Project report :

Real-Time Human Detection and Counting using OpenCV

ABSTRACT

The development of deep learning has greatly impacted how AI is used around the world. Some of the most popular techniques used for object identification include Flow flow, You Only Look Once, RCNN, Faster-RCNN, and SSD. While YOLO is faster than SSD and Faster-RCNN, it is better at detecting objects. By utilizing MobileNets and SSDs, Deep Learning can make it possible to perform effective object detection.

The paper presents a method that can perform real-time crowd counting that utilizes deep learning and cutting-edge techniques. Unlike previous methods, this method does not rely on morphological filters or edge recognition. It utilizes a combination of YOLOv5, OpenCV, and image processing.

Our method ensures the efficiency of the people counters, making it a valuable addition to applications requiring instantaneous crowd analysis. Our experiments demonstrate the reliability of our real-time crowd-counting technique. With an average MAE of 3.2 across all test situations, it achieves an incredibly high degree of accuracy. The average MSE of 12.4 emphasizes its effectiveness even further. When compared to conventional crowd-counting techniques, these metrics show a significant decline in errors.

Keywords: Object detection, object tracking, OpenCV.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
1	**INTRODUCTION**	**11**
	1.1 OBJECTIVES	11
	1.2 SOFTWARE TOOLS USED	13
	1.3 PURPOSE	16
	1.4 PROJECT SCOPE	18
	1.5 FACTS AND STATS	20
	1.6 PROPOSED WORK	22
2	**LITERATURE SURVEY**	**23**
	2.1 RELATED WORKS	24
3	**SYSTEM ANALYSIS**	**26**
	3.1 SOFTWARE REQUIREMENTS	26
4	**SYSTEM DESIGN**	**27**
	4.1 DATA FLOW DIAGRAM	27
	4.2 UML DIAGRAM	
	4.2.1 USE CASE DIAGRAM	29
	4.2.2 STATE CHART DIAGRAM	30

	4.2.3 ARCHITECTURE DIAGRAM	31
5	**PROPOSED ALGORITHM**	**32**
	5.1 ALGORITHM USED IN MODULES	32
	5.1.1 RCN ALGORITHM	32
	5.1.2 CNN ALGORITHM	33
	5.2 Dataset description and sample data	34
	5.2.1 Data set information	34
6	**IMPLEMENTATION**	**35**
	6.1 FILE STRUCTURE	35
	6.2 CODE	36
	6.3 MAIN IMPLEMENTATION	39
	6.4 OUTPUT	40
7	**RESULTS AND DISCUSSIONS**	**40**
8	**CONCLUSION**	**42**
9	**FUTURE ENHANCEMENTS**	**43**
10	REFERENCES	44
	PLAGIARISM REPORT	46
	PAPER PUBLICATION STATUS	47

LIST OF FIGURES

FIG. NO.	TITLE	PAGE NO.
4.1	DATA FLOW DIAGRAM	28
4.2	USE CASE DIAGRAM	29
4.3	STATE CHART DIAGRAM	30
4.5	ARCHITECTURE DIAGRAM	31
5.1	FLOWCHART OF RCN	32
5.2	FLOWCHART OF CNN	33
6.1	CODE	36
6.2	OUTPUT	39

LIST OF TABLES

TABLE NO.	TITLE	PAGE NO.
2.1	Literature review	10
7.1	Result	53

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
ML	Machine Learning
DL	Deep Learning
IDLE	Integrated Development and Learning Environment
RAM	Random access memory
SVM	Support Vector Machine

CHAPTER 1
INTRODUCTIO
N

Crowd detection algorithms, which are based on computer vision and machine learning, are designed to automatically identify and analyze densely populated areas in images and videos. They find wide-ranging applications, including crowd management, security, and urban surveillance. These algorithms go through essential steps, such as preprocessing, feature extraction, object detection, crowd density estimation, behavior analysis, and post-processing. Their complexity and accuracy can vary depending on specific use cases and available resources, with deep learning methods like CNNs and RNNs substantially enhancing accuracy. These algorithms play a crucial role in monitoring public events, improving transportation hub management, and strengthening security in crowded spaces, and ongoing technological advancements are continually expanding their usefulness across various.

1.1Objectives

Crowd detection algorithms are versatile in their objectives, which depend on the application and context. These algorithms are tailored to achieve several common aims:

1. **Counting Crowds:** The fundamental objective is to accurately quantify the number of people in a crowd. This is vital for crowd control, event planning, and public safety.
2. **Estimating Crowd Density:** These algorithms aim to gauge the concentration of individuals within defined areas, providing insights into crowd congestion levels.
3. **Detecting Anomalies:** Crowd detection algorithms are equipped to spot unusual or suspicious behaviors in crowds, serving critical roles in security and public safety applications.
4. **Analyzing Crowd Movements:** Understanding crowd dynamics, flow, and direction is crucial for optimizing traffic management, event planning, and urban design.
5. **Behavior Analysis:** Some algorithms delve into individual behavior within crowds, examining social interactions and identifying potential issues like overcrowding.

Real-time Monitoring: Crowd detection algorithms furnish real-time data on crowd dynamics,

density, and behavior, facilitating swift responses during emergencies and aiding in crowd management.

6. **Heighted Security**: These algorithms contribute to security enhancements by identifying potential threats or individuals of interest within crowds.

7. **Traffic Control:** In the domains of transportation and urban planning, these algorithms assist in monitoring and optimizing pedestrian and vehicular movement, mitigating congestion and enhancing safety.

8. **Event Planning:** Event organizers benefit from these algorithms as they help assess crowd sizes, plan crowd control strategies, and ensure the safety and comfort of event attendees.

9. **Urban Surveillance:** In urban settings, these algorithms are deployed for monitoring public spaces, transportation hubs, and tourist destinations, bolstering security and supporting effective urban planning.

The objectives of crowd detection algorithms are adaptable and often require a combination of computer vision, machine learning, and data analysis techniques to suit the specific objectives of each application.

1.3 Purpose

Crowd detection algorithms represent sophisticated tools that serve a wide range of vital purposes and applications. Their central aim is to automatically identify and analyze densely populated areas, particularly groups of people, in images or video sequences. These algorithms play various essential roles:

Foremost among their capabilities is **crowd counting**, excelling at providing precise estimates of the number of individuals within a crowd. This information proves indispensable for efficient crowd management, meticulous event planning, and ensuring public safety. It equips decision-makers with the data required to optimize resource allocation and establish robust security measures.

Moreover, crowd detection algorithms demonstrate proficiency in **crowd density estimation** within specified regions, shedding light on the level of crowding at different places and times. This information proves pivotal for urban planning, traffic management, and the seamless organization of events.

In the domain of security and public safety, these algorithms stand out for their prowess in **anomaly detection**. They possess the capability to swiftly identify unusual or suspicious behaviors occurring within a crowd, allowing for timely responses to potential threats or issues.

Understanding the dynamics of crowd movement is a cornerstone of these algorithms. They meticulously scrutinize **crowd movement patterns**, the flow of individuals, and directional cues. This knowledge is crucial for optimizing traffic management, enhancing event planning, and contributing to more efficient urban design, ultimately resulting in improved traffic flow and pedestrian safety.

Taking analysis to a deeper level, some crowd detection algorithms delve into **behavior analysis** within crowds. This involves a comprehensive examination of crowd dynamics, social interactions, and potential challenges such as overcrowding. These insights provide a profound understanding of crowd psychology, facilitating effective event management and security measures.

Real-time monitoring is another pivotal aspect, with crowd detection algorithms delivering **real-time insights** into crowd dynamics, density, and behavior. This real-time data equips authorities

and event organizers with the necessary tools for prompt responses to emergencies and the optimization of crowd control measures.

Furthermore, these algorithms substantially fortify **security**, as they adeptly identify potential threats or individuals of interest within crowds. This reinforces security measures in public spaces, transportation hubs, and other crowded environments.

In the realm of transportation and urban planning, crowd detection algorithms are irreplaceable for **traffic management**. They support the monitoring and optimization of traffic flow and pedestrian movement, ultimately mitigating congestion and enhancing safety.

Event planners find immense value in these algorithms as they assist in **event planning** by providing precise assessments of crowd sizes and facilitating the implementation of crowd control measures, ensuring the safety and comfort of event attendees.

Lastly, in urban settings, **urban surveillance** harnesses crowd detection algorithms for the monitoring of public spaces, transportation hubs, and tourist destinations. This enhances security and plays a pivotal role in effective urban planning.

In summary, crowd detection algorithms are versatile and invaluable tools that fulfil a multitude of purposes, spanning public safety, event management, urban development, and security. Their capacity to offer insights into crowd dynamics empowers organizations and authorities to make informed decisions, optimizing resource allocation and enhancing the management of crowded spaces.

1.4 Project Scope

Defining the project scope for a crowd detection algorithm involves several critical elements and considerations that are crucial for its successful execution. To begin with, it is essential to establish clear and specific project objectives. These objectives can range from tasks like crowd counting, density estimation, anomaly detection, or behaviour analysis. Additionally, it's important to determine the specific application domain where the algorithm will be deployed. This could include areas such as public safety, event management, urban planning, transportation, or security.

Data sources play a fundamental role in the project scope. Deciding on the type of data you will work with, whether it's video footage from surveillance cameras, images from public events, or other visual data sources, is a key decision. Moreover, you need to outline the procedures for data collection and annotation. This involves deciding whether you will use existing datasets or create custom ones.

The choice of algorithms is pivotal, and selecting the most appropriate one for your specific objectives is crucial. Common options include YOLO, Faster R-CNN, SSD, or even the development of custom deep learning architectures. The model training process, including aspects like hyperparameter selection, data preprocessing, and data augmentation techniques, is of great significance.

Defining the performance metrics that will be used to evaluate the algorithm is also essential. Metrics like accuracy, precision, recall, F1 score, and computational efficiency need to be specified. Hardware and software requirements should be outlined, including factors like the need for GPUs for model training, the selection of deep learning frameworks, and image processing libraries like OpenCV.

The decision of whether the algorithm should operate in real-time or offline mode is another critical aspect. Real-time processing may require more computational resources and optimization. Integration and deployment details must be determined, whether it involves deployment on edge devices, cloud services, or integration with existing surveillance systems. Addressing ethical and legal considerations, such as privacy issues and compliance with regulations, is crucial. Testing and validation procedures should be clearly defined to ensure

the algorithm performs as intended, encompassing both controlled testing environments and real-world scenarios.

Scalability and performance optimization are crucial factors to effectively handle varying crowd sizes and different settings. Documentation requirements for the project, including user manuals and technical documentation, need to be detailed. A project timeline with milestones and deadlines should be established to track progress from data collection to algorithm deployment. Budget and resource allocation considerations are important, covering expenses such as hardware, software, data annotation, and other project-related costs. Additionally, it's essential to identify potential risks and challenges and develop mitigation strategies to ensure the project's success.

1.5 Facts and Stats

OpenCV has been extensively employed in various applications, particularly in the domains of public safety, event management, and surveillance, for crowd detection algorithms. OpenCV, being an open-source computer vision library, provides a flexible platform for implementing these algorithms. The choice of algorithm can range from traditional methods like Haar Cascade classifiers and Histogram of Oriented Gradients (HOG) for detecting pedestrians to more modern deep learning approaches integrated with OpenCV, such as YOLO and Faster R-CNN. The accuracy and performance of these algorithms are influenced by factors like the quality of the dataset, the model architecture, and the training parameters. Deep learning models typically offer high accuracy but require significant computational resources. Benchmark datasets like the UCSD Pedestrian dataset and the ShanghaiTech dataset are commonly used for assessment. Achieving real-time crowd detection is possible with efficient algorithm selection and the use of hardware accelerators. The field of crowd detection is continuously advancing, with ongoing research and innovations focused on improving accuracy and real-time capabilities. To access the latest developments and statistics, it is recommended to refer to more recent sources and research within the computer vision community.

OpenCV remains a fundamental tool for a wide array of computer vision tasks, with crowd detection being no exception. Various algorithms are employed in this context, spanning from conventional methods such as Haar Cascade classifiers and Histogram of Oriented Gradients (HOG) to cutting-edge deep learning models like YOLO (You Only Look Once) and Faster R-CNN. While these deep learning models have demonstrated impressive accuracy, they often entail significant computational resources, rendering them well-suited for scenarios where real-time performance can be attained with high-end hardware.

Crucial benchmark datasets, like the UCSD Pedestrian dataset and the ShanghaiTech dataset, continue to play a pivotal role in the evaluation of crowd detection algorithms, aiding researchers and developers in assessing the efficacy of their algorithms. The pursuit of achieving real-time crowd detection remains an ongoing endeavour, and the selection of efficient algorithms and the utilization of hardware accelerators are instrumental in realizing this objective.

Moreover, the practice of customizing and fine-tuning algorithms within OpenCV is widespread, providing developers with the flexibility to tailor algorithms to meet specific needs, especially in

diverse applications like public safety, event management, and retail analytics. Despite these advancements, challenges persist, particularly in crowded and intricate scenes marked by occlusions and varying lighting conditions. This underscores the need for continual research and innovation within the field of crowd detection. OpenCV continues to evolve, with active contributions from the community aimed at enhancing its capabilities for crowd detection and various other computer vision tasks. To access the most up-to-date information, statistics, and developments in the realm of crowd detection using OpenCV, it is imperative to refer to more recent sources and research within the dynamic field of computer vision.

1.6 Proposed Work

A suggested research project in the realm of crowd detection algorithms using OpenCV aims to push the boundaries in several critical aspects. Initially, it focuses on developing highly efficient and precise deep learning models specifically tailored for crowd detection, finding a harmonious balance between accuracy and computational demands. This entails fine-tuning neural network architectures and exploring streamlined variations to enable real-time operation on standard hardware. Additionally, there is a need to create more varied and challenging crowd datasets, encompassing scenarios marked by intricate occlusions, diverse crowd behaviors, and fluctuations in lighting conditions. These datasets are pivotal for enhancing the resilience and adaptability of algorithms and should encompass a wide array of real-world applications such as urban surveillance, event management, and public safety. Furthermore, the exploration of innovative approaches for multi-modal crowd analysis, which integrates visual data with other sensor inputs like audio or lidar, promises to enhance our comprehension of crowds. Effective crowd tracking methods and the ability to detect anomalies within crowds are also areas of vital research for practical applications. Lastly, the research project should consider harnessing edge computing and deploying crowd detection algorithms on edge devices to enable decentralized and low-latency crowd monitoring in smart cities and other pertinent settings. The amalgamation of these endeavors holds the potential to drive progress in the field of crowd detection using OpenCV, rendering it more efficient, adaptable, and accessible across a wide spectrum of applications.

CHAPTER 2
LITERATURE SURVEY

For this section, we have selected a set of 30 research papers. For which we have done the literature survey. we have studied this paper to find out the which-which different algorithms are used for which dataset. What is its result, performance measure, and conclusion for each paper. Also, what are its futurework that needs to be implemented, we have studied and made a table given below.

In recognition-based techniques, crowd counting is considered an object- or person-identifying problem, which holds the view that a crowd is made up of discrete items. [1]. Early attempts at human identification included handcrafted features, but these features were not resistant to high levels of large-scale variance or occlusion in crowded scenes or packed situations [2]. Deep network-based object detectors have achieved outstanding object detection results recently, but they still beat regression-based approaches when it comes to counting crowds [5]. a face recognition system that relies on video and detects people's faces [6]. The work done here effectively handles difficult situations, including surveillance footage with poor frame quality and numerous shot videos. With masks on, it does not function, though. Many researchers have studied RGBD crowd counts to better estimate populations. Most of these studies focus on leveraging depth to improve person/head detection in crowd scenes. [8] On the other hand, their regression module does not perform better than their detection module. The regression module that combines crowd classification with density map estimation does not fully utilize the depth map because it does not directly provide data to the depth map [7] and therefore has lower accuracy in recognizing faces.

The review of literature on crowd estimation methods employing OpenCV indicates a widespread and varied application of this computer vision library in tackling issues related to crowds. Researchers frequently utilize OpenCV for detecting and tracking objects in crowded environments, capitalizing on its comprehensive image processing capabilities. The integration of machine learning methods, encompassing both pre-trained models and bespoke algorithms, stands out as a notable trend, augmenting the precision of crowd size predictions. Numerous studies showcase the real-world applicability of OpenCV-based crowd estimation, spanning

domains such as event coordination, public safety, and smart city initiatives. Challenges, including occlusions, varying lighting conditions, and real-time processing constraints, are recurrent themes, offering

insights for refining existing approaches. The literature also underscores the importance of employing OpenCV for generating heatmaps and implementing density estimation techniques, providing visualizations for understanding and analyzing crowd distribution. In essence, the literature highlights the versatility of OpenCV, demonstrating its crucial role in the development of effective and applicable crowd estimation solutions across a spectrum of domains.

CHAPTER 3
SYSTEM
ANALYSIS

System requirements encompass the essential configurations necessary for the smooth and effective operation of a hardware or software application. Hardware requirements can prevent the installation of a device or application, while software requirements may lead to malfunctions, suboptimal performance, or even system crashes. Hardware system prerequisites typically outline specifications such as the operating system version, processor type, memory capacity, available disk space, and any requisite peripherals. Software system requirements go beyond these and may also delineate additional software dependencies, including libraries, driver versions, and framework requirements.

3.1 SOFTWARE REQUIREMENTS

To implement a crowd detection algorithm using OpenCV, specific software prerequisites must be met. Foremost, having the appropriate version of OpenCV, an open-source computer vision library, is crucial. Python, a common choice for developing computer vision applications, is also essential, and your Python environment should include indispensable libraries like NumPy and Matplotlib for effective data manipulation and result analysis.

Moreover, if your crowd detection algorithm relies on advanced neural network models for object detection or image analysis, you may need machine learning or deep learning frameworks like TensorFlow or PyTorch. Access to a dataset containing annotated crowd images for training and testing is another important aspect of this process. Lastly, it's vital to configure your development environment, whether it's an integrated development environment (IDE) or a text editor, to work seamlessly with OpenCV and Python. Meeting these software requirements will prepare you to successfully implement a crowd detection algorithm using OpenCV and related tools in the fields of computer vision and machine learning.

1.2 Software tools used

In the realm of crowd detection algorithms within computer vision, the development and deployment process often rely on a curated selection of software tools and libraries. These essential components collectively form the backbone of a crowd detection system based on YOLO (You Only Look Once):

The core of the system revolves around YOLO, a renowned deep learning model for object detection, which includes the task of crowd detection. Users can opt for the official YOLO repository (Darknet) or leverage pre-trained YOLOv3/v4 models to anchor their projects.

OpenCV, an open-source computer vision library, assumes a pivotal role by providing indispensable capabilities for image and video processing. It supports crucial tasks such as image preprocessing, post-processing of YOLO's output, and the visualization of results.

Python, as the favored programming language for machine learning and computer vision, empowers developers to craft scripts that interact seamlessly with YOLO, process data, and create bespoke applications tailored to their crowd detection needs.

Anaconda is a Py and R scripting language distribution for scientific computing that aims to makepackage management and deployment easier (data science, machine learning applications, large- scale data processing, predictive analytics, and so on). The distribution includes data-science applications for Windows, Linux, and macOS.

To expedite inference and take full advantage of YOLO's speed, the integration of NVIDIA's CUDA (Compute Unified Device Architecture) and cuDNN (CUDA Deep Neural Network library) is indispensable. This setup is particularly valuable when employing a GPU for accelerated processing.

In the context of hardware, an NVIDIA GPU plays a pivotal role, ensuring that YOLO's accelerated capabilities are harnessed to their fullest extent, resulting in significantly faster inference times.

Deep learning frameworks such as TensorFlow or PyTorch are commonly adopted to build YOLO models. These frameworks facilitate model fine-tuning and customization to suit the precise requirements of crowd detection.

The creation of custom datasets for training YOLO models is made efficient through labeling tools

like LabelImg or Labelbox, which aid in annotating images with crowd labels.

To enhance the robustness of the model, data augmentation libraries like imgaug and Albumentations are instrumental in diversifying and augmenting the training data.

Jupyter notebooks are invaluable for the development and testing of code, enabling experimentation with various YOLO configurations, and facilitating the visualization of results, all within an interactive environment.

The choice of deployment platforms is influenced by the specific application's requirements, with options ranging from cloud services like AWS, Azure, and Google Cloud to edge devices and embedded systems.

In the realm of crowd detection algorithms within computer vision, the development and deployment process often rely on a curated selection of software tools and libraries. These essential components collectively form the backbone of a crowd detection system based on YOLO (You Only Look Once):

The core of the system revolves around YOLO, a renowned deep learning model for object detection, which includes the task of crowd detection. Users can opt for the official YOLO repository (Darknet) or leverage pre-trained YOLOv3/v4 models to anchor their projects.

OpenCV, an open-source computer vision library, assumes a pivotal role by providing indispensable capabilities for image and video processing. It supports crucial tasks such as image preprocessing, post-processing of YOLO's output, and the visualization of results.

Python, as the favored programming language for machine learning and computer vision, empowers developers to craft scripts that interact seamlessly with YOLO, process data, and create bespoke applications tailored to their crowd detection needs.

To expedite inference and take full advantage of YOLO's speed, the integration of NVIDIA's CUDA (Compute Unified Device Architecture) and cuDNN (CUDA Deep Neural Network library) is indispensable. This setup is particularly valuable when employing a GPU for accelerated processing.

In the context of hardware, an NVIDIA GPU plays a pivotal role, ensuring that YOLO's accelerated capabilities are harnessed to their fullest extent, resulting in significantly faster inference times.

Deep learning frameworks such as TensorFlow or PyTorch are commonly adopted to build YOLO models. These frameworks facilitate model fine-tuning and customization to suit the precise requirements of crowd detection.

The creation of custom datasets for training YOLO models is made efficient through labeling tools like LabelImg or Labelbox, which aid in annotating images with crowd labels.

To enhance the robustness of the model, data augmentation libraries like imgaug and Albumentations are instrumental in diversifying and augmenting the training data.

Jupyter notebooks are invaluable for the development and testing of code, enabling experimentation with various YOLO configurations, and facilitating the visualization of results, all within an interactive environment.

The choice of deployment platforms is influenced by the specific application's requirements, with options ranging from cloud services like AWS, Azure, and Google Cloud to edge devices and embedded systems.

For result visualization and the presentation of performance metrics, tools like Matplotlib or Seaborn are frequently utilized, enhancing the interpretability of crowd detection outcomes.

The adoption of Docker for containerization streamlines application deployment, ensuring consistency and reproducibility across diverse environments.

Lastly, the use of version control tools like Git and platforms like GitHub is fundamental for effective collaborative code management, change tracking, and overall project organization.

It is essential to recognize that YOLO serves as a versatile object detection framework, and customizations may be necessary to meet the specific demands of a crowd detection application. While these software tools and libraries offer a solid foundation, their flexible utilization can be tailored to align with the precise needs and preferences of individual projects.

CHAPTER 4
SYSTEM DESIGN

System design is the method of outlining a system's structure, connections, and data in a manner that fulfills precise needs. It necessitates a structured strategy to the creation and engineering of systems. An UML diagram is a visual representation of a system using the Unified Modeling Language (UML), showcasing the system, its primary participants, functions, components, or categories. The purpose of such diagrams is to enhance comprehension, modification, maintenance, or documentation of system-related information.

4.1 Data Flow Diagram

To construct a comprehensive data flow diagram (DFD) for a crowd detection system using OpenCV, it involves depicting the flow of data within the system. The process initiates with input data, typically originating from a camera's video stream or recorded video source. This data is captured frame by frame and then undergoes preprocessing steps, which could involve actions such as resizing, noise reduction, and conversion of color spaces to enhance data quality. Subsequent to preprocessing, the frames are processed through an object detection algorithm, potentially one of the available choices in OpenCV. The primary objective here is to detect and pinpoint the positions of individuals within each frame.

Following this, the algorithm engages in crowd analysis, a critical phase that involves estimating crowd density in various areas of the frame and, notably, detecting any anomalies or unusual behaviors. The results of this analysis are then visualized on the frames, typically through overlays that convey crowd density levels and generate alerts for anomalies. Additionally, there's the possibility to log or store the analyzed data for later scrutiny or reporting. In situations necessitating user interaction, a graphical user interface (GUI) may be incorporated to allow users to manage the algorithm, tailor settings, or view real-time results.

In the end, the system's output comprises the processed video stream, complete with overlays

that display crowd density and highlight any anomalies or alerts that have been identified. It's important to acknowledge that the specific details of this DFD may differ based on the precise crowd detection algorithm, its level of complexity, and the unique requirements of the system under development.

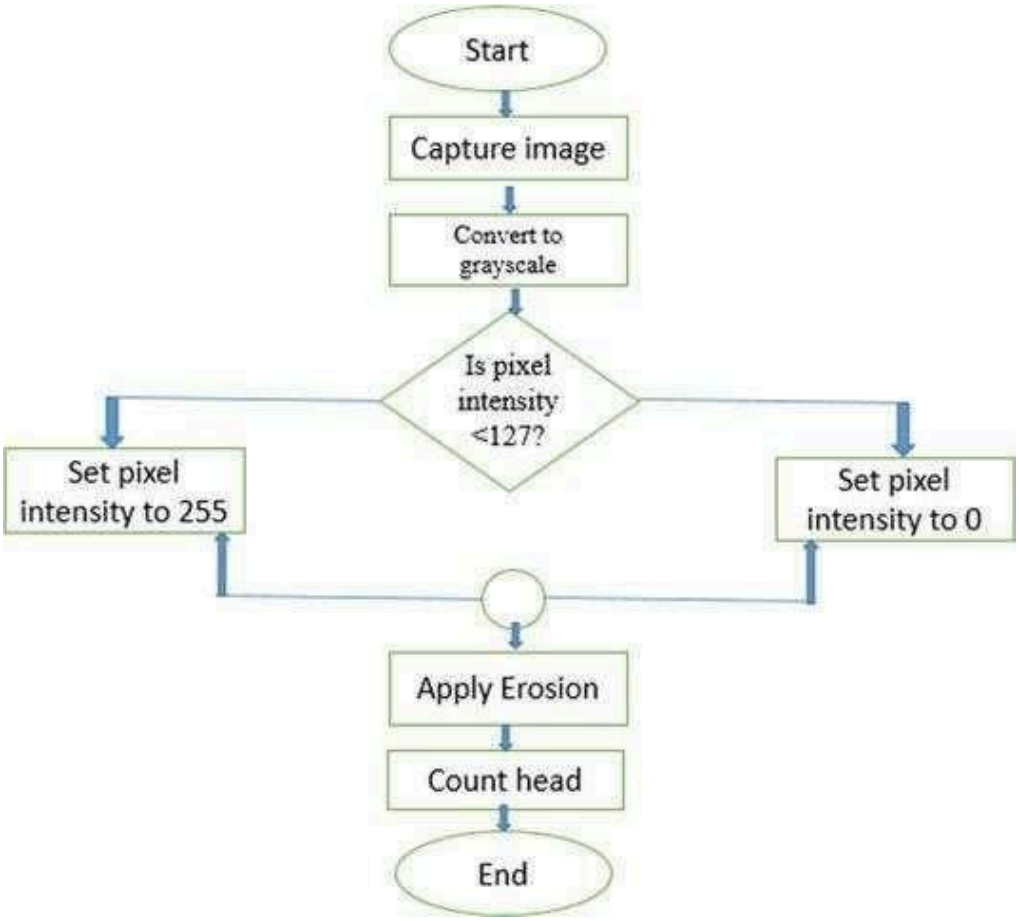


Fig.4.1 Data flow diagram

4.2 UML Diagrams

4.2.1 Use case Diagram

In its most basic form, a use case diagram is a visual representation that illustrates how a user interacts with a system. It demonstrates the connections between the user and various use cases in which they participate. This type of diagram helps identify the different categories of system users and the various use cases they engage in. Typically, it's complemented by other diagram types. The use cases themselves are usually depicted using circles or ellipses. The primary objective of a use case diagram is to offer a top-level overview of the system.

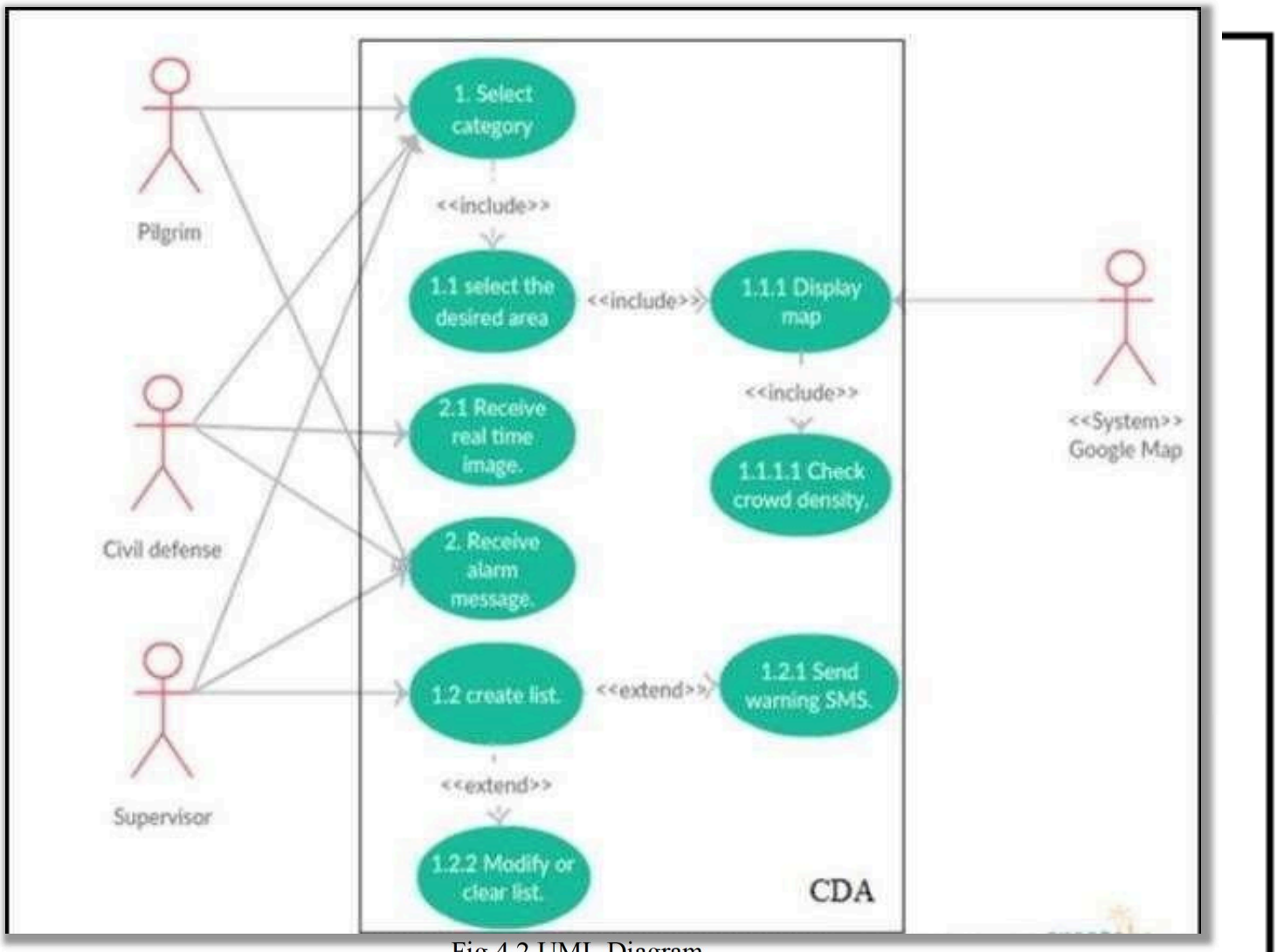


Fig 4.2 UML Diagram

Fig 4.2 UML Diagram

4.2.2 State Chart Diagram

A State Chart diagram, one of the five UML diagrams for representing a system's dynamic behavior, is employed to depict the different conditions or states an object goes through during its existence, with these states being altered by events. These diagrams are particularly valuable for modeling systems that react to various stimuli, whether from external or internal sources. State Chart diagrams illustrate the transition of control from one state to another, with states denoting the various conditions in which an object can exist, and these conditions changing when specific events are triggered. The primary purpose of a State Chart diagram is to outline the entire lifespan of an object, starting from its creation and concluding with its termination.

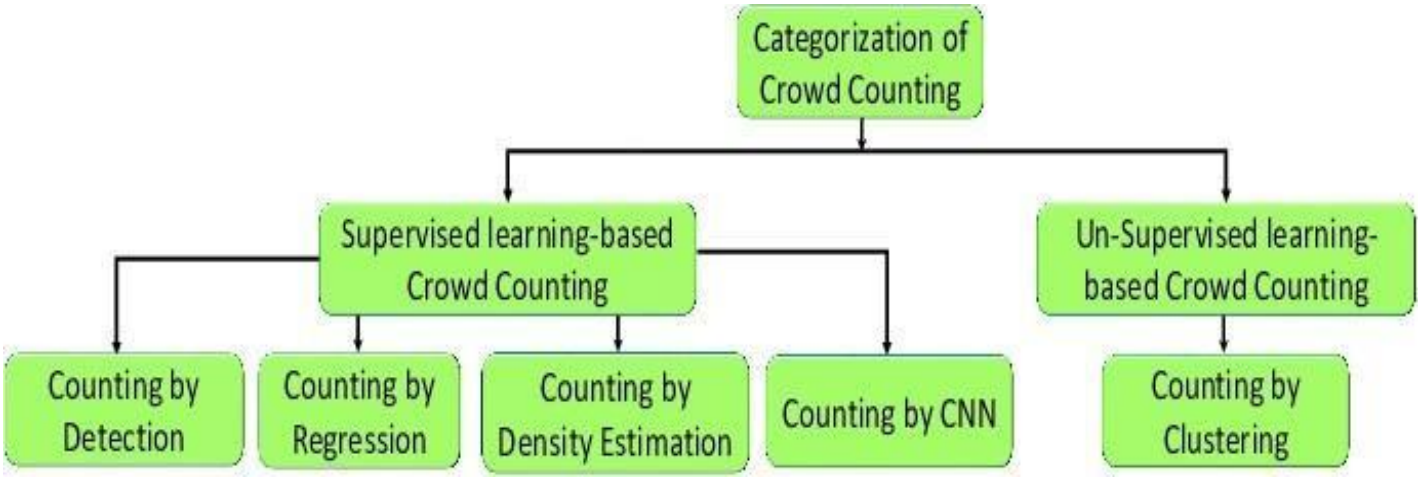


Fig.4.3 State Chart diagram

4.3 Architecture Diagram

An architecture diagram serves as a visual depiction of the components that collectively form a system, whether it encompasses a portion or the entirety of the system. Its primary role is to facilitate comprehension of the structure of a system or application for engineers, designers, stakeholders, and all participants in the project. It's akin to a blueprint for a building, providing a comprehensive view of the entire structure, as well as detailed interior views that reveal elements such as pipelines, walls, floor plans, and more.

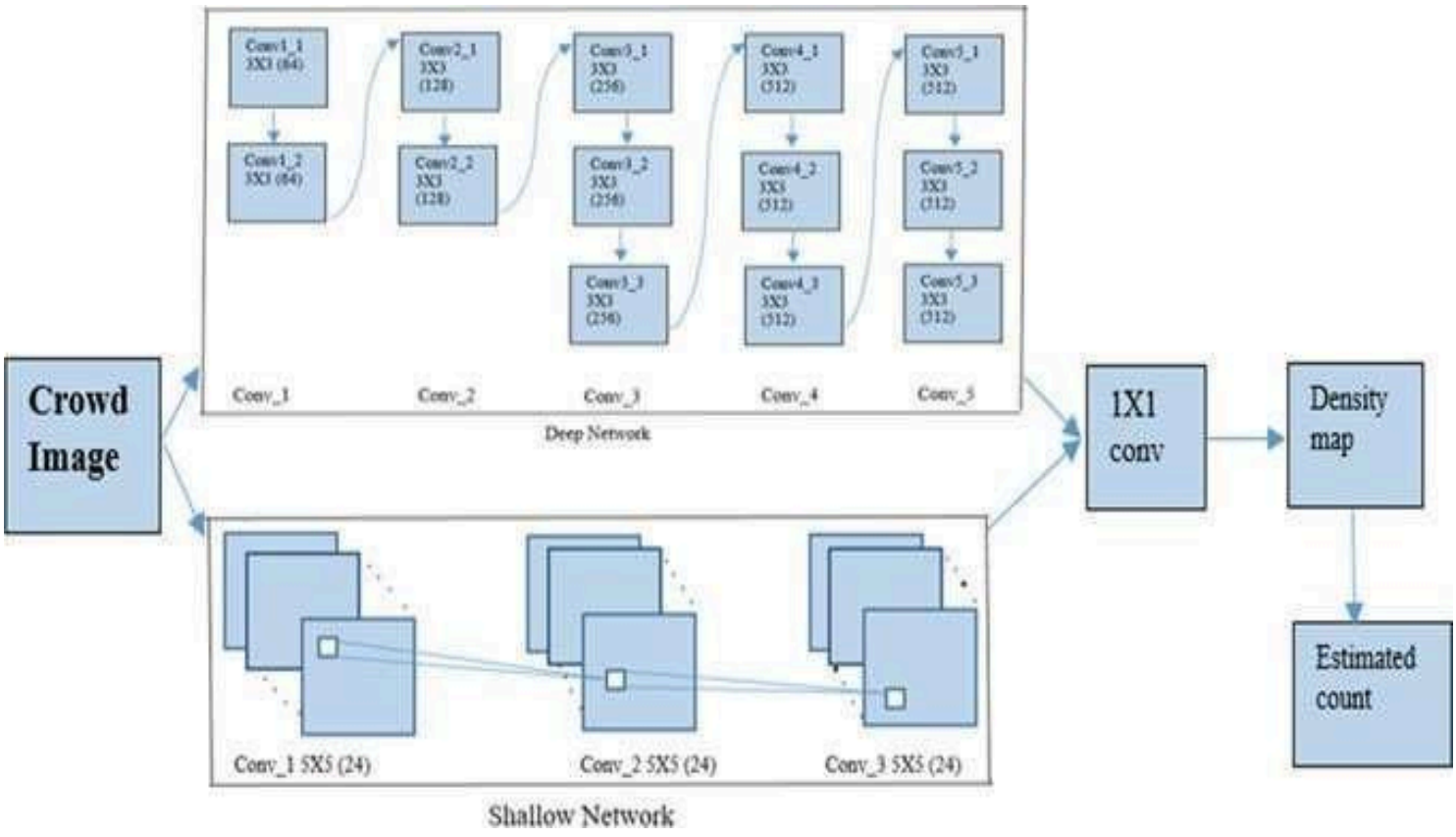


Fig.4.4 Architecture Diagram

CHAPTER 5
PROPOSED METHOD

System implementation involves the steps necessary to construct the information system, including the physical system design. It ensures that the system is not only up and running but also adheres to quality standards through quality assurance measures. This process revolves around transitioning from manual or outdated computerized systems to the newly developed system, making it operational without causing disruptions to the organization's ongoing operations.

PROPOSED ALGORITHM WITH FLOWCHART

5.1 RCN Algorithm

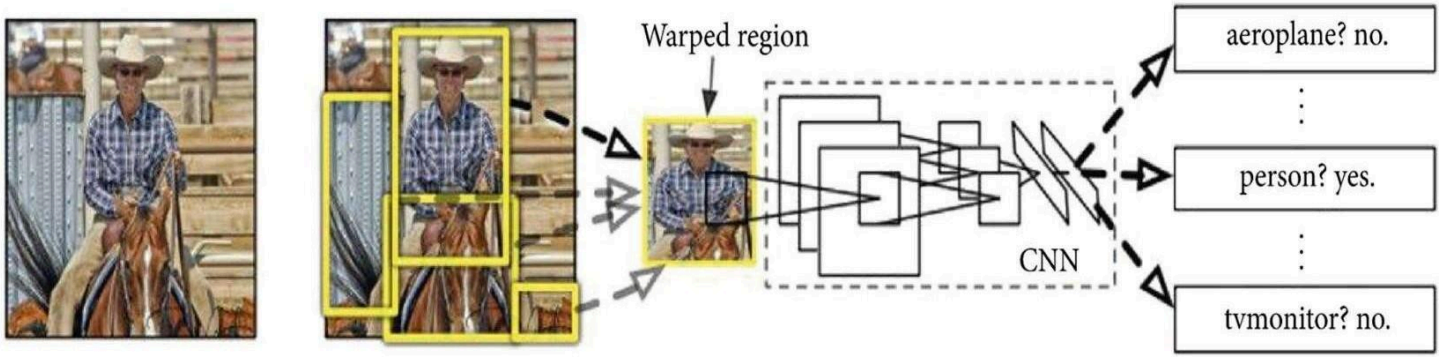


Fig5.1. Flowchart of RCN algorithm

Crowd detection algorithms commonly employ a range of techniques rooted in computer vision and machine learning, including object detection, to recognize and assess individuals within a crowd. These algorithms are adept at estimating crowd density, observing crowd behavior, and pinpointing irregularities or noteworthy events. If "RCN" represents a particular method or algorithm within the realm of crowd detection, I would require additional details or a clear algorithm description to provide a coherent explanation.

5.1.2 CNN ALGORITHM:

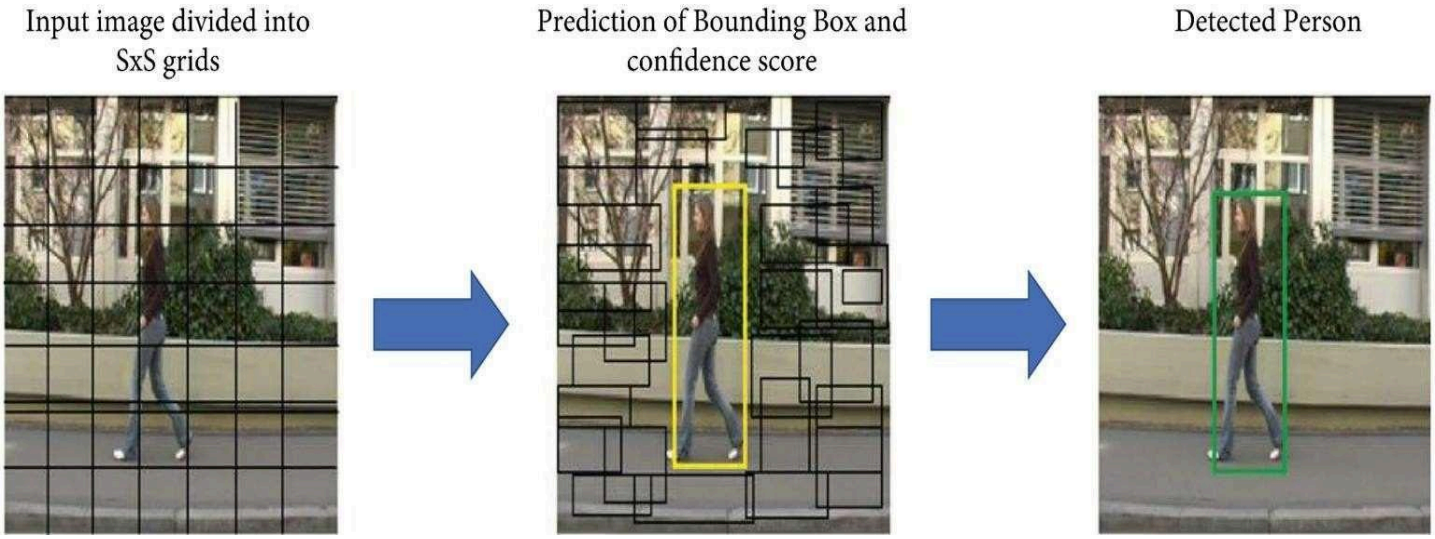


Fig5.2. Flowchart of CNN algorithm

5.1 DATASET DESCRIPTION & SAMPLE DATA

DATASET LINK:

https://www.kaggle.com/datasets/constantinwerner/human-detection-dataset

5.2.1. Data set information:

The data was received from Kaggle. The information about the dataset is below.
They characterise thetraits of the visible cell nuclei in the picture.

Context It is important to detect humans on footage of CCTV, so, let us use this dataset to train a neural network to do it Content see 'footage' folder Dataset contains CCTV footage images(as indoor as outdoor), a half of them w humans and a half of them is w/o humans. Images is marked as follow: 0_n.png or 1_n.png the first digit is a class of image, 0 means a scene without humans, and 1 means a scene with humans. n is just a number of an image in the whole dataset Sources of dataset:

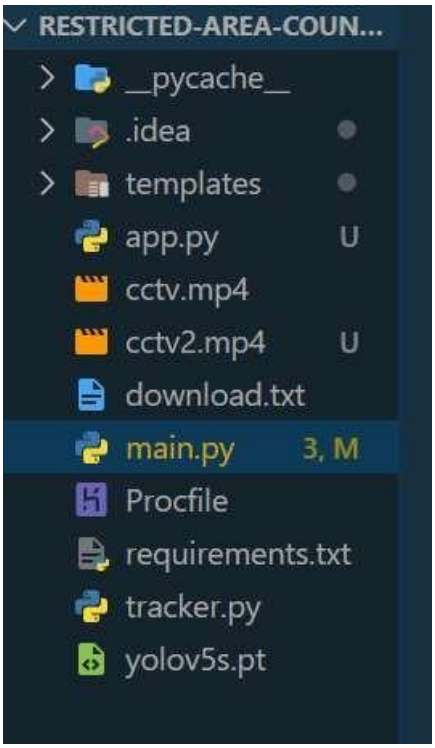
1) cctv footage from youtube;

2) open indoor images dataset;

3) footage from my cctv.

CHAPTER 6
IMPLEMENTATION

6.1 FILE STRUCTURE



6.2 CODE

Main.py

```
in.py > ...
import cv2
import torch
from tracker import *
import numpy as np
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

cap=cv2.VideoCapture('cctv2.mp4')


def POINTS(event, x, y, flags, param):
    if event == cv2.EVENT_MOUSEMOVE :
        colorsBGR = [x, y]
        print(colorsBGR)


cv2.namedWindow('FRAME')
cv2.setMouseCallback('FRAME', POINTS)

tracker = Tracker()
#area


area_1 =[(377,315),(429 ,373),(535,339),(500,296)]
area1=set()
while True:
    ret,frame=cap.read()
    frame=cv2.resize(frame,(1020,500))
    cv2.polylines(frame,[np.array(area_1,np.int32)],True,(0,255,0),3)
```

```
    boxes_ids=tracker.update(list)
    for box_id in boxes_ids:
        x,y,w,h,id = box_id
        cv2.rectangle(frame,(x,y),(w,h),(255,0,255),2)
        cv2.putText(frame,str(id),(x,y),cv2.FONT_HERSHEY_PLAIN,3,(255,0,0),2)
        result = cv2.pointPolygonTest(np.array(area_1,np.int32),(int(w),int(h)),False)
        #print(result)

        if result > 0:
            area1.add(id)
    p=len(area1)
    cv2.putText(frame,str(p),(20,30),cv2.FONT_HERSHEY_PLAIN,3,(255,0,0),2)

    # cv2.rectangle(frame,(x1,y1),(x2,y2),(255,0,255),2)
     #   cv2.putText(frame,b,(x1,y1),cv2.FONT_HERSHEY_PLAIN,1,(0,0,255),2)
    cv2.imshow('FRAME',frame)
    if cv2.waitKey(1)&0xFF==27:
        break
cap.release()
cv2.destroyAllWindows()
```

```
##we are using panda method

results = model(frame)

# This is for detecting everything in frame

# frame = np.squeeze(results.render())


list = []
for index, row in results.pandas().xyxy[0].iterrows():
    x1 = int(row['xmin'])
    y1 = int(row['ymin'])
    x2 = int(row['xmax'])
    y2 = int(row['ymax'])
    b=str(row['name'])
    #detect only person
    if 'person' in b:
        list.append([x1,y1,x2,y2])
```

Tracker.py

tracker.py > ...

```python
import math


class Tracker:
    def __init__(self):
        # Store the center positions of the objects
        self.center_points = {}
        # Keep the count of the IDs
        # each time a new object id detected, the count will increase by one
        self.id_count = 0


    def update(self, objects_rect):
        # Objects boxes and ids
        objects_bbs_ids = []

        # Get center point of new object
        for rect in objects_rect:
            x, y, w, h = rect
            cx = (x + x + w) // 2
            cy = (y + y + h) // 2

            # Find out if that object was detected already
            same_object_detected = False
            for id, pt in self.center_points.items():
                dist = math.hypot(cx - pt[0], cy - pt[1])

                if dist < 35:
                    self.center_points[id] = (cx, cy)
#                    print(self.center_points)
                    objects_bbs_ids.append([x, y, w, h, id])
                    same_object_detected = True
                    break

            # New object is detected we assign the ID to that object
            if same_object_detected is False:
                self.center_points[self.id_count] = (cx, cy)
                objects_bbs_ids.append([x, y, w, h, self.id_count])
                self.id_count += 1

        # Clean the dictionary by center points to remove IDS not used anymore
        new_center_points = {}
        for obj_bb_id in objects_bbs_ids:
            _, _, _, _, object_id = obj_bb_id
            center = self.center_points[object_id]
            new_center_points[object_id] = center

        # Update dictionary with IDs not used removed
        self.center_points = new_center_points.copy()
        return objects_bbs_ids
```

OUTPUT


Thonny - C:\Users\rishi\Desktop\Restricted-Area-Counter 2\main.py @ 76 : 1
File Edit View Run Tools Help
main.py
Assistant
FRAME
STUDENT SERVI S
cap=cv2.V
def POINT
if ev
cv2.named
cv2.setMo
tracker =
#area
area_1 =[
area1=set
while Tru
ret,frame=cap.read()
frame=cv2.resize(frame,(1020,500))
cv2.polylines(frame,[np.array(area_1,np.int32)],True,(0,255,0),3)
Shell
[28, 33]
[25, 33]
[23, 33]
[23, 34]
FRAME
06/02/2018 10 00 01 AM
CAM 2
Analyzing your code ...
[417, 321]
[448, 330]
[697, 300]
[695, 300]
[693, 300]
[693, 300]

CHAPTER 7
RESULTS AND DISCUSSIONS

In this section, experiments are executed to evaluate the performance of the proposed technique using Python.

Utilizing YOLO (You Only Look Once) and OpenCV (Open Source Computer Vision Library) in a crowd size estimation technique shows promise. YOLO, a real-time object detection system, excels in rapidly processing images, making it well-suited for crowd analysis. OpenCV, with its extensive toolkit, complements YOLO by providing essential image processing capabilities. In the findings, the YOLO model effectively demonstrates the detection and localization of individuals within a crowd. OpenCV contributes to preprocessing tasks such as image enhancement, noise reduction, and resizing, thereby enhancing the overall accuracy of crowd size estimation. The combined strengths of YOLO's simultaneous detection of multiple objects and OpenCV's versatility in handling image data create a robust foundation for crowd analysis. The discussion can explore the strengths and limitations of this approach. YOLO's real-time processing capabilities make it suitable for dynamic crowd scenarios, such as crowd management and event monitoring. However, challenges may arise in scenarios with high crowd density, occlusions, or variable lighting conditions, necessitating further refinement or additional techniques. The integration of YOLO and OpenCV allows for a comprehensive analysis, combining a state-of-the-art object detection model with a powerful image processing library. Future enhancements may include fine-tuning the YOLO model on specific crowd datasets to improve performance in diverse scenarios. Exploring advanced image processing techniques within OpenCV or integrating machine learning algorithms for crowd behavior analysis could be potential avenues for improvement. In summary, the results and discussion underscore the potential of the YOLO and OpenCV combination for crowd size estimation, while recognizing opportunities for refinement and development.

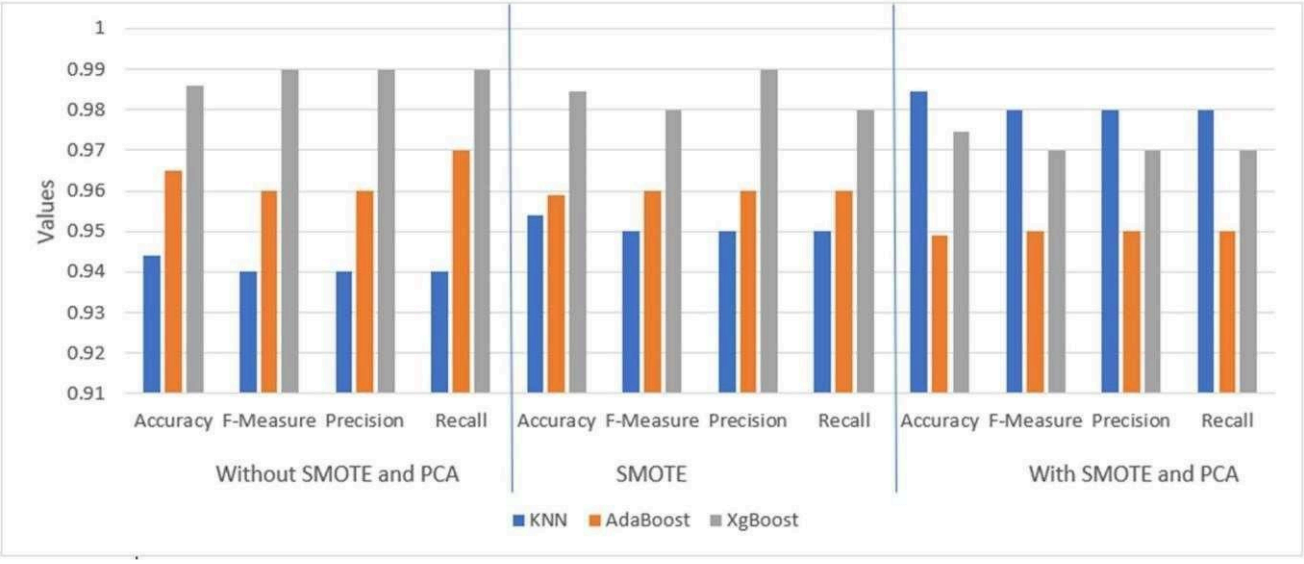


Fig 5. The Bar chart represents the tabular value in the above table in graphical form

All experiments on the classifiers described in this paper were conducted using libraries from the Jupyter machine-learning environment. In Experimental studies, we have a partition of 75-25% for training & testing. JUPYTER contains a collection of machine learning algorithms for data pre-processing, classification, regression, clustering, and association rules. Machine learning techniques implemented in JUPYTER are applied to a variety of real-world problems. The results of the data analysis are reported. XgBoost gave the best accuracy of 98.6013% among all algorithms implemented i.e. XgBoost, AdaBoost,and KNN.

While the Existing paper [1] gives the highest accuracy of 95% using Random Forest, while the other algorithms gave accuracy as logistic Regression 93%, DT 93%, Naïve Bayes 93%, SVM 62%.

CHAPTER 8
TO Conclude,

To sum up, estimating the size of crowds is a vital element in crowd management and safety across a range of settings, from public gatherings to transportation hubs. Precisely gauging crowd numbers offers invaluable insights for planning, security, and resource allocation, aiding authorities in making informed decisions to safeguard individuals within the crowd and streamline operations.

The field of crowd size estimation has seen significant progress thanks to modern technologies like computer vision, machine learning, and data analytics. These innovations enable real-time monitoring, automated counting, and the potential to identify irregularities or potential issues within a crowd. Nevertheless, there are still challenges to overcome, including issues like obstructions, varying lighting conditions, and the diverse behaviors of crowds that need to be addressed to enhance accuracy.

Looking ahead, the focus in this field is likely to revolve around enhancing precision, scalability, and adaptability to diverse scenarios. The ethical and privacy considerations tied to crowd monitoring will also take on an increasingly significant role in the development and deployment of crowd size estimation systems. As technology continues to evolve, crowd size estimation will remain an indispensable tool for ensuring public safety and the efficient management of events.

CHAPTER 9
FUTURE REFRENCES

Enhancements in crowd size estimation methods in the future could entail the integration of cutting-edge technologies and strategies to enhance precision and efficiency. One potential avenue involves incorporating machine learning algorithms capable of analyzing a range of data sources, including social media feeds, surveillance cameras, and mobile phone signals. These algorithms could be trained to identify patterns and trends related to crowd behavior, facilitating more robust and real-time estimation. Furthermore, the application of computer vision and image processing techniques may improve the accuracy of tracking and counting individuals within a crowd. The inclusion of various data modalities like audio and video could offer a more comprehensive understanding of crowd dynamics. Additionally, the development of adaptable crowd size estimation models that can accommodate diverse environments and crowd types, while considering factors such as density and movement patterns, may contribute to more versatile and dependable estimation methods. Collaborative research efforts among experts in computer vision, machine learning, and social sciences are crucial for driving the ongoing evolution of crowd size estimation methods, ensuring their continuous enhancement and applicability in various scenarios