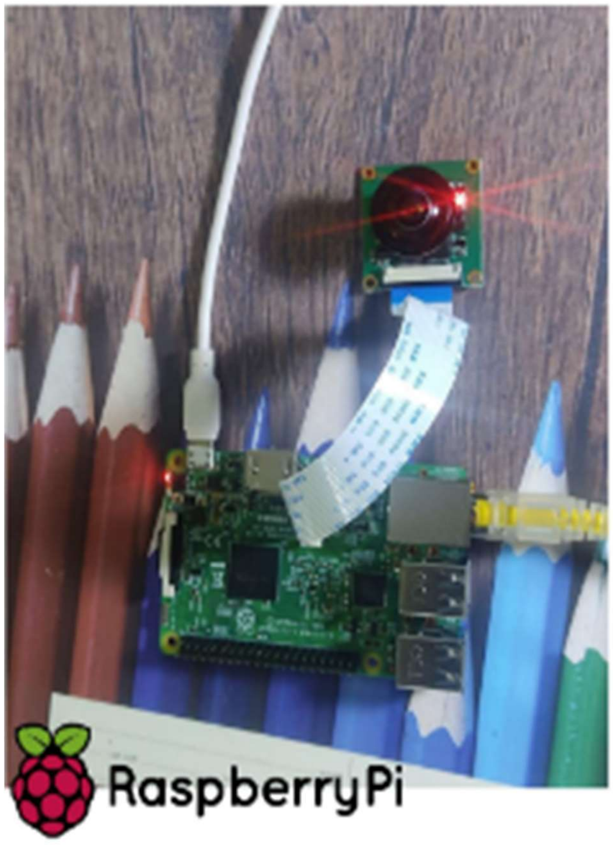


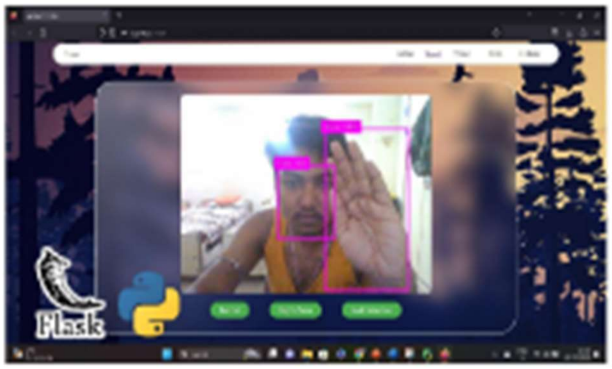
Roboflow	**ClearML ⭐ NEW**	**Comet ⭐ NEW**	**Neural Magic ⭐ NEW**
Label and export your custom datasets directly to YOLOv8 for training with Roboflow	Automatically track, visualize and even remotely train YOLOv8 using ClearML (open-source!)	Free forever, Comet lets you save YOLOv8 models, resume training, and interactively visualize and debug predictions	Run YOLOv8 inference up to 6x faster with Neural Magic DeepSparse

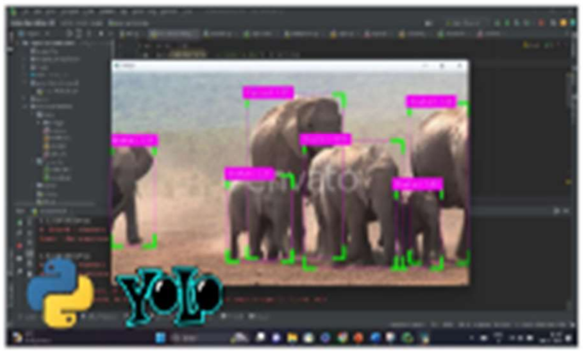
Ultralytics HUB

Experience seamless AI with Ultralytics HUB ⭐, the all-in-one solution for data visualization, YOLOv5 and YOLOv8 (coming soon) 🚀 model training and deployment, without any coding. Transform images into actionable insights and bring your AI visions to life with ease using our cutting-edge platform and user-friendly Ultralytics App. Start your journey for **Free** now!

Demo of the Project







Conclusion

This project is a great example of how computer vision can be used to solve real-world problems. By detecting and counting the number of "wild animals" (hands) and "domestic animals or people" (faces) in real-time video, this system could be used to improve security in public spaces or monitor the number of people in a building.

Team Members:

KUNDETI NAGA ARAVIND	- RA2011028010067
SHAYAN HORE	- RA2011028010073