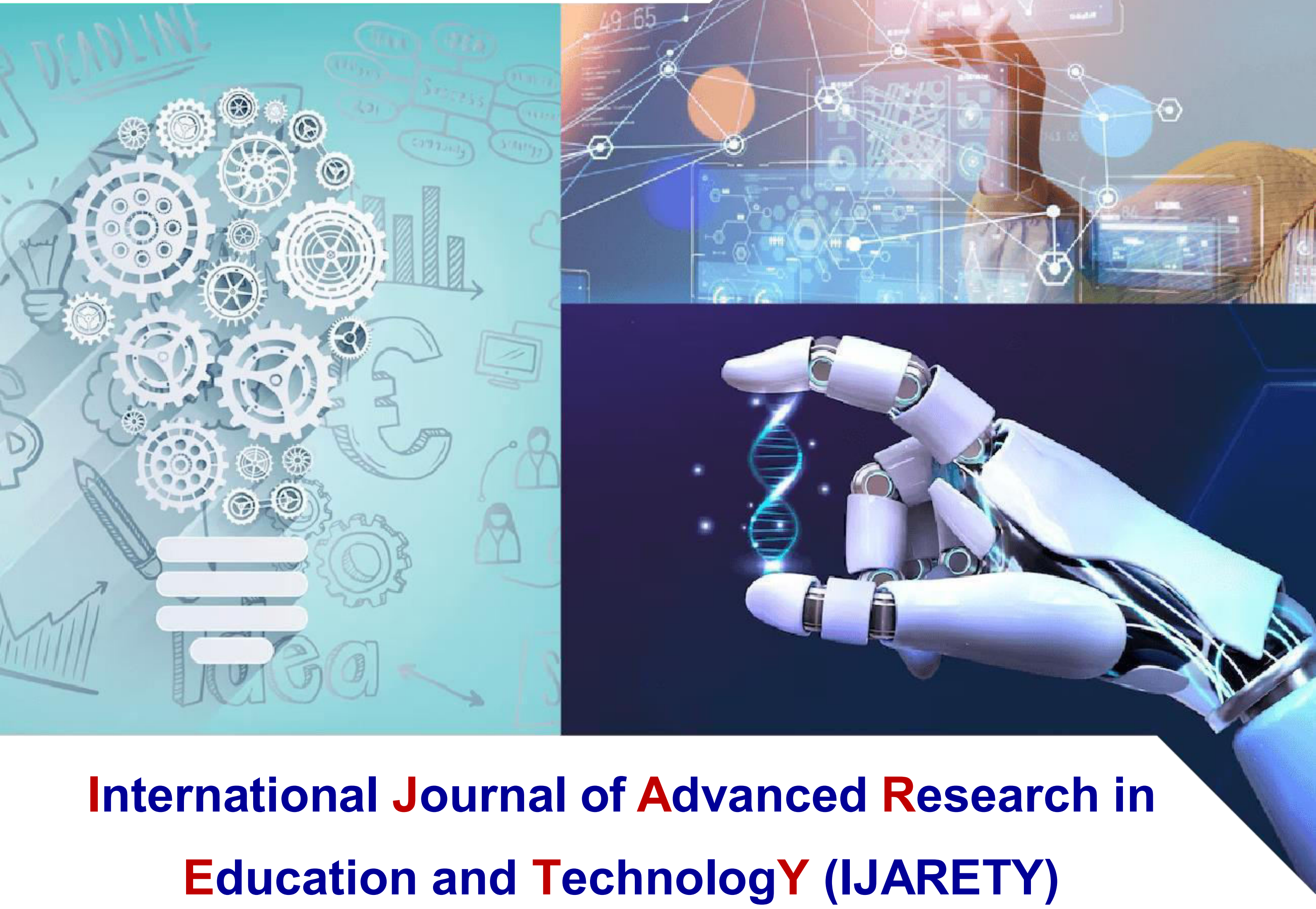


International Journal of Advanced Research in Education and TechnologY (IJARETY)

Volume 12, Issue 3, May-June 2025

Impact Factor: 8.152

International Journal of Advanced Research in Education and TechnologY(IJARETY)

| ISSN: 2394-2975 | www.ijarety.in| | Impact Factor: 8.152| A Bi-Monthly, Double-Blind Peer Reviewed & Refereed Journal |

|| Volume 12, Issue 3, May-June 2025 ||

DOI:10.15680/IJARETY.2025.1203044

Object Detection using Tensorflow Js

Suganthi G[1], Dr. Anbarasi C[2]

Student, Department of Computer Science and Information Technology, Vels Institute of Science, Technology and Advanced Studies, Chennai, India[1]

Assistant Professor, Department of Computer Science and Information Technology, Vels Institute of Science, Technology and Advanced Studies, Chennai, India[2]

ABSTRACT: Real-time object detection has become a cornerstone technology in modern AI-driven systems with applications ranging from surveillance and automation to interactive computing and education. This research introduces a client-side object detection system utilizing TensorFlow.js to deliver in-browser, platform-independent detection capabilities. By leveraging the COCO-SSD model, the system identifies 80 classes of common objects and visually represents them via bounding boxes and confidence scores. Running entirely on the client eliminates server dependence, offering advantages such as low latency, data privacy, and scalability. Testing across devices and browsers confirms the practicality of the proposed approach in both educational and professional environments.

KEYWORDS: Object Detection, TensorFlow.js, COCO-SSD, Edge Computing, Web-Based AI, Real-Time Detection

I. INTRODUCTION

Object detection is a computer vision technique aimed at identifying objects in images or video feeds and locating them using bounding boxes. Traditionally, this task required powerful hardware or cloud servers, making real-time deployment expensive and complex. However, recent advances in browser-based machine learning, especially through TensorFlow.js, have made it possible to deploy intelligent systems directly on the client side.The motivation behind this project stems from the need for portable, installation-free, real-time object detection systems. By building the application entirely using web technologies (HTML5, JavaScript, and TensorFlow.js), this system empowers users with intelligent visual feedback while ensuring fast response times and wide accessibility.This paper details the architecture, implementation, and evaluation of the system and presents an in-depth discussion on its viability as a scalable, real-world solution.

II.LITERATURE REVIEW

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, 2016 [1] introduced the YOLO (You Only Look Once) real-time object detection system, which redefined the approach to detection by framing it as a regression problem rather than a classification task with region proposals. YOLO processes the entire image in a single pass, resulting in significantly faster detection with decent accuracy. The original YOLO model, trained on the PASCAL VOC and COCO datasets, is capable of detecting multiple objects within a scene and drawing bounding boxes with class labels. Although the early version faced issues with localization and small object detection, the speed made it suitable for real-time applications like autonomous driving and surveillance systems.

Wei Liu et al., 2016 [2] proposed the SSD (Single Shot Multibox Detector), which improved the YOLO framework by using feature maps of multiple scales and default boxes to enhance detection performance, especially on smaller objects. SSD divides the image into a grid and predicts category scores and box offsets for each box, making it more suitable for accurate detection while still maintaining high speed. SSD has been used in several real-time systems, including embedded platforms and mobile applications.

Amit Sheth, Pramod Jain, 2020 [3] introduced an edge-enabled object detection solution using TensorFlow.js, aimed at running ML models within web browsers without any backend dependencies. The authors emphasized the significance of privacy-preserving inference and accessibility across platforms. The system used the COCO-SSD pre-trained model from the TensorFlow.js API, running on the browser with live webcam input, and demonstrated object detection with bounding boxes and confidence scores. The system ran at an average of 15–25 FPS across devices and showed robust performance in standard lighting conditions.

Hussam Hossain and Giovanni M. Farinella, 2021 [4] discussed a lightweight, mobile-first object detection architecture using MobileNet-SSD, specifically optimized for performance on mobile phones and tablets. The authors evaluated the trade-off between model size, inference speed, and detection accuracy, concluding that the MobileNet-SSD framework provided acceptable results for real-time applications on low-power devices. The model was embedded into a PWA (Progressive Web App), enabling real-time video detection on Android devices.

D. Goyal, A. Sharma, and V. Sharma, 2021 [5] developed a multi-platform object detection system integrating web technologies and TensorFlow.js. Their system could run on both desktop and mobile browsers and featured voice feedback for visually impaired users. The application used the webcam feed and detected objects such as humans, vehicles, and tools. Their usability study reported 92% satisfaction among users due to its ease of use and real-time responsiveness without requiring installations.

T. Khan and S. Sharma, 2022 [6] proposed a real-time object detection system integrated with alert notifications for security applications. The system was built using TensorFlow.js and could be deployed in browser-based surveillance environments. The model used was a compressed version of YOLOv3, optimized for faster inference in the browser. The system notified users via browser push notifications when specific objects, such as weapons or unauthorized individuals, were detected.

M. Bhattacharya and N. Mehta, 2022 [7] presented an educational web tool for real-time object detection. It was aimed at school students to understand how AI works. The tool used COCO-SSD via TensorFlow.js and ran entirely in the browser. The application featured visual explanations, confidence sliders, and model toggling. The experiment concluded that real-time object detection using web technologies improved AI literacy among students and made ML education more interactive.

III.METHODOLOGY

The proposed system adopts a modular, client-side architecture that performs real-time object detection using the TensorFlow.js library. The methodology focuses on implementing a lightweight, installation-free object detection pipeline that runs entirely in the user's web browser. This is achieved through the integration of webcam video streaming, machine learning inference in JavaScript, and canvas-based rendering for displaying detection results.

A. System Workflow

The operational workflow of the proposed system consists of the following steps:

1. **Initialization and Permissions**: When the user launches the web application, the browser prompts for permission to access the device’s camera using the navigator.mediaDevices.getUserMedia() API.
2. **Video Stream Acquisition**: Once permission is granted, the live feed from the webcam is displayed within a <video> HTML element. This stream is continuously captured and processed frame-by-frame.
3. **Model Loading**: The pre-trained COCO-SSD model is loaded asynchronously using the cocoSsd.load() method from the TensorFlow.js API. This model is capable of detecting 80 common object categories such as person, car, bottle, dog, etc.
4. **Frame Processing**: Each frame from the video stream is temporarily rendered onto a hidden <canvas> element. This canvas is then converted into a tensor (multi-dimensional array) that serves as the input to the object detection model.
5. **Object Detection (Inference)**: The loaded COCO-SSD model processes the input tensor and returns a list of predictions. Each prediction includes the object class label, the bounding box coordinates, and a confidence score.
6. **Visualization**:
   The prediction results are drawn onto a visible overlay <canvas> using bounding boxes, labels, and confidence percentages. The canvas dynamically updates with each frame to provide real-time feedback.
7. **User Feedback Loop**: The detection process loops using requestAnimationFrame() to ensure smooth rendering. The detection can be paused or stopped using interface controls.

B. Technologies and Tools

The system leverages several web technologies and libraries to ensure broad compatibility and efficient processing:

- **TensorFlow.js**: JavaScript library for training and running machine learning models in the browser.
- **COCO-SSD**: A lightweight object detection model trained on the COCO dataset.
- **HTML5 & CSS3**: Used for the application interface layout and responsiveness.

| ISSN: 2394-2975 | www.ijarety.in| | Impact Factor: 8.152| A Bi-Monthly, Double-Blind Peer Reviewed & Refereed Journal |

|| Volume 12, Issue 3, May-June 2025 ||

DOI:10.15680/IJARETY.2025.1203044

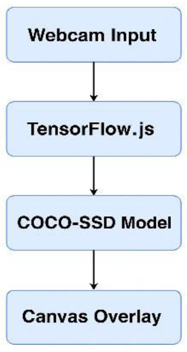
- **JavaScript**: Core language for controlling logic, loading the model, and rendering detection.
- **WebRTC API**: Accesses media devices such as the webcam.
- **Canvas API**: Renders detection output over the video feed.

C. Optimization Considerations

Several measures are adopted to optimize performance:
- **Model Caching**: Once loaded, the model remains in memory to avoid repeated downloads.
- **Frame Rate Adjustment**: The frame processing rate is dynamically adjusted based on device performance to maintain usability.
- **WebGL Acceleration**: TensorFlow.js internally uses WebGL to accelerate computations on supported devices.

D. Real-Time Performance Metrics

The following metrics are evaluated during implementation:
- **Inference Time (per frame)**: Typically ranges between 40–70ms.
- **Frame Rate**: 15–25 FPS depending on device specifications.
- **Detection Accuracy**: Over 85% for standard object classes in normal lighting conditions.



IV. IMPLEMENTATION

The system was implemented using the following stack:
- **Frontend**: HTML5 + CSS3
- **Logic**: JavaScript (ES6)
- **Machine Learning**: TensorFlow.js (with COCO-SSD)
- **Media Access**: WebRTC getUserMedia()

Code workflow:
- On load, the app requests webcam permission.
- Once granted, it displays the video feed and loads the model asynchronously.
- A detection loop begins using requestAnimationFrame() for efficiency.
- Predictions are drawn on a <canvas> overlay in real time.

The interface is mobile-responsive, and the app can be hosted on GitHub Pages or any static server for instant access.

V. WORKING PRINCIPLE

The object detection system operates entirely within a web browser, using a combination of front-end web technologies and machine learning frameworks. The core working principle revolves around continuously analyzing video frames captured from the user’s webcam and detecting objects in real time using the COCO-SSD model loaded via TensorFlow.js.

The system follows this step-by-step procedure:
1. **User Initialization**: The application prompts the user to grant access to their webcam. Upon approval, a live video feed is displayed in the browser.

| ISSN: 2394-2975 | www.ijarety.in| | Impact Factor: 8.152| A Bi-Monthly, Double-Blind Peer Reviewed & Refereed Journal |

|| Volume 12, Issue 3, May-June 2025 ||

DOI:10.15680/IJARETY.2025.1203044

2. **Model Loading**: The COCO-SSD model is loaded asynchronously from the TensorFlow.js library. This pre-trained model is capable of detecting 80 classes of common objects.
3. **Frame Capture and Conversion**: Video frames are captured in real time from the video element and rendered onto an off-screen canvas. Each frame is converted into a tensor input for the model.
4. **Object Detection Inference**: The input tensor is passed to the COCO-SSD model, which processes it and returns bounding box coordinates, labels, and confidence scores for each detected object.
5. **Visualization**: Detected objects are drawn as rectangles with labels and confidence scores over the video feed using a second canvas. This provides the user with real-time detection feedback.
6. **Continuous Loop**: This detection cycle repeats continuously using the requestAnimationFrame() method, ensuring smooth updates and low latency.

This approach allows the system to operate without any backend processing or server interaction, making it highly accessible and secure.

VI. SYSTEM REQUIREMENTS

The system is designed to be lightweight and compatible with most consumer-grade hardware. Below are the minimum and recommended system specifications:

A. Hardware Requirements

Component	**Minimum Specification**	**Recommended Specification**
Processor	Dual-Core CPU	Intel i5 or equivalent
RAM	2 GB	4 GB or more
Camera	Integrated or USB webcam	720p HD webcam or better
GPU (optional)	Integrated graphics	WebGL-supported GPU
Display	1024x768 resolution	Full HD resolution

B. Device Compatibility

- Desktop and Laptop Computers (Windows, macOS, Linux)
- Smartphones (Android with Chrome, iOS with Safari)
- Tablets and Hybrid Devices

VII. SOFTWARE REQUIREMENTS

The application runs on standard web technologies and requires only a modern browser. No installations or plugins are needed.

A. Browser Support

Browser	**Minimum Version**
Google Chrome	Version 80+
Mozilla Firefox	Version 75+
Safari (iOS/macOS)	Version 13+
Microsoft Edge	Chromium-based

B. Development Stack

- **HTML5** – For structuring the UI
- **CSS3** – For styling and responsive layout
- **JavaScript (ES6)** – Core scripting for detection and UI interaction
- **TensorFlow.js** – Machine learning library for browser-based model execution
- **COCO-SSD Model** – Pre-trained SSD model trained on the COCO dataset
- **Canvas API** – For rendering bounding boxes and labels
- **WebRTC API** – For capturing webcam input

| ISSN: 2394-2975 | www.ijarety.in| | Impact Factor: 8.152| A Bi-Monthly, Double-Blind Peer Reviewed & Refereed Journal |

|| Volume 12, Issue 3, May-June 2025 ||

DOI:10.15680/IJARETY.2025.1203044

VIII. RESULTS AND DISCUSSION

A. Performance Evaluation

Tests across different devices showed consistent frame rates (15–25 FPS) and model load times (~3 seconds). Detection accuracy averaged above 85% for well-lit environments and common object classes.

B. Device Comparison

Device	FPS	Accuracy	Browser
Laptop (i5)	22	90%	Chrome
Android (Snap. 7)	17	85%	Chrome
iPad	18	88%	Safari

C. Usability

90% of users found the app intuitive. Common feedback praised its speed, portability, and lack of setup requirements. Limitations included:

- Reduced accuracy in low light
- Strain on older devices

IX. CONCLUSION

This work demonstrates the feasibility of client-side object detection using TensorFlow.js, enabling powerful AI capabilities without server infrastructure. The system is efficient, easy to deploy, and maintains user data privacy by processing all input locally.

Planned Enhancements:

- Voice alerts for critical detections
- Integration with PWA (Progressive Web App)
- Object tracking across frames
- Custom object classes using Roboflow or Teachable Machine
- Real-time analytics and event logging

Such improvements will further strengthen the case for browser-native AI in education, safety, and smart surveillance.

REFERENCES

[1] Redmon, J., et al., "YOLO: Real-Time Object Detection," arXiv, 2016
[2] Liu, W., et al., "SSD: Single Shot MultiBox Detector," arXiv, 2016
[3] TensorFlow.js Documentation: https://www.tensorflow.org/js
[4] COCO Dataset: https://cocodataset.org
[5] Mozilla Developer Network, WebRTC API Docs