International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 06 | June - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Real Time Object Detection Using OpenCV

Azmeera Lavanya [1], N Naveen Kumar[2]

[1]Post-Graduate Student, Department of Information Technology, Software Engineering, Jawaharlal Nehru Technological University, Hyderabad, India.

[2]Professor, Department of Information Technology, Jawaharlal Nehru Technological University, Hyderabad, India

ABSTRACT - Object detection has evolved into a vital component of intelligent vision systems. It involves identifying instances of objects from specific categories in digital images or videos. This paper presents a real-time object detection approach using the Single Shot Multibox Detector (SSD) model with a MobileNet backbone, optimized for speed and low-resource devices. Implemented using OpenCV and TensorFlow, the system supports both still images and live video input. The model demonstrates an effective balance between speed and accuracy, offering high detection precision on standard computing hardware. Applications include surveillance, autonomous systems, and industrial monitoring, where real-time object detection is crucial.

Keywords: Real-Time Detection, SSD (Single Shot Multibox Detector), MobileNet, Deep Learning, OpenCV, Computer Vision.

1. INTRODUCTION

In the era of rapid technological advancement, the ability of machines to interpret and understand visual information has become a cornerstone of artificial intelligence. Object detection, as a subfield of computer vision, plays a pivotal role in enabling this capability by allowing systems to recognize, classify, and locate multiple objects within an image or a video stream. This technology is not only critical for emerging domains such as autonomous vehicles, intelligent surveillance, and robotics, but it also enhances user experience in everyday applications like photo tagging, virtual assistants, and mobile augmented reality.

Traditional methods of object detection often relied on handcrafted features and classic machine learning algorithms, such as Haar cascades and Histogram of Oriented Gradients (HOG) combined with Support Vector Machines (SVMs). While these approaches provided a foundation for early breakthroughs, they were limited in scalability and adaptability to complex visual environments. Their dependence on feature engineering made them less flexible and more susceptible to noise, occlusion, and lighting variations.

The introduction of deep learning, particularly Convolutional Neural Networks (CNNs), revolutionized object detection by enabling end-to-end learning of features directly from raw image data. This advancement led to the development of robust detection frameworks such as R-CNN, YOLO (You Only Look Once), and SSD (Single Shot MultiBox Detector). Each of these models brought distinct advantages in terms of speed, accuracy, and computational efficiency. SSD, in particular, gained popularity for its balanced trade-off between real-time performance and detection accuracy.

To further optimize object detection for real-time applications, lightweight neural networks like MobileNet were introduced. Designed with efficiency in mind, MobileNet uses depthwise separable convolutions to significantly reduce model size and computation time without major compromises in performance. When combined with SSD, MobileNet forms a powerful architecture capable of delivering fast and accurate object detection on resource-constrained devices.

This study details the development of a real-time object detection system that combines the SSD architecture with a MobileNet foundation, utilizing the capabilities of OpenCV and TensorFlow. The goal is to build a practical, scalable, and low-latency system that can perform robust object detection in both static and dynamic environments. The remainder

of this paper discusses the motivation behind this project, a review of related literature, a detailed overview of the system architecture, implementation methodology, experimental results, and potential areas for future enhancement.

2. PROPOSED METHODOLOGY

Tools and Frameworks Used:

Python Programming Language- Python is chosen as the core development language due to its clarity, concise syntax, and extensive support for scientific computing. Its widespread use in machine learning and computer vision research ensures compatibility with major frameworks. Python also allows seamless integration of visualization, data manipulation, and inference logic within a unified environment, speeding up both development and testing.

OpenCV, the Open Source Computer Vision Library, is an essential resource for tasks involving image and video processing. It provides high-level interfaces for capturing real-time frames from camera devices, resizing and normalizing images, drawing bounding boxes, and displaying annotated results. Of particular importance in this project is OpenCV's Deep Neural Network (DNN) module, which enables the execution of pre-trained deep learning models without relying on an external training setup.

TensorFlow Framework- TensorFlow serves as the engine behind the object detection model. It allows for loading and executing frozen inference graphs, managing the model weights, and handling computations on multidimensional data structures (tensors). In this context, TensorFlow operates as the backbone for SSD MobileNet execution, enabling integration through OpenCV's DNN backend, thus removing the need for building the entire deep learning pipeline from scratch.

COCO Dataset (Common Objects in Context)- The SSD MobileNet model used in this project is trained on the COCO dataset, which includes a broad variety of everyday objects such as people, vehicles, animals, and furniture. This dataset is instrumental in ensuring that the detection model generalizes well to real-world scenes. Its diversity and balance across 80 object categories improve detection robustness in practical deployments.

NumPy a python library is widely employed for manipulating video frames and performing pixel-level computations. It is vital for tasks such as blob creation (the transformation of image data into a format suitable for neural network input), normalization of pixel values, and coordinate transformations for drawing bounding boxes.

Jupyter Notebook Environment- Jupyter Notebook is used as the interactive coding platform for this project. It supports live code execution, inline visualization, and modular script management. This environment aids in experimentation, debugging, and explanation of each processing step, making it an effective interface for iterative development and documentation.

Hardware Configuration- To emphasize accessibility and efficiency, the model was tested on a standard personal computer equipped with an Intel i5 processor and 8 GB RAM, without the use of a dedicated GPU. The system's ability to deliver real-time performance under these conditions showcases its suitability for deployment in low-cost embedded systems and edge devices.

Dataset:

- The system incorporates a pre-trained SSD MobileNet V3 model, which was trained using the COCO dataset.

- The COCO dataset comprises 80 distinct object categories, including common items like bicycles, cars, people, dogs, and chairs.

- Model files used: frozen_inference_graph.pb and ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt.

- Class labels are loaded from a Labels.txt file.

Implementation Workflow:

1. Load model and label files.

2. Read input from webcam, video file, or image.

3. Preprocess frame: resize, normalization, blob creation.

4. Perform forward pass using OpenCV DNN module.

5. The process involves extracting the class index, confidence score, and bounding box coordinates for detected objects.

6. Apply threshold to filter low-confidence detections.

7. Draw bounding boxes and labels on frames.

3. ARCHITECTURE

The SSD-MobileNet architecture is engineered to deliver real-time performance within resource-constrained environments.

Input Preprocessing: Video frames are captured and then resized to 300x300 pixels to conform to the input dimensions required by the SSD MobileNet model. Pixel values are normalized, and the image is converted into a blob.

Feature Extraction (MobileNet): MobileNet serves as the backbone, providing high-level feature maps through depthwise separable convolutions. This process extracts critical spatial features, simultaneously reducing the overall complexity of the model.

Detection Head (SSD Layers): The SSD head comprises multiple convolution layers that predict class probabilities and bounding box offsets. Multiple bounding boxes, varying in aspect ratio and scale, are predicted at each location on the feature map, enabling this multi-scale method to ensure reliable detection of both small and large objects.

Post-processing: The raw outputs undergo confidence thresholding to eliminate low-confidence predictions. Redundant overlapping bounding boxes are removed using Non-Maximum Suppression (NMS), with only the most pertinent predictions being kept for the final visual output.

Output Visualization: Using OpenCV, the final detections are drawn on the original frame with bounding boxes and associated class labels. This process repeats for each frame in a video stream, enabling live object detection.

This modular architecture ensures a balance of speed and accuracy, enabling the system to function effectively on embedded systems, mobile devices, and low-end CPUs.

4. TEST SCENARIOS

To ensure that the model performs well in practical use, several test scenarios were created. Each scenario aimed to replicate real-world conditions where object detection is typically used. These scenarios aided in assessing the consistency and adaptability of the SSD with MobileNet model.

Scenario 1: Detection in Cluttered Backgrounds This test involved images and video footage with complex backgrounds—such as busy streets, crowded rooms, and supermarket aisles. The goal was to evaluate the model's capability to detect multiple objects without being confused by visual noise. The system managed to correctly detect most target objects, although some overlapping items posed a challenge.

Scenario 2: Varying Lighting Conditions Images and video streams were captured under different lighting—bright daylight, dim indoor lighting, and nighttime with artificial lights. The model performed well in bright and moderate lighting but showed slight degradation in accuracy in darker settings. Applying brightness normalization helped improve detection slightly in low light.

Scenario 3: Moving Objects in Video Streams This test was conducted using both pre-recorded and live webcam video to simulate real-time scenarios with people walking, vehicles passing, or objects moving across the frame. The model demonstrated effective motion tracking, consistently high detection confidence, and seamless handling of partially occluded or reappearing objects.

Scenario 4: Object Detection from Different Angles and Scales We tested how well the model detects objects viewed from different angles—sideways, tilted, or partially blocked—and at various distances (close-up and far away). While most common objects were detected reliably, performance decreased for small, distant, or highly skewed objects, indicating room for improvement with advanced training techniques or fine-tuned models.

Scenario 5: Real-Time Detection with CPU Limitations This scenario focused on evaluating how the model performs without GPU support. The tests were done on a regular laptop with no external acceleration. Results showed stable real-time detection with 20–25 FPS and minimal latency, proving the system is suitable for edge deployment where hardware is limited.

These collective test scenarios affirmed the capability, flexibility, and practical applicability of the SSD and MobileNet-based object detection system for real-world implementation. They also provided insight into potential enhancements for better performance across diverse use cases.

5. EXPERIMENTAL RESULTS

The model was tested on static images, video files, and real-time webcam feeds. The detections were accurate in most conditions, with slight variations under extreme lighting or motion blur.

Performance Summary:

- Average Accuracy: 75% across test inputs.

- Real-time FPS: 20–25 frames per second on Intel i5 CPU.

- Detected Categories: Human, automobile, bottles, furniture, etc.

- Confidence Threshold: Detections above 50% confidence were considered valid.

5.1 OUTPUT SCREENS

- Fig 1: Output from a webcam feed showing real-time object detection.

- Fig 2: Processed frame from a video showing multiple object detections.

- Fig 3: displays a processed frame from a webcam, illustrating multiple object detections.

6. CONCLUSION

This work showcases an effective real-time object detection solution using SSD and MobileNet implemented with OpenCV. The model performed efficiently on commonly available hardware, demonstrating practical use for real-time scenarios. The system's modularity allows for easy integration into other vision-based applications.

Future Enhancements:

- Integration with object tracking algorithms.

- Potential deployment includes platforms such as Raspberry Pi or NVIDIA Jetson Nano.

- Custom datasets can be employed for specialized object detection tasks within specific domains.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who supported and encouraged me throughout the development of this project. I am especially grateful to my project guide, N Naveen Kumar, whose guidance, valuable insights, and continuous support played a crucial role in shaping this research. Their feedback and encouragement motivated me to explore deeper and work more efficiently.

REFERENCES

[1] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In Proceedings of the European Conference on Computer Vision (ECCV).

[2] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861.

[3] OpenCV Developers. (2023). OpenCV Documentation. Retrieved from https://docs.opencv.org

[4] TensorFlow Team. (2023). TensorFlow Object Detection API. Retrieved from https://github.com/tensorflow/models

[5] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Dollár, P. (2014). Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV)*.