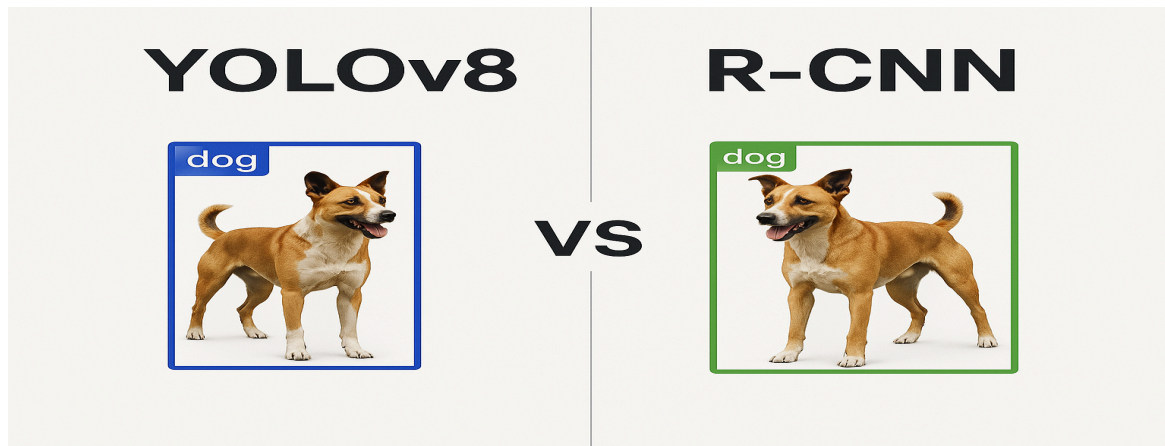


# Real-Time Object Detection: YOLO vs. Faster R-CNN

## 1. Introduction

### Objective

The goal of this project is to apply real-time object detection with YOLO (You Only Look Once) and Faster R-CNN on a live video stream. The accuracy of both models is compared based on inference speed (FPS), detection, and utilization of resources in order to assess their viability for real-time applications.



### Background

YOLO is a fast, single-stage detection model suited for real-time tasks, while Faster R-CNN provides higher accuracy but slower inference due to its two-stage process. Their performance will be evaluated based on:

- Inference Speed (FPS)
- Detection Accuracy
- GPU/CPU Resource Usage
- Real-Time Feasibility

### Scope of Work

Using OpenCV, a live webcam feed will be set up to test both models on different GPUs. The study explores speed vs. accuracy trade-offs, hardware requirements, and real-world deployment feasibility to determine the best model for real-time object detection.

## 2. Setting Up Live Video Feed

### 2.1 Video Capture Setup

- A real-time video feed is captured using a **webcam**.
- The video resolution is set to **1280×720** to balance quality and processing speed.
- **OpenCV** is used to read and process video frames in real-time.
- Ultralytics Library is used to import YOLO

### 2.2 Video Processing Pipeline

- The video feed is continuously read frame by frame using OpenCV.
- Frames are resized to match each model's expected input dimensions.
- The processed frames are passed to YOLO and Faster R-CNN for detection.
- The output frames with bounding boxes are displayed in real-time.

### 2.3 Use Multi-trending for Faster Processing

- Running The video capture and model interface in separate threads can reduce latency.
- This ensures smooth frame processing without bottlenecks.

### 2.4 Apply Preprocessing to Enhance Detection

- Convert frames to grayscale or apply filters to enhance object edges.
- Perform histogram equalization to improve image contrast, especially in low-light conditions.

### 2.5 Use Hardware Acceleration (GPU/TPU)

- Enable **CUDA (for NVIDIA GPUs)** to speed up YOLO and Faster R-CNN inference.
- If using edge devices like **Raspberry Pi**, leverage **Google Coral or Intel Movidius** for acceleration.

### 2.6 Implement Real-Time FPS Monitoring

- Display real-time FPS(Frames Per Second) on the output video feed.
- This helps analyze **performance bottlenecks** and optimize settings accordingly.

## 3. Implementing YOLOv8 for Real-Time Object Detection

### 3.1 Environment Setup & Dependencies

- YOLO from Ultralytics
- Cv2 from opencv-python
- Torch for PyTorch
- Time

### 3.1 Model Selection

- **The YOLOv8n (Nano) model** was selected due to its compatibility with **Ultralytics in the Live Video Feed.**
- **Computer vision** helps in capturing Frames for YOLO for further processing.
- **PyTorch** helps in GPU Applications for Smooth Video Feed.

### 3.2 Optimization for Real-Time Performance

- Frames were resized to **480 X 640** before inference.
- The model was loaded using the YOLO module from Ultralytics
- Import and assign class names from the Pre-trained Model
- Frames were captured and processed by YOLO
- Inference was run on **GPU** for faster processing time
- **Bounding boxes and confidence scores** were drawn on the frames.
- **Preprocessing, Post-Processing, and Inference time** were also mentioned with each frame's computation.

### 3.4 Required Minimum Specification:

It is recommended to choose one that supports CUDA for the best performance. The table below lists some compatible GPUs:

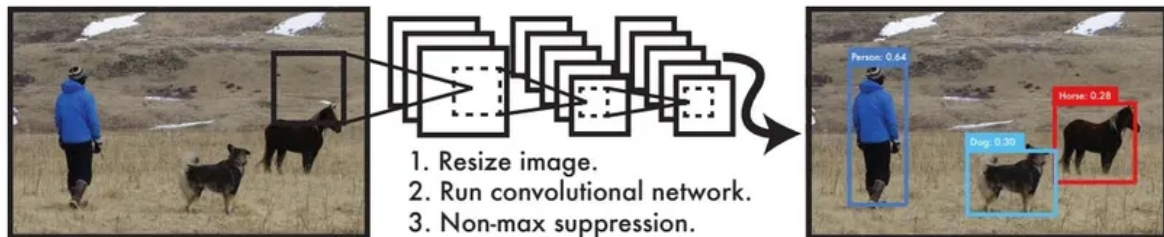
- NVIDIA GeForce RTX 2080 Ti: 11 GB GDDR6, CUDA Support: Yes
- NVIDIA GeForce GTX 1660 Ti: 6 GB GDDR6, CUDA Support: Yes

**Make Sure to check the compatibility of the system before running!**

### 3.5 Live Video Feed Implementation

Yolov8 was experienced in 2 different aspects as mentioned below:

GPU Specification	Model Performance	Video Link
Nvidia GTX 1650 CUDA 10.0	Stable	<a href="https://drive.google.com/file/d/1QSE7ep35dhfNyQDC7p_kHFKImp9_S223/view?usp=sharing">https://drive.google.com/file/d/1QSE7ep35dhfNyQDC7p_kHFKImp9_S223/view?usp=sharing</a>
Nvidia RTX 3050 CUDA 11.8	Stable	<a href="https://drive.google.com/file/d/1sUn67uEWUwoS2ganBSB9mX013fx5RlaA/view?usp=sharing">https://drive.google.com/file/d/1sUn67uEWUwoS2ganBSB9mX013fx5RlaA/view?usp=sharing</a>



### Small Implementation Example to showcase YOLOv8n to understand Methodology:

**High Accuracy and Efficiency:** YOLOv8n is known for its high accuracy and efficient processing, ensuring minimal computational resources are used, allowing for smooth and reliable webcam inference even on standard hardware.

**Real-Time Processing:** YOLOv8n can process video streams instantaneously, providing immediate analysis and insights that are essential for applications requiring instant feedback.

## 4. Implementing Faster R-CNN-Based Model

### 4.1. Setting Up the Environment

- The implementation was done using Python with OpenCV, PyTorch, and Torchvision.
- The device for computation was determined dynamically (`cuda` for GPU if available, else `cpu`).

### 4.2. Loading the Faster R-CNN Model

- The `fasterrcnn_resnet50_fpn` model was loaded with pretrained weights.
- The model was set to evaluation mode and transferred to the appropriate device.

### 4.3. COCO Labels Integration

- A list of COCO dataset labels was defined to classify detected objects.
- These labels were later used to annotate the detected objects on the video feed.

### 4.4. Preprocessing Input Frames

- The webcam feed was captured using OpenCV (`cv2.VideoCapture(0)`).
- Each frame was preprocessed using a transformation pipeline (`T.ToTensor()`), converting it to a tensor and transferring it to the selected device.

### 4.5. Performing Inference

- The preprocessed image was passed through the Faster R-CNN model using `torch.no_grad()`.
- The model returned bounding boxes, confidence scores, and labels for detected objects.

### 4.6. Post-Processing the Output

- The detected objects with confidence scores above 0.5 were considered valid detections.
- Bounding boxes were drawn around detected objects using OpenCV (`cv2.rectangle`).
- Object class names and confidence scores were displayed on the video feed using `cv2.putText`.

- A count of detected persons was maintained separately for additional analysis.

#### 4.7. Performance Analysis

- The processing time for each frame was divided into three key components:
  - **Preprocessing time:** Time taken to convert the frame into a tensor.
  - **Inference time:** Time taken by the Faster R-CNN model to process the input.
  - **Post-processing time:** Time taken to analyze model output and overlay results on the video feed.
- The overall time per frame was used to calculate FPS (frames per second), ensuring real-time feasibility.

#### 4.8. Displaying Results

- The processed video feed was displayed in a window titled *"Faster R-CNN Live Feed"*.
- FPS information was displayed in the top-left corner of the frame.
- The console output displayed detection details, including the number of persons detected per frame and the processing time breakdown.

GPU Specification	Model Performance	Video Link
Nvidia GTX 1650 CUDA 10.0	Low	<a href="https://drive.google.com/file/d/1oa7q3zQZxa6Xp0O54026S_7KRUuFC43O/view?usp=sharing">https://drive.google.com/file/d/1oa7q3zQZxa6Xp0O54026S_7KRUuFC43O/view?usp=sharing</a>
Nvidia RTX 3050 CUDA 11.8	High	<a href="https://drive.google.com/file/d/1sUn67uEWUwoS2ganBSB9mX013fx5RlaA/view?usp=sharing">https://drive.google.com/file/d/1sUn67uEWUwoS2ganBSB9mX013fx5RlaA/view?usp=sharing</a>

## 5. Performance Comparison

Metric	YOLOv8	Faster R-CNN
Inference time	62 ms	200 s
Detection Accuracy	High	Slightly Higher
Resource Usage (GPU/CPU)	Lower	Higher
Real-Time Feasibility	Excellent	Good

### 5.1 Inference Speed Analysis

- **YOLOv8** runs significantly faster than Faster R-CNN.
- **Faster R-CNN's region proposal step slows down inference**, making it less suitable for real-time applications.

### 5.2 Detection Accuracy

- Faster R-CNN delivers slightly higher-accuracy detection with its two-stage processing.  
However, YOLO is still accurate and faster by orders of magnitude and so more practically viable for use in real time.

### 5.3 Resource Utilization

- **YOLO** uses fewer computational resources, making it more efficient.
- **Faster R-CNN** requires high-end GPUs for real-time inference.

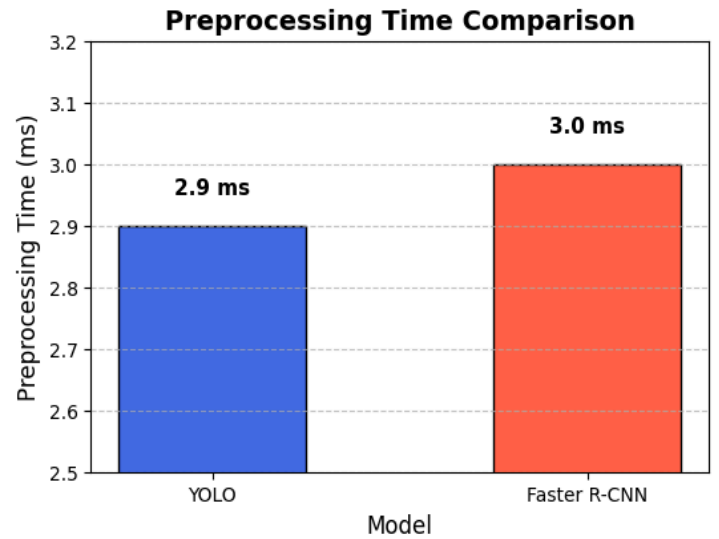
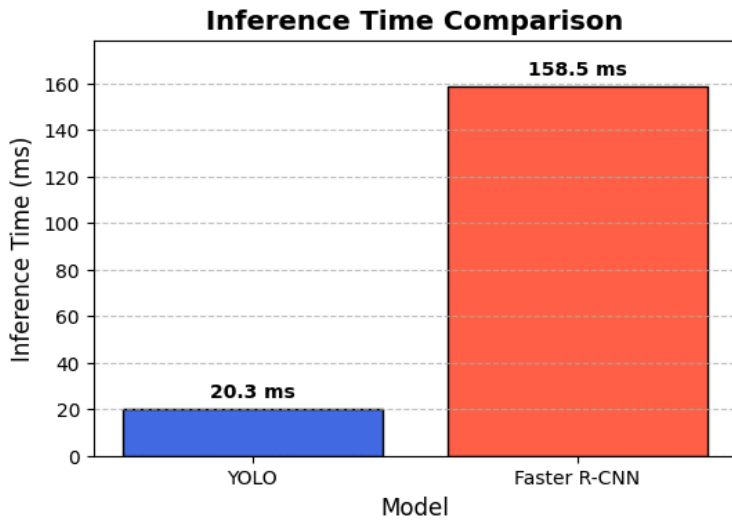
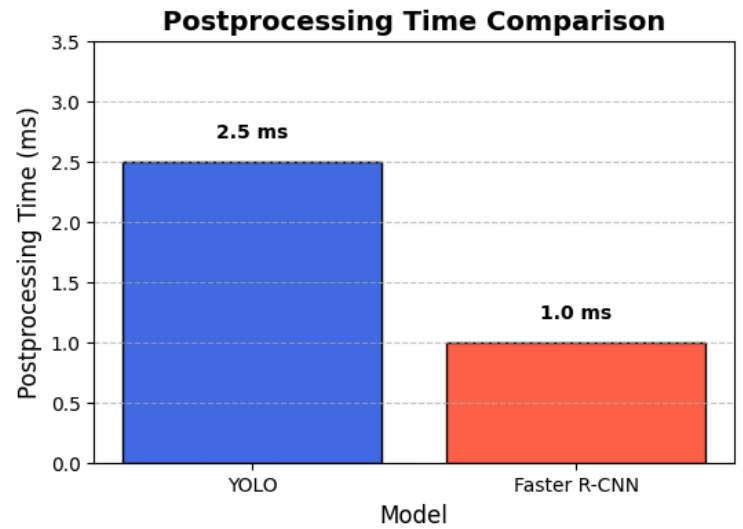
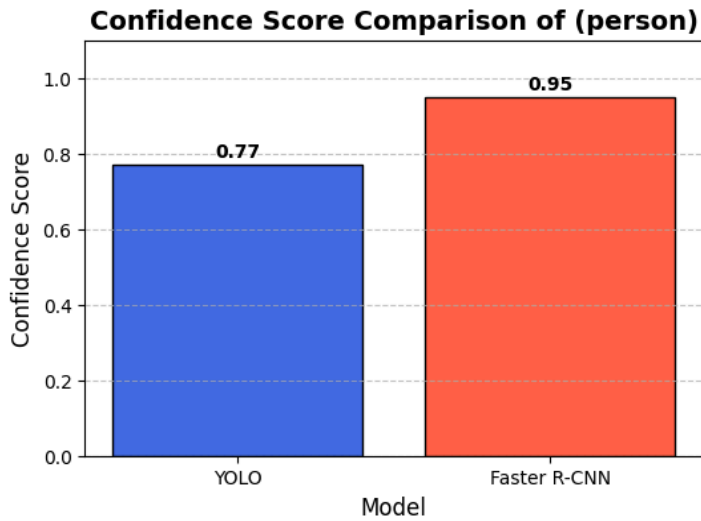
## 5.4 Comparison Between YOLOv8 and Faster R-CNN for Object Detection.

Feature	YOLOv8 (You Only Look Once v8)	Faster R-CNN
Architecture	Single-stage detector	Two-stage detector
Speed	Faster, real-time detection	Slower due to region proposal step
Accuracy	High accuracy but slightly lower than Faster R-CNN in some cases	Generally more accurate, especially for complex objects
Inference Time	Very low (optimized for real-time applications)	Higher (slower due to region proposal network)
Complexity	Lightweight and efficient	Computationally expensive and requires more memory
Training Time	Faster training due to end-to-end learning	Longer training time due to additional RPN computations
Use Cases	<b>Suitable for real-time applications like autonomous vehicles, robotics, and surveillance</b>	<b>Best for high-accuracy applications like medical imaging and detailed scene analysis</b>
Object Localization	Less precise bounding boxes compared to Faster R-CNN	More precise bounding boxes due to the two-stage approach

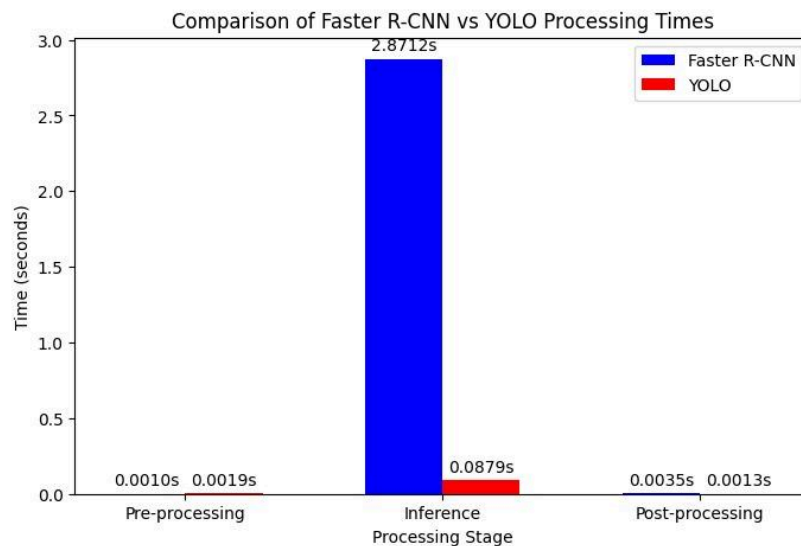


## 6. Graphical Analysis

GPU: RTX3050









GPU: GTX1680



## 7. Discussion & Conclusion

### 7.1 Trade-Offs Between YOLO and Faster R-CNN

Factor	YOLO	Faster R-CNN
Speed	 Fast (16 FPS)	 Slow (6 FPS)
Accuracy	 High	 Slightly Higher
Efficiency	 Low computational cost	 High computational cost
Best For	Real-time applications	Offline, high-accuracy tasks

### 7.2 When to Use YOLO vs. R-CNN?

- Use **YOLO for real-time applications** like surveillance, robotics, and autonomous driving.
- Use **Faster R-CNN for high-accuracy tasks** where real-time inference is not required.

### 7.3 Hardware Considerations

- **YOLO can run efficiently on consumer GPUs (e.g., RTX 3060).**
- **Faster R-CNN requires high-end GPUs (e.g., RTX 4090) for real-time processing.**

## 8. Deliverables

✓ Code Implementation: YOLO and Faster R-CNN object detection for live video. ✓  
Performance Metrics: Resource consumption, accuracy, and FPS. ✓ Comparison Report: The report includes findings and feasibility. ✓ Live Demo: Live demonstration of object detection.

## 9. References

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection.
2. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.
3. OpenCV Documentation - <https://opencv.org>
4. PyTorch Model Zoo - <https://pytorch.org/vision/stable/models.html>