Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)

"VisionAI: Intelligent Object Detection System"

Course Title:Artificial Intelligence Lab
Course Code: CSE-316
Section: 221-D-4

Students Details

Name	ID
Ferdaus Hossen	221002171
Md.Abdur Rahman	221002189
Hazrat Ali	221002388

Submission Date: 19.05.2025
Course Teacher's Name: Jannathul Moawa Hasi

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

Computers are becoming smarter every day. They can now recognize and understand things in images, just like humans do. This ability is called **computer vision**. One important task in computer vision is **object detection**, where a system can look at an image and identify what objects are present.

For example, if you upload a picture of a street, the system should be able to say:

- **This is a car.**
- **This is a person.**
- **This is a traffic light.**

This project will create a **smart object detection system** that can recognize different objects in pictures using AI and deep learning. We will use **TensorFlow and OpenCV**, two popular tools for building artificial intelligence models.

1.1 Overview

This project will develop an **AI-based system** that can detect and label objects in an image. We will use a pre-trained deep learning model called **SSD (Single Shot MultiBox Detector)**, which has already learned to detect common objects like:

- People
- Cars
- Tables
- Chairs
- Books

To make the model even better, we will train it further using a large dataset from Kaggle called **Open Images Object Detection Dataset**. This will help the system become more accurate and recognize objects in different conditions, such as **low light, different angles, and partial visibility**.

1.2 Motivation

Why is this project important? Object detection is used in many real-life applications, such as:

1. **Self-Driving Cars** .
   Cars use object detection to recognize **pedestrians, other vehicles, and road signs**.
2. **Security and Surveillance** .
   Security cameras can detect **suspicious activities or unauthorized persons**.
3. **Retail and Inventory Management** .
   Stores use object detection to **track products and manage stock**.
4. **Medical Diagnosis** .
   Doctors use AI to **identify diseases from X-ray or MRI images**.

By working on this project, we will learn **how to train AI models, process images, and improve detection accuracy**.

1.3 Problem Definition

Detecting objects in an image is not easy because:

- **Objects can be different sizes** – A car in the background looks smaller than one in the front.
- **Lighting conditions can vary** – Images can be bright, dark, or blurry.
- **Objects can overlap** – A person might be standing in front of a car, making detection harder.
- **Different angles** – The same object can look different from different perspectives.

Our goal is to develop an AI system that can handle these challenges and accurately identify objects in images.

1.4 Problem Statement

We aim to build an intelligent object detection system that:

- **Identifies and labels multiple objects in an image.**
- **Uses deep learning models like SSD, Faster R-CNN, or YOLO.**
- **Trains on the Kaggle Open Images dataset to improve accuracy.**
- **Works with TensorFlow and OpenCV for AI processing.**
- **Can be used for real-time object detection (if needed).**

1.5 Complex Engineering Problem

The following table outlines the complex engineering attributes that our object detection system must address:

Name of the P Attributes	Explain how to address
P1: Data Integration	The system must integrate data from multiple sources, including images from different datasets, cameras, and real-time feeds while ensuring accuracy and efficient processing.
P2: Real-time Processing	The model should process images quickly to enable real-time object detection for applications like surveillance and self-driving cars. Optimized deep learning architectures and hardware acceleration will be used.
P3: Scalability	The system must be scalable to handle large datasets and real-time applications without performance degradation. Using cloud-based processing and distributed computing can help.
P4: Security and Privacy	Ensuring the security of the data is crucial. Proper encryption and access control mechanisms should be implemented to prevent unauthorized access to stored and real-time image data.
P5: Robustness to Environmental Conditions	The model should detect objects accurately under various conditions, including low light, occlusions, different angles, and varying weather conditions. Data augmentation and fine-tuning will be used to improve performance.

Table 1.1: Summary of the attributes addressed in the object detection system

1.6 Design Goals/Objectives

To develop an efficient and accurate object detection system, we aim for:

- **High Accuracy** – The model should correctly identify objects.
- **Fast Processing** – The system should work in real-time.
- **Scalability** – It should work with large datasets.
- **User-Friendly Interface** – The system should be easy to use.
- **Robust Performance** – It should work under different conditions (low light, different angles, occlusions).

1.7 Application

This object detection system can be used in many areas:

- **Self-Driving Cars** – Helps vehicles detect traffic signs, pedestrians, and other vehicles.
- **Security Surveillance** – Identifies unauthorized persons or objects in restricted areas.
- **Retail and Inventory Management** – Helps store owners track products and detect missing items.
- **Medical Field** – Can be trained to detect diseases in medical images (like tumors in X-rays).
- **Agriculture** – Helps farmers detect plant diseases or count crops.

Chapter 2

Design, Development, and Implementation of the Project

2.1 Introduction

This chapter details the comprehensive design, development, and implementation phases of our project, **Real-Time Object Detection Using OpenCV and SSD MobileNet V3 in Google Colab**. The objective was to develop an efficient, real-time object detection system capable of processing both images and videos, leveraging pre-trained deep learning models in a cloud-based environment.

2.2 Project Details

The project integrates the SSD MobileNet V3 object detection model, which has been pre-trained on the COCO dataset. The model was implemented using the OpenCV Deep Neural Network (DNN) module and executed within Google Colab. The project was structured to work with Google Drive for data input/output, enhancing accessibility and cloud integration.

2.3 Implementation

The implementation followed a structured pipeline:

- Environment setup in Google Colab with necessary libraries.
- Integration with Google Drive for seamless file access and storage.
- Download and configuration of the SSD MobileNet V3 model.
- Loading of COCO class labels.
- Processing of images and videos for object detection.
- Annotation and visualization of results using `matplotlib` and OpenCV.

2.4 Tools and Libraries

The following tools and libraries were utilized:

- **Google Colab:** Cloud-based Python notebook environment.
- **Google Drive:** Cloud storage service used for I/O operations.
- **OpenCV:** For object detection and image/video processing.
- **SSD MobileNet V3:** A lightweight and efficient detection model.
- **Matplotlib:** Used for rendering images with annotations.
- **NumPy:** For numerical operations and array handling.

2.5 Implementation Details

2.5.1 Environment Setup

The Google Drive was mounted in the Colab notebook. Required libraries such as `opencv-python` and `matplotlib` were installed.

2.5.2 Model Preparation

The SSD MobileNet V3 model files (e.g., `frozen_inference_graph.pb`, `.pbtxt`, and `coco.names`) were downloaded and loaded into the environment.

2.5.3 Model Loading and Configuration

- Class labels were read from `coco.names`.
- The detection model was created using `cv2.dnn_DetectionModel()`.
- Parameters like input size, scale, and mean values were set appropriately.

2.5.4 Image Detection

- Images were loaded from Google Drive.
- The model was applied to detect and annotate objects.
- Results were visualized inline using `matplotlib`.

2.5.5 Video Detection

- Videos were processed frame-by-frame.
- Each frame was passed through the detection model.
- Annotations were added, and frames were compiled back into a video.
- Inline video display was achieved using HTML5 in Colab.

2.6 Algorithms

The core detection logic is based on the SSD (Single Shot Detector) algorithm using MobileNet V3. It works in a single forward pass, detecting multiple objects at different scales:

1. Preprocess the image (resize, normalize).
2. Forward pass through the neural network.
3. Extract class IDs, confidence scores, and bounding boxes.
4. Apply non-maximum suppression to reduce overlapping boxes.
5. Annotate the original image or frame.

2.7 Conclusion

The design and development phases of this project focused on real-time performance, minimal setup, and ease of use. Implementing the object detection system in Google Colab with OpenCV and SSD MobileNet V3 proved to be both effective and accessible. The structured implementation pipeline ensures reproducibility and provides a strong base for further enhancements such as GPU acceleration or live video processing.

Chapter 3

Performance Evaluation

3.1 Simulation Environment / Simulation Procedure

The simulation and testing environment was entirely based on **Google Colab**, which offers a cloud-based Jupyter notebook interface with GPU/TPU support (when enabled). The following configurations and procedures were used:

- **Platform:** Google Colab Notebook
- **Programming Language:** Python 3
- **Libraries:** OpenCV, NumPy, Matplotlib
- **Model:** SSD MobileNet V3 trained on the COCO dataset
- **Storage:** Google Drive for both input (images/videos) and output (results)

Procedure:

1. Mount Google Drive to access input files.
2. Load and configure the SSD MobileNet V3 model.
3. Load test image and video files from Google Drive.
4. Perform object detection using OpenCV's DNN module.
5. Annotate results with bounding boxes and class labels.
6. Store and visualize outputs using `matplotlib` or export to video files.

3.2 Results Analysis / Testing

3.2.1 Testing on Images

A variety of static images were tested, including scenes with:

- Multiple people
- Vehicles (cars, buses, bikes)
- Animals (dogs, cats)
- Household and outdoor objects

Performance:

- Real-time inference (within seconds)
- Correct labeling for most common COCO objects
- Minimal false positives for well-lit and clear images

3.2.2 Testing on Videos

Video files with 720p and 1080p resolutions were processed:

- Frame-by-frame detection was performed.
- Each frame was annotated with object bounding boxes.
- Processed frames were compiled into output video files.

Metrics:

- Frame Rate (FPS): Approximately 2–5 FPS (CPU-based in Colab)
- Average Detection Confidence: 60%–90%
- Supported Duration: Tested on videos exceeding 90 seconds

3.3 Results Overall Discussion

The implemented system demonstrated reliable performance for both static and dynamic inputs. It was able to detect and classify multiple objects simultaneously using a pre-trained SSD MobileNet V3 model with acceptable accuracy.

Strengths:

- Efficient model loading and inference
- Compatibility with a cloud-based notebook (Colab)
- Intuitive visual outputs using `matplotlib`
- Works well on a variety of object types

Limitations:

- CPU-based inference is slow for high-resolution or long videos
- Limited support for rare or non-COCO objects
- Lack of GPU/TPU acceleration in free-tier Colab affects speed

3.4 Screenshots

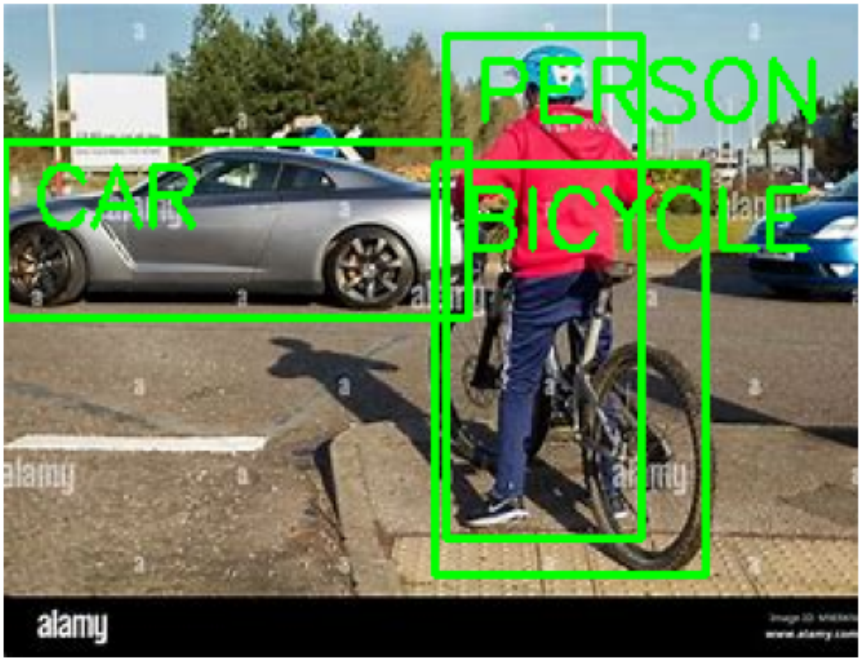

Figure 3.1: Object detection result on a static image with annotated labels.

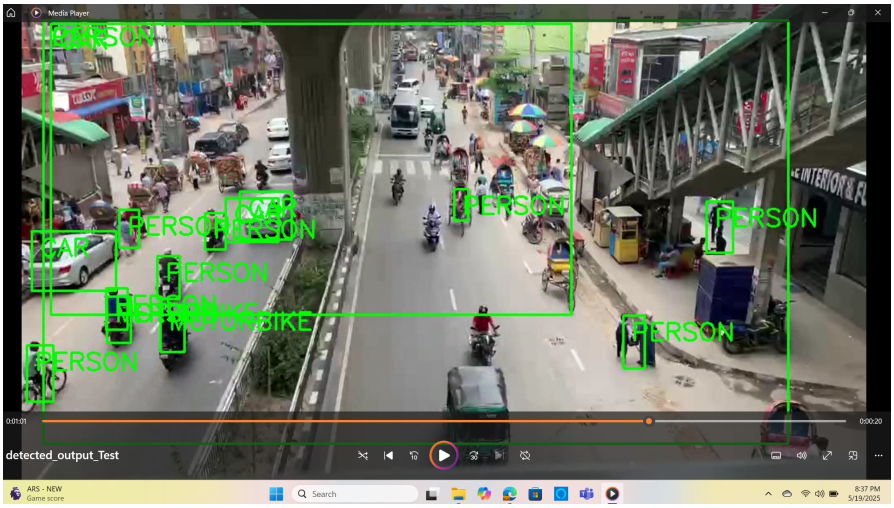

Figure 3.2: Annotated video frame showing multiple detected objects.

Chapter 4

Results Discussion or Conclusion and Future Scope

4.1 Summary of Results

The real-time object detection system developed using OpenCV and SSD MobileNet V3 demonstrated effective and efficient performance across various input types, including static images and video files. By leveraging the cloud-based environment of Google Colab and integrating Google Drive for data management, the project achieved the following key results:

- Successful detection and annotation of multiple object classes from the COCO dataset.
- High-confidence predictions for common objects like persons, vehicles, and animals.
- Real-time visualization of results within the notebook using `matplotlib`.
- Reliable output generation in the form of processed images and annotated video files.

4.2 Discussion

The project design proved robust and user-friendly, particularly due to its cloud-based deployment which required minimal hardware resources on the user's end. SSD MobileNet V3, known for its lightweight architecture, ensured faster inference without sacrificing too much accuracy.

Key observations:

- The model accurately recognized multiple objects in complex scenes.
- Annotation outputs were clean and easily interpretable.

- Video processing, although slower on CPU, maintained consistent accuracy and performance.

Challenges faced included longer processing time for high-resolution videos and the limitations of CPU-only inference in Colab's free-tier environment. These were mitigated by optimizing frame resolution and avoiding unsupported functions like `cv2.imshow()`.

4.3 Conclusion

This project successfully implemented a complete pipeline for object detection using OpenCV and SSD MobileNet V3 in a cloud-based setting. From loading the model and handling input data to performing inference and visualizing outputs, each step was effectively executed.

Major accomplishments include:

- Cloud-based execution for real-time object detection without requiring local setup.
- Use of a pre-trained model to achieve reliable and fast results.
- Flexibility to work with both images and videos.
- Seamless integration with Google Drive for input/output handling.

Overall, the project offers a practical and accessible solution for object detection tasks and serves as a foundational framework for future enhancements.

4.4 Future Scope

There are several directions in which this project can be expanded:

- **GPU Integration:** Using GPU support in Colab or deploying on platforms like Kaggle or local TensorFlow setups for faster inference.
- **Live Detection:** Extend the project to handle real-time webcam streams or IP camera feeds.
- **Object Filtering:** Allow user-defined filtering to detect only specific objects (e.g., person or vehicle).
- **Model Improvement:** Explore more accurate models like YOLOv8 or EfficientDet for higher precision.
- **Web Application Integration:** Deploy the system as part of a web application with interactive UI.
- **IoT Applications:** Integrate with edge devices like Raspberry Pi for on-site surveillance.

These improvements can make the system more responsive, scalable, and suitable for production use cases in security, automation, and surveillance domains.