Volume-09 Issue 03, March-2025 ISSN: 2456-9348
Impact Factor: 8.232

International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

REAL-TIME OBJECT DETECTION AND TRACKING USING YOLOV8 AND OPENCV

Dr. AMIT GUPTA
JBIET College, Hyderabad

MOHAMMAD TOUFEEQ
SHROFF FAWAZ PASHA
G. HARSHA VARDAN
UG Student, Department of Artificial Intelligence and Data Science,
JBIET College Hyderabad

ABSTRACT
This project develops a real-time object detection and tracking system using YOLOv8. It processes live video from a webcam or external camera, detecting multiple objects dynamically. YOLOv8's efficient architecture ensures high-speed and accurate object classification. Detected objects are displayed with bounding boxes, labels, and confidence scores. OpenCV enhances video frame processing for minimal latency and smooth visualization. A custom color scheme differentiates object classes for improved visual clarity. The system supports applications in surveillance, traffic monitoring, and automation. Its real-time performance makes it ideal for safety-critical AI applications. Dynamic tracking ensures continuous updates for every frame in the video feed. This project showcases a robust and efficient solution for modern object detection tasks.

Keywords:
Object Dection , AI, CNN, Web Application, Crop Health,
Diagnosis, YOLOv8,Dection Mobile Support.

INTRODUCTION

Object detection and tracking are crucial for AI applications, enabling real-time visual analysis. This project utilizes YOLOv8, a deep learning model known for its speed and accuracy. Live video feeds are processed using OpenCV to capture and visualize detection results.YOLOv8 detects multiple objects, classifies them, and tracks movements with minimal latency. Bounding boxes, class labels, and confidence scores provide clear visual representation. Advanced computer vision and deep learning enhance detection accuracy and tracking. The system is applicable in surveillance, autonomous vehicles, and industrial automation. It addresses challenges like dynamic tracking, fast inference, and environmental adaptability. The project demonstrates the potential of modern AI for real-time detection tasks. Further sections discuss objectives, methodology, implementation, and applications. Agricultural practices.

OBJECTIVES

The objective of this project is to develop a **real-time object detection and tracking system** using the **YOLOv8** deep learning framework. The system processes live video input from a **webcam or external camera** to detect and classify multiple objects with **high accuracy and speed**. By integrating **OpenCV**, the system efficiently processes video frames and dynamically updates **bounding boxes, class labels, and confidence scores** for real-time visualization. A **custom color scheme** enhances object differentiation for better clarity. This project aims to provide a **robust and efficient solution** for applications in **surveillance, traffic monitoring, autonomous systems, and safety-critical environments**, demonstrating the effectiveness of **modern AI in real-time object detection and tracking**.

METHODOLOGY

- **Data Acquisition & Preprocessing**
  - Collect live video input from a **webcam or external camera**.
  - Perform **frame extraction** and preprocessing (resizing, normalization) for optimal model input.
- **YOLOv8 Model Implementation**
  - Utilize the **YOLOv8 deep learning model** for real-time object detection.
  - Load pre-trained weights or fine-tune the model on a custom dataset for specific object categories.
- **Object Detection & Tracking**
  - Detect multiple objects within each video frame and classify them into **predefined categories**.
  - Generate **bounding boxes, class labels, and confidence scores**, dynamically updating them in real time.

IJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

- **Integration with OpenCV**
  - Use **OpenCV** for efficient **video frame processing** and visualization.
  - Optimize performance to ensure minimal latency and smooth real-time tracking.
- **Custom Visualization Enhancements**
  - Implement a **custom color scheme** to differentiate object classes for improved clarity.
  - Overlay bounding boxes, labels, and confidence scores directly on the video feed..
- **Application & Deployment**
  - Test the system in different environments such as **surveillance, traffic monitoring, and automation**.
  - Deploy the solution on **edge devices, cloud platforms, or local machines** for real-

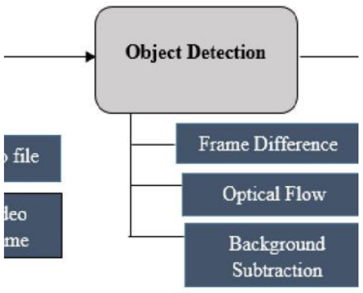


Fig. 1 Architecture

Figure 1 illustrates the architecture of the proposed system, which involves multiple stages from data collection to deployment. The process begins with defining the problem statement, followed by data collection and preprocessing. The cleaned data is used for model training using a Convolutional Neural Network (CNN). Once the model is trained, it is exported and converted into a TensorFlow Lite (TFLite) model for efficient deployment.The ability of Xception to extract intricate hierarchical information from images is one of its key advantages. Its deep network architecture and depthwise separable convolutions improve classification performance by simultaneously extracting low-level and high-level features. This is particularly crucial for plant disease identification, where distinguishing between similar disease symptoms requires precise feature extraction. The Xception architecture is derived from Inception-V3, but with modifications in the inception blocks that allow for greater efficiency and accuracy.
For accessibility, the system includes both a web application built using React.js and a mobile application developed with React Native. The trained model is deployed to Google Cloud Platform (GCP) using Google Cloud Functions and is accessible through FastAPI. This setup ensures seamless interaction between the frontend applications and the backend AI model.In conclusion, a thorough analysis of deep learning-based image classification techniques was conducted for model selection. Considering our specific research objectives, Xception CNN—with its innovative depthwise separable convolutions—outperformed other architectures in terms of accuracy and computational efficiency.
Figure 2 illustrates the block diagram of the grape leaf disease detection system. The process begins with the Grape Leaf Dataset, which contains images of healthy and diseased grape leaves. These images undergo Data Pre-processing, which includes noise removal, normalization, and resizing to ensure consistency before model training.
A Pre-trained Model (EfficientNet B7) is used for Transfer Learning, where the model is fine-tuned on the grape leaf dataset to enhance its feature extraction capabilities. The extracted features pass through a Fully Connected Layer, which helps in classification by learning the most relevant patterns.
To improve the robustness of the model, Variance Control is applied to handle variations in leaf color, texture, and lighting conditions. The refined Feature Vector is then mapped, and the final step is Disease Detection, where the model classifies whether a grape leaf is healthy or diseased.

iJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

This approach leverages the power of EfficientNet B7, known for its efficiency and high accuracy, making it a

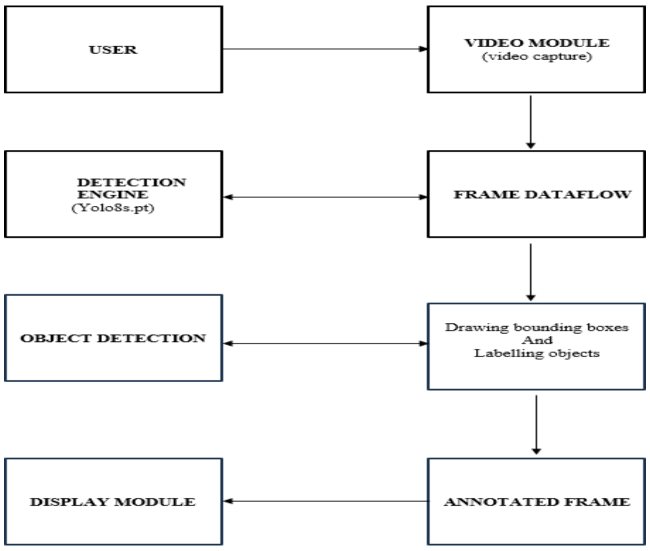

USER
VIDEO MODULE (video capture)
DETECTION ENGINE (Yolo8s.pt)
FRAME DATAFLOW
OBJECT DETECTION
Drawing bounding boxes And Labelling objects
DISPLAY MODULE
ANNOTATED FRAME

Fig.2 Block Diagram

A. MODEL:

The **Xception (Extreme Inception) model**, introduced by **François Chollet in 2017**, represents a paradigm shift in **Convolutional Neural Network (CNN) design**, specifically aiming to enhance **efficiency without compromising accuracy**. The key innovation of **Xception** is the integration of **depthwise separable convolutions** throughout the network, which significantly reduces the number of parameters and computations while maintaining high performance.

CLASS DIAGRAM

The Class Diagram provides a structured view of the object detection system, showing the relationships between different components. The core class in the system is YOLO, which loads the pre-trained detection model and performs real-time object tracking. It interacts with VideoCapture, which captures live video frames using OpenCV. The detected objects are stored in the BoundingBox class, which contains attributes such as coordinates and confidence scores. The ObjectClassifier assigns class labels to detected objects based on the YOLO model's predictions.For visualization, the system uses ColorManager, which assigns different colors to object classes, and DisplayManager, which overlays bounding boxes and labels onto video frames. The SystemController handles the overall process, managing video input, detection, tracking, and user interactions. When the user presses 'q', the system stops processing and exits gracefully. The Class Diagram helps developers understand the system's object- oriented structure, making it easier to modify or extend functionalities.These UML diagrams provide a clear visualization of the system's behavior (Use Case Diagram) and structure (Class Diagram), ensuring an organized approach to software development. Let me know if you need further refinements

Example: usecase of vechiles tracking

Volume-09 Issue 03, March-2025 ISSN: 2456-9348
Impact Factor: 8.232

IJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

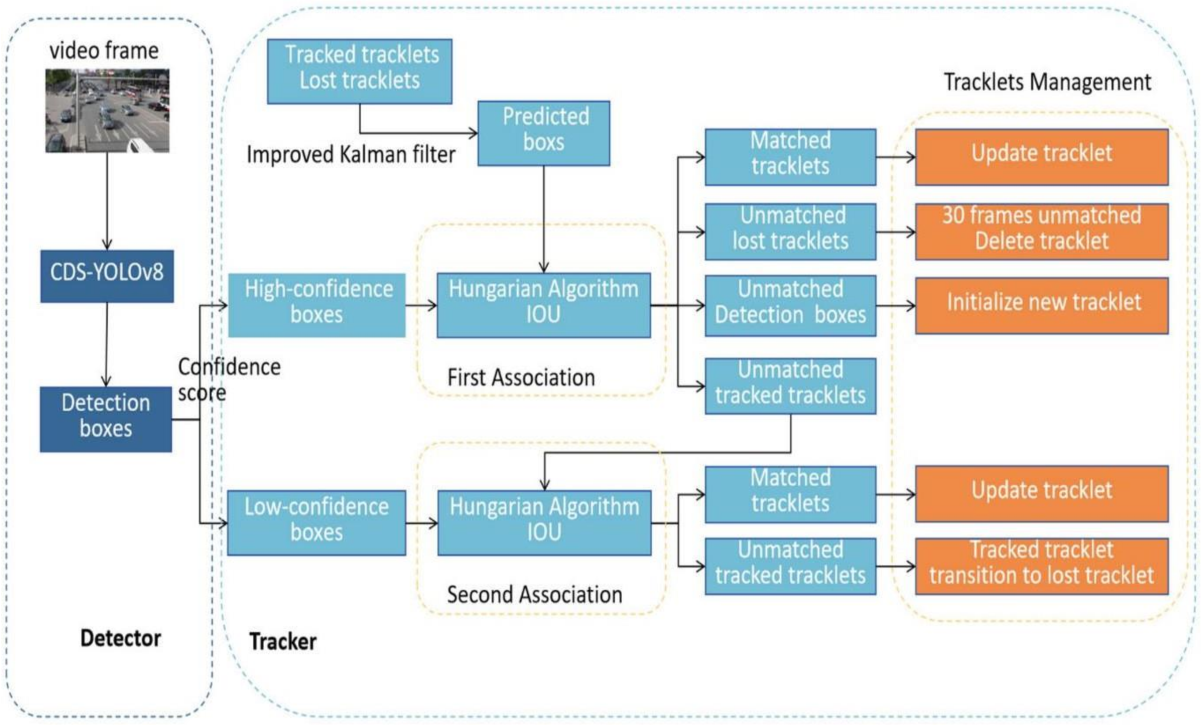
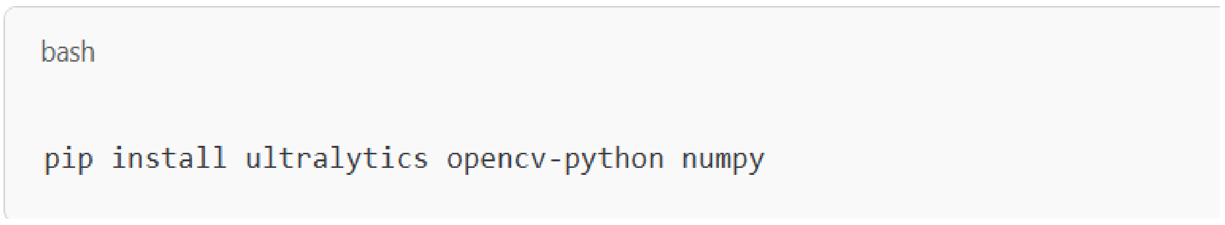


Fig:3 Class Diagram

INTRODUCTION
The implementation of an Object Detection System using YOLOv8 and OpenCV focuses on real-time object recognition in video streams. YOLOv8 provides high-speed, accurate detection, while OpenCV handles video capture and visualization. The system processes frames, detects objects, and displays bounding boxes with class names and confidence scores. This implementation is applicable in areas like smart surveillance, autonomous systems, and industrial automation. The following sections outline the setup, model integration, and real- time processing for an efficient detection system.

6.1.1 Importing Modules:
System Setup:
Using pip install ultralytics opencv-python numpy

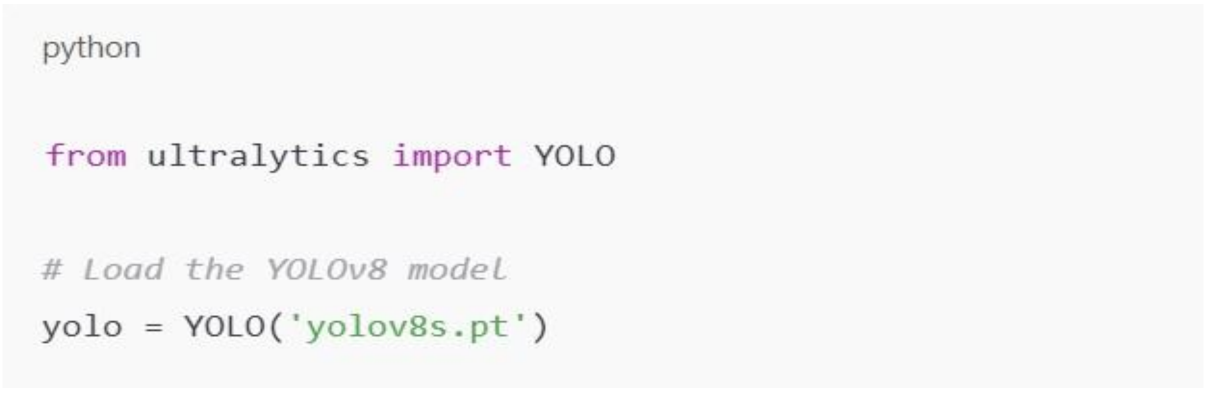


```bash
pip install ultralytics opencv-python numpy
```

Load YOLO Model:

iJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

YOLOv8 model is loaded using the Ultralytics library

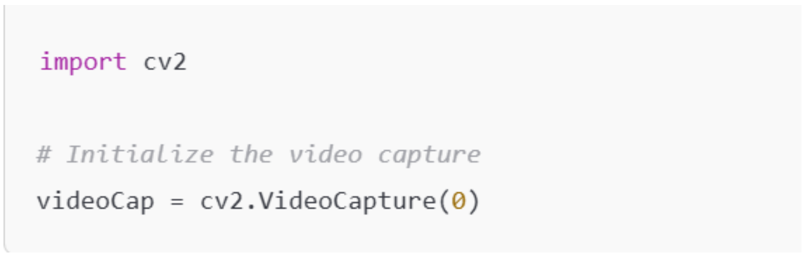


```
python

from ultralytics import YOLO

# Load the YOLOv8 model
yolo = YOLO('yolov8s.pt')
```

Import cv2 cam:
A real-time video stream is captured using **OpenCV**. The frames are processed one by one.



```
import cv2

# Initialize the video capture
videoCap = cv2.VideoCapture(0)
```

IJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

6.1.2 CODE

```
import cv2
from ultralytics import YOLO # Load the
model
yolo = YOLO('yolov8s.pt') # Load the
video capture
videoCap = cv2.VideoCapture(0) # Function
to get class colors
def getColours(cls_num):
    base_colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255)]
    color_index = cls_num % len(base_colors) increments = [(1, -2, 1),
    (-2, 1, -1), (1, -1, 2)]
    color = [base_colors[color_index][i] + increments[color_index][i] * (cls_num //
    len(base_colors)) % 256 for i in range(3)]
    return tuple(color) while
True:
    ret, frame = videoCap.read() if not ret:
        continue
    results = yolo.track(frame, stream=True) for result in
    results:
        # get the classes names classes_names =
        result.names # iterate over each box
        for box in result.boxes:
            # check if confidence is greater than 40 percent if box.conf[0]
            > 0.4:
                # get coordinates
                [x1, y1, x2, y2] = box.xyxy[0] # convert
                to int
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2) # get the class
                cls = int(box.cls[0]) # get the
                class name
                class_name = classes_names[cls] # get the
                respective colour  colour = getColours(cls)
                # draw the rectangle
                cv2.rectangle(frame, (x1, y1), (x2, y2), colour, 2) # put the class
                name and confidence on the image
                    cv2.putText(frame, f'{classes_names[int(box.cls[0])]} {box.conf[0]:.2f}', (x1, y1),
     cv2.FONT_HERSHEY_SIMPLEX, 1, colour, 2)
    # show the image cv2.imshow('frame',
    frame)
    # break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'): break
# release the video capture and destroy all windows videoCap.release()
cv2.destroyAllWindows()
```

iJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

7. OUTPUT SCREEN

FIG:7.1

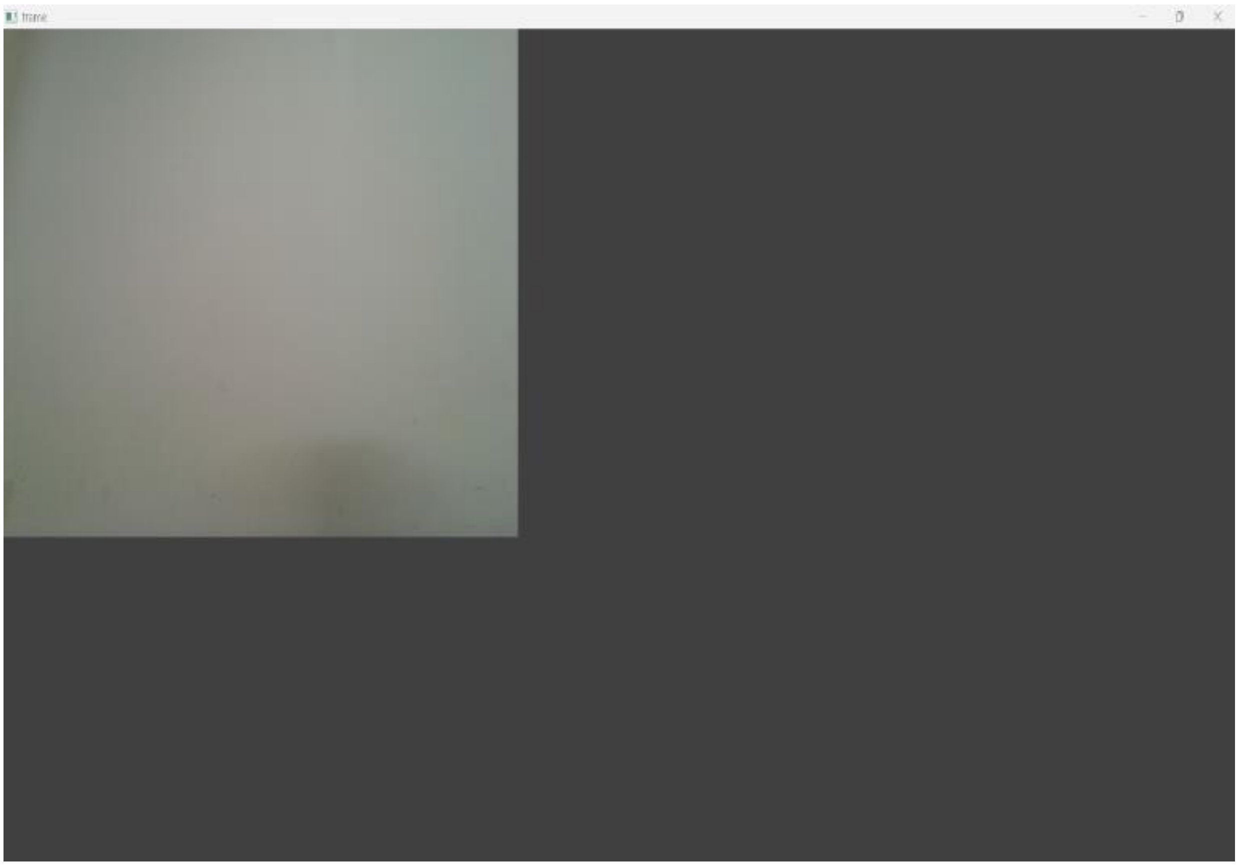
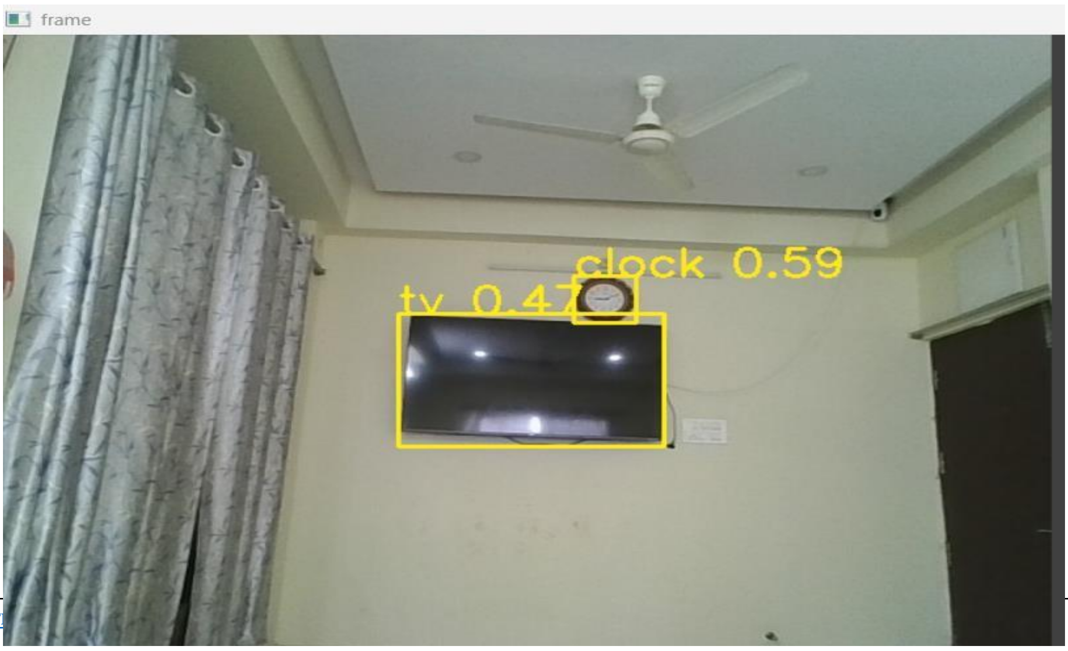


FIG:7.2



iJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

FIG:7.3

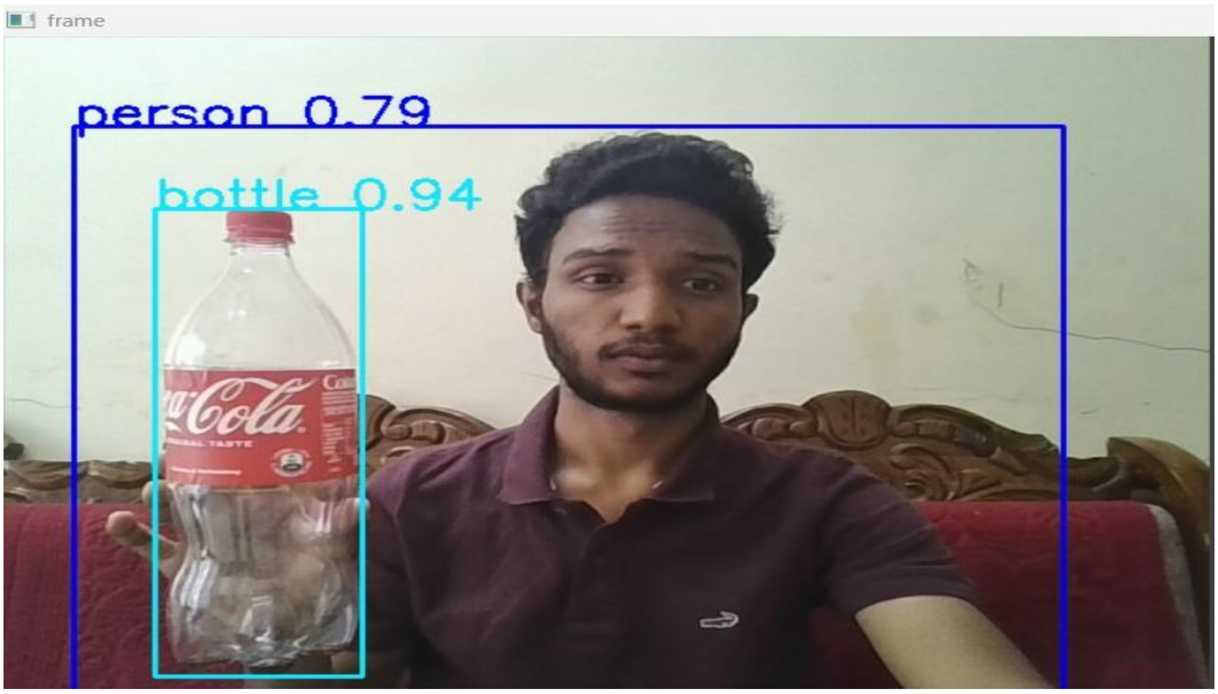
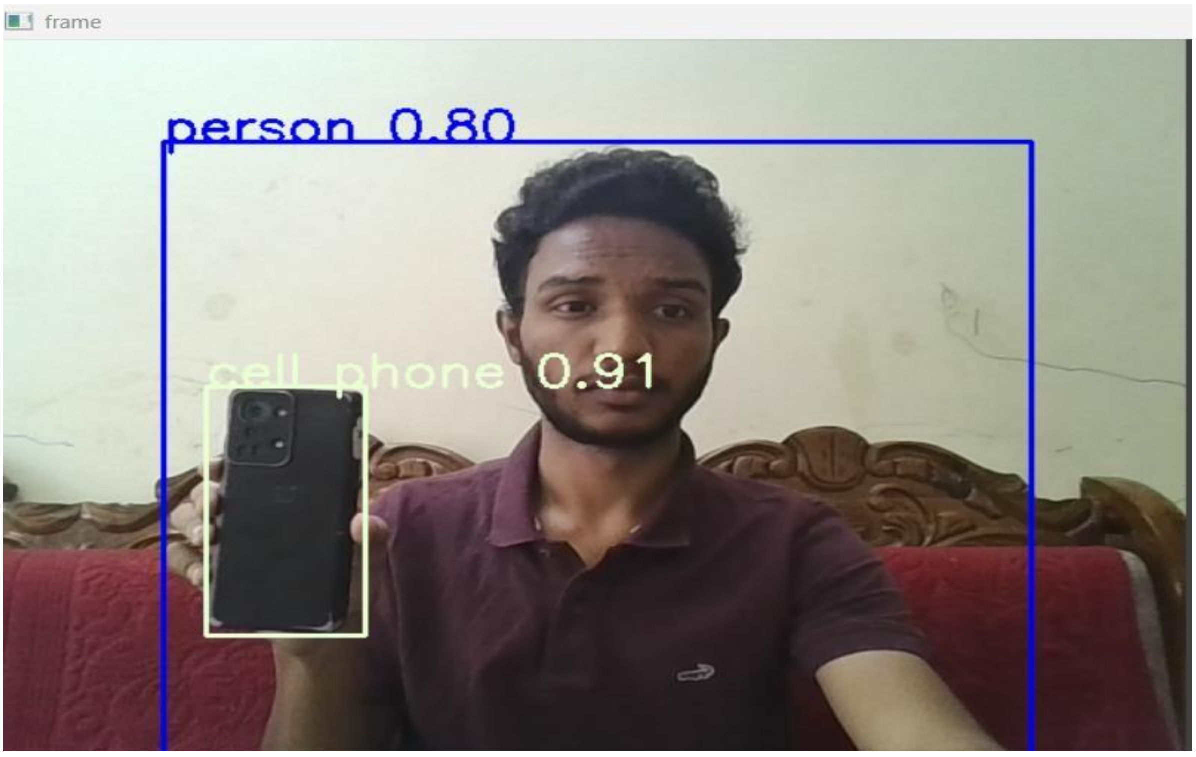


FIG:7.4



iJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

CONCLUSION

The development and implementation of the Object Detection System using YOLOv8 and OpenCV demonstrate the effectiveness of deep learning in real-time object recognition. This project successfully captures video frames, processes them using the YOLO model, and accurately identifies objects within the scene. By leveraging computer vision techniques and neural networks, the system achieves high-speed detection with minimal computational overhead.

One of the key strengths of this system is its ability to provide real-time insights through video analysis. The integration of bounding boxes, class labels, and confidence scores enhances its usability across various applications such as security surveillance, autonomous vehicles, retail analytics, and industrial automation. The use of OpenCV for image processing and YOLO for deep learning-based detection ensures a robust and scalable solution. Additionally, the modular approach to development makes it easy to extend the system with additional features, such as multi-camera support, object tracking, and cloud integration. The project's implementation highlights the potential of AI-powered solutions in transforming traditional object detection methods, offering greater accuracy, efficiency, and automation.

Overall, this object detection system serves as a strong foundation for future advancements in computer vision. It underscores the importance of deep learning models in practical applications, providing a framework that can be optimized for various industries. The results demonstrate that real-time object detection is not only feasible but also highly impactful in solving real-world challenges.

FUTURE SCOPE

The future scope of the Object Detection System is vast, with numerous advancements that can enhance its efficiency, accuracy, and real-world applicability. One of the key areas of improvement is the enhancement of model performance by integrating more advanced deep learning architectures such as EfficientDet or Vision Transformers (ViTs). Fine-tuning the system with custom datasets will further improve accuracy in domain-specific applications. Additionally, incorporating real-time object tracking through algorithms like SORT and DeepSORT will enable seamless object identification across multiple frames. Multi-camera integration can also be explored for broader surveillance and monitoring applications.

Another significant advancement lies in deploying the system on edge computing devices like Raspberry Pi and NVIDIA Jetson Nano, reducing dependence on high-end hardware while improving real-time processing. Cloud-based solutions using platforms like AWS, Google Cloud, or Azure can facilitate large-scale object detection, enabling remote accessibility and analytics. Furthermore, the integration of the Internet of Things (IoT) can enhance automation, where detected objects trigger smart responses such as security alarms or automated vehicle navigation. The application of 5G and AI-powered IoT will allow faster data transmission and improved real-time monitoring in sectors such as healthcare, smart cities, and intelligent transportation.

Beyond traditional object detection, this system can be extended to autonomous vehicles and robotics, enabling self-driving cars and drones to detect and navigate around obstacles. Pose estimation and gesture recognition can also be implemented for human-robot interaction and advanced surveillance. Moreover, incorporating big data analytics will allow predictive insights from detection patterns, which can be useful in traffic management, supply chain monitoring, and retail analytics. The combination of Augmented Reality (AR) and Mixed Reality (MR) can further revolutionize applications by overlaying real-time object information, benefiting industries like gaming, education, and virtual shopping.

In conclusion, the future of object detection extends beyond simple recognition to intelligent automation, decision-making, and real-time analytics. With the integration of AI, IoT, cloud computing, and edge computing, the system can be widely adopted in industries such as security, healthcare, retail, and autonomous systems. The continuous evolution of deep learning models and technological innovations will ensure that object detection remains a crucial component of smart, efficient, and data-driven solutions for real-world applications.

Volume-09 Issue 03, March-2025

ISSN: 2456-9348
Impact Factor: 8.232

IJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

B. BLOCKCHAIN:

.

REFERENCES

- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016) – *You Only Look Once: Unified, Real-Time Object Detection*, IEEE CVPR.
- Redmon, J., & Farhadi, A. (2018) – *YOLOv3: An Incremental Improvement*, arXiv Preprint.
- Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020) – *YOLOv4: Optimal Speed and Accuracy of Object Detection*, arXiv Preprint.
- Jocher, G., et al. (2023) – *YOLOv8: Ultralytics Documentation & Model Implementation*, Ultralytics Research.
- Ren, S., He, K., Girshick, R., & Sun, J. (2015) – *Faster R-CNN: Towards Real- Time Object Detection with Region Proposal Networks*, NeurIPS.
- Bradski, G. (2000) – *The OpenCV Library*, Dr. Dobb's Journal of Software Tools.
- Paszke, A., et al. (2019) – *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, NeurIPS.
- Wojke, N., Bewley, A., & Paulus, D. (2017) – *Simple Online and Realtime Tracker (SORT) and DeepSORT: A Deep Learning-Based Multi-Object Tracker*, IEEE Transactions on Pattern Analysis and Machine Intelligence.
- Howard, A. G., et al. (2017) – *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, IEEE Transactions on Neural Networks and Learning Systems.
- Amazon Web Services (AWS) (2022) – *AWS IoT and Cloud-Based Object Detection*, AWS Documentation.
- Google Cloud AI (2021) – *Using TensorFlow and AutoML for Cloud-Based Object Detection*, Google Cloud White Paper.
- Ultralytics (2023) – *YOLOv8 Official Documentation & Model Training Guide*, GitHub Repository.
- OpenCV Team (2023) – *OpenCV-Python Tutorials and Video Processing Documentation*, OpenCV.org.
- Zhang, Y., Wang, X., & Li, J. (2022) – *Real-Time Object Detection for Smart Surveillance Systems*, IEEE Transactions on Image Processing.
- Nguyen, T., & Lee, Y. (2021) – *Autonomous Vehicles and Real-Time Object Detection Using Deep Learning*, International Journal of Computer Vision.