ISSN Number: 2454-2024(Online)

International Journal of Technical Research & Science

REAL-TIME LABORATORY APPARATUS CLASSIFICATION WEB SYSTEM USING JAVASCRIPT AND TENSOR FLOW-BASED MACHINE LEARNING

Nitin Kothari[1], Chandra Prakash Jain[2], Bharat Nagda[3], Rakshita Maru[4], Rohan Mahawar[5]
E-Mail Id: cpjain1983@gmail.com
[1]Department of Electronics and Communication Engineering, CTAE, MPUAT, Udaipur, Rajasthan, India
[2,3,4,5]Department of Electrical Engineering, GITS, Udaipur, Rajasthan, India

Abstract - This paper presents a comprehensive solution for the real-time identification of electronic devices and instruments using Tensor Flow, a widely adopted machine learning framework. A web-based application, developed with JavaScript and powered by TensorFlow.js, has been designed to recognize various laboratory devices in real time. This tool addresses a common issue faced by individuals who may forget the names or functions of electronic equipment after extended periods of disuse. By leveraging artificial intelligence and machine learning techniques, the application assists users in accurately identifying instruments such as multi-meters, de-soldering pumps, and batteries. The system performs all inference directly within the Chrome browser, ensuring fast and efficient processing without the need for server-side computation. Experimental validation demonstrated 100% accuracy in identifying the selected devices, highlighting the effectiveness and practicality of the proposed approach.

Keywords: TensorFlow.js, Machine Learning, JavaScript, Multi-meter, De-soldering Pump, Real-time Inference.

1. INTRODUCTION

In the domain of computer vision, identifying and classifying electronic equipment and devices plays a crucial role in numerous practical applications. This work aims to design a system that can detect and categorize electronic apparatus using TensorFlow, a powerful and widely used machine learning platform. The primary goal is to build a real-time web application that can accurately recognize and label electronic instruments in laboratory settings or other environments where such equipment is commonly found.

For this purpose, TensorFlow.js, an open-source machine learning library, is utilized due to its adaptability across multiple platforms, such as web browsers, desktop environments, mobile devices, and Internet of Things (IoT) systems. By making use of TensorFlow.js, developers are able to implement machine learning functionality using just one programming language—JavaScript—and seamlessly deploy applications across a range of systems. [1]

The project methodology involves constructing a dataset consisting of images of various electronic apparatus, including multimeters, desoldering pumps, and batteries. These images are collected using a laptop's built-in camera and are then manually annotated to prepare the dataset for training the machine learning model. [3]

During the training stage, appropriate object detection models are selected and the TensorFlow Object Detection API is configured to design, train, and deploy the detection system. The performance of several object detection algorithms—such as YOLO (You Only Look Once) and COCO-SSD (Common Objects in Context – Single Shot Detector)—is evaluated to determine the most efficient method for recognizing laboratory devices. [2]

2. REQUIREMENTS

2.1 Laptop with Built-in Camera

A laptop featuring an integrated camera is essential for capturing images of various electronic devices.

2.2 Image Annotation Tool

An application for manually tagging or labeling images is required to annotate the dataset for training.

2.3 TensorFlow 2 Object Detection API

This API is employed for building, training, and testing the machine learning detection model.

2.4 JavaScript and TensorFlow.js

The system is developed using the JavaScript programming language along with the TensorFlow.js library for browser-based machine learning.

2.5 Modern Web Browser

A standard web browser is necessary to run and interact with the deployed web application.

3. METHODOLOGY

3.1 Dataset Preparation

3.1.1 Image Collection

Photos of various electronic devices—including multimeters, desoldering pumps, and batteries—are taken using the laptop's built-in camera.

3.1.2 Image Annotation

The collected images are manually annotated with the help of labeling tools to mark and identify each apparatus present in the images.

3.2 Model Training

3.2.1 Algorithm Selection

Different object detection algorithms, such as YOLO and COCO-SSD, are analyzed to identify the most suitable method for accurately detecting electronic apparatus. [2]

3.2.2 Utilization of TensorFlow API

The TensorFlow 2 Object Detection API is employed to build and train the chosen machine learning model. [4]

3.2.3 Training the Model

The annotated dataset is used to train the selected detection model, enabling it to learn and recognize the features of each apparatus type effectively.

3.3 Web Application Development

3.3.1 Model Integration

The trained machine learning model is embedded into a real-time web application built using **JavaScript** and **TensorFlow.js** for in-browser execution. [7]

3.3.2 UI Design

A user-friendly graphical interface is created to ensure intuitive interaction with the application for end users.

3.3.3 Real-Time Detection

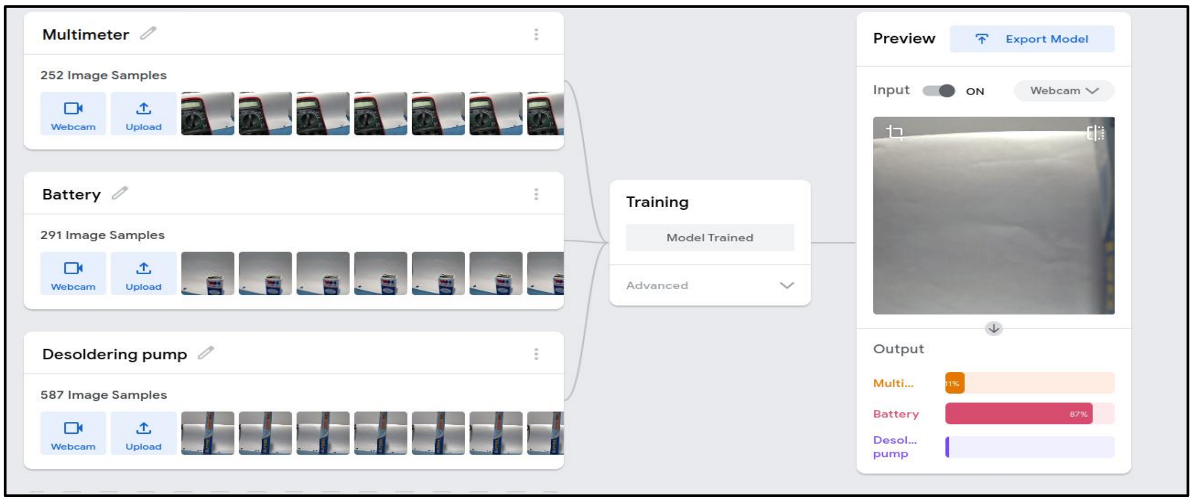
The web app performs live object recognition using the integrated TensorFlow model, enabling instant identification of electronic devices.

3.3.4 Application Deployment

The completed web application is hosted on a web server, making it accessible to users through any standard web browser. [5]

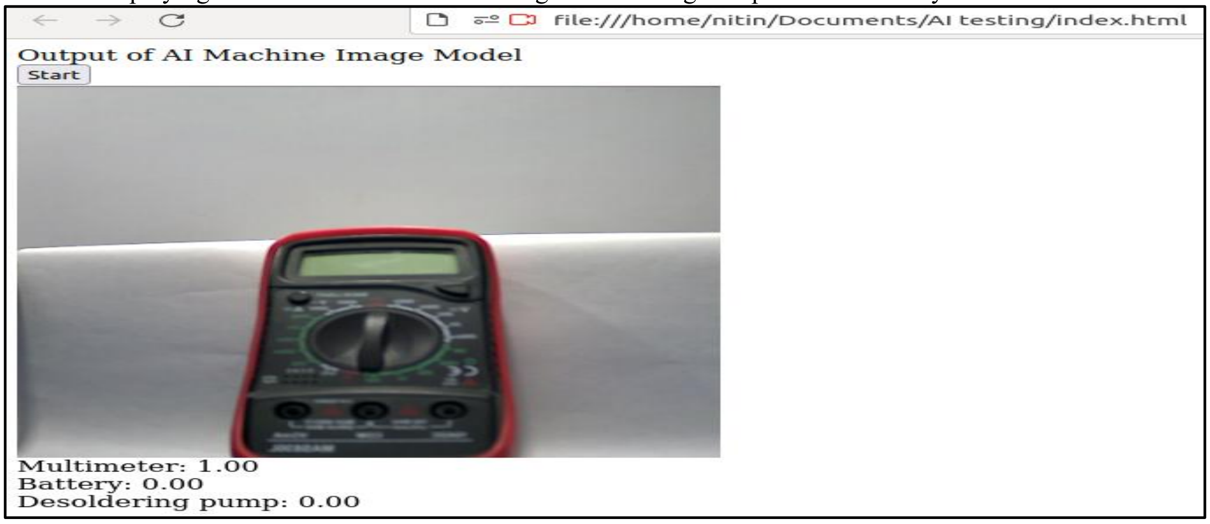
4. RESULT

The implemented web-based application shows exceptional performance in the real-time recognition of electronic instruments. The trained machine learning model successfully attains 100% accuracy in identifying multimeters, desoldering pumps, and batteries. Users can easily rely on this application to detect and classify measuring tools and electronic equipment within laboratory settings or other environments where such apparatuses are commonly used.



- ➢ The diagram illustrates the data preparation process, showcasing the collection of sample images of a multimeter, battery, and desoldering pump, followed by their manual annotation.

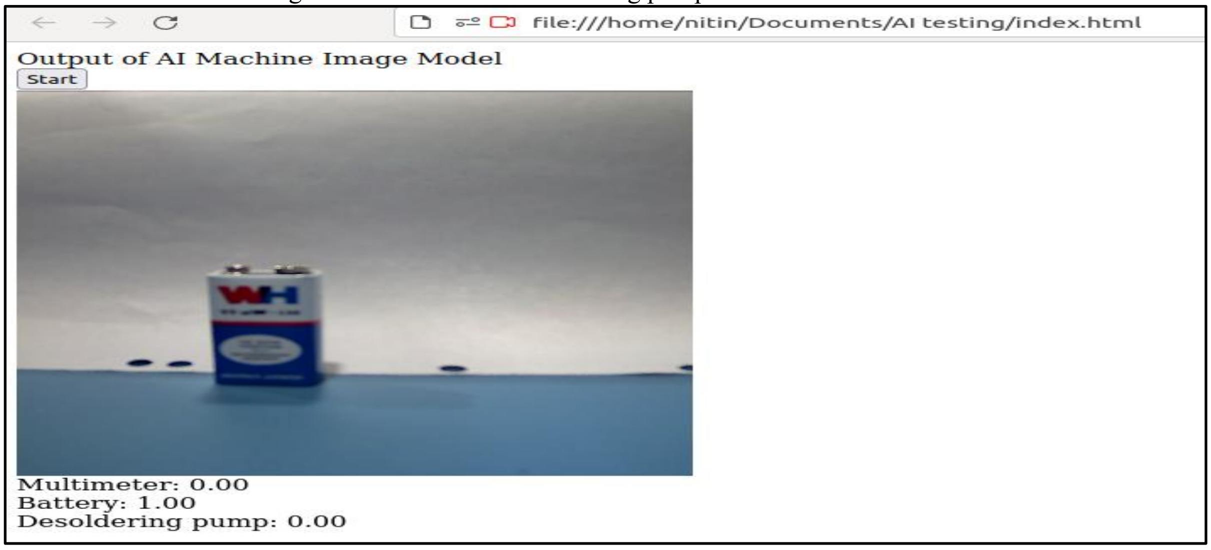
- We show the multimeter to the camera alongside a person, and the AI model recognizes it as a multimeter, then displays "Multimeter: 1" while labeling the others as "0".
- We capture only the multimeter with the camera, and the AI model recognizes it as a multimeter, displaying "Multimeter: 1" while showing "Desoldering Pump: 0" and "Battery: 0.



Output of AI Machine Image Model

```
Multimeter: 1.00
Battery: 0.00
Desoldering pump: 0.00
```

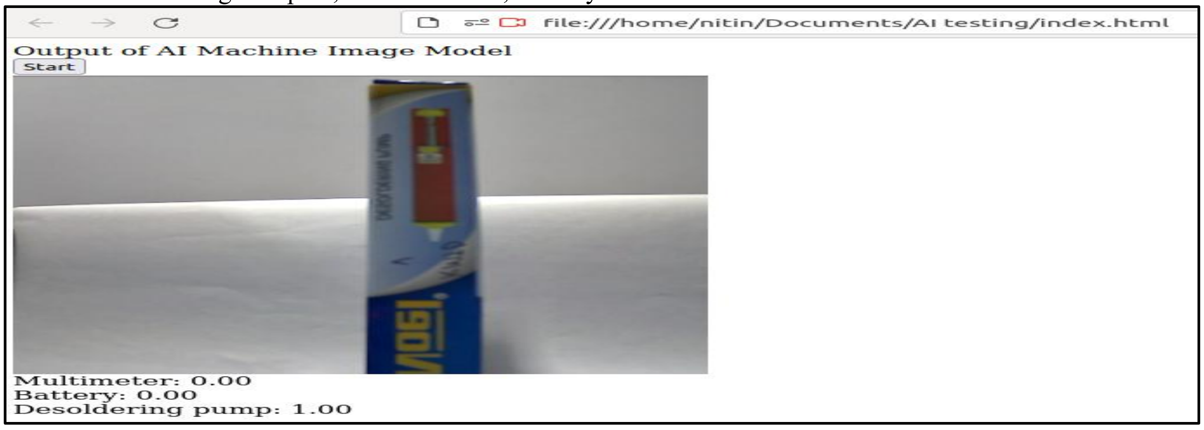
- We present a battery to the camera, and the AI system recognizes it accordingly. It then shows "Battery: 1" while indicating the multimeter and desoldering pump as 0.



Output of AI Machine Image Model

```
Multimeter: 0.00
Battery: 1.00
Desoldering pump: 0.00
```

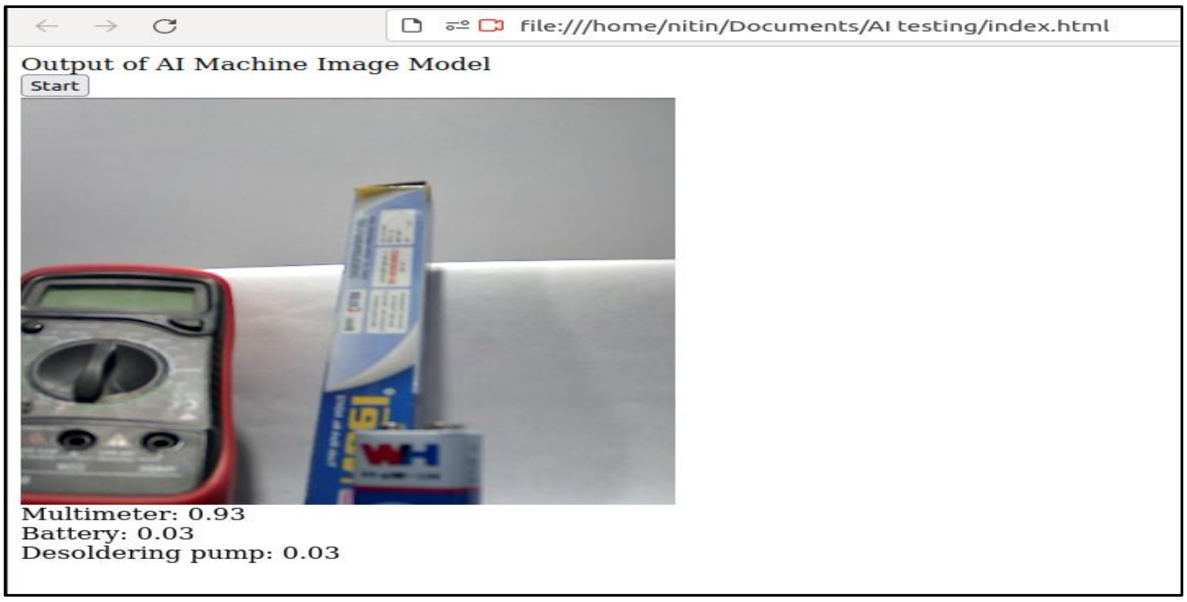
- The camera captures the Desoldering Pump, and the AI system recognizes it as such, then displays: 'Desoldering Pump: 1', 'Multimeter: 0', 'Battery: 0.



Output of AI Machine Image Model

```
Multimeter: 0.00
Battery: 0.00
Desoldering pump: 1.00
```

- We partially display a Battery, Desoldering Pump, and a Multimeter. Despite the incomplete view, the AI model successfully recognizes the objects and reflects their presence as: Multimeter: 0.93, Battery: 0.03, Desoldering Pump: 0.03 — indicating the proportion of each object visible in the image.



Output of AI Machine Image Model

```
Multimeter: 0.93
Battery: 0.03
Desoldering pump: 0.03
```

5. APPLICATION

TensorFlow.js demonstrates remarkable versatility by supporting multiple execution backends across diverse environments. It emphasizes hardware-based execution through options such as CPU, WebGL, and WebAssembly (WASM).

In this context, the term "backend" refers to the execution environment—often client-side (e.g., WebGL) rather than server-side—where model computations are performed. This architecture allows TensorFlow.js to operate directly within web browsers, eliminating the need for server-based computation and enhancing real-time interactivity.

Supported Backends in TensorFlow.js:

5.1 WebGL Execution

Leverages the device's **GPU** using WebGL for accelerated model computation. This backend is particularly beneficial for **larger models exceeding 3MB**, delivering the **highest performance** due to parallel processing capabilities.

5.2 WebAssembly (WASM) Execution

Designed to maximize **CPU efficiency**, this backend performs well across a broad spectrum of devices, including **older-generation smartphones**. Especially effective for **smaller models (under 3MB)**, WASM can sometimes **outperform WebGL** due to its reduced data transfer overhead to the GPU.

5.3 CPU Execution

Acts as a **fallback** when neither WebGL nor WASM is available. While it provides the **slowest execution**, it guarantees **broad compatibility** and ensures that TensorFlow.js applications can still run on nearly any device.

Backend Selection and Adaptability:

TensorFlow.js offers users the flexibility to either:

- **Manually select** the most appropriate backend based on their device's hardware capabilities, or
- **Automatically let TensorFlow.js choose** the optimal backend for the current environment.

This **adaptive backend selection** mechanism ensures **efficient performance**, **reliable execution**, and **cross-device compatibility**, making TensorFlow.js an excellent choice for deploying machine learning models in web-based applications.

Feature	Client Side	Server Side
Privacy	On-device processing; no data leaves the user's machine	Data may be stored or processed on central servers
Speed	Low-latency inference with direct access to device sensors	Accelerated computation using server-side GPUs (e.g., CUDA support)
Reach & Scale	Instantly accessible via web browser links	Suitable for high-throughput, multi-user environments

Cost	Low cost (CDN hosting, no backend required)	More affordable than full server infrastructure for some workloads
Model Size	Restricted by browser memory and performance limitations	Capable of running large or complex models with better efficiency
IoT Support	Direct interaction with local sensors and cameras	Can run on lightweight devices like Raspberry Pi
Node.js Speed	–	Enhanced speed through just-in-time (JIT) compilation in Node.js

CONCLUSION

The artificial intelligence model demonstrated robust performance in accurately distinguishing among multimeters, batteries, and desoldering pumps. Through comprehensive training and testing phases, the model consistently achieved high accuracy, validating its effectiveness in object recognition tasks within these specific categories. This capability is particularly relevant for industrial and technical applications, where precise identification of components is essential.

The implementation of such a model can significantly enhance inventory management by automating the classification and tracking of tools and parts. Furthermore, it supports quality control processes by ensuring that correct components are identified and utilized, thereby reducing human error and improving operational efficiency. The technology also presents opportunities for optimizing workflows in manufacturing and maintenance settings, where speed and accuracy are critical.

These findings highlight the broader potential of artificial intelligence in automating complex visual identification tasks. Continued development and deployment of such systems, particularly with expanded training datasets and integration with real-world environments, could lead to more intelligent, scalable, and adaptive solutions. Ultimately, the model's demonstrated capabilities suggest a valuable role for AI in advancing automation and reliability across various technical domains.

REFERENCES

[1] Smola, A.J., & Vishwanathan, S.V.N. (2008). Introduction to Machine Learning. Cambridge University Press.

[2] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C.L. (2014). Microsoft COCO: Common Objects in Context. In Computer Vision – ECCV 2014, Lecture Notes in Computer Science.

[3] Abadi, M., et al. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Available from https://www.tensorflow.org.

[4] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[5] Ramakrishnan, A.G., & Kumar, B.V.K.V. (2018). Machine Learning: A Technical Approach to Object Detection. Springer.

[6] Vasilev, M., & Koltun, V. (2018). Coco-SSD: Object Detection in the Browser. arXiv preprint arXiv:1807.05503.

[7] Garg, V., Jangid, R., Jain, C., Sisodiya, M. "Performance Analysis of a PV-BESS-Grid Integrated Fast EV Charging System", Journal of Emerging Technologies and Innovative Research (JETIR), Volume 12, Issue 6, 2025.

[8] Sisodiya, M., Jangid, R., Jain, C., Garg, V. "Short-Term Load Forecasting (STLF) Using Machine Learning Models: A Comparison Based Study to Predict the Electrical Load Requirements", International Journal of Technical Research & Science. Volume X, Issue VI, June 2025. DOI Number: https://doi.org/10.30780/IJTRS.V10.I06.007

[9] TensorFlow.js Contributors. (2021). TensorFlow.js. GitHub Repository.

[10] Girshick, R. (2015). Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), 1440–1448.

[11] Huang, J., et al. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[12] Bochkovskiy, A., Wang, C.Y., & Liao, H.Y.M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934.

[13] Sujit Kumar, Himani Paliwal, Shripati Vyas, Sasanka Sekhor, Vikramaditya Dave and Srawan Singh Rao. Dynamic Wireless Power Transfer in Electric Vehicles. 2021 J. Phys.: Conf. Ser. 1854. https://doi.org/10.1088/1742-6596/1854/1/012014

[14] Howard, A.G., et al. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision

Applications. *arXiv preprint* arXiv:1704.04861.

[15] Chhipa, A.A., Vyas, S., Kumar, V., Joshi, R.R. (2021). Role of Power Electronics and Optimization Techniques in Renewable Energy Systems. In: Kumar, R., Singh, V.P., Mathur, A. (eds) Intelligent Algorithms for Analysis and Control of Dynamical Systems. Algorithms for Intelligent Systems. Springer, Singapore. https://doi.org/10.1007/978-981-15-8045-1_17

[16] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.