DETECTION AND TRACKING OF OBJECTS USING DRONE IMAGES

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by
Addepalli Sravani, 221910310003
Sowmya Nekkanti, 221910310036
Sameera Jampala, 221910310048
Bhargav Kiran Surapaneni, 221910310052

Under the esteemed guidance of
Dr. Rupesh Kumar Mishra
Asst. professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GITAM
(Deemed to be University)
HYDERABAD

OCTOBER 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled "**DETECTION AND TRACKING OF OBJECTS USING DRONE IMAGES**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. In Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration No(s)	Name(s)	Signature(s)
221910310003	Addepalli Sravani	
221910310036	Sree Lakshmi Sowmya .N	
221910310048	Sameera Jampala	
221910310052	Bhargav Kiran Surapaneni	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM
(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled **"DETECTION AND TRACKING OF OBJECTS USING DRONE IMAGES**" is a bonafide record of work carried out by **Addepalli Sravani (221910310003), Sree Lakshmi Sowmya .N (221910310036), Sameera Jampala (221910310048) and Bhargav Kiran Surapaneni (221910310052)** students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	**Project coordinator**	**Head of the Department**
Dr. Rupesh Kumar Mishra	**Dr.S. Aparna**	**Dr.S. Phani Kumar**
Assistant Professor Dept. of CSE	Assistant Professor Dept. of CSE	Professor & HOD Dept. of CSE

TABLE OF CONTENTS

LIST OF FIGURE

1. ABSTRACT

Drones, or general UAVs, equipped with cameras have been fast deployed with a wide range of applications, including agriculture, aerial photography, and surveillance. Consequently, automatic understanding of visual data collected from drones becomes highly demanding, bringing computer vision and drones more and more closely. To promote and track the developments of object detection and tracking algorithms, we have organized three challenge workshops in conjunction with ECCV 2018, ICCV 2019 and ECCV 2020, attracting more than 100 teams around the world. We provide a large-scale drone captured dataset, VisDrone, which includes four tracks, i.e., (1) image object detection, (2) video object detection, (3) single object tracking, and (4) multi-object tracking. In this paper, we first present a thorough review of object detection and tracking datasets and benchmarks, and discuss the challenges of collecting large-scale drone-based object detection and tracking datasets with fully manual annotations. After that, we describe our VisDrone dataset, which is captured over various urban/suburban areas of 14 different cities across China from North to South. Being the largest such dataset ever published, VisDrone enables extensive evaluation and investigation of visual analysis algorithms for the drone platform. We provide a detailed analysis of the current state of the field of large-scale object detection and tracking on drones, and conclude the challenge as well as propose future directions. We expect the benchmark largely boost the research and development in video analysis on drone platforms.

2. INTRODUCTION

Drones lot of attention in recent years. The commercial drone (or UAVs) equipped with cameras are receiving a market report describes that the global commercial drone market size will reach 501.4 billion by 2028, with a compound annual growth rate of 57:5% from 2021 to 2028. Equipped with embedded devices, drones are able to analyze the captured data and spawn a variety of new application scenarios, e.g., Agriculture. Drones can provide valuable insights to help farmers or ranchers optimize agriculture operations, monitor crop growth and keep herds safe, etc. Aerial photography. Drones are used to extract aerial photography images instead of expensive cranes and helicopters. Shipping and delivery. Drones can efficiently send packages such as medical supplies, food, or other goods to the designated places. Security and surveillance. Drones can provide realtime visibility into security threats and emergency situations by monitoring large regions. Search and rescue. Drones are useful to help search missing persons, fugitives, or rescue survivors and drop supplies in difficult terrains and harsh conditions. Consequently, automatic understanding of visual data collected from drones become highly demanding, which brings computer vision to drones more and more closely. As two fundamental problems in computer vision, object detection and object tracking are under extensive investigation. However, although the great progress has been made in various application scenarios such as Internet and security surveillance, they are not usually optimal for dealing with drone captured sequences or images.

2.1 SOFTWARE REQUIREMENTS

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Platform – In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming languages and their runtime libraries.

Operating system is one of the first requirements mentioned when defining system requirements (software). Software may not be compatible with different versions of same line of operating systems, although some measure of backward compatibility is often maintained. For example, most software designed for Microsoft Windows XP does not run on Microsoft Windows 98, although the converse is

not always true. Similarly, software designed using newer features of Linux Kernel v2.6 generally does not run or compile properly (or at all) on Linux distributions using Kernel v2.2 or v2.4.

APIs and drivers – Software making extensive use of special hardware devices, like high-end display adapters, needs special API or newer device drivers. A good example is DirectX, which is a collection of APIs for handling tasks related to multimedia, especially game programming, on Microsoft platforms.

Web browser – Most web applications and software depending heavily on Internet technologies make use of the default browser installed on system. Microsoft Internet Explorer is a frequent choice of software running on Microsoft Windows, which makes use of ActiveX controls, despite their vulnerabilities.

1) Visual Studio Community Version

2) Nodejs (Version 12.3.1)

3) Python IDEL (Python 3.7)

2.2 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, a hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

Architecture – All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture. See also a list of common operating systems and their supporting architectures.

Processing power – The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored. This definition of power is often erroneous, as AMD Athlon

and Intel Pentium CPUs at similar clock speed often have different throughput speeds. Intel Pentium CPUs have enjoyed a considerable degree of popularity, and are often mentioned in this category.

Memory – All software, when run, resides in the random access memory (RAM) of a computer. Memory requirements are defined after considering demands of the application, operating system, supporting software and files, and other running processes. Optimal performance of other unrelated software running on a multi-tasking computer system is also considered when defining this requirement.

Secondary storage – Hard-disk requirements vary, depending on the size of software installation, temporary files created and maintained while installing or running the software, and possible use of swap space (if RAM is insufficient).

Display adapter – Software requiring a better than average computer graphics display, like graphics editors and high-end games, often define high-end display adapters in the system requirements.

Peripherals – Some software applications need to make extensive and/or special use of some peripherals, demanding the higher performance or functionality of such peripherals. Such peripherals include CD-ROM drives, keyboards, pointing devices, network devices, etc.

1) Operating System: Windows Only
2) Processor: i5 and above
3) Ram: 4 GB and above
4) Hard Disk: 50 GB.

3. FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are:

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

3.1 ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

3.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

3.3 SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity, as he is the final user of the system.

4. LITERATURE SURVEY

4.1 A BENCHMARK AND SIMULATOR FOR "UAV" TRACKING:

https://link.springer.com/chapter/10.1007/978-3-319-46448-0_27

In[2], M. Mueller, N. Smith, and B. Ghanem proposed a new aerial video dataset and benchmark for low altitude UAV target tracking, as well as, a photo-realistic UAV simulator that can be coupled with tracking methods. Our benchmark provides the first evaluation of many state-of-the-art and popular trackers on 123 new and fully annotated HD video sequences captured from a low-altitude aerial perspective. Among the compared trackers, we determine which ones are the most suitable for UAV tracking both in terms of tracking accuracy and run-time. The simulator can be used to evaluate tracking algorithms in real-time scenarios before they are deployed on a UAV "in the field", as well as, generate synthetic but photo-realistic tracking datasets with automatic ground truth annotations to easily extend existing real-world datasets. Both the benchmark and simulator are made publicly available to the vision community on our website to further research in the area of object tracking from UAVs.

4.2 DRONESURF: BENCHMARK DATASET FOR DRONE-BASED FACE RECOGNITION:

https://ieeexplore.ieee.org/document/8756593

In [3], I. Kalra, M. Singh, S. Nagpal, R. Singh, M. Vatsa, and P. B. Sujit, used Unmanned Aerial Vehicles (UAVs) or drones are often used to reach remote areas or regions which are inaccessible to humans. Equipped with a large field of view, compact size, and remote control abilities, drones are deemed suitable for monitoring crowded or disaster-hit areas, and performing aerial surveillance. While research has focused on area monitoring, object detection and tracking, limited attention has been given to person identification, especially face recognition, using drones. This research presents a novel large-scale drone dataset, DroneSURF: Drone Surveillance of Faces, in order to facilitate research for face recognition. The dataset contains 200 videos of 58 subjects, captured across 411K frames, having over 786K face annotations. The proposed dataset demonstrates variations across two surveillance use cases: (i) active and (ii) passive, two locations, and two acquisition times. DroneSURF encapsulates challenges due to the effect of motion, variations in pose, illumination, background, altitude, and resolution, especially due to the large and varying distance between the drone and the subjects. This research presents a detailed description of the proposed DroneSURF dataset, along with information regarding the data distribution, protocols for evaluation, and baseline results.

4.3 THE UNMANNED AERIAL VEHICLE BENCHMARK: OBJECT DETECTION AND TRACKING:

https://openaccess.thecvf.com/content_ECCV_2018/papers/Dawei_Du_The_Unmanned_Aerial_ECCV_2018_paper.pdf

In [4], D. Du, Y. Qi, H. Yu, Y. Yang, K. Duan, G. Li, W. Zhang, Q. Huang, and Q. Tian, with the advantage of high mobility, Unmanned Aerial Vehicles (UAVs) are used to fuel numerous important applications in computer vision, delivering more efficiency and convenience than surveillance cameras with fixed camera angle, scale and view. However, very limited UAV datasets are proposed, and they focus only on a specific task such as visual tracking or object detection in relatively constrained scenarios. Consequently, it is of great importance to develop an unconstrained UAV benchmark to boost related researches. In this paper, we construct a new UAV benchmark focusing on complex scenarios with new level challenges. Selected from 10 hours raw videos, about 80, 000 representative frames are fully annotated with bounding boxes as well as up to 14 kinds of attributes (e.g., weather condition, flying altitude, camera view, vehicle category, and occlusion) for three fundamental computer vision tasks: object detection, single object tracking, and multiple object tracking. Then, a detailed quantitative study is performed using most recent state-of-the-art algorithms for each task. Experimental results show that the current state-of-the-art methods perform relative worse on our dataset, due to the new challenges appeared in UAV based real scenes, e.g., high density, small object, and camera motion. To our knowledge, our work is the first time to explore such issues in unconstrained scenes comprehensively.

4.4 DRONE-BASED OBJECT COUNTING BY SPATIALLY REGULARIZED REGIONAL PROPOSAL NETWORK:

https://openaccess.thecvf.com/content_ICCV_2017/papers/Hsieh_Drone-Based_Object_Counting_ICCV_2017_paper.pdf

In [5], Existing counting methods which often adopt regression-based approaches and cannot precisely localize the target objects, which hinders the further analysis (e.g., high-level understanding and fine-grained classification). In addition, most of prior work mainly focus on counting objects in static environments with fixed cameras. Motivated by the advent of unmanned flying vehicles (i.e., drones), M. Hsieh, Y. Lin, and W. H. Hsu they worked on detecting and counting objects in such dynamic environments. We propose Layout Proposal Networks (LPNs) and spatial kernels to simultaneously count and localize target objects (e.g., cars) in videos recorded by the drone. Different from the

conventional region proposal methods, we leverage the spatial layout information (e.g., cars often park regularly) and introduce these spatially regularized constraints into our network to improve the localization accuracy. To evaluate our counting method, we present a new large-scale car parking lot dataset (CARPK) that contains nearly 90,000 cars captured from different parking lots. To the best of our knowledge, it is the first and the largest drone view dataset that supports object counting, and provides the bounding box annotations.

4.5 LEARNING SOCIAL ETIQUETTE: HUMAN TRAJECTORY UNDERSTANDING IN CROWDED SCENES

https://svl.stanford.edu/assets/papers/ECCV16social.pdf

In [6], A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese argued that Humans navigate crowded spaces such as a university campus by following common sense rules based on social etiquette. In this paper, that in order to enable the design of new target tracking or trajectory forecasting methods that can take full advantage of these rules, we need to have access to better data in the first place. To that end, we contribute a new large-scale dataset that collects videos of various types of targets (not just pedestrians, but also bikers, skateboarders, cars, buses, golf carts) that navigate in a real world outdoor environment such as a university campus. Moreover, we introduce a new characterization that describes the "social sensitivity" at which two targets interact. We use this characterization to define "navigation styles" and improve both forecasting models and state-of-the-art multi-target tracking - whereby the learnt forecasting models help the data association step.

5. SYSTEM ANALYSIS

5.1 EXISTING SYSTEM:

It is necessary to develop and evaluate new vision algorithms for drone captured visual data. However, as pointed out in [2], [5], studies toward this goal are seriously limited by the lack of publicly available large-scale benchmarks or datasets. Some recent efforts [2], [5], [6] have been devoted to construct datasets captured by drones focusing on object detection or tracking. These datasets are still limited in size and scenarios covered, due to the difficulties in data collection and annotation. Thorough evaluations of existing or newly developed algorithms remain an open problem. A more general and comprehensive benchmark is desired for further boosting video analysis research on drone platforms.

5.1.1 DISADVANTAGES OF EXISTING SYSTEM:

These datasets are still limited in size and scenarios covered, due to the difficulties in data collection and annotation

5.2 PROPOSED SYSTEM:

This paper focuses on analyzing various representative object detection and tracking algorithms thoroughly. Specifically, we discuss and analyze the advantages and disadvantages of the submitted methods in terms of model design and training strategies. After that, we advocate several future research directions of object detection and tracking on drone-captured videos. We expect such comprehensive review and analysis to largely boost the research and development in video analysis on drones.

5.2.1 ADVANTAGES OF PROPOSED SYSTEM:

1. Provides a comprehensive evaluation platform for object detection and tracking.
2. Effectiveness and efficiency are both important aspects for algorithms in real applications.

5.3 FUNCTIONAL REQUIREMENTS

1. Data Collection
2. Data Pre-processing
3. Training and Testing
4. Modelling
5. Final Outcomes

5.4 NON FUNCTIONAL REQUIREMENTS

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system. Example of nonfunctional requirement, *"how fast does the website load?"* Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. Non- functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

- Usability requirement
- Serviceability requirement
- Manageability requirement
- Recoverability requirement
- Security requirement
- Data Integrity requirement
- Capacity requirement
- Availability requirement
- Scalability requirement
- Interoperability requirement
- Reliability requirement
- Maintainability requirement
- Regulatory requirement
- Environmental requirement

6. SYSTEM DESIGN

6.1 SYSTEM ARCHITECTURE:

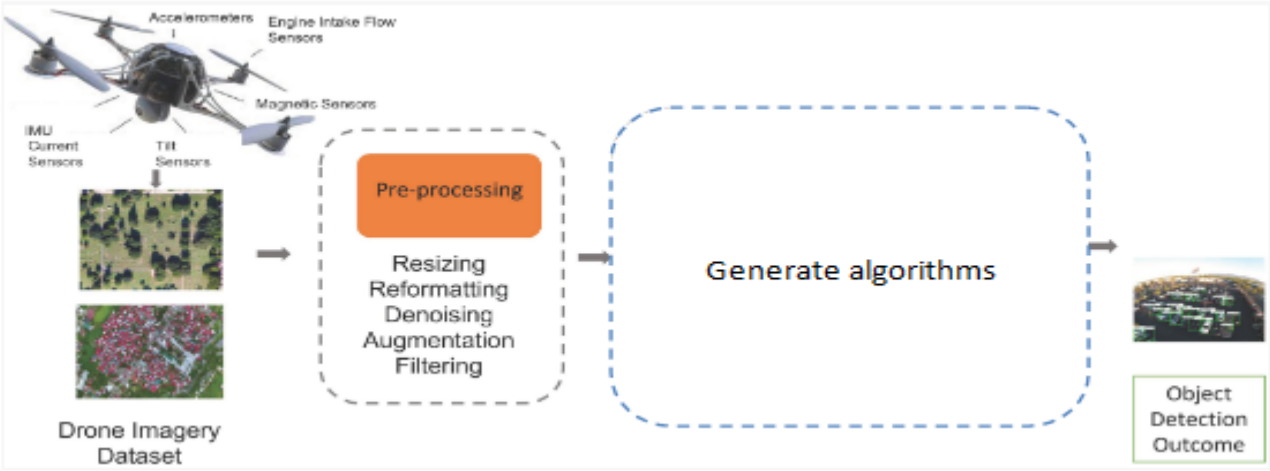


Fig.6.1.1 System architecture

DATA FLOW DIAGRAM:

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

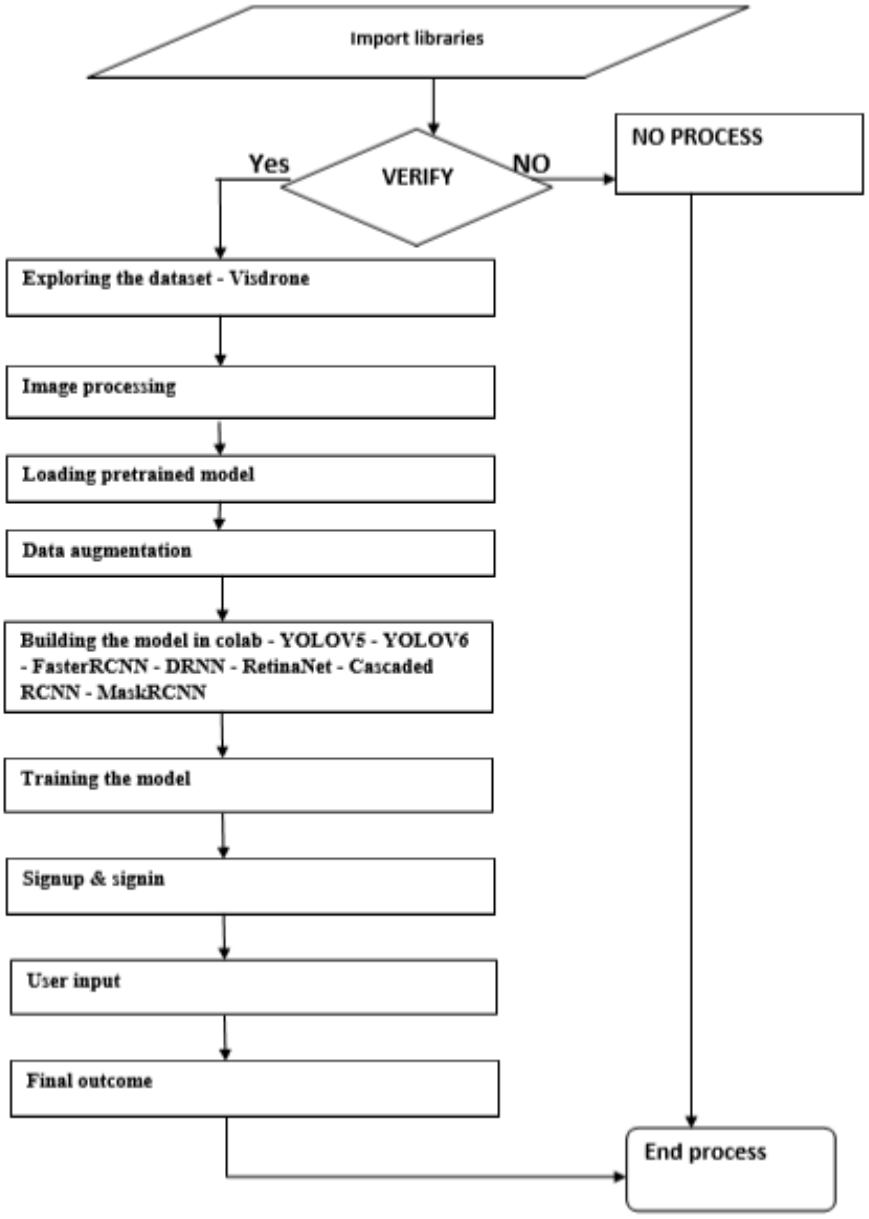


Fig.6.1.2 Dataflow diagram

6.2 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

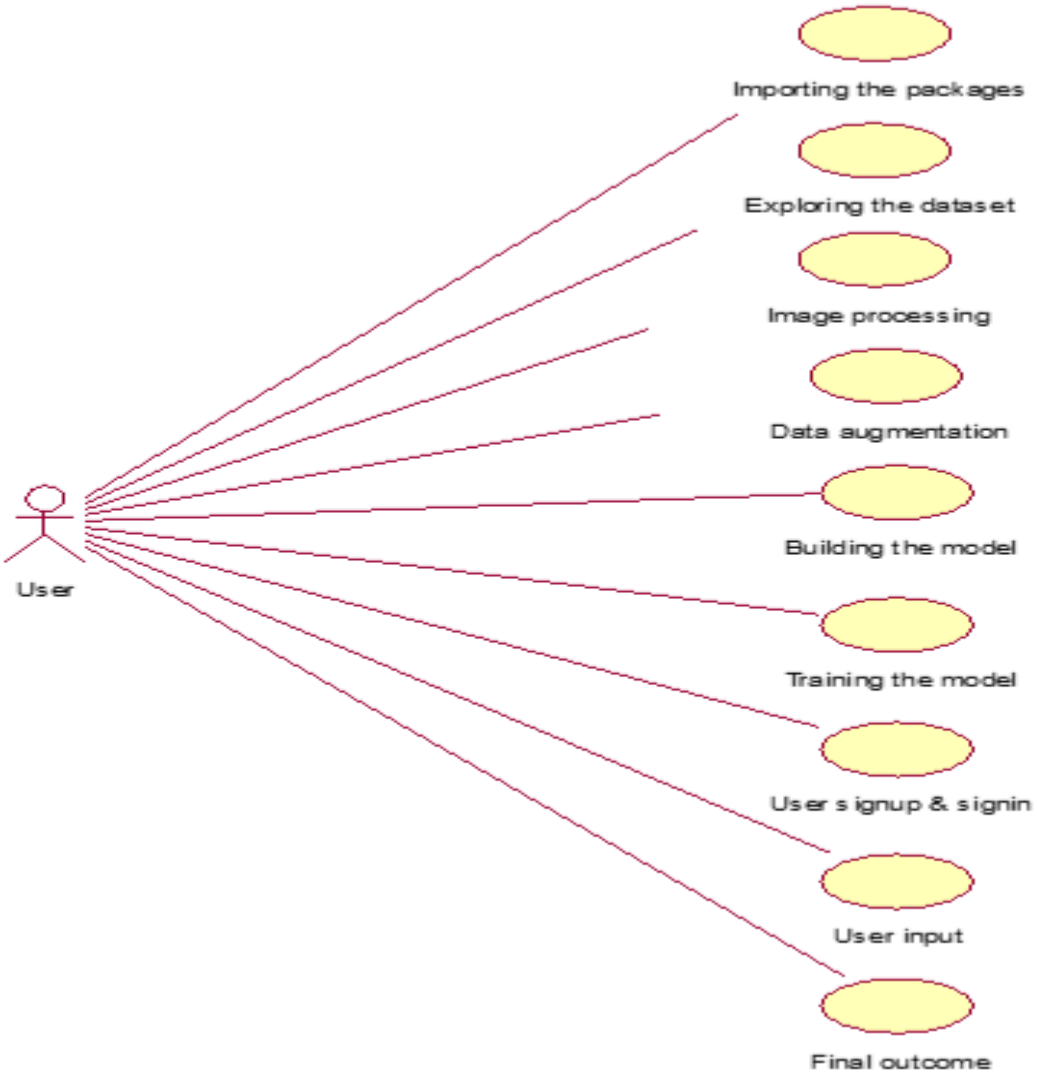


Fig.6.2.1 Use case diagram

CLASS DIAGRAM:

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

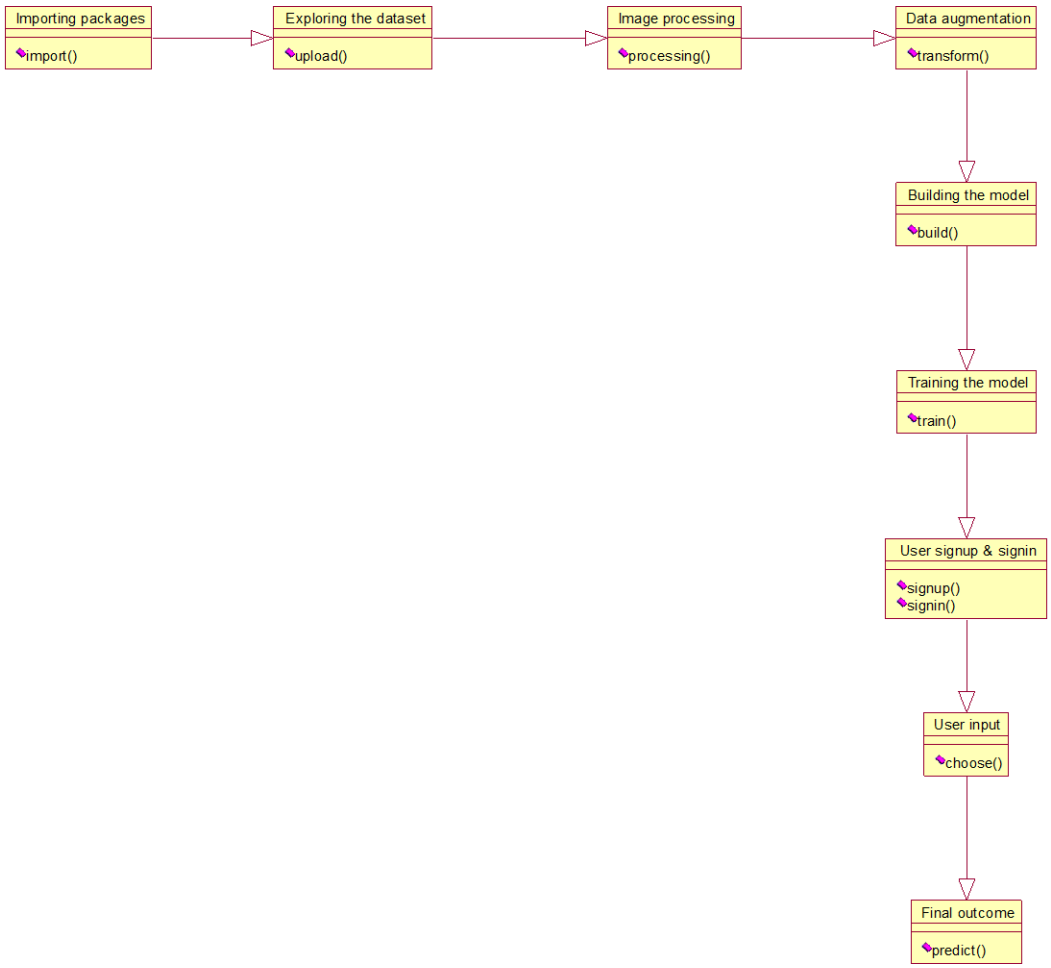


Fig.6.2.2 Class diagram

ACTIVITY DIAGRAM:

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.

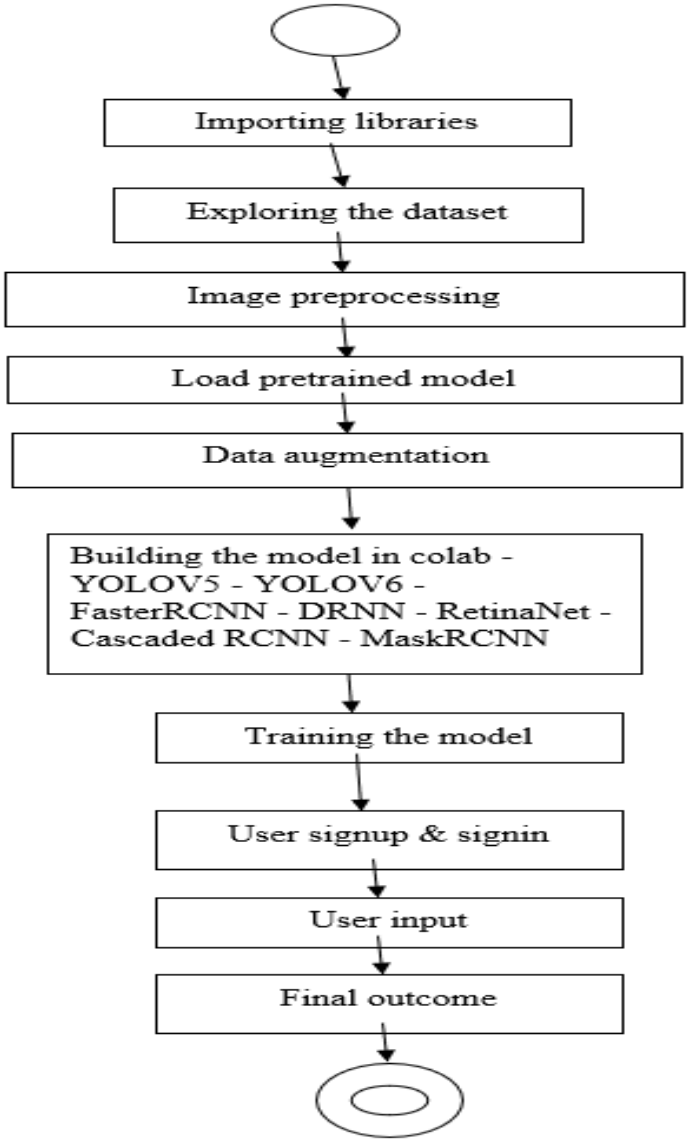


Fig.6.2.3 Activity diagram

SEQUENCE DIAGRAM:

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

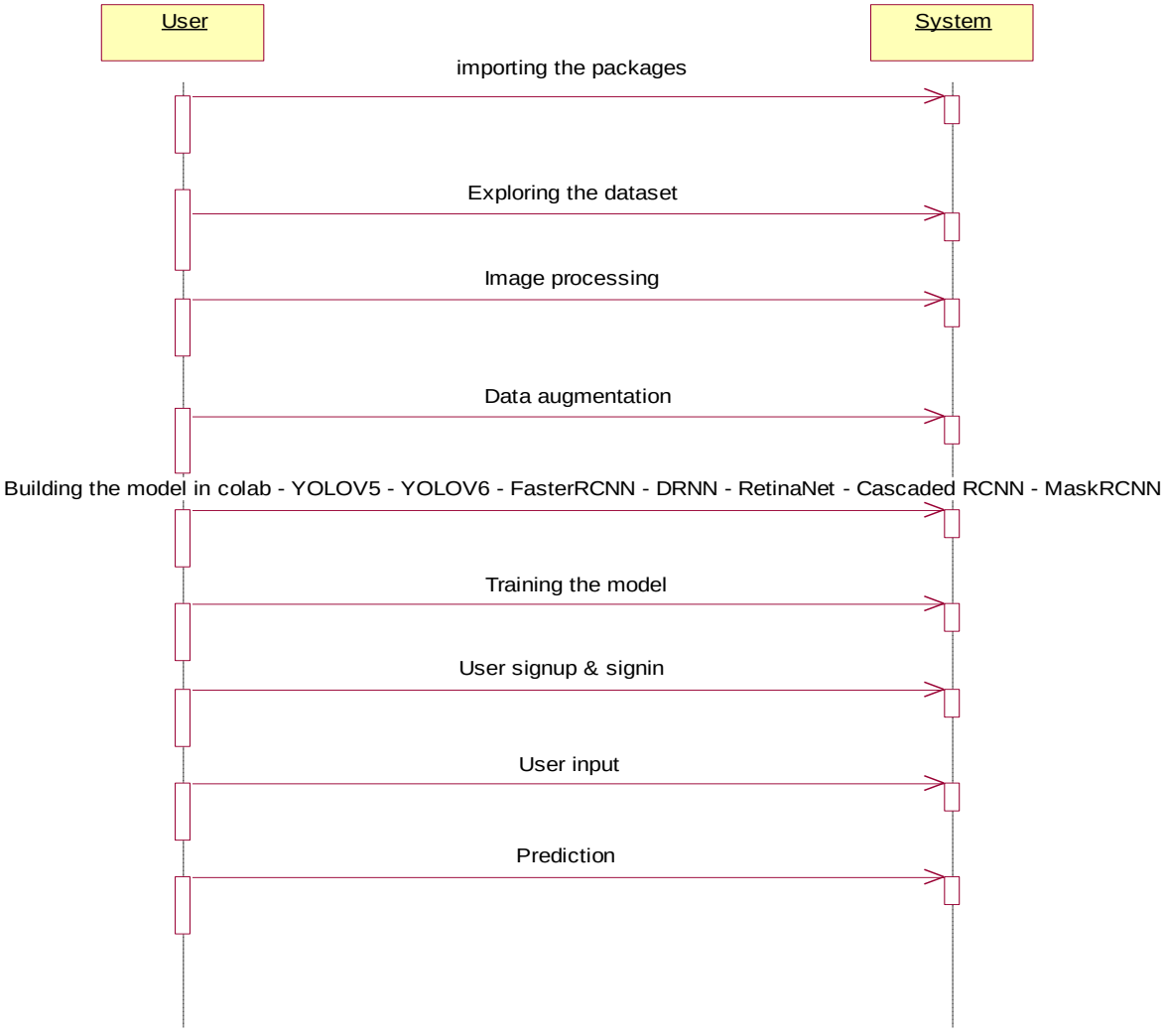


Fig.6.2.4 Sequence diagram

COLLABORATION DIAGRAM:

A collaboration diagram groups together the interactions between different objects. The interactions are listed as numbered interactions that help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with other objects.

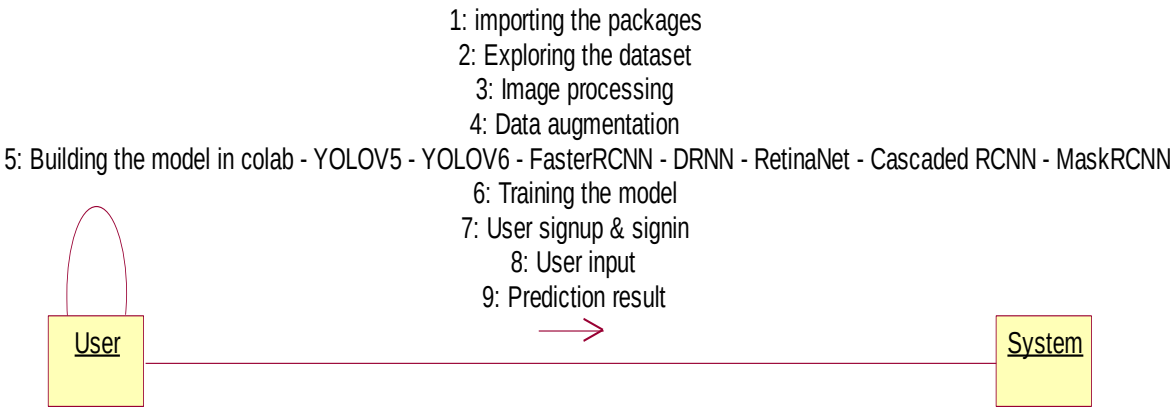


Fig.6.2.5 Collaboration diagram

COMPONENT DIAGRAM:

The component diagram represents the high-level parts that make up the system. This diagram depicts, at a high level, what components form part of the system and how they are interrelated. A component diagram depicts the components culled after the system has undergone the development or construction phase.

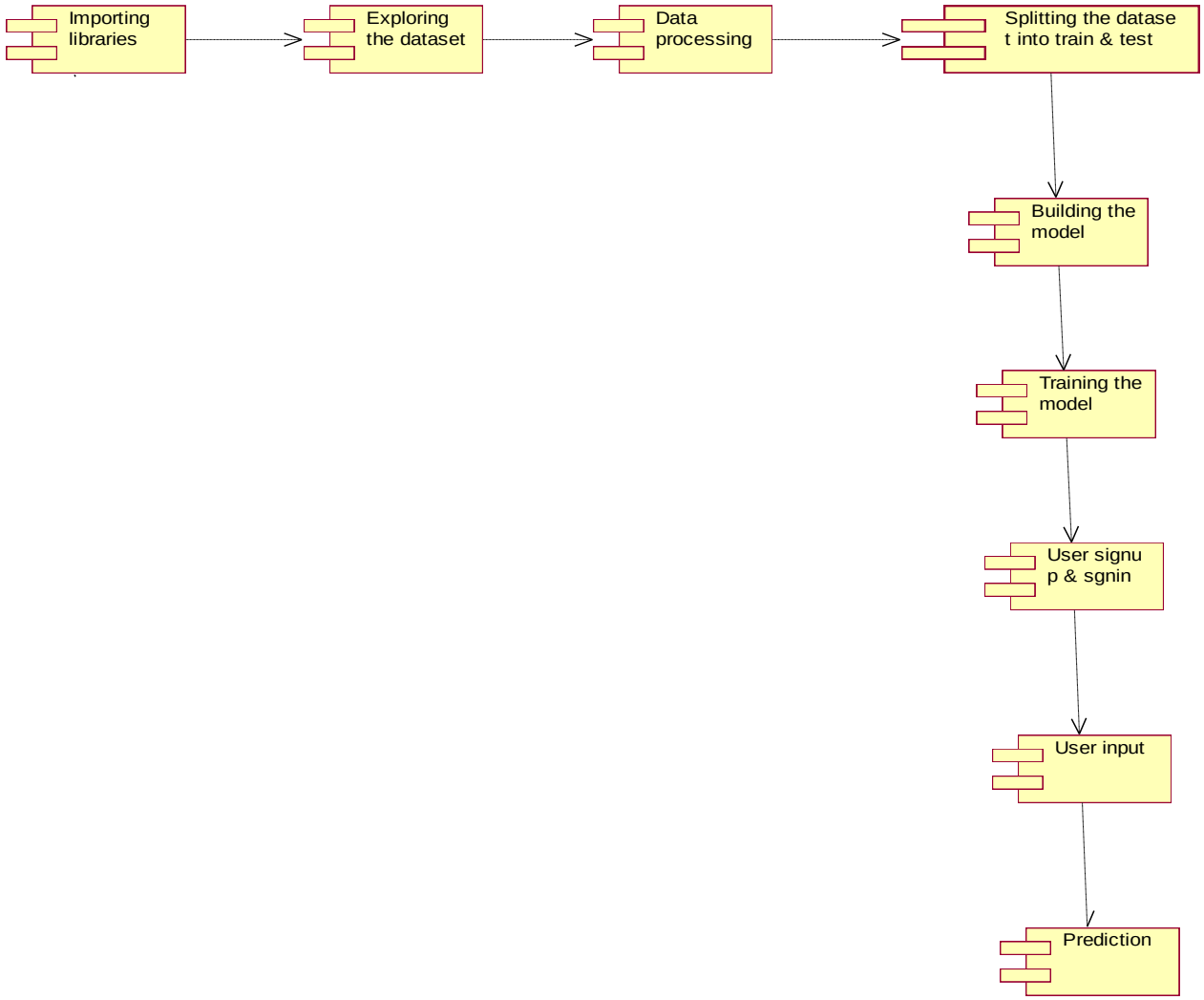


Fig.6.2.6 Component diagram

DEPLOYMENT DIAGRAM:

The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far most useful when a system is built and ready to be deployed.

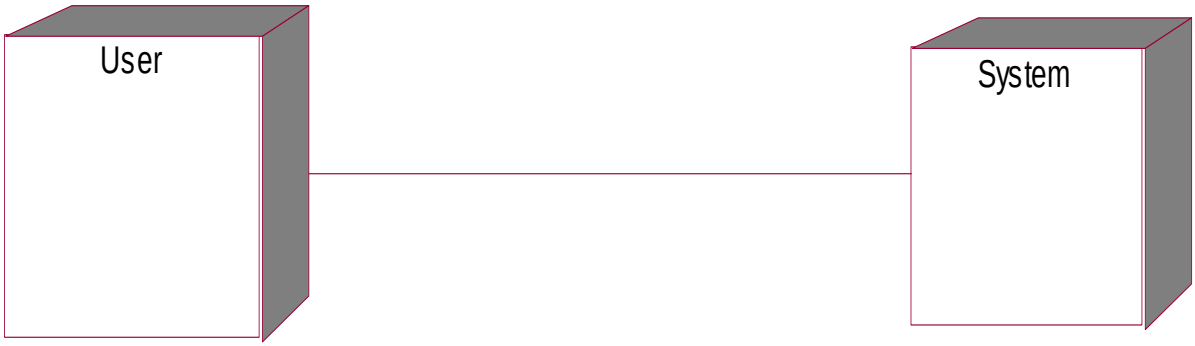


Fig.6.2.7 Deployment diagram

7. IMPLEMENTATION

MODULES:

- Data exploration: using this module we will load data into system
- Processing: Using the module we will read data for processing
- Data augmentation: using this module to artificially increase the amount of data by generating new data points from existing data.
- Model generation: Building the model in colab - YOLOV5 - YOLOV6 - FasterRCNN - DRNN - RetinaNet - Cascaded RCNN - MaskRCNN. Algorithms accuracy calculated.
- User signup & login: Using this module will get registration and login
- User input: Using this module will give input for prediction
- Prediction: final predicted displayed

ALGORITHMS:

YOLOV5- It is a novel convolutional neural network (CNN) that detects objects in real-time with great accuracy. This approach uses a single neural network to process the entire picture, then separates it into parts and predicts bounding boxes and probabilities for each component.

YOLOV6 – YOLOv6 is a single-stage object detection framework dedicated to industrial applications, with hardware-friendly efficient design and high performance. It outperforms YOLOv5 in detection accuracy and inference speed, making it the best OS version of YOLO architecture for production applications.

FasterRCNN – Faster R-CNN is a single-stage model that is trained end-to-end. It uses a novel region proposal network (RPN) for generating region proposals, which save time compared to traditional algorithms like Selective Search. It uses the ROI Pooling layer to extract a fixed-length feature vector from each region proposal.

DRNN – Recurrent neural networks (RNNs) are the state of the art algorithm for sequential data and are used by Apple's Siri and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data.

RetinaNet – RetinaNet is one of the best one-stage object detection models that has proven to work well with dense and small scale objects. For this reason, it has become a popular object detection model to be used with aerial and satellite imagery.

Cascaded RCNN – Cascade R-CNN is an object detection architecture that seeks to address problems with degrading performance with increased IoU thresholds (due to overfitting during training and inference-time mismatch between IoUs for which detector is optimal and the inputs).

MaskRCNN- Mask R-CNN is a state of the art model for instance segmentation, developed on top of Faster R-CNN. Faster R-CNN is a region-based convolutional neural networks, that returns bounding boxes for each object and its class label with a confidence score.

6.2 SAMPLE CODE:

```
Import
io
        import os
        import cv2
        import numpy as np
        from PIL import Image
        from keras import backend
        from base64 import b64encode
        from keras.models import Model
        import matplotlib.pyplot as plt
        from keras_retinanet import models
        from keras_retinanet.utils.colors import label_color
        from keras_retinanet.utils.visualization import draw_box, draw_caption
        from keras_retinanet.utils.image import read_image_bgr, preprocess_image, resize_image
        class Core:
            def __init__(self, model_filename: str = "/Trained-Model/drone-detection-v5.h5") -> None:
                self.current_path = os.getcwd()
                self.model_path = self.current_path + model_filename
                self.labels_to_names = {0: 'drone', 1: 'dummy'}
                self.model = None
            def get_model(self) -> Model:
                print(self.model_path)
```

```
        return models.load_model(self.model_path, backbone_name='resnet50')
    def set_model(self, model: Model) -> 'Core':
        self.model = model
        return self
    @staticmethod
    def load_image_by_path(filename: str) -> np.ndarray:
        return read_image_bgr(filename)
    @staticmethod
    def load_image_by_memory(file: bytes) -> np.ndarray:
        image = np.asarray(Image.open(io.BytesIO(file)).convert('RGB'))
        return image[:, :, ::-1].copy()
    @staticmethod
    def convert_rgb_to_bgr(image: np.ndarray) -> np.ndarray:
        return image[:, :, ::-1].copy()
    @staticmethod
    def pre_process_image(image: np.ndarray) -> tuple:
        pre_processed_image = preprocess_image(image)
        resized_image, scale = resize_image(pre_processed_image)
        return resized_image, scale
    @staticmethod
    def predict(model: Model, image: np.ndarray, scale: float) -> tuple:
        boxes, scores, labels = model.predict_on_batch(np.expand_dims(image, axis=0))
        boxes /= scale
        return boxes, scores, labels
    def predict_with_graph_loaded_model(self, image: np.ndarray, scale: float) -> tuple:
        with backend.get_session().as_default():
            with backend.get_session().graph.as_default():
                return self.predict(self.model, image, scale)
    def predict_with_graph(self, model: Model, image: np.ndarray, scale: float) -> tuple:
        with backend.get_session().graph.as_default() as g:
            return self.predict(model, image, scale)
    @staticmethod
    def clear_graph_session() -> None:
        backend.clear_session()
        return None
    @staticmethod
    def get_drawing_image(image: np.ndarray) -> np.ndarray:
        # copy to draw on
        drawing_image = image.copy()
        drawing_image = cv2.cvtColor(drawing_image, cv2.COLOR_BGR2RGB)
        return drawing_image
    def draw_boxes_in_image(self, drawing_image: np.ndarray, boxes: np.ndarray, scores:
np.ndarray,
                            threshold: float = 0.3) -> list:
        detections = []
        for box, score in zip(boxes[0], scores[0]):
            # scores are sorted so we can break
            if len(detections) > 0:
                threshold = 0.5
```

```
            if score < threshold:
                break
            detections.append({"box": [int(coord) for coord in box], "score": int(score * 100)})
            color = label_color(0)
            b = box.astype(int)
            draw_box(drawing_image, b, color=color)
            caption = "{} {:.3f}".format(self.labels_to_names[0], score)
            draw_caption(drawing_image, b, caption)
        return detections
    @staticmethod
    def visualize(drawing_image: np.ndarray) -> None:
        plt.figure(figsize=(15, 15))
        plt.axis('off')
        plt.imshow(drawing_image)
        plt.show()
    @staticmethod
    def convert_numpy_array_to_base64(image: np.ndarray, extension: str = ".png") -> bytes:
        data = cv2.imencode(extension, image)[1].tostring()
        return b64encode(data)
```

7. SOFTWARE ENVIRONMENT

PYTHON LANGUAGE:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python is a dynamic, high-level, free open source, and interpreted programming language. It supports object-oriented programming as well as procedural-oriented programming. In Python, we don’t need to declare the type of variable because it is a dynamically typed language. For example, x = 10 Here, x can be anything such as String, int, etc.

Features in Python:

There are many features in Python, some of which are discussed below as follows:

1. **FREE AND OPEN SOURCE**

Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

2. EASY TO CODE

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

3. EASY TO READ

As you will see, learning Python is quite simple. As was already established, Python's syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

4. OBJECT-ORIENTED LANGUAGE

One of the key features of Python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

5. GUI PROGRAMMING SUPPORT

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

6. HIGH-LEVEL LANGUAGE

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

7. EXTENSIBLE FEATURE

Python is an Extensible language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

8. EASY TO DEBUG

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program's issues once you understand how to interpret Python's error traces. Simply by glancing at the code, you can determine what it is designed to perform.

9. PYTHON IS A PORTABLE LANGUAGE

Python language is also a portable language. For example, if we have Python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

10. PYTHON IS AN INTEGRATED LANGUAGE

Python is also an Integrated language because we can easily integrate Python with other languages like C, C++, etc.

11. INTERPRETED LANGUAGE:

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called bytecode.

12. LARGE STANDARD LIBRARY

Python has a large standard library that provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in Python such as regular expressions, unit-testing, web browsers, etc.

13. DYNAMICALLY TYPED LANGUAGE

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

14. FRONTEND AND BACKEND DEVELOPMENT

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like javascript. Backend is the strong forte of Python it's extensively used for this work cause of its frameworks like Django and Flask.

15. ALLOCATING MEMORY DYNAMICALLY

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

8. SYSTEM TESTING

System testing, also referred to as system-level tests or system-integration testing, is the process in which a quality assurance (QA) team evaluates how the various components of an application interact together in the full, integrated system or application. System testing verifies that an application performs tasks as designed. This step, a kind of black box testing, focuses on the functionality of an application. System testing, for example, might check that every kind of user input produces the intended output across the application.

Phases of system testing:

A video tutorial about this test level. System testing examines every component of an application to make sure that they work as a complete and unified whole. A QA team typically conducts system testing after it checks individual modules with functional or user-story testing and then each component through integration testing.

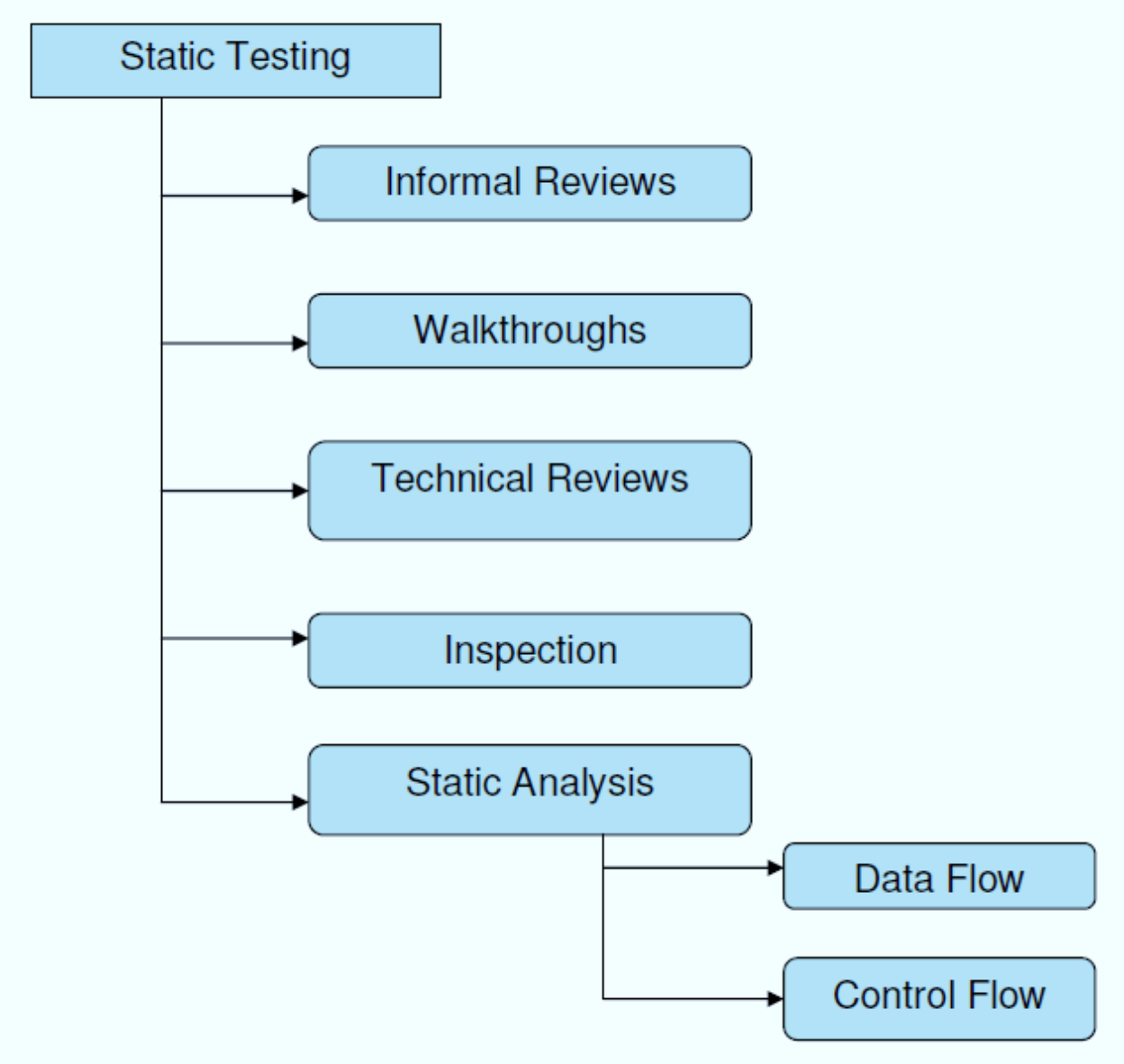
If a software build achieves the desired results in system testing, it gets a final check via acceptance testing before it goes to production, where users consume the software. An app-dev team logs all defects, and establishes what kinds and amount of defects are tolerable.

8.1SOFTWARE TESTING STRATEGIES:

Optimization of the approach to testing in software engineering is the best way to make it effective. A software testing strategy defines what, when, and how to do whatever is necessary to make an end-product of high quality. Usually, the following software testing strategies and their combinations are used to achieve this major objective:

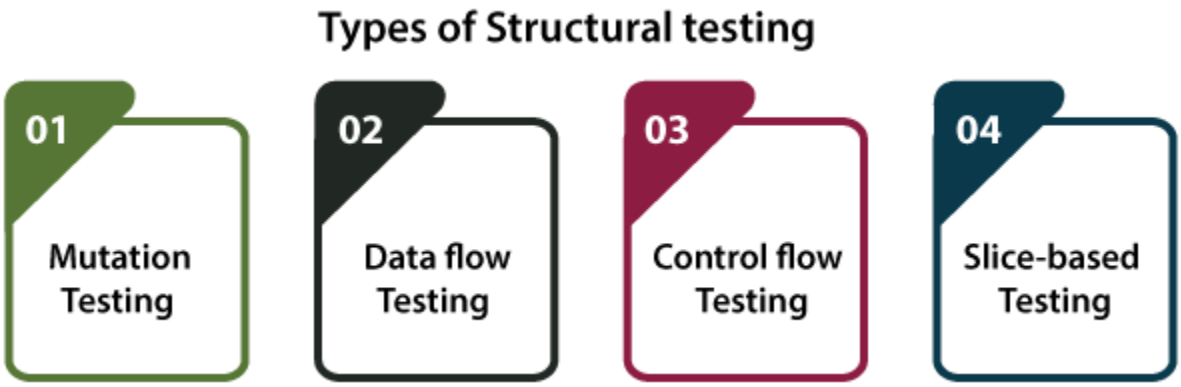
Static Testing:

The early-stage testing strategy is static testing: it is performed without actually running the developing product. Basically, such desk-checking is required to detect bugs and issues that are present in the code itself. Such a check-up is important at the pre-deployment stage as it helps avoid problems caused by errors in the code and software structure deficits.



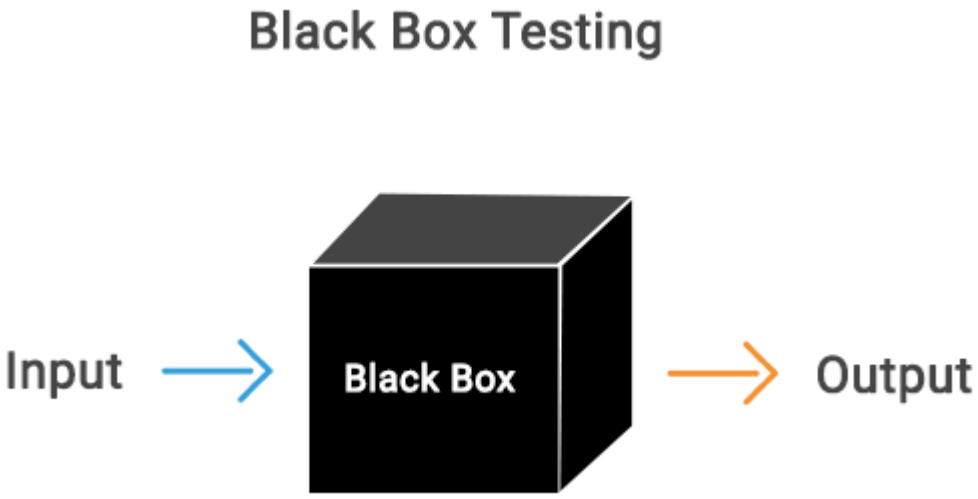
Structural Testing:

It is not possible to effectively test software without running it. Structural testing, also known as white-box testing, is required to detect and fix bugs and errors emerging during the pre-production stage of the software development process. At this stage, unit testing based on the software structure is performed using regression testing. In most cases, it is an automated process working within the test automation framework to speed up the development process at this stage. Developers and QA engineers have full access to the software's structure and data flows (data flows testing), so they could track any changes (mutation testing) in the system's behavior by comparing the tests' outcomes with the results of previous iterations (control flow testing).



Behavioral Testing:

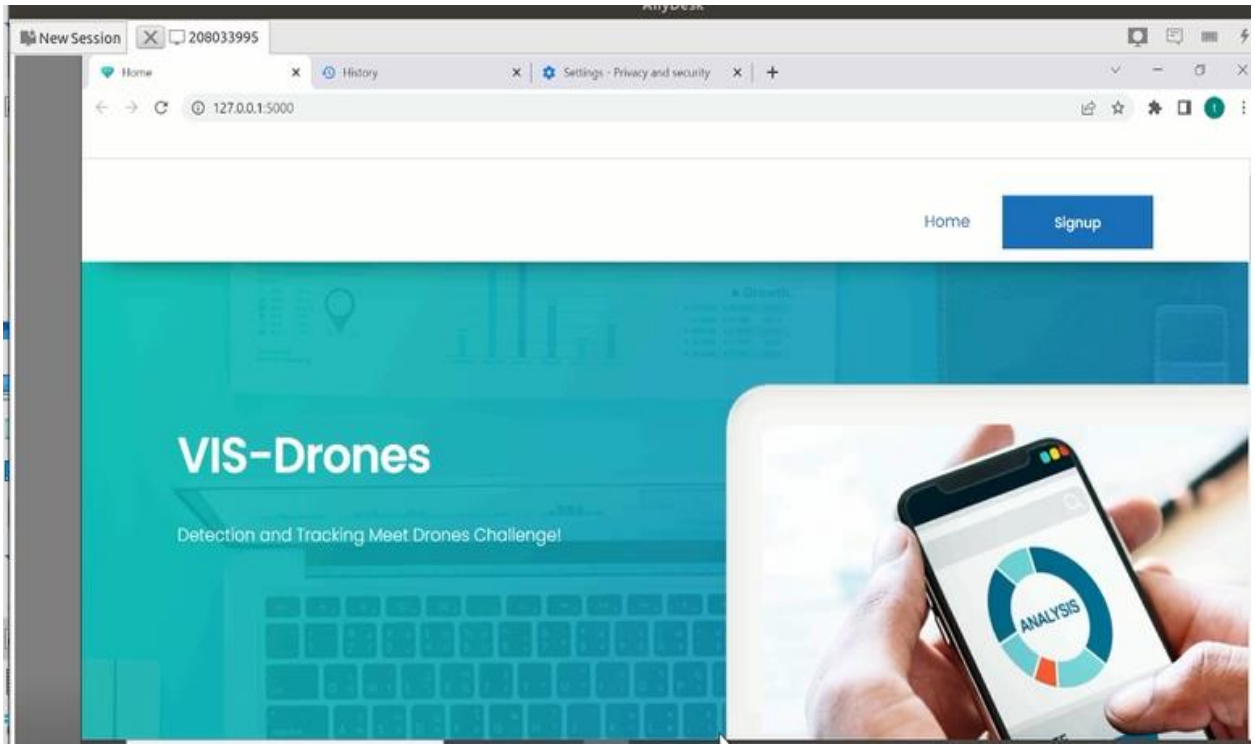
The final stage of testing focuses on the software's reactions to various activities rather than on the mechanisms behind these reactions. In other words, behavioral testing, also known as black-box testing, presupposes running numerous tests, mostly manual, to see the product from the user's point of view. QA engineers usually have some specific information about a business or other purposes of the software ('the black box') to run usability tests, for example, and react to bugs as regular users of the product will do. Behavioral testing also may include automation (regression tests) to eliminate human error if repetitive activities are required. For example, you may need to fill 100 registration forms on the website to see how the product copes with such an activity, so the automation of this test is preferable.



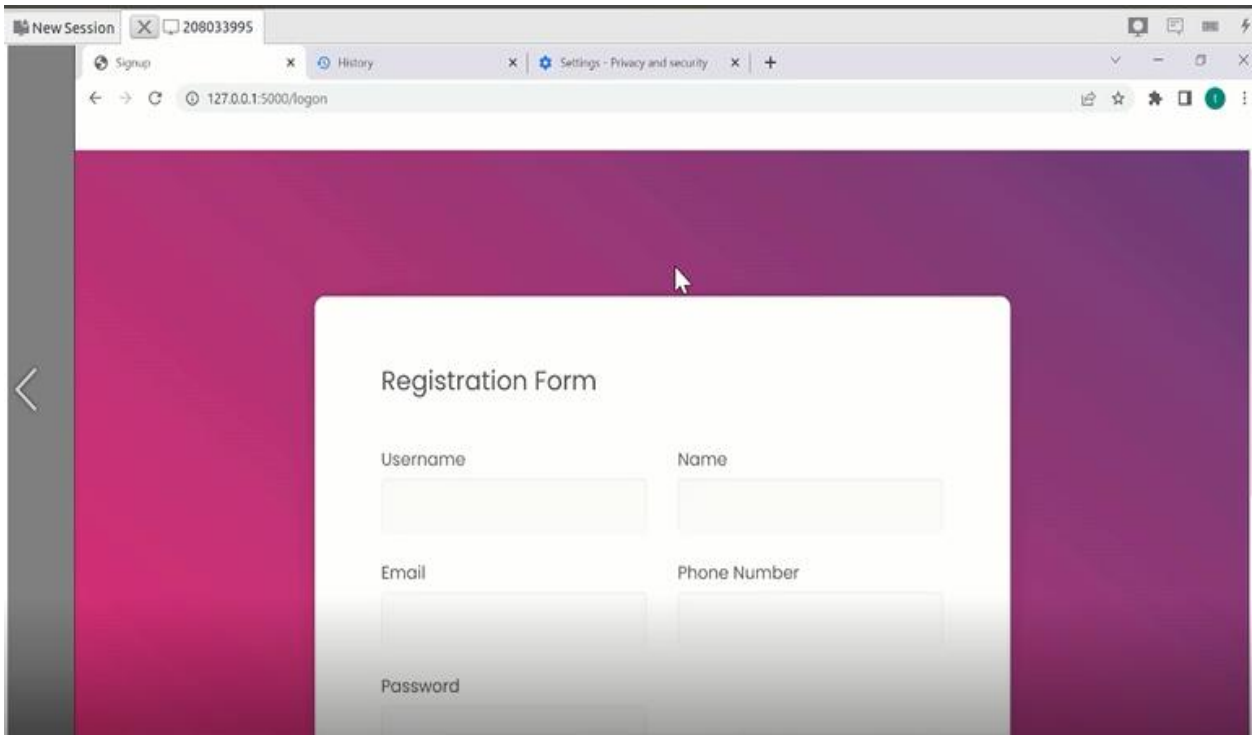
8.2 TEST CASES:

S.NO	INPUT	If available	If not available
1	User signup	User get registered into the application	There is no process
2	User signin	User get login into the application	There is no process
3	Enter input for prediction	Prediction result displayed	There is no process

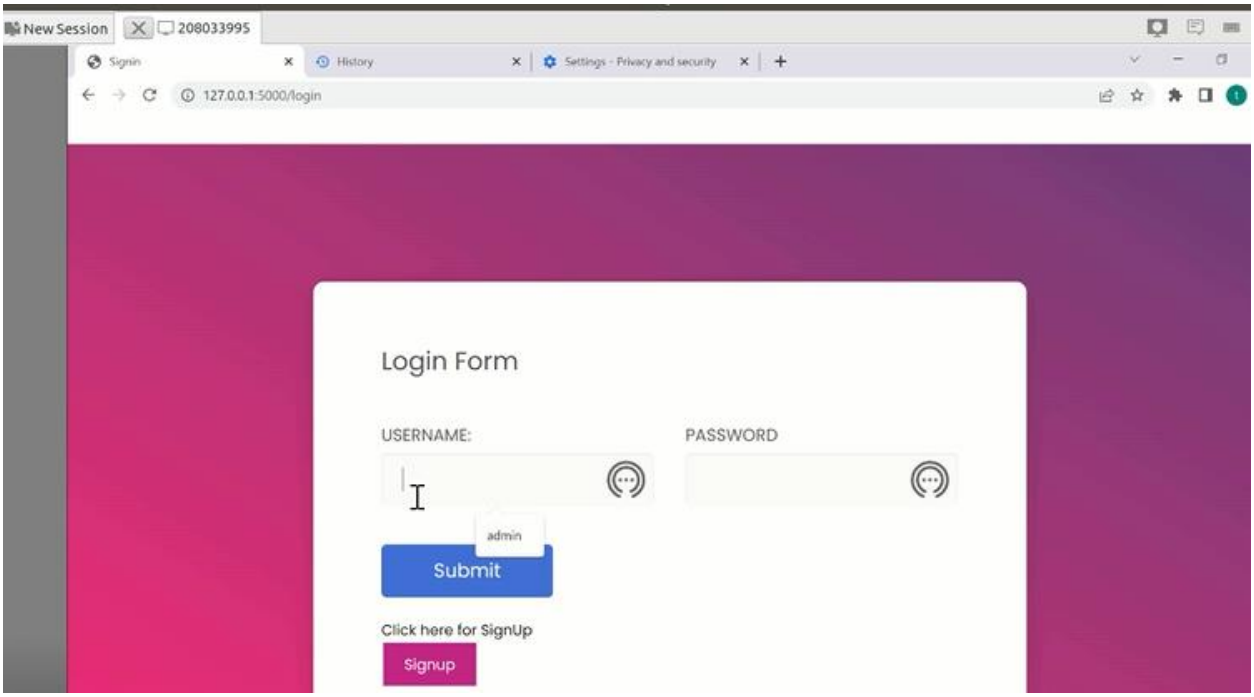
10. Outcomes



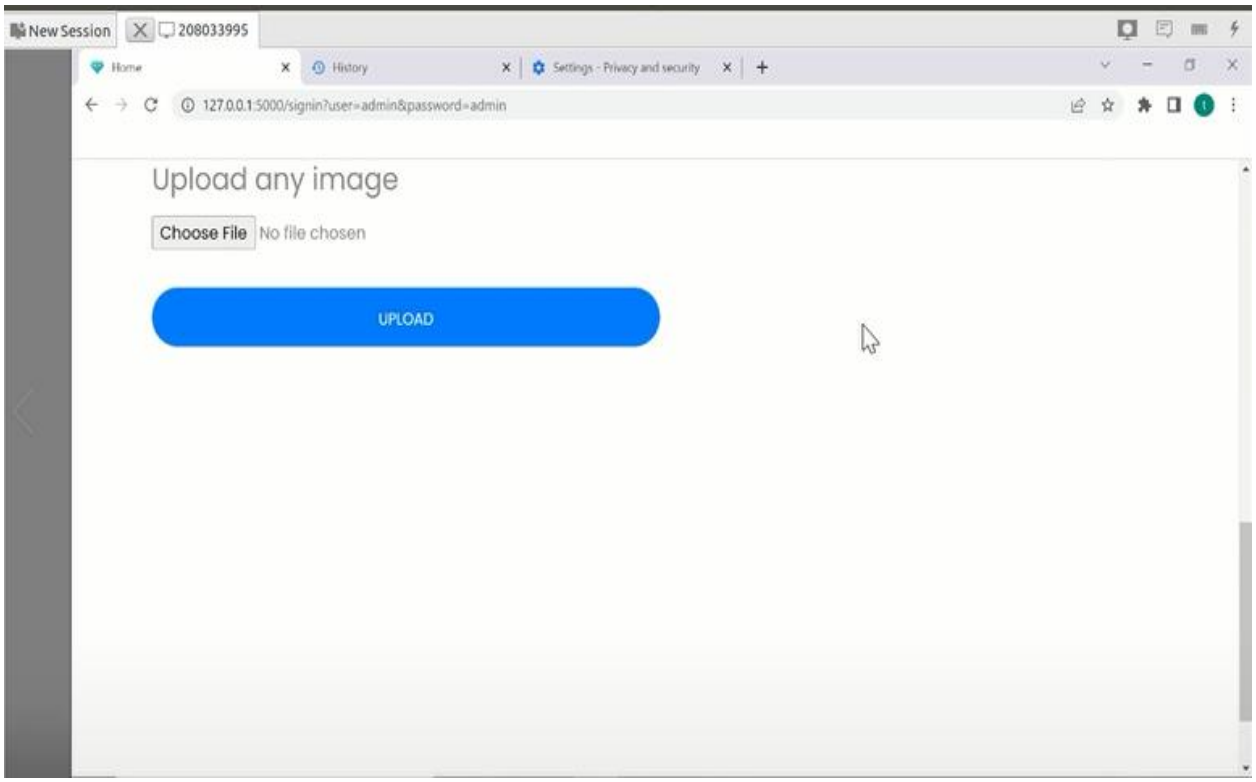
User interface



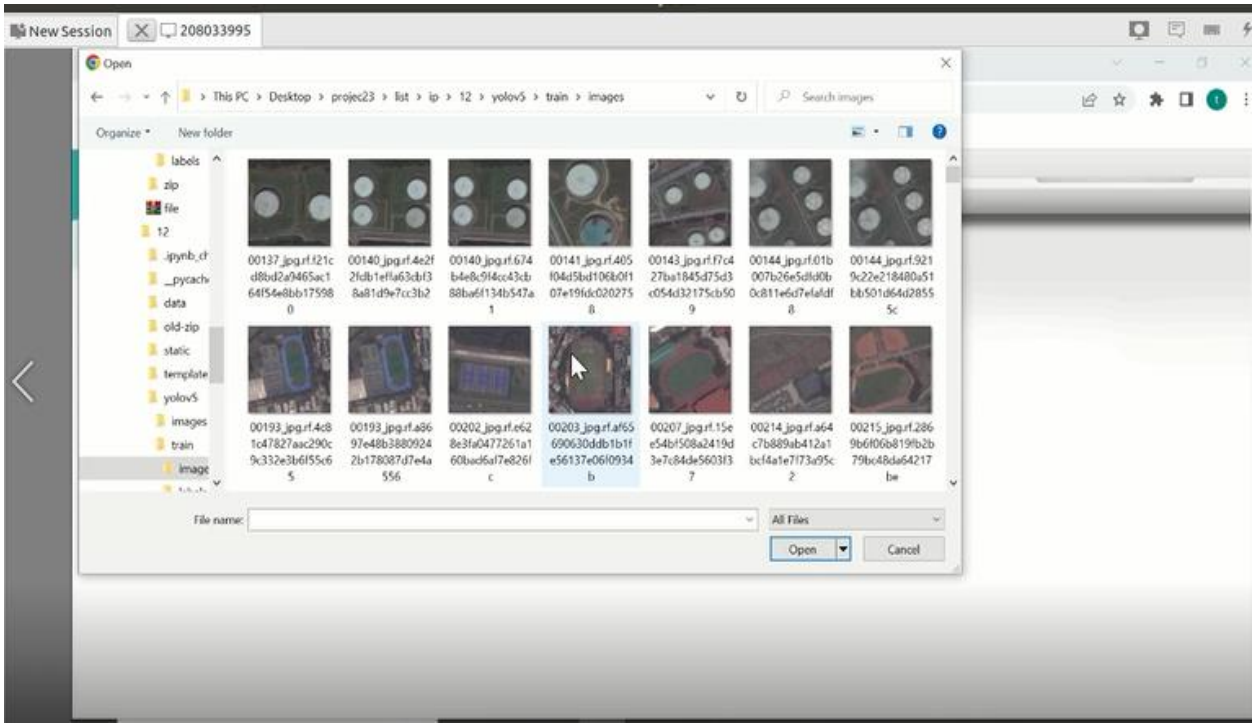
Sign-up page



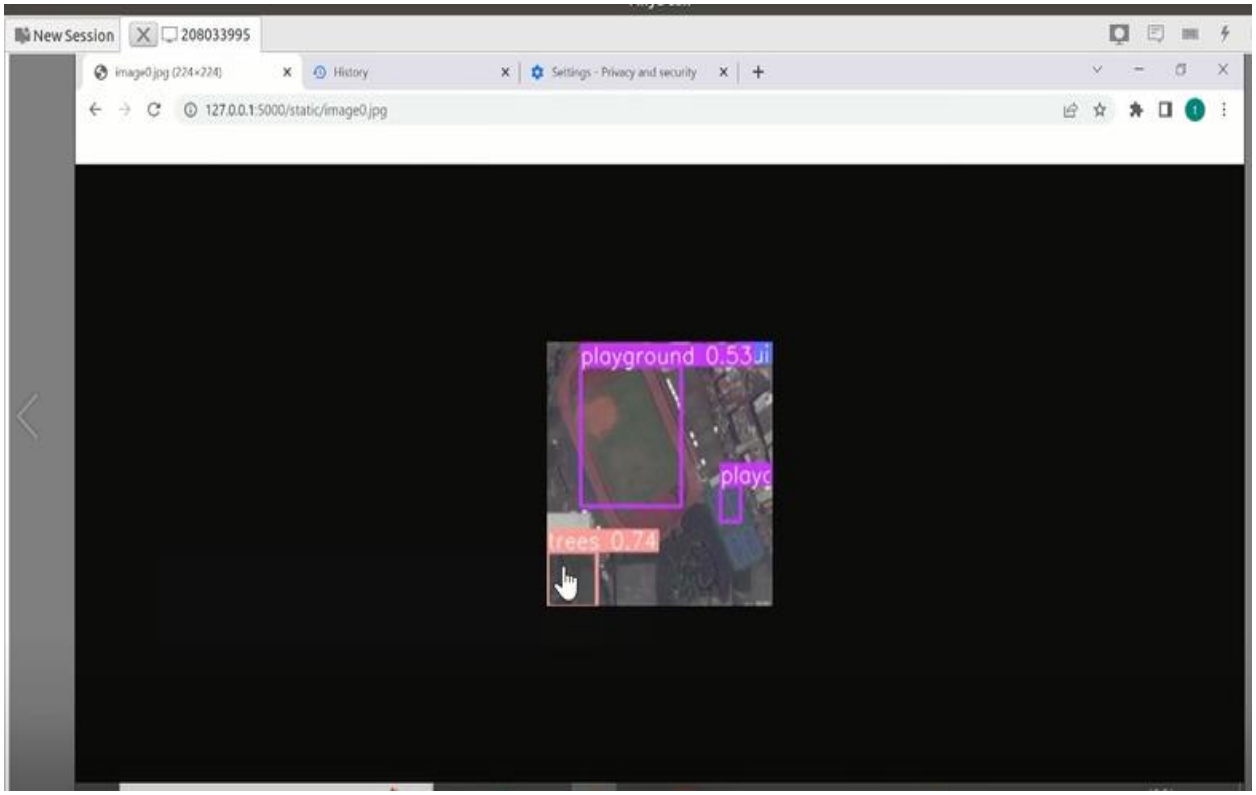
Login page



Choosing Image



Selecting image



Detection of the object

11. CONCLUSION

We introduce a new large-scale benchmark, VisDrone, to facilitate the research of object detection and tracking on drone captured imagery. With over 6,000 worker hours, a vast collection of object instances are gathered, annotated, and organized to drive the advancement of object detection and tracking algorithms.

12. REFERENCES

[1] G. V. Research, “Commercial drone market size, share & trends analysis report by product (fixed-wing, rotary blade, hybrid), by application, by end-use, by region, and segment forecasts, 2021 - 2028,” https://www.reportlinker.com/p06068219/?utm source=GNW, 2021.

[2] M. Mueller, N. Smith, and B. Ghanem, “A benchmark and simulator for UAV tracking,” in ECCV, 2016, pp. 445–461.

[3] I. Kalra, M. Singh, S. Nagpal, R. Singh, M. Vatsa, and P. B. Sujit, “Dronesurf: Benchmark dataset for drone-based face recognition,” in FG, 2019, pp. 1–7.

[4] D. Du, Y. Qi, H. Yu, Y. Yang, K. Duan, G. Li, W. Zhang, Q. Huang, and Q. Tian, “The unmanned aerial vehicle benchmark: Object detection and tracking,” in ECCV, 2018, pp. 375–391.

[5] M. Hsieh, Y. Lin, and W. H. Hsu, “Drone-based object counting by spatially regularized regional proposal network,” in ICCV, 2017.

[6] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, “Learning social etiquette: Human trajectory understanding in crowded scenes,” in ECCV, 2016, pp. 549–565.

[7] P. Zhu, L. Wen, D. Du, X. Bian, H. Ling, Q. Hu, and et al., “Visdrone-det2018: The vision meets drone object detection in image challenge results,” in ECCV Workshops, 2018, pp. 437–468.

[8] L. Wen, P. Zhu, D. Du, X. Bian, H. Ling, Q. Hu, and et al., “Visdrone-sot2018: The vision meets drone single-object tracking challenge results,” in ECCV Workshops, 2018, pp. 469–495.

[9] P. Zhu, L. Wen, D. Du, X. Bian, H. Ling, Q. Hu, and et al., “Visdrone-vdt2018: The vision meets drone video detection and tracking challenge results,” in ECCV Workshops, 2018, pp. 496– 518.

[10] D. Du, P. Zhu, L. Wen, X. Bian, H. Ling, Q. Hu, and et al., “Visdrone-det2019: The vision meets drone object detection in image challenge results,” in ICCV Workshops, 2019.