(Annexure-I)

A Project Report

On

Real Time Object Detection using python and OpenCV/Tensorflow

Submitted in partial fulfillment of the requirements for the award of the degree of

Master of Computer Applications

In

Session: 2020-2022

Submitted by Supervised By

Rongsensashi Walling Prabhsharan Kaur

UID: 20MCA1510 Assistant professor

Dept of University Institute of Computing

Chandigarh University



University Institute of Computing

CHANDIGARH UNIVERSITY, GHARUAN, MOHALI, PUNJAB, 140413

Project Coordinator Head-UIC

**ACKNOWLEDGEMENT**

I would like to give my heartfelt gratitude to *Satnam Singh Sandhu*, *Chancellor*, *Chandigarh University*, for providing me all the facilities and support and thereby giving this great opportunity for enhancing and developing my talent through this seminar report.

I would like to thank our department HOD, Manish Malhotra and all the teaching faculty members of University Institute of Computing Department for providing me with all the necessary information needed during the development phase of the seminar report.

Also I would also like to give my heartfelt gratitude to my *Internal Guide* *Prabhsharan Kaur*, *Assistant Professor, Department of University Institute Of Computing*, *Chandigarh University* for helping me out in doing this seminar report.

Rongsensashi Walling

**SELF CERTIFICATE**

I hereby declare that the work on the project titled **“Real Time Object Detection using python and OpenCV/Tensorflow”** has been carried out by me under the Internal Guidance of **Prabhsharan Kaur, Assistant Professor, Dept. of University Institute Of Computing, Chandigarh University** and submitted in partial fulfillment for qualifying the Project Reportin the final year of the MCA program (Class of 2020 – 22).

I also declare hereby that any or all contents incorporated in the report has not been submitted in any form whatsoever for any other degree or diploma under any other university.

Rongsensashi Walling

University Enrollment No: 20MCA1510

**INTERNAL GUIDE’S CERTIFICATE**

This is to certify that **Mr. Rongsensashi Walling,** bearing **University Enrollment No. 20MCA1510** has carried out the Projecttitled **“Real Time Object Detection using python and OpenCV/Tensorflow”** under my *Internal Supervision* and *Guidance* during *Aug-Nov 2021*.

The said Project report work has not been submitted for the purpose of any other degree or diploma given by any University.

This Project report has not been submitted for any other examination and does not form any part of any other course(s) undergone by the candidate.

Dated: **07/11 /2021**

Place: **Chandigarh**

**Prabhsharan Kaur**

**Assistant Professor,**

**Dept. University Institute of Computing**

**Chandigarh University**

**Preface**

Humans can easily detect and identify objects present in an image. The human visual system is fast and accurate and can perform complex tasks like identifying multiple objects and detect obstacles with little conscious thought. With the availability of large amounts of data, faster GPUs, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy. In this project, we will explore terms such as object detection, object localization, loss function for object detection and localization, and finally explore an object detection algorithm using Python and OpenCV.

**Abstract.**

An accurate object recognition/detection based on image processing using real time video footage is required to be embedded in various applications, where the real-time image processing is expected to recognize the image accurately. To achieve object recognition in real time, an accurate recognition algorithm needs to be implemented in the system so that it can process the data and categorize the objects accordingly. In this project, we implemented an accurate recognition algorithm with good amount of accuracy and speed. We show that the program can achieve real-time object recognition on QVGA video at 22 fps with multiple objects. Furthermore, we constructed a real-time object recognition system using OpenCV, one of the most widely used object recognition algorithm, and demonstrated object recognition with it.

Contents

[1. Introduction 1](#_Toc88137277)

[ Background 1](#_Toc88137278)

[ Features of the project 3](#_Toc88137279)

[ Objectives of the Project 4](#_Toc88137280)

[ Purpose 5](#_Toc88137281)

[ Scope 6](#_Toc88137282)

[2. System Analysis 7](#_Toc88137283)

[ Need for identification 7](#_Toc88137284)

[ Preliminary Investigation 7](#_Toc88137285)

[ Feasibility Study 8](#_Toc88137286)

[ For a successful software project the following steps can be followed 8](#_Toc88137287)

[ Gaps of Existing System 10](#_Toc88137288)

[ Dual priorities: object classification and localization 10](#_Toc88137289)

[ Speed for real-time object detection 10](#_Toc88137290)

[ Limited data 10](#_Toc88137291)

[ Class imbalance 11](#_Toc88137292)

[ Planning and Requirement Gathering 12](#_Toc88137293)

[ Project Scheduling 13](#_Toc88137294)

[Table: For Gantt chart 13](#_Toc88137295)

[Fig: Pert chart 13](#_Toc88137296)

[ ER Diagrams of tables created 14](#_Toc88137297)

[Fig 13.1: ER Diagram for Object Recognition 14](#_Toc88137298)

[ Software Requirement Specification 15](#_Toc88137299)

[ Tools, Hardware and Software Specification 16](#_Toc88137300)

[ Technologies Used 17](#_Toc88137301)

[ Python 17](#_Toc88137302)

[ OpenCV 17](#_Toc88137303)

[ Mobilenet-SSD 18](#_Toc88137304)

[ PyCharm 18](#_Toc88137305)

[ Data Flow Diagram (Level-0, Level-1, Level-2) 19](#_Toc88137306)

[ Object Recognition System DFD Level-0 19](#_Toc88137307)

[Fig 9.1: Level-0 DFD for Object Recognition 19](#_Toc88137308)

[ Object Recognition System DFD Level-1 20](#_Toc88137309)

[Fig 9.2: Level-1 DFD for Object Recognition 20](#_Toc88137310)

[ Object Recognition System DFD Level-2 21](#_Toc88137311)

[Fig 9.3: Level-2 DFD for Object Recognition 21](#_Toc88137312)

[3. System Design 22](#_Toc88137313)

[ Modularization details 22](#_Toc88137314)

[Fig 12.1: Screenshot 22](#_Toc88137315)

[Fig 12.2: Screenshot 23](#_Toc88137316)

[Fig 12.3: Screenshot 23](#_Toc88137317)

[Fig 12.4: Screenshot 24](#_Toc88137318)

[Fig 12.5: Screenshot 24](#_Toc88137319)

[Fig 12.6: Screenshot 25](#_Toc88137320)

[Fig 12.7: Screenshot 25](#_Toc88137321)

[Fig 12.8: Screenshot 26](#_Toc88137322)

[Fig 12.9: Screenshot 26](#_Toc88137323)

[Fig 12.10: Screenshot 27](#_Toc88137324)

[ User Interface Design 28](#_Toc88137325)

[ Test Cases (Unit Test Cases and System Test Cases) 28](#_Toc88137326)

[4. Coding 30](#_Toc88137327)

[ Complete Project Coding 30](#_Toc88137328)

[ main.py 30](#_Toc88137329)

[ coco.names 31](#_Toc88137330)

[ ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt 33](#_Toc88137331)

[ Comments and Description of Coding segments 43](#_Toc88137332)

[5. Testing 44](#_Toc88137333)

[ Testing Strategy used & description of test cases and use cases 44](#_Toc88137334)

[ One-stage vs. two-stage deep learning object detectors 47](#_Toc88137335)

[ Work Progress as per timeline 48](#_Toc88137336)

[ Number of Objectives Achieved 49](#_Toc88137337)

[6. Documentation 50](#_Toc88137338)

[ Cost Estimation of the Project along with Cost Estimation Model. 50](#_Toc88137339)

[7. Conclusion and Future Scope 51](#_Toc88137340)

[ Conclusion 51](#_Toc88137341)

[ Future scope and further enhancement of the Project 52](#_Toc88137342)

[References 53](#_Toc88137343)

[Plagiarism Report 55](#_Toc88137344)

[Github Repository 56](#_Toc88137345)

# Introduction

Humans can easily detect and identify objects present in an image. The human visual system is fast and accurate and can perform complex tasks like identifying multiple objects and detect obstacles with little conscious thought. With the availability of large amounts of data, faster GPUs, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy. In this blog, we will explore terms such as object detection, object localization, loss function for object detection and localization, and finally explore an object detection algorithm.

## Background

We all know that humans can easily detect and identify objects present in an image or in real time scenarios. The human visual system is fast and accurate compared to AI computer vision used by computers and can perform complex tasks like identifying multiple objects at the same time using our peripheral vision and detect obstacles with little to no conscious thought as compared to other beings. With the availability of large amounts of data (increasing day by day), faster GPUs, and better algorithms, we can easily train computers to detect and distinguish between multiple objects present on an image with high speed and accuracy. In this project, we will go through topics such as object detection, object localization, and finally run and test a project to see how it runs and check its performance.

Object detection aims at recognizing and locating all known objects in a given image or in real time footage. Most preferably in a 3-Dimensional environment as recovering a 3D pose of an object is very important for robotic control systems and AI learning. Passing on intelligence to machines and making computers more and more autonomous and AI driven has been a technological dream of mankind. It is our dream to let the machines take on our tiresome, repetitive, and hazardous tasks so that we can focus our time on more creative tasks which requires human interaction. Unluckily, the intelligent part seems to be still lagging behind and cannot be fully entrusted. Speaking of present day scenario, to attain this goal, we need to work on both the hardware and software so that we can enable computers to do the work with little to no human interactions or in other words, act independently. One of the key components regarding this project is providing the computer with vision, apart from other internal factors like, learning and cognitive thinking. A computer cannot be considered as intelligent if it cannot see and adapt to a dynamic environment surrounding it or comprehend the data that is provided. The recognition process used by computers in real time scenario is very difficult and far from perfection. Till date, There is no effective solution for this problem. Despite after a lot of research in this field, the methods used and developed so far are not efficient, time consuming, and the problems only increases with the implementation of real time applications, and are most often limited to a small class. Object detection is relatively effortless if the computer is looking for a particular object. However, recognizing all the objects in a frame automatically requires the skill to differentiate one object from the other which in some cases, may be quiet similar in appearance. Such problem is very difficult for computers to distinguish, if they are not provided with the endless possibilities of objects in the universe.

## Features of the project

Some of the features of Object detection are;

1. Object detection can be used to name objects that are unknown to a person or may not know the name of a particular given object
2. It can be used to read road signs which many at times, only have a board with a certain type of logo/image which the most people are unaware of what it means.
3. It can be use for identifying plants and distinguishing it from poisonous and safe/medicinal plants.
4. Real time object detection which use video footage can be used to estimate the number of people in a crowd
5. Object detection is also implemented in automated self-driving cars.  An AI powered self-driving vehicle can only be able to navigate safely through the streets if it could detect all the objects such as people, road signs on the road, other vehicles, in order to decide what action to take and navigate through the streets safely.
6. It can be used in security and surveillance as object detection would play an important role in tracking vehicles during a car chase.
7. With object tracking it would be easier to track a person in a video where a camera has to keep focus on a person in a football match.
8. In the field of traffic monitoring too object tracking plays a crucial role as it can keep count of how many vehicles passes through a particular location
9. It helps in identifying/detecting infected crops and helps the farmers take measures accordingly.
10. It could also help identify various skin problems/diseases in healthcare. Although it needs more research and advancement in the medical field

## Objectives of the Project

1. Object detection aims to detect all elements present in an image from a known class, such as phones, people, vehicles etc.
2. The object that it detects is reported with some form of class information. It is then categorized to which class it belongs and determines what kind of object it is.
3. In some other situations, the information provided in the form of an image is more detailed and contains the parameters of a linear or non-linear transformation
4. The face detection in a face detector may compute the locations of the eyes, nose and mouth, in addition to the bounding box of the face may be different from bicycle detection in an image that specifies the locations of certain part. The image can also be defined by a three-dimensional transformation specifying the location of the object relative to the camera.
5. Object detection systems always construct a model for an object class from a set of training examples.
6. In the case of a fixed rigid object in an image, only one example may be needed, but more generally multiple training examples are necessary to capture certain aspects of class variability that includes more than one object in one given image.
7. The gaming industry has started using object recognition technology along with AR to their advantage which provides gamers with a realistic experience.
8. The Image recognition system can be used to add great value in the educational sector by enabling students with learning disabilities to register knowledge in a way that is easier for them.

## Purpose

The ability for computer vision to distinguish between objects when it is in motion is an important feature of object detection, mainly in video surveillance. Since we are limited by our present technology and resources, the detection of objects in motion using Convolutional Neural Networks still remains a challenge. In this project, we aim to improving efficiency, reliability and accuracy of the detection of moving objects. We can achieve this by using a 2-level methodology based upon deep learning, also known as Object Detection with Binary Classifiers. The first level takes the candidate regions from the given frame and the second level uses a technique called “Binarization” based on a CNN classifier.

Most AI and AR based computer tasks are based on computer vision and object detection. Such problems can be solved by using modern deep learning models especially when the objects are different in a frame for example, different color, size, shape and distance. However, this task becomes more complicated when the targeted objects are far or not clearly visible which may be caused by poor camera quality or reduced number of pixels. Till date, the most efficient models of object detection are based on deep CNN as these models automatically learn the distinctive features of objects from a large set of labeled data and hope to implement this model in this project.

## Scope

There are many optimizations that could be made in this project. For example, the project can further be implemented and modified into other projects that require face recognition for unlocking phones without the use of passwords. And furthermore, the code can be optimized as per user preference and add or remove features accordingly. With more time and effort, this project could perhaps run and function smoothly with the introduction of new technology. Another feature that could be added is to allow the computer to scan and detect only a particular part of an image and not the entire image thus increasing speed and accuracy. This however, needs to be tested first as it may backfire and result in more time consuming and errors while showing result. The main objective for now is to detect simple objects and distinguish according to their class and not go deeper for industrial or medical level perfection. Perfecting it will most certainly take time and we can speed it up by using more powerful computers with better hardware and faster software. We can also make use of more than one computer allowing for more processing to be done per frame. One example is to install a good quality camera which will ultimately improve the real time footage captured and processed by the computer. Better video quality means more frame rate and higher resolution images. This would enable the computer to filter out non-relevant objects from the image and also provide a fast tracking while using real time object tracking.

# System Analysis

## Need for identification

Considering the growing demand of computer vision, many organizations are investing in image recognition to interpret and analyze data coming primarily from visual sources for a number of uses such as medical image analysis, identifying objects in autonomous cars, face detection for security purpose, etc. **Image recognition is the ability of a system or software to identify objects, people, places, and actions in images.** It uses machine vision technologies with artificial intelligence and trained algorithms to recognize images through a camera system. Much fuelled by the recent advancements in machine learning and an increase in the computational power of the machines, image recognition has taken the world by storm.

## Preliminary Investigation

A wide range of [computer vision applications](https://viso.ai/applications/computer-vision-applications/) has become available for object detection and tracking. As a result, numerous real-world applications, such as healthcare monitoring, autonomous driving, video surveillance, anomaly detection, or robot vision, are based on deep learning object detection.

Imaging technology has greatly progressed in recent years. Cameras are smaller, cheaper, and of higher quality than ever before. Meanwhile, computing power has dramatically increased and became much more efficient. In past years, computing platforms moved toward parallelization through multi-core processing, graphical processing unit (GPU), and AI accelerators such as tensor processing units (TPU)

Such hardware allows to perform computer vision for object detection and tracking in near real-time implementations. Hence, rapid development in deep convolution neural networks (CNN) and GPU’s enhanced computing power are the main drivers behind the great advancement of computer vision based object detection.

## Feasibility Study

As per present availability of resources and technology, object detection is feasible to a certain extend but needs more development to be implemented into day to day life. For smaller projects, we can most definitely make use of the resources available and go ahead to test and observe what are the possibilities and scope of the project in the future. Although we may not get expected outcome or return on our investment, we can always improve and work on our project to make it more efficient in the future. Since we can make use of open source resources like OpenCV and Tensorflow, it makes our task much easier as they have almost of the resources that we require to develop and run the project and do not need to pay to use their resources. To run this program the hardware does not need to be of industrial level, a standard computer machine with up-to-date features serves as a moderate environment to test and run the program. As data is becoming more and more prevalent, we can use that to our advantage with all kinds of technologies and discussion forums online for new ideas and innovations. It is safe to say that the demand in market and corporation for object detection and its applications will only increase with more and more people are opting for AI oriented tasks and machines.

## For a successful software project the following steps can be followed

* Select a project
* Identifying project’s aims and objectives.
* Understanding requirements and specification.
* Methods of analysis, design and implementation
* Testing techniques.
* Documentation
* Project milestones and deliverables
* Budget allocation
* Exceeding limits within control
* Project estimates
* Cost
* Time
* Size of code
* Duration
* Resource allocation
* Hardware
* Software
* Previous relevant project information
* Risk management
* Risk avoidance
* Risk detection

## Gaps of Existing System

The goal of object detection is to identify different objects and draw rectangular boxes around the object, and determine the type of object that it discovers. The operation of object detection arise in [**many different areas**](https://www.quora.com/What-are-some-interesting-applications-of-object-detection)including detecting people on the road for self driving AI vehicles, scanning agricultural crops, and it is used even in sports for tracking real-time ball movement.

Some of the major challenges faced by object detection are as follows:

## Dual priorities: object classification and localization

One of the major complications that object detection faces is its multiple task at the same time: the computer not only have to classify image objects and its class but also need to determine the objects’ current positions in real time, which is referred to as the object localization task. To get rid of this problem, researchers mostly use a technique called the multi-task loss function to address both misclassifications of objects and localization errors.

## Speed for real-time object detection

Object detection algorithms need to accurately classify and localize important objects as well as they need to be really fast at prediction the time to meet the real-time demands of video processing that requires a fast computer with high processing speed. Improvement of technology over the years have drastically increased the speed of these algorithms with better processors and better camera that provides the computer with better quality video footage.

## Limited data

The limited amount of data currently available for object detection and its application is also one of the major drawbacks. Object detection datasets mostly contain ground truth examples ranging from about few to a couple of hundred classes of objects, on the other hand, image classification datasets can have more than 100,000 known classes or even more. Although, open crowd-sourcing and communities often provide image classification tags for free, gathering ground truth labels along with accurate bounding boxes for object detection still remains incredibly boring and time consuming work.

## Class imbalance

A class imbalance in tags proves to be a major issue for most object classification problems, and the issue only becomes severe with the introduction of object detection in real time. One object may belong to a specific class and other similar object may belong to other class, but with the lack of data, the computer cannot differentiate between the two classes and reads it as one, thus giving a false output.

## Planning and Requirement Gathering

For a project to be successful, we must plan ahead and plan properly to avoid any problem that may arise while implementing the project. We must also keep in mind, the time frame allotted for each set of task which may include study, testing, implementation, debugging etc. It is better to separate weeks for specific tasks and try to finish within the given time frame.

For this project, brief case study was conducted to check the possible outcome and to check for hardware and software requirements. We also checked the current technology available and if it is possible to execute the project with low cost. Testing and running the code comes after the case study. We plan to make this project as simple as possible and just work on its functionality for now and not aim to implement it in industries. Just detecting and recognizing simple objects fulfills of this project and we can improve the project further in the future if we want. The scope of Object detection is endless ranging from small AI toy robots to applications in space exploration. The project will take approximately around 3-4 months to complete for testing and running and hope to make modifications and improvement in the coming days.

## Project Scheduling

An elementary Gantt chart or Timeline chart for the development plan is given below.

The plain explains the tasks versus the time (in weeks) they will take to complete.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | August | | | | | September | | | | | October | | | | | |  |
| Requirement Gathering |  | |  | | |  | | | | |  | | | | | |
| Analysis |  | |  | | |  | | | | |  | | | | | |
| Design |  | | | | |  |  | | | |  | | | | | |
| Coding |  | | | | |  |  | | | |  |  | | | | |
| Testing |  | | | | |  | | | | |  |  | | |  | |
| Implement |  | | | | |  | | | | |  | | | |  | |
|  | W1 | W2 | | W3 | W4 | W1 | | W2 | W3 | W4 | W1 | | W2 | W3 | | W4 |

W=week

### Table: For Gantt chart

Requirement Gathering Design Testing

5

3

1

0 20 80

6

4

2

Analysis 10 Coding 40 Implement 120

### Fig: Pert chart

## ER Diagrams of tables created

**RETRIVING THE STORED DATA**

### Fig 13.1: ER Diagram for Object Recognition

This ER diagram for object detection/recognition shows the overall model of the project. As this is not exported and just need to run on the IDE, it is fairly simple and the diagram depicts the overall hierarchy of the overall system.

## Software Requirement Specification

Minimum:

64-bit dual core 2 Ghz CPU with SSE2 support

Windows 7

1 GB free space on hard disk

4 GB RAM

1280×768 display

Graphics card with 1 GB RAM

Recommended:

64-bit quad core CPU

Windows 10

5 GB free space on hard disk

8 GB RAM

Full HD display

Graphics card with 4 GB RAM

Optimal:

64-bit quad core CPU

Windows 10

10 GB free space on hard disk

8 GB RAM

Full HD displays

Graphics card with +4 GB RAM

## Tools, Hardware and Software Specification

1. External camera(minimum 2MP)
2. Python 3.7 or higher
3. Install Image-AI and its dependencies like Tensorflow, OpenCV, etc.
4. Windows 10(preferred)
5. 32 or 64 bit OS
6. 4 GB RAM or more
7. 1 GB free space on Hard Disk

## Technologies Used

## Python

Python is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [high-level](https://en.wikipedia.org/wiki/High-level_programming_language) [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). Its design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with its use of [significant indentation](https://en.wikipedia.org/wiki/Off-side_rule). Its [language constructs](https://en.wikipedia.org/wiki/Language_construct) as well as its [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aim to help [programmers](https://en.wikipedia.org/wiki/Programmers) write clear, logical code for small and large-scale projects.

Python is [dynamically-typed](https://en.wikipedia.org/wiki/Type_system#DYNAMIC) and [garbage-collected](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [structured](https://en.wikipedia.org/wiki/Structured_programming) (particularly, [procedural](https://en.wikipedia.org/wiki/Procedural_programming)), object-oriented and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). It is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

[Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) began working on Python in the late 1980s, as a successor to the [ABC programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)), and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension) and a garbage collection system using [reference counting](https://en.wikipedia.org/wiki/Reference_counting). Python 3.0 was released in 2008 and was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility). Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages.

## OpenCV

OpenCV (Open Source Computer Vision Library) is a [library of programming functions](https://en.wikipedia.org/wiki/Library_(computing)) mainly aimed at real-time [computer vision](https://en.wikipedia.org/wiki/Computer_vision). Originally developed by [Intel](https://en.wikipedia.org/wiki/Intel_Corporation), it was later supported by [Willow Garage](https://en.wikipedia.org/wiki/Willow_Garage) then Itseez (which was later acquired by Intel). The library is [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) and free for use under the [open-source](https://en.wikipedia.org/wiki/Open-source_software) [Apache 2 License](https://en.wikipedia.org/wiki/Apache_License). Starting with 2011, OpenCV features GPU acceleration for real-time operations.

Officially launched in 1999 the OpenCV project was initially an [Intel Research](https://en.wikipedia.org/wiki/Intel_Research_Lablets) initiative to advance [CPU](https://en.wikipedia.org/wiki/Central_processing_unit)-intensive applications, part of a series of projects including [real-time](https://en.wikipedia.org/wiki/Real-time_computing) [ray tracing](https://en.wikipedia.org/wiki/Ray_tracing_(graphics)) and [3D display](https://en.wikipedia.org/wiki/3D_Display) walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described

## Mobilenet-SSD

The “ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt” is an external, open source downloadable files that is used in computer vision for object detection (The above given code for ssd\_mobilenet is less than half of the total code). The mobilenet-ssd model is a Single-Shot multi-box Detection (SSD) network intended to perform object detection. This model is implemented using the Caffe\* framework.

The model input is a blob that consists of a single image of 1, 3, 300, 300 in BGR order, also like the densenet-121 model. The BGR mean values need to be subtracted as follows: [127.5, 127.5, 127.5] before passing the image blob into the network. In addition, values must be divided by 0.007843.

For more information on ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt, visit the website <https://docs.openvino.ai/latest/omz_models_model_mobilenet_ssd.html>

## PyCharm

PyCharm is an [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) used in [computer programming](https://en.wikipedia.org/wiki/Computer_programming), specifically for the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) language. It is developed by the [Czech](https://en.wikipedia.org/wiki/Czech_Republic) company [JetBrains](https://en.wikipedia.org/wiki/JetBrains) (formerly known as IntelliJ). It provides code analysis, a graphical debugger, an integrated unit tester, integration with [version control systems](https://en.wikipedia.org/wiki/Revision_control) (VCSes), and supports web development with [Django](https://en.wikipedia.org/wiki/Django_(web_framework)) as well as [data science](https://en.wikipedia.org/wiki/Data_science) with [Anaconda](https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)).[[6]](https://en.wikipedia.org/wiki/PyCharm#cite_note-6)

PyCharm is [cross-platform](https://en.wikipedia.org/wiki/Cross-platform), with [Windows](https://en.wikipedia.org/wiki/Windows), [macOS](https://en.wikipedia.org/wiki/MacOS) and [Linux](https://en.wikipedia.org/wiki/Linux) versions. The Community Edition is released under the [Apache License](https://en.wikipedia.org/wiki/Apache_License), and there is also Professional Edition with extra features – released under a [proprietary license](https://en.wikipedia.org/wiki/Proprietary_software).

## Data Flow Diagram (Level-0, Level-1, Level-2)

## Object Recognition System DFD Level-0

To start with, let us familiarize what is Object Recognition System DFD level 0. The Object Recognition System DFD level 0 is also known as context diagram. It’s supposed to be an abstract view, with the mechanism represented as a single process with external parties.  
  
This DFD for the Object Recognition System depicts the overall structure as a single bubble. It comes with incoming/outgoing indicators showing input and output data.

Object Recognition Information

Object Recognition System

Trace Object Structure

System/User

Record/Count of Objects

### Fig 9.1: Level-0 DFD for Object Recognition

In this data flow diagram you will see the general process done in Object Recognition System monitoring. This will also serve as a guide as you go through the deeper processes of the Object Recognition System data flow diagrams.  
As you see, when you build the levels of data flow diagrams, the connections of the transactions and data also broaden and gets more specific.

## Object Recognition System DFD Level-1

Object Structure Object Confirmation in

Object Confirmation Object Confirmation out

**Manage Coming In/Out**

**System**

**Manage Object Recognition**

Object Confirmation Object

Confirmation in

**User**

Object StructureObject Confirmation out

Reports

**Generate Reports**

Object Recognition Records Coming In/Out Records

### Fig 9.2: Level-1 DFD for Object Recognition

These procedures require information such as record of familiar objects, unfamiliar objects and size or dimension of the object from which served as the bases for admin to manage the Object Recognition System. This type of data is represented by a data store. In addition to that, this may also serve as your reference on how the inputs or data fed on the system. Then you will be also informed about the outputs that the system gives.

## Object Recognition System DFD Level-2

**Manage Object Recognition**

Object Structure Object Structure

Object Structure Recognized Objects

Unrecognized Objects

**User**

**Object Detection Database**

Recognition In

**Manage Coming In/Out**

Recognition Out Object

Recognition

Records

Recognition in

**System**

**Reports Database**

Recognition Out

**Generate Reports**

Reports Reports

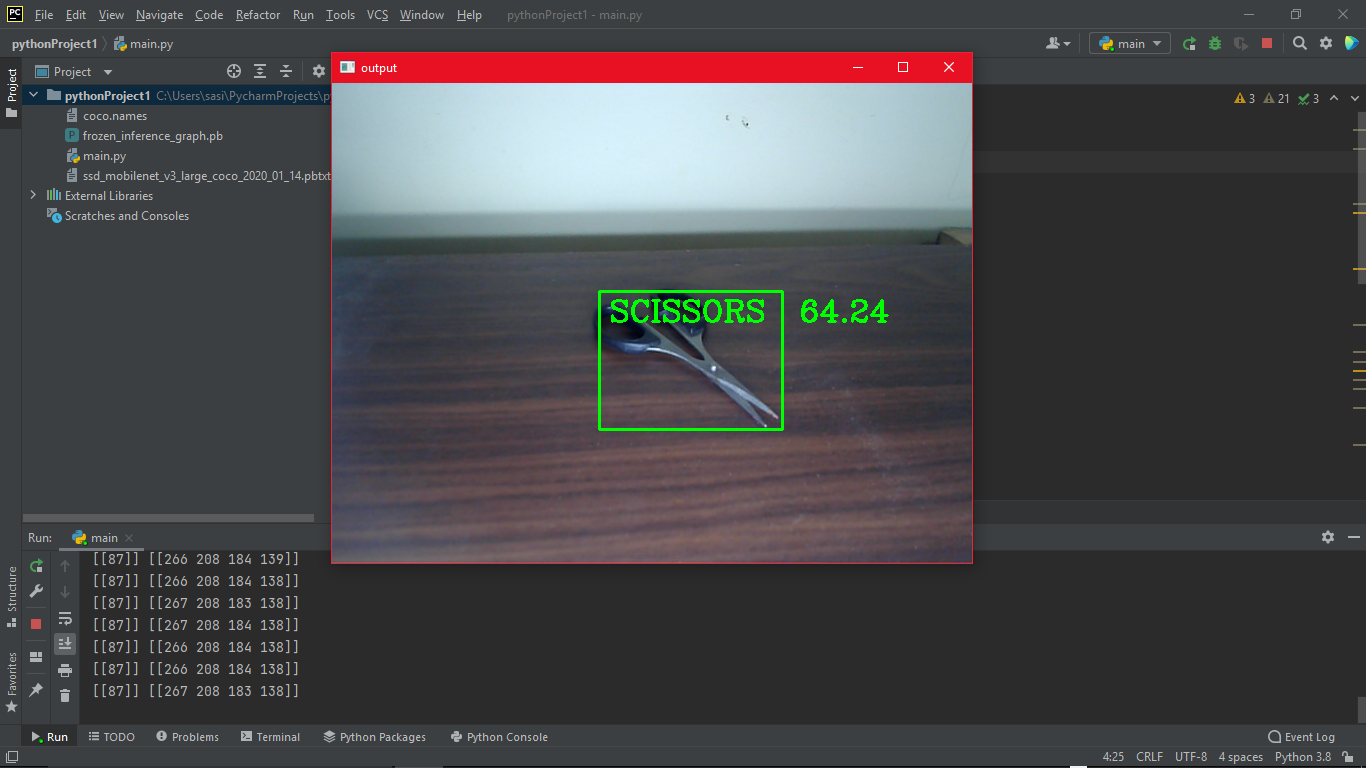
### Fig 9.3: Level-2 DFD for Object Recognition

DFD level 2 let’s you know the ideas on where does the data inputs goes and inputs comes within the Object Recognition System. Considering the dataflow levels mentioned above, you can determine well the importance of breaking the processes into more specific manner thus making it more efficient.

# System Design

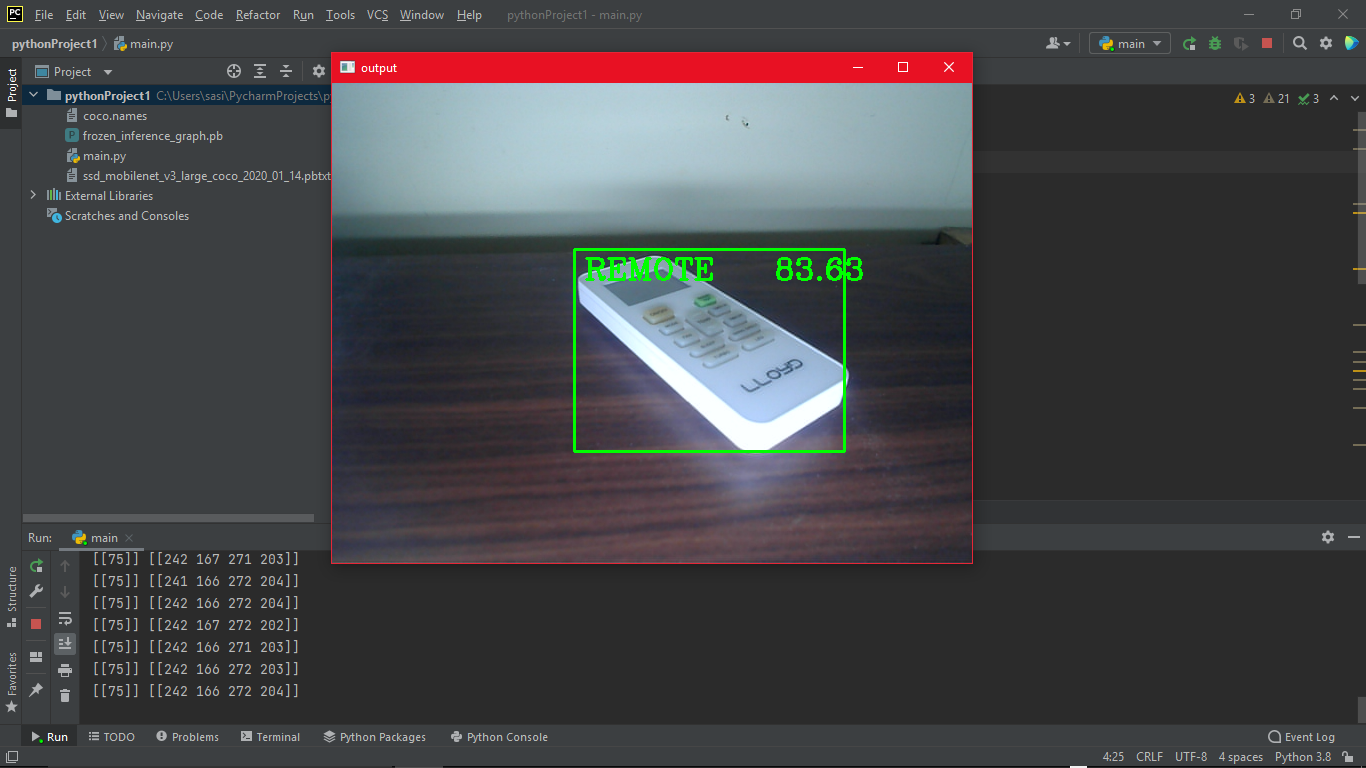
## Modularization details

The overall project is 90% complete and only requires minor changes. Even though, this project is in a running state, there are minor issues which will be stated below using the snapshots of the output modules.



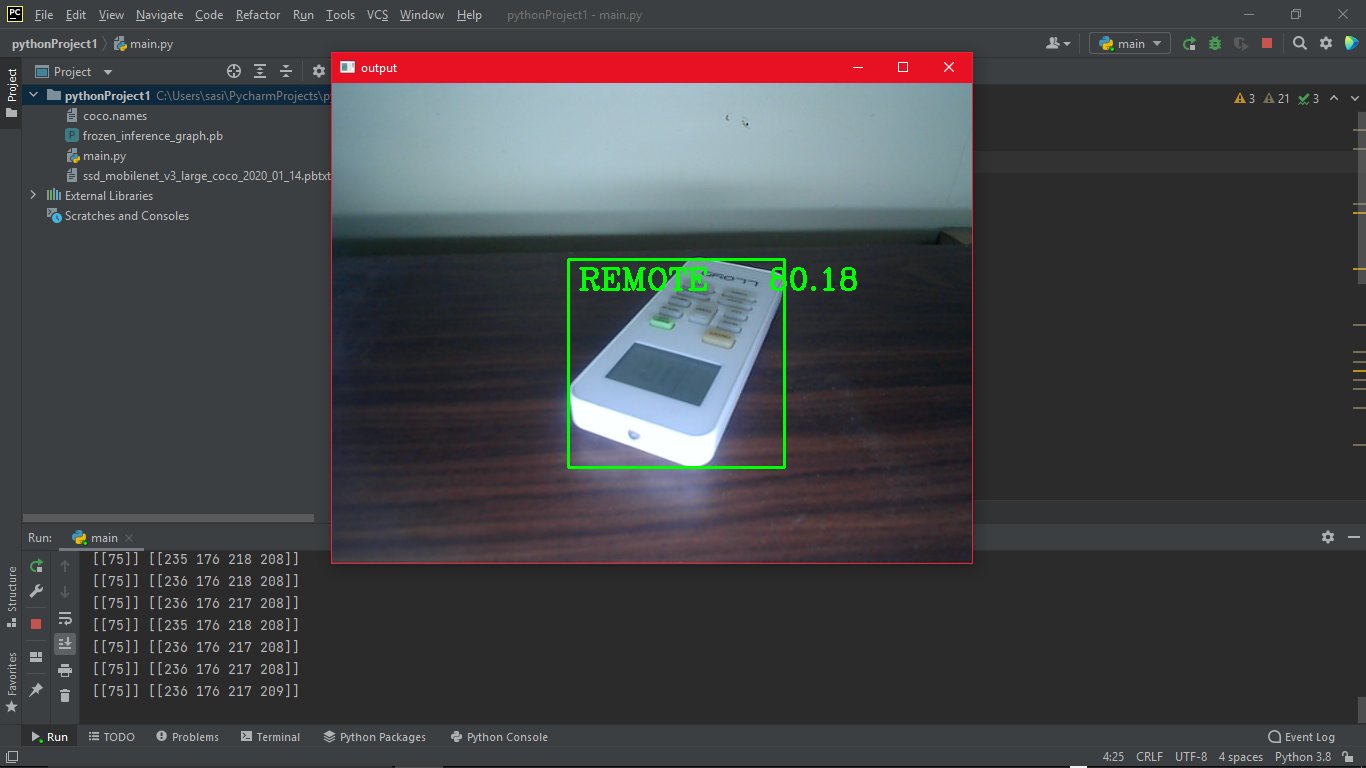
### Fig 12.1: Screenshot

In fig 12.1, the object detection seems to be running fine and as we can see, it detects the object in real time through the camera. This is the expected outcome with a bounding box around the detected object with the threshold value next to it. The threshold value shows the percentage of similarity to an object class to its stored data. The higher the threshold, the more accurate it is. We can change the lighting and angle of the camera to give the system a better visual input which will result in more accurate data. The object (Scissors) in the snapshot is a common item and the system was able to detect it easily but some issues will arise when the system detects an object that is unfamiliar or belongs to two or more object class.



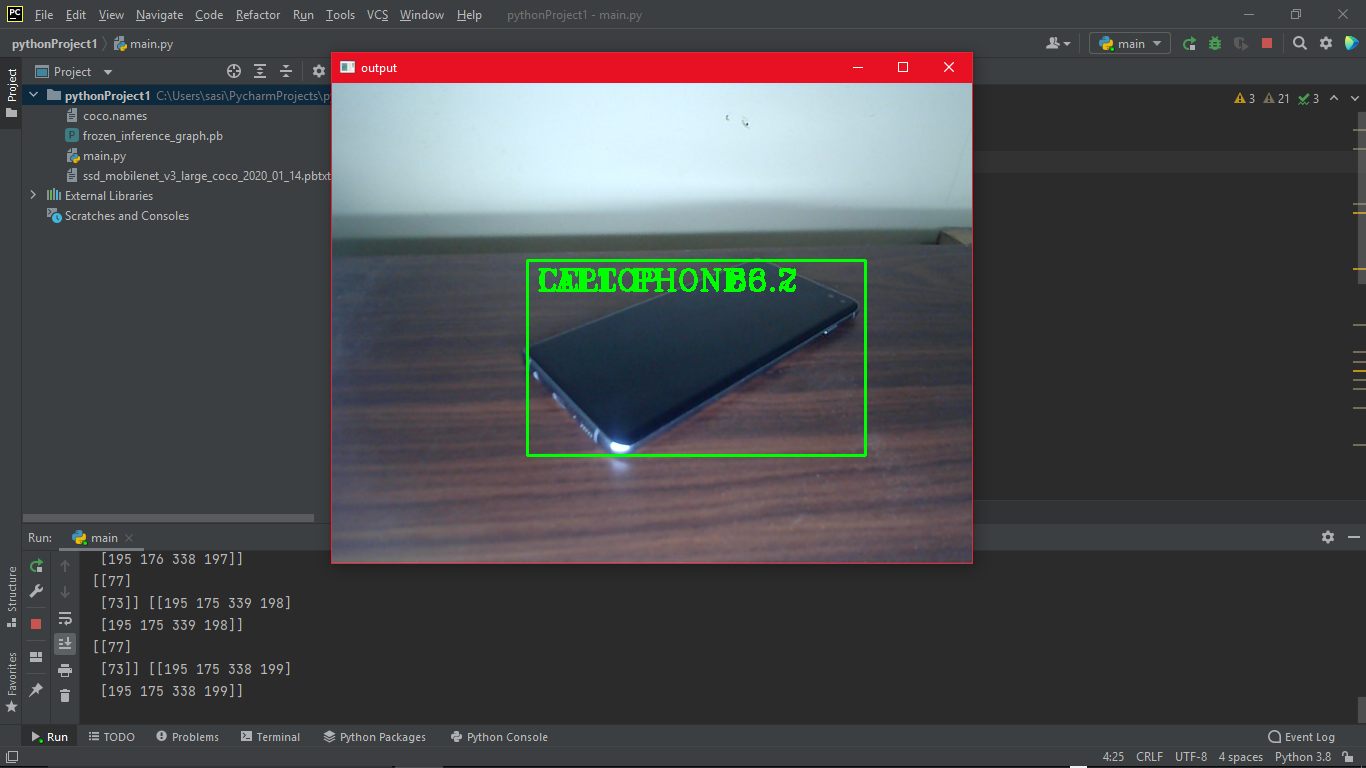
### Fig 12.2: Screenshot

Here in fig 12.2, the system detects an object with fairly more accuracy as we can see the threshold value is more as compared to the threshold value in fig 12.1.



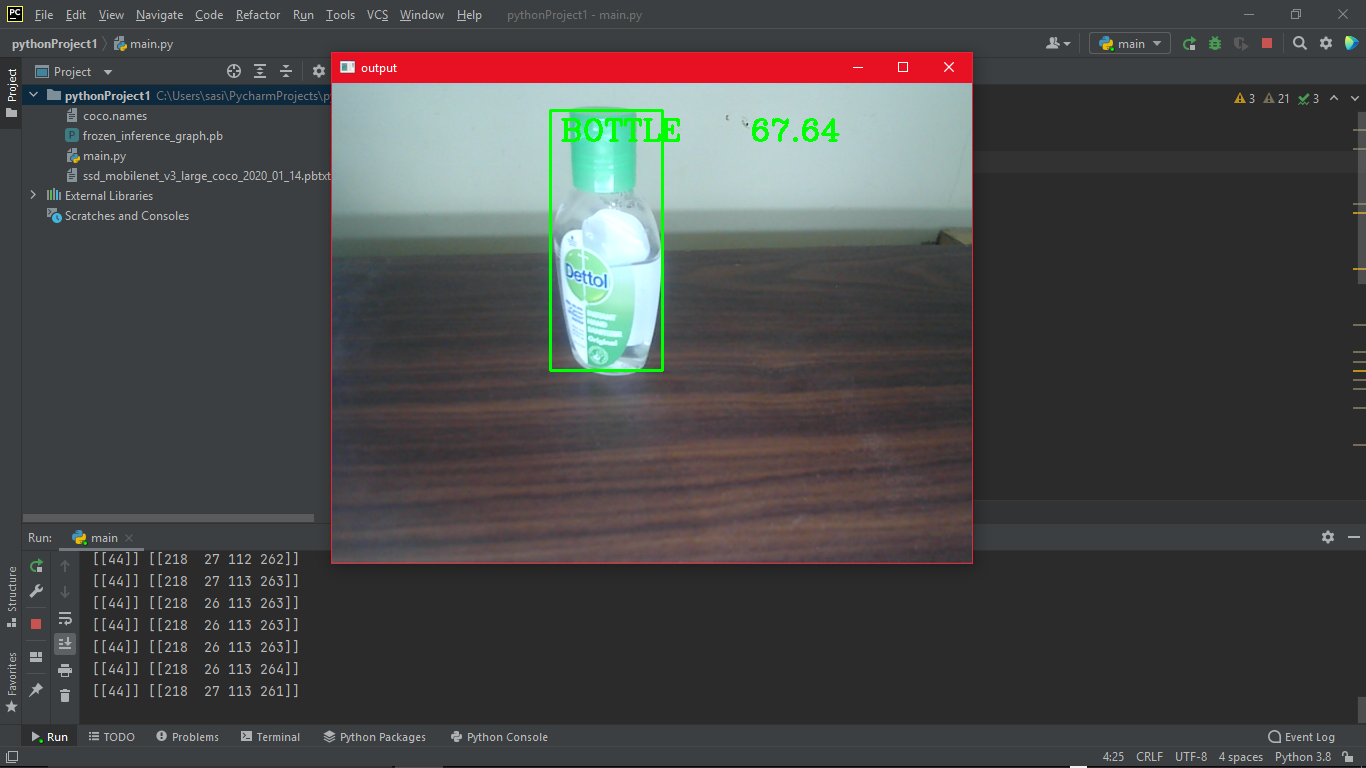
### Fig 12.3: Screenshot

The threshold value almost remains relatively same even if we change the position and angle of the object in fig 12.3. This shows that the system is able to detect everyday common objects fairly easily.

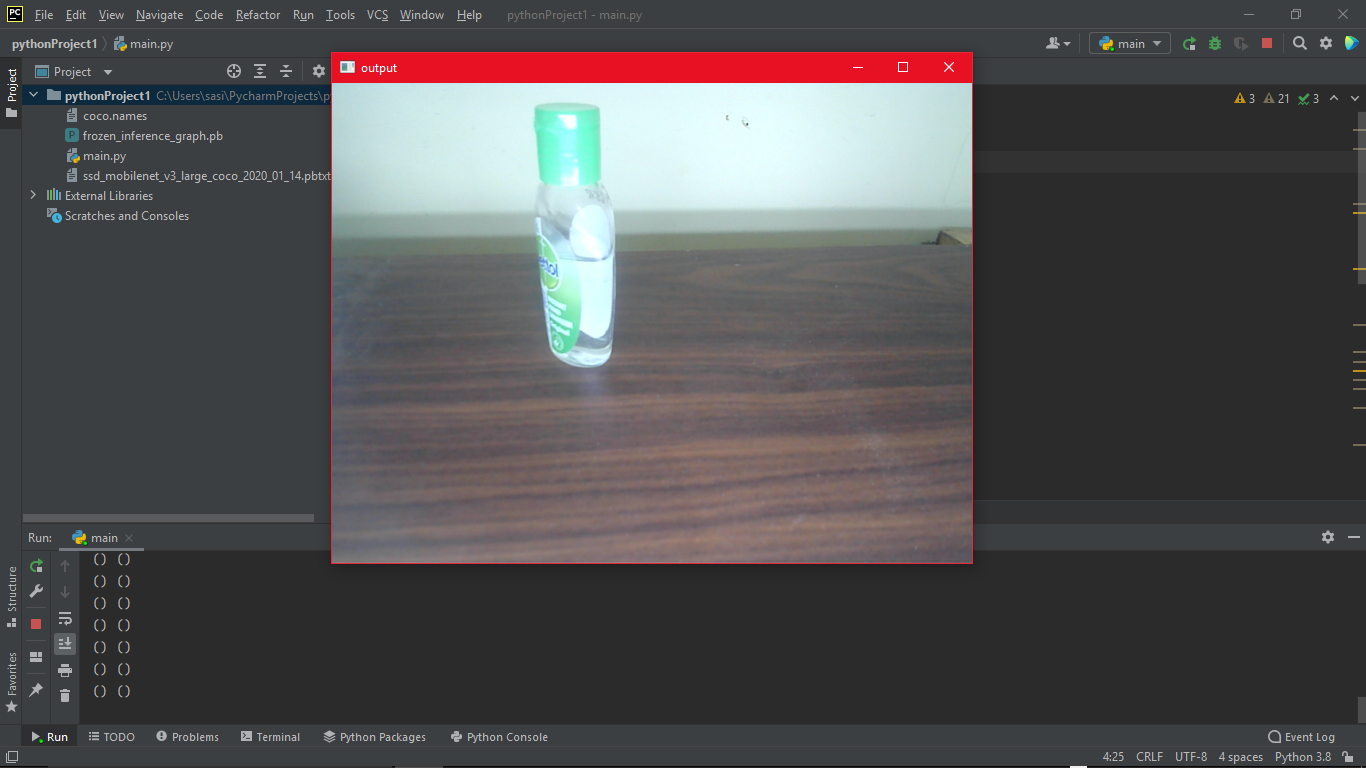
****

### Fig 12.4: Screenshot

In fig 12.4, we can see that it detects the object accurately but the name of the object and threshold value overlaps making it visually hard to read. We can change that by making small changes in the code that separates them further or by making the threshold value appear outside of the box.

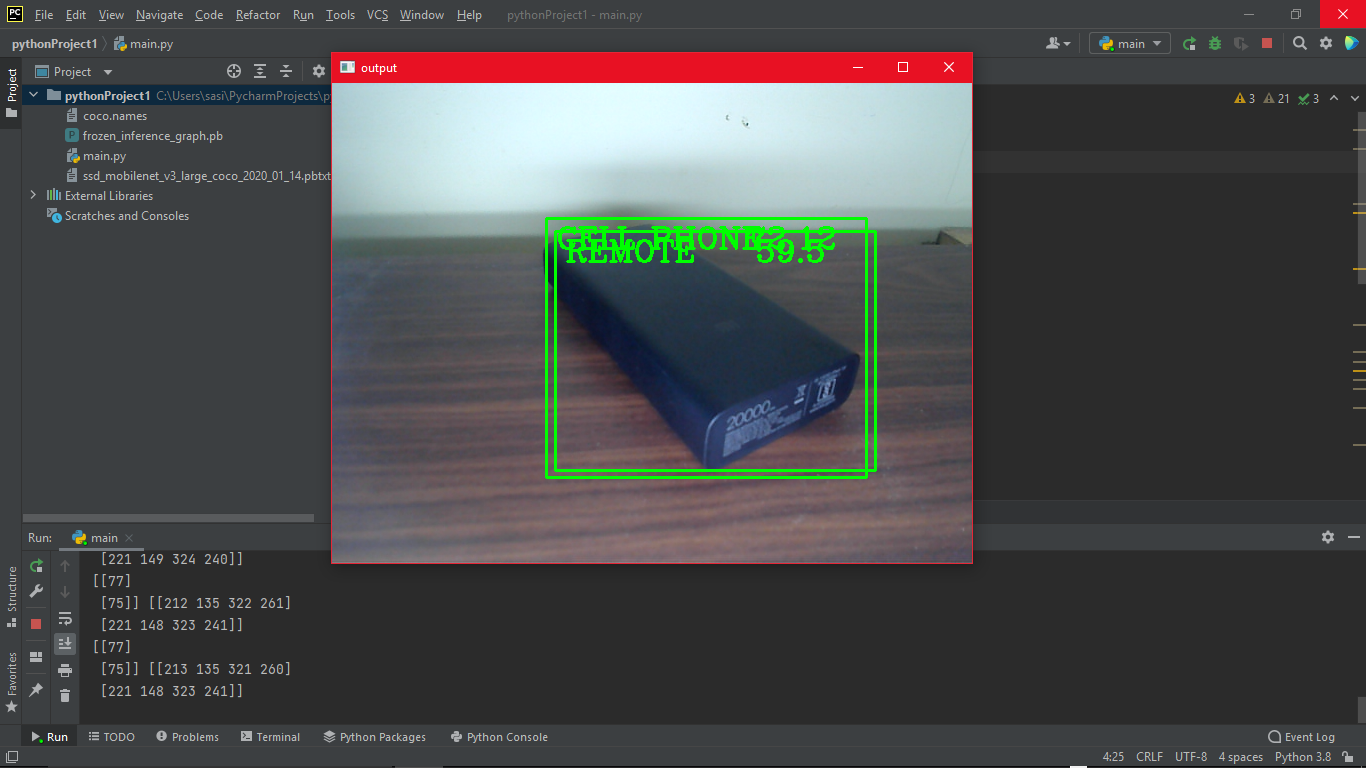


### Fig 12.5: Screenshot

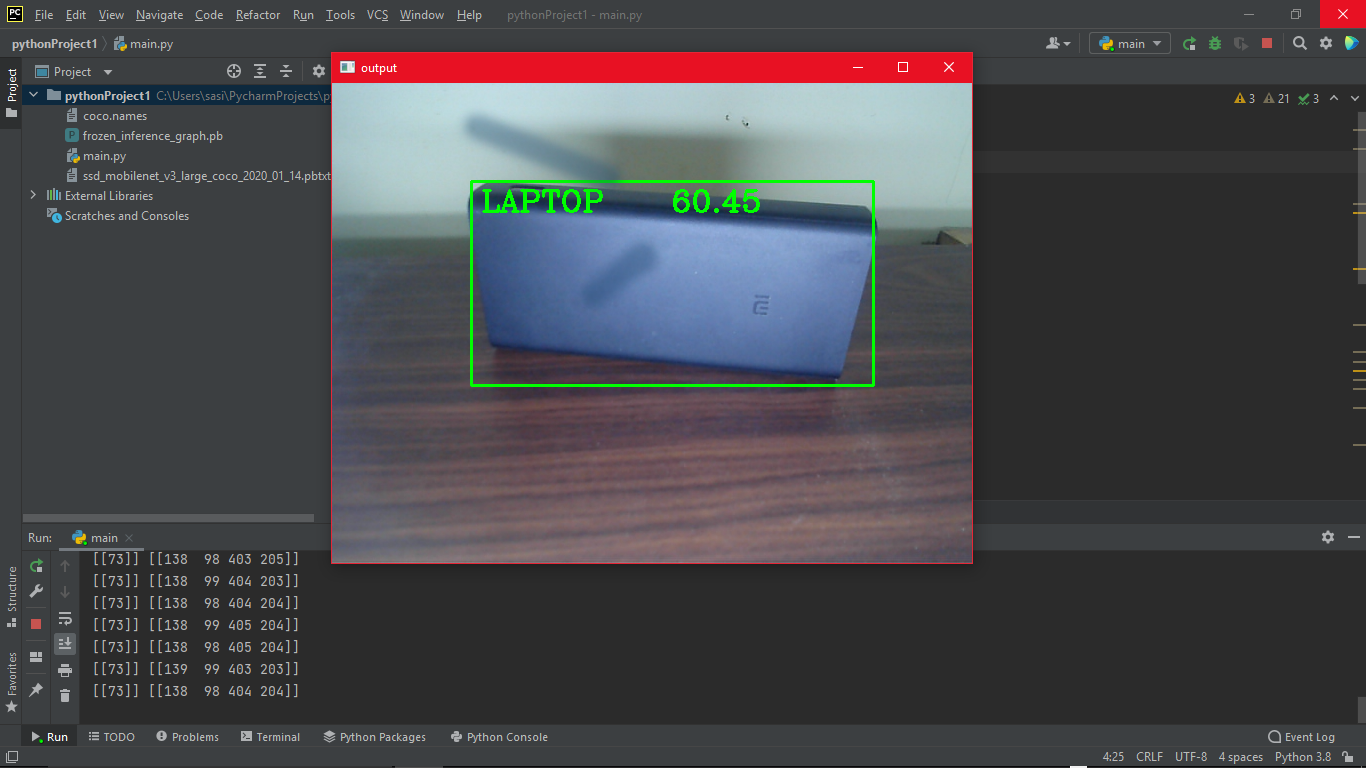


### Fig 12.6: Screenshot

In figure 12.5 and fig 12.6, we can see that the object is same but in different angle. The object is detected fairly accurately in the first figure but fails to detect it in the second. We can solve this by increasing or decreasing the lighting and changing the angle of the object. It is recommended to keep the object that you want to detect right in front of the camera for maximum efficiency.

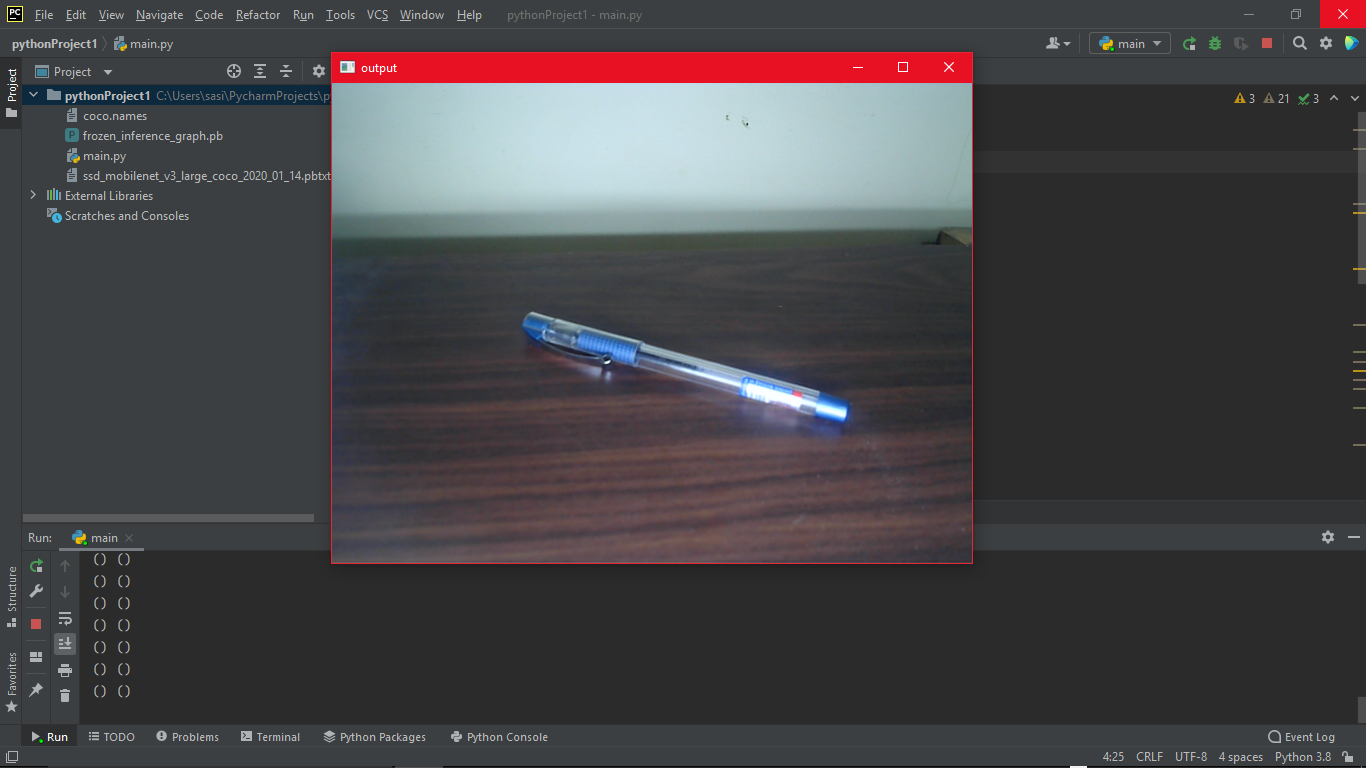


### Fig 12.7: Screenshot

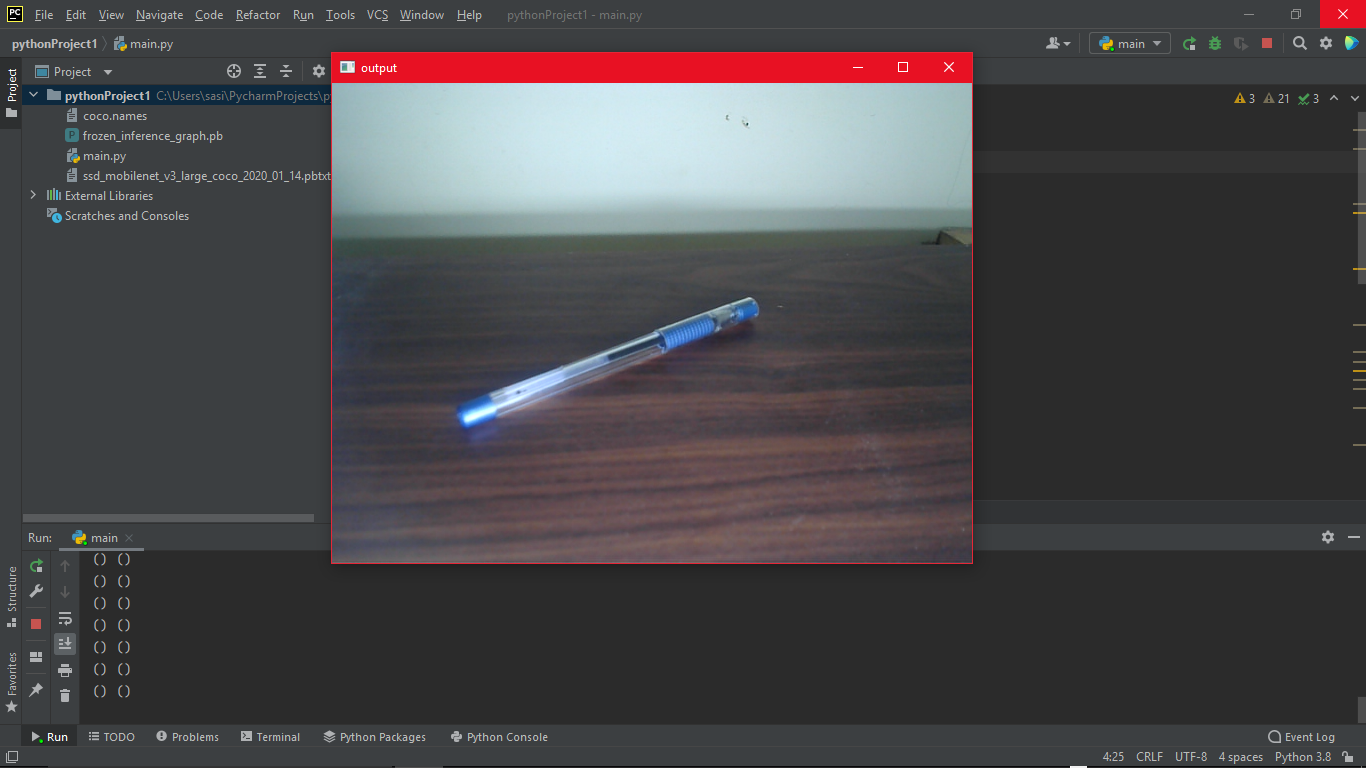


### Fig 12.8: Screenshot

In fig 12.7 and fig 12.8, a power bank is placed in front of the camera but the system does the recognize it and on the first figure it gives two outputs for the same object with overlaying bounding box. And when we change the angle of the object in the second figure, it still gives the wrong output. The system detects the object and classifies it to its closest object class which may always not be correct. This shows that this system is not 100% reliable and need further development.

****

### Fig 12.9: Screenshot

****

### Fig 12.10: Screenshot

In figure 12.9 and figure 12.10, we can see that the object is a common item but the system fails to detect the object. Even after changing the angle of the object, the system still does not detect the object in the frame. This may be caused due to insufficient data as its object class was not present on the system. This further proves that object detection is still evolving and developing and not fully furnished as of today.

## User Interface Design

The user interface of this project is fairly simple and not hard to use. As this project runs on a PyCharm and is not exported as external software. We just need to run the code on the IDE and start testing the project. The visuals of the project and how it runs can be seen above (Fig. 12.1- Fig 12.10).

## Test Cases (Unit Test Cases and System Test Cases)

For testing the project we are using Moblienet-SSD and the reason we are using the Mobilenet-SSD is because it is one of the best methods right now which has a good balance between accuracy and speed, also, it doesn’t require a heavy machine with high processing speed. So, we will be able to run with our CPU almost real time with a decent CPU and it will be able to detect a lot of objects with a good amount of accuracy. There are also other ways with which we can test for example YOLO, but with YOLO we have to use GPU otherwise it will be too slow and if we go for the YOLO tiny it's not that good as compared to YOLO, so if we want to detect some objects with good amount of speed and accuracy, Mobilenet-SSD does the job pretty well. Also with YOLO tiny, it fails to detect some common objects. Mobilenet-SSD can also be used with raspberry pi and jetson nano as well because it is lightweight and you can get a decent amount of accuracy so this is the reason that we have chosen Mobilenet-SSD for testing.

In test cases scenario, we can use random objects and allow the system to detect it using the camera. By using camera to input the data, we can achieve almost real time object detection. Whereas, if we use a photo file, we cannot detect objects in real time and our main objective is to detect objects in real time and not using a photo file. To test the system, we need a camera to feed the system with live footage of the objects that it sees and the system processes and try to detect the object and classifies according to the object class it belongs to. Then it names the object within a bounding box that is displayed when it detects a familiar object. And depending to the percentage of familiarity of the object present in the data, it shows the probability of an object belonging to a certain class. We can change the threshold value which makes us clear that it belongs to a specific object class. The expected outcome is a green (can be any color) bounding box around the detected object with the name of class and threshold value.

In use cases, as this project is not exported as a separate software package, there are not many complications in running the program and GUIs are kept at minimum to make it simple and easy to use. The interface is quiet simple and minimum making it easy to use. We just need to run the codes on our IDE and point the camera at any object that we want to detect and we get the expected outcome on the screen with the object class and threshold inside a bounding box.

# Coding

## Complete Project Coding

### main.py

import cv2  
  
thres=0.5  
cap = cv2.VideoCapture(0)  
cap.set(3,640)  
cap.set(4, 480)  
  
classNames = []  
classFiles = 'coco.names'  
with open(classFiles,'rt') as f:  
 classNames = f.read().rstrip('\n').split('\n')  
  
 configPath = 'ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt'  
 weightsPath = 'frozen\_inference\_graph.pb'  
  
 net = cv2.dnn\_DetectionModel(weightsPath, configPath)  
 net.setInputSize(320, 320)  
 net.setInputScale(1.0 / 127.5)  
 net.setInputMean((127.5, 127.5, 127.5))  
 net.setInputSwapRB(True)  
while True:  
 success, img = cap.read()  
 classIds, confs , bbox = net.detect(img, confThreshold=thres)  
 print(classIds, bbox)  
  
 if len(classIds) !=0:  
 for classId, confidence, box in zip(classIds.flatten(), confs.flatten(),bbox):  
 cv2.rectangle(img,box,color=(0,255,0), thickness=2)  
 cv2.putText(img, classNames[classId-1].upper(), (box[0]+10,box[1]+30),  
 cv2.FONT\_HERSHEY\_COMPLEX,1,(0,255,0),2)  
  
 cv2.putText(img,str (round(confidence\*100,2)), (box[0] + 250, box[1] + 30),  
 cv2.FONT\_HERSHEY\_COMPLEX, 1, (0, 255, 0), 2)  
  
 cv2.imshow("output", img)  
 cv2.waitKey(1)

### coco.names

person  
bicycle  
car  
motorcycle  
airplane  
bus  
train  
truck  
boat  
traffic light  
fire hydrant  
street sign  
stop sign  
parking meter  
bench  
bird  
cat  
dog  
horse  
sheep  
cow  
elephant  
bear  
zebra  
giraffe  
hat  
backpack  
umbrella  
shoe  
eye glasses  
handbag  
tie  
suitcase  
frisbee  
skis  
snowboard  
sports ball  
kite  
baseball bat  
baseball glove  
skateboard  
surfboard  
tennis racket  
bottle  
plate  
wine glass  
cup  
fork  
knife  
spoon  
bowl  
banana  
apple  
sandwich  
orange  
broccoli  
carrot  
hot dog  
pizza  
donut  
cake  
chair  
couch  
potted plant  
bed  
mirror  
dining table  
window  
desk  
toilet  
door  
tv  
laptop  
mouse  
remote  
keyboard  
cell phone  
microwave  
oven  
toaster  
sink  
refrigerator  
blender  
book  
clock  
vase  
scissors  
teddy bear  
hair drier  
toothbrush  
hair brush

### ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt

node {  
 name: "normalized\_input\_image\_tensor"  
 op: "Placeholder"  
 attr {  
 key: "dtype"  
 value {  
 type: DT\_FLOAT  
 }  
 }  
 attr {  
 key: "shape"  
 value {  
 shape {  
 dim {  
 size: 1  
 }  
 dim {  
 size: 320  
 }  
 dim {  
 size: 320  
 }  
 dim {  
 size: 3  
 }  
 }  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/Conv/Conv2D"  
 op: "Conv2D"  
 input: "normalized\_input\_image\_tensor"  
 input: "FeatureExtractor/MobilenetV3/Conv/weights"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "dilations"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
 attr {  
 key: "explicit\_paddings"  
 value {  
 list {  
 }  
 }  
 }  
 attr {  
 key: "padding"  
 value {  
 s: "SAME"  
 }  
 }  
 attr {  
 key: "strides"  
 value {  
 list {  
 i: 1  
 i: 2  
 i: 2  
 i: 1  
 }  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/Conv/BatchNorm/FusedBatchNormV3"  
 op: "FusedBatchNormV3"  
 input: "FeatureExtractor/MobilenetV3/Conv/Conv2D"  
 input: "FeatureExtractor/MobilenetV3/Conv/BatchNorm/gamma"  
 input: "FeatureExtractor/MobilenetV3/Conv/BatchNorm/beta"  
 input: "FeatureExtractor/MobilenetV3/Conv/BatchNorm/moving\_mean"  
 input: "FeatureExtractor/MobilenetV3/Conv/BatchNorm/moving\_variance"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "epsilon"  
 value {  
 f: 0.001  
 }  
 }  
 attr {  
 key: "U"  
 value {  
 type: DT\_FLOAT  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/add"  
 op: "AddV2"  
 input: "FeatureExtractor/MobilenetV3/Conv/BatchNorm/FusedBatchNormV3"  
 input: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/add/y"  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/Relu6"  
 op: "Relu6"  
 input: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/add"  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/mul"  
 op: "Mul"  
 input: "FeatureExtractor/MobilenetV3/Conv/BatchNorm/FusedBatchNormV3"  
 input: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/Relu6"  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/mul\_1"  
 op: "Mul"  
 input: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/mul"  
 input: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/mul\_1/y"  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/depthwise"  
 op: "DepthwiseConv2dNative"  
 input: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/mul\_1"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/depthwise\_weights"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "dilations"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
 attr {  
 key: "padding"  
 value {  
 s: "SAME"  
 }  
 }  
 attr {  
 key: "strides"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/BatchNorm/FusedBatchNormV3"  
 op: "FusedBatchNormV3"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/depthwise"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/BatchNorm/gamma"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/BatchNorm/beta"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/BatchNorm/moving\_mean"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/BatchNorm/moving\_variance"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "epsilon"  
 value {  
 f: 0.001  
 }  
 }  
 attr {  
 key: "U"  
 value {  
 type: DT\_FLOAT  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/Relu"  
 op: "Relu"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/BatchNorm/FusedBatchNormV3"  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv/project/Conv2D"  
 op: "Conv2D"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/depthwise/Relu"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/project/weights"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "dilations"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
 attr {  
 key: "explicit\_paddings"  
 value {  
 list {  
 }  
 }  
 }  
 attr {  
 key: "padding"  
 value {  
 s: "SAME"  
 }  
 }  
 attr {  
 key: "strides"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
}

node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv/project/BatchNorm/FusedBatchNormV3"  
 op: "FusedBatchNormV3"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/project/Conv2D"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/project/BatchNorm/gamma"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/project/BatchNorm/beta"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/project/BatchNorm/moving\_mean"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/project/BatchNorm/moving\_variance"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "epsilon"  
 value {  
 f: 0.001  
 }  
 }  
 attr {  
 key: "U"  
 value {  
 type: DT\_FLOAT  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv/add"  
 op: "AddV2"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/project/BatchNorm/FusedBatchNormV3"  
 input: "FeatureExtractor/MobilenetV3/Conv/hard\_swish/mul\_1"  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/input"  
 op: "Identity"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv/add"  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/Conv2D"  
 op: "Conv2D"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/input"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/weights"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "dilations"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
 attr {  
 key: "explicit\_paddings"  
 value {  
 list {  
 }  
 }  
 }  
 attr {  
 key: "padding"  
 value {  
 s: "SAME"  
 }  
 }  
 attr {  
 key: "strides"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/BatchNorm/FusedBatchNormV3"  
 op: "FusedBatchNormV3"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/Conv2D"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/BatchNorm/gamma"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/BatchNorm/beta"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/BatchNorm/moving\_mean"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/BatchNorm/moving\_variance"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "epsilon"  
 value {  
 f: 0.001  
 }  
 }  
 attr {  
 key: "U"  
 value {  
 type: DT\_FLOAT  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/Relu"  
 op: "Relu"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/BatchNorm/FusedBatchNormV3"  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/depthwise"  
 op: "DepthwiseConv2dNative"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/expand/Relu"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/depthwise\_weights"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "dilations"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
 attr {  
 key: "padding"  
 value {  
 s: "SAME"  
 }  
 }  
 attr {  
 key: "strides"  
 value {  
 list {  
 i: 1  
 i: 2  
 i: 2  
 i: 1  
 }  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/BatchNorm/FusedBatchNormV3"  
 op: "FusedBatchNormV3"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/depthwise"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/BatchNorm/gamma"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/BatchNorm/beta"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/BatchNorm/moving\_mean"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/BatchNorm/moving\_variance"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "epsilon"  
 value {  
 f: 0.001  
 }  
 }  
 attr {  
 key: "U"  
 value {  
 type: DT\_FLOAT  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/Relu"  
 op: "Relu"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/BatchNorm/FusedBatchNormV3"  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/project/Conv2D"  
 op: "Conv2D"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/depthwise/Relu"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/project/weights"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "dilations"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
 attr {  
 key: "explicit\_paddings"  
 value {  
 list {  
 }  
 }  
 }  
 attr {  
 key: "padding"  
 value {  
 s: "SAME"  
 }  
 }  
 attr {  
 key: "strides"  
 value {  
 list {  
 i: 1  
 i: 1  
 i: 1  
 i: 1  
 }  
 }  
 }  
}  
node {  
 name: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/project/BatchNorm/FusedBatchNormV3"  
 op: "FusedBatchNormV3"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/project/Conv2D"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/project/BatchNorm/gamma"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/project/BatchNorm/beta"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/project/BatchNorm/moving\_mean"  
 input: "FeatureExtractor/MobilenetV3/expanded\_conv\_1/project/BatchNorm/moving\_variance"  
 attr {  
 key: "data\_format"  
 value {  
 s: "NHWC"  
 }  
 }  
 attr {  
 key: "epsilon"  
 value {  
 f: 0.001  
 }  
 }

The “ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt” is an external, open source downloadable files that is used in computer vision for object detection (The above given code for ssd\_mobilenet is less than half of the total code). The mobilenet-ssd model is a Single-Shot multi-box Detection (SSD) network intended to perform object detection. This model is implemented using the Caffe\* framework.

The model input is a blob that consists of a single image of 1, 3, 300, 300 in BGR order, also like the densenet-121 model. The BGR mean values need to be subtracted as follows: [127.5, 127.5, 127.5] before passing the image blob into the network. In addition, values must be divided by 0.007843.

For more information on ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt, visit the website <https://docs.openvino.ai/latest/omz_models_model_mobilenet_ssd.html>

## Comments and Description of Coding segments

For the project to run without any errors, we need to have main.py, coco.names, ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt, frozen\_inference\_graph.pb and OpenCV on out PyCharm. OpenCV is an open source python extension for Computer Vision. We can download it by going to add extensions and searching “OpenCV”, click add and install. The main.py is basically the GUI design for the project and the other resources helps in detecting the object and labeling them according to their classes. The coco.names is the list of some of the object class that the computer vision will be able to detect in real time with the help of an external camera. The ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt may be defined as the brain of the project that detects all the objects and labels it. With the development of new and better technologies, there are better and faster alternatives than the one we are using but due to limited processing power and other resources, ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt serves as an effective alternative with relatively fast and accurate object detection.

# Testing

## Testing Strategy used & description of test cases and use cases

For testing the project we are using Moblienet-SSD and the reason we are using the Mobilenet-SSD is because it is one of the best methods right now which has a good balance between accuracy and speed, also, it doesn’t require a heavy machine with high processing speed. So, we will be able to run with our CPU almost real time with a decent CPU and it will be able to detect a lot of objects with a good amount of accuracy. There are also other ways with which we can test for example YOLO, but with YOLO we have to use GPU otherwise it will be too slow and if we go for the YOLO tiny it's not that good as compared to YOLO, so if we want to detect some objects with good amount of speed and accuracy, Mobilenet-SSD does the job pretty well. Also with YOLO tiny, it fails to detect some common objects. Mobilenet-SSD can also be used with raspberry pi and jetson nano as well because it is lightweight and you can get a decent amount of accuracy so this is the reason that we have chosen Mobilenet-SSD for testing.

In test cases scenario, we can use random objects and allow the system to detect it using the camera. By using camera to input the data, we can achieve almost real time object detection. Whereas, if we use a photo file, we cannot detect objects in real time and our main objective is to detect objects in real time and not using a photo file. To test the system, we need a camera to feed the system with live footage of the objects that it sees and the system processes and try to detect the object and classifies according to the object class it belongs to. Then it names the object within a bounding box that is displayed when it detects a familiar object. And depending to the percentage of familiarity of the object present in the data, it shows the probability of an object belonging to a certain class. We can change the threshold value which makes us clear that it belongs to a specific object class. The expected outcome is a green (can be any color) bounding box around the detected object with the name of class and threshold value.

In use cases, as this project is not exported as a separate software package, there are not many complications in running the program and GUIs are kept at minimum to make it simple and easy to use. The interface is quiet simple and minimum making it easy to use. We just need to run the codes on our IDE and point the camera at any object that we want to detect and we get the expected outcome on the screen with the object class and threshold inside a bounding box.

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user. It verifies and validates whether the program is working correctly with no bugs; and also ensures that the system is working according to the specifications. It is a series of processes which is designed to make sure that the computer code does what it was designed to do. Testing is vital to the success of the system as, inadequate testing or non-testing leads to errors that may not appear unless testing is done.

The different types of testing are listed below:

* **Unit Testing:** This is the most basic testing mechanism at the developer level. This covers very narrow and well defined scope. Unit tests focus on very small unit of functionality. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. In unit testing issues are found at early stage.
* **System Testing:** System testing is designed to uncover weaknesses that were not found in earlier tests. This includes forced system failure and validation of the total system as it will be implemented by its user in the operational environment.
* **White-Box Testing:** White-box testing is a method of testing the application at the level of the source code. It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability. It is also known as clear, open, structural, and glass box testing.
* **Black Box Testing:** Black box testing is a method of testing that examines the external functionality of the software (e.g. what the software does) without the knowledge of its internal structure or working of the software. Black box testing is conducted from the user point of view.
* **Alpha Testing:** It is one of the most common software testing strategies used in software development. This test takes place at the developer’s site. It is the testing of an application when development is about to complete.
* **Beta testing:** In software development, a beta test is the second phase of software testing in which a sampling of the intended audience tries the product out. It is also known as field testing. The goal of beta testing is to place your application in the hands of real users outside of your own engineering team to discover any flaws or issues from the user’s perspective.
* **Performance testing:** It is the testing which is performed to ascertain how the components of a system are performing given a particular situation. Resource usage, scalability and reliability of the product are also validated under this testing.

## One-stage vs. two-stage deep learning object detectors

The state-of-the-art object detection methods can be categorized into two main types: One-stage vs. two-stage object detectors.

In general, deep learning based object detectors extract features from the input image or video frame. An object detector solves two subsequent tasks:

**Task #1:** Find an arbitrary number of objects (possibly even zero), and

**Task #2:** Classify every single object and estimate its size with a bounding box.

To simplify the process, you can separate those tasks into two stages. Other methods combine both tasks into one step (single-stage detectors) to achieve higher performance at the cost of accuracy.

**One-stage detectors:** One-stage detectors predict bounding boxes over the images without the region proposal step. This process consumes less time and can therefore be used in real-time applications.

* One-stage object detectors prioritize inference speed and are super fast but not as good at recognizing irregularly shaped objects or a group of small objects.
* The most popular one-stage detectors include the YOLO, SSD, and RetinaNet. The latest real-time detectors are YOLOv4-Scaled (2020) and YOLOR (2021). View the benchmark comparisons below.
* The main advantage of single-stage is that those algorithms are generally faster than multi-stage detectors and structurally simpler.

**Two-stage detectors:**In two-stage object detectors, the approximate object regions are proposed using deep features before these features are used for the classification as well as bounding box regression for the object candidate.

* The two-stage architecture involves (1) object region proposal with conventional Computer Vision methods or deep networks, followed by (2) object classification based on features extracted from the proposed region with bounding-box regression.
* Two-stage methods achieve the highest detection accuracy but are typically slower. Because of the many inference steps per image, the performance (frames per second) is not as good as one-stage detectors.
* Various two-stage detectors include region convolution neural network (RCNN), with evolutions Faster R-CNN or Mask R-CNN. The latest evolution is the granulated RCNN (G-RCNN).
* Two-stage object detectors first find a region of interest and use this cropped region for classification. However, such multi-stage detectors are usually not end-to-end trainable because cropping is a non-differentiable operation.

## Work Progress as per timeline

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TASK | Estimated Start date | Estimated Finish date | Description | Estimated Duration |
| Project Proposal | 22-08-2021 | 25-08-2021 | Working and submission of project proposal | 3 days |
| Project Report-1 | 26-08-2021 | 20-09-2021 | Start of project. Brief case study on the topic and implementation | 2-3 weeks |
| Project Report-2 | 21-09-2021 | 15-10-2021 | Testing and debugging | 3-4 weeks |
| Final Project Report | 16-10-2021 | 05-11-2021 | Finalizing of project report and submission of project | 2-3 weeks |

## Number of Objectives Achieved

* Research related to the topic completed
* Base design and GUI completed
* Importing required data and packages completed.
* Testing and implementation completed.
* Debugging of errors completed.

The project is in a running state and all the major bugs and errors have been fixed. The project aims at detecting object using real time camera footage and accomplishes it quiet accurately and efficiently without taking much storage space of without a powerful processor.

# Documentation

## Cost Estimation of the Project along with Cost Estimation Model.

As this project relied mostly on open soured software, the software cost comprises a small percentage of overall computer-based system cost. There are a number of factors, which are considered, that can affect the ultimate cost of the software such as human, technical, hardware and software availability etc.

The main point that was considered during the cost estimation of project was its sizing in spite of complete software sizing, function point and approximate lines of code were also used to size each element of the software and their costing. The cost estimation done by me for project also depend upon the baseline metrics collected from past projects and these were used in conjunction with estimation variables to develop cost and effort projections.

We have basically estimated this project mainly on two bases:

**Effort Estimation:** This refers to the total man-hours required for the development of the project. It even includes the time required for doing documentation and user manual.

**Hardware Required Estimation:** This includes the cost of the PCs and the hardware cost required for development of this project.

# Conclusion and Future Scope

## Conclusion

The ability for computer vision to distinguish between objects when it is in motion is an important feature of object detection, mainly in video surveillance. Since we are limited by our present technology and resources, the detection of objects in motion using Convolutional Neural Networks still remains a challenge. In this project, we aim to improving efficiency, reliability and accuracy of the detection of moving objects. We can achieve this by using a 2-level methodology based upon deep learning, also known as Object Detection with Binary Classifiers. The first level takes the candidate regions from the given frame and the second level uses a technique called “Binarization” based on a CNN classifier.

Most AI and AR based computer tasks are based on computer vision and object detection. Such problems can be solved by using modern deep learning models especially when the objects are different in a frame for example, different color, size, shape and distance. However, this task becomes more complicated when the targeted objects are far or not clearly visible which may be caused by poor camera quality or reduced number of pixels. Till date, the most efficient models of object detection are based on deep CNN as these models automatically learn the distinctive features of objects from a large set of labeled data and hope to implement this model in this project.

The detection accuracy is mostly measured on a given test set where the expected outcome for a detection sample is compared to the actual outcome of the object detection system .Detection accuracy refers to the percentage of samples for which the expected outcome matches the actual outcome of the detection system. It may most definitely not be accurate always and show wrong output but we can use that data to prevent similar errors from happening again. We plan to show the output on an external monitor where objects that the system detects are shown inside a rectangular bordered box. And if it matches the class then it will show what kind of object that is along with percentage. The system checks for similarities and shows how much it matches a particular class.

## Future scope and further enhancement of the Project

Object detection is one of the most fundamental and challenging problems in computer vision. It has received great attention in recent years, especially with the success of deep learning methods that currently dominate the recent state-of-the-art detection methods. Object detection is increasingly important for computer vision applications in any industry There are many optimizations that could be made in this project. For example, the project can further be implemented and modified into other projects that require face recognition for unlocking phones without the use of passwords. And furthermore, the code can be optimized as per user preference and add or remove features accordingly. With more time and effort, this project could perhaps run and function smoothly with the introduction of new technology. Another feature that could be added is to allow the computer to scan and detect only a particular part of an image and not the entire image thus increasing speed and accuracy. This however, needs to be tested first as it may backfire and result in more time consuming and errors while showing result. The main objective for now is to detect simple objects and distinguish according to their class and not go deeper for industrial or medical level perfection. Perfecting it will most certainly take time and we can speed it up by using more powerful computers with better hardware and faster software. We can also make use of more than one computer allowing for more processing to be done per frame. One example is to install a good quality camera which will ultimately improve the real time footage captured and processed by the computer. Better video quality means more frame rate and higher resolution images. This would enable the computer to filter out non-relevant objects from the image and also provide a fast tracking while using real time object tracking.

# References

[1] Jalled, F., & Voronkov, I. (2016). Object detection using image processing. *arXiv preprint arXiv:1611.07791*.

[2] Khan, F. S., Anwer, R. M., Van De Weijer, J., Bagdanov, A. D., Vanrell, M., & Lopez, A. M. (2012, June). Color attributes for object detection. In *2012 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3306-3313). IEEE.

[3] Lee, C. C. (2007). Moving object detection at night. *Department of Electrical-Electronic Engineering, University of Technology, Malayasia*.

[4] Manap, M. S. A., Sahak, R., Zabidi, A., Yassin, I., & Tahir, N. M. (2015, March). Object detection using depth information from Kinect sensor. In *2015 IEEE 11th International Colloquium on Signal Processing & Its Applications (CSPA)* (pp. 160-163). IEEE.

[5] Ophoff, T., Van Beeck, K., & Goedemé, T. (2019). Exploring RGB+ Depth fusion for real-time object detection. *Sensors*, *19*(4), 866.

[6] Paine, M. (2003). Heavy vehicle object detection systems. *Vehicle Design and Research Pty Lmited for VicRoads*, 1-22.

[7] Papageorgiou, C., & Poggio, T. (2000). A trainable system for object detection. *International journal of computer vision*, *38*(1), 15-33.

[8] Papageorgiou, C. P., Oren, M., & Poggio, T. (1998, January). A general framework for object detection. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)* (pp. 555-562). IEEE.

[9] Szegedy, C., Reed, S., Erhan, D., Anguelov, D., & Ioffe, S. (2014). Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*.

[10] Strickland, R. N., & Hahn, H. I. (1997). Wavelet transform methods for object detection and recovery. *IEEE transactions on image processing*, *6*(5), 724-735.

[11] Tang, S., & Yuan, Y. (2015). Object detection based on convolutional neural network. In *International Conference-IEEE–2016*.

[12] Tijtgat, N., Van Ranst, W., Goedeme, T., Volckaert, B., & De Turck, F. (2017). Embedded real-time object detection for a UAV warning system. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (pp. 2110-2118).

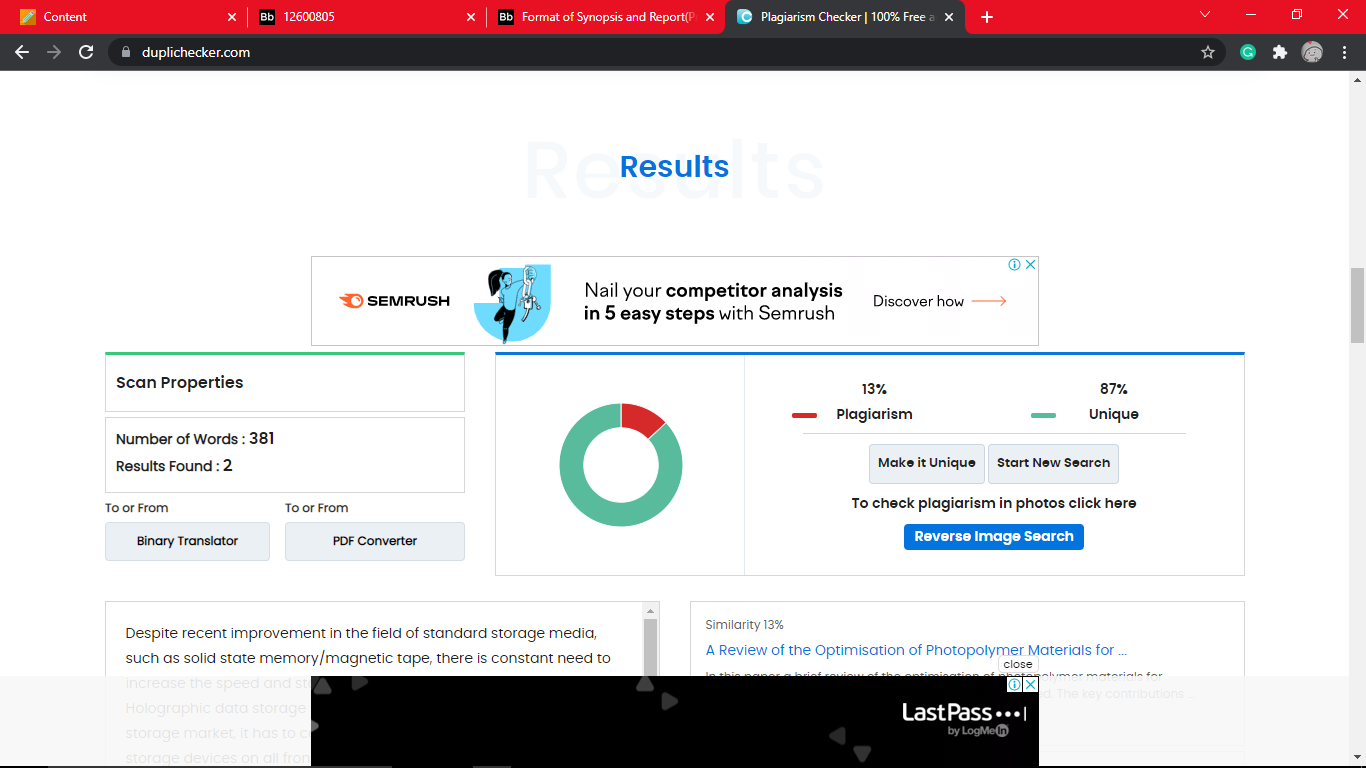
[13] Tripathi, S., Dane, G., Kang, B., Bhaskaran, V., & Nguyen, T. (2017). Lcdet: Low-complexity fully-convolutional neural networks for object detection in embedded systems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (pp. 94-103).

[14] Viola, P., & Jones, M. (2001). Robust real-time object detection. *International journal of computer vision*, *4*(34-47), 4.

[15] Zhou, J., Tan, X., Shao, Z., & Ma, L. (2019, October). FVNet: 3D front-view proposal generation for real-time object detection from point clouds. In *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)* (pp. 1-8). IEEE.

# Plagiarism Report

Via <https://www.duplichecker.com/>



12% Plagiarism

87% Unique

# Github Repository

Link: <https://github.com/SashiWalling/Real-Time-object-detection-using-Python-and-OpenCV>

