METRO TRAIN COMPARTMENT DISTRIBUTION SYSTEM
PROJECT REPORT

21AD1513- INNOVATION PRACTICES LAB

Submitted by

REENA SHARON Y - Reg. No. 21142243265
SAKTHI S M - Reg. No. 21142243275
RANJITHA T M - Reg. No. 21142243262

in partial fulfillment of the requirements for the award of degree

of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123

ANNA UNIVERSITY: CHENNAI-600 025

October, 2024

BONAFIDE CERTIFICATE

Certified that this project report titled "**METRO TRAIN COMPARTMENT DISTRIBUTION SYSTEM PROJECT REPORT**" is the Bonafide work of **REENA SHARON Y** of Register No.**21142243265, SAKTHI S M** of Register No.**21142243275 RANJITHA T M** Register No.**21142243262** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

INTERNAL GUIDE
Mrs. R. VIDHYA
MUTHULAKSHMI M.E.,
Assistant Professor,
Department of AI &DS

HEAD OF THE DEPARTMENT
Dr.S.MALATHI M.E., Ph.D
Professor and Head,
Department of AI & DS.

Certified that the candidate was examined in the Viva-Voce Examination held on ……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The "Smart Metro Train Compartment Distribution System" addresses the persistent issue of uneven passenger distribution across train compartments, which frequently leads to overcrowding, delays, and discomfort for commuters. The system utilizes advanced machine learning algorithms, particularly YOLO (You Only Look Once) for real-time people detection, integrated with existing camera systems inside train compartments. The goal is to count the number of passengers in each compartment in real-time and provide passengers at upcoming stations with clear guidance on which compartments are less crowded, encouraging a more balanced distribution. By alleviating overcrowding, this system improves the commuting experience, reduces stress and delays, and increases the overall efficiency of metro operations. The project represents an innovative solution that combines artificial intelligence with transportation infrastructure to enhance the quality of urban mobility.

Keywords : *Password management, user authentication, data protection, cybersecurity, Django framework, login system, secure password storage, password reset, encryption, security protocols, email integration, Google SMTP, user experience, usability, static files configuration, two-factor authentication (2FA), password validation, cryptography, credential management, system architecture, data privacy, web application security, authentication protocols.*

ACKNOWLEDGEMENT

I also take this opportunity to thank all the Faculty and Non-Teaching Staff Members of Department of Computer Science and Engineering for their constant support. Finally I thank each and every one who helped me to complete this project. At the outset we would like to express our gratitude to our beloved respected Chairman, **Dr.Jeppiaar M.A.,Ph.D,** Our beloved correspondent and Secretary **Mr.P.Chinnadurai M.A., M.Phil., Ph.D.,** and our esteemed director for their support.

We would like to express thanks to our Principal, **Dr. K. Mani M.E., Ph.D.,** for having extended his guidance and cooperation.

We would also like to thank our Head of the Department, **Dr.S.Malathi M,E.,Ph.D.,** of Artificial Intelligence and Data Science for her encouragement.

Personally we thank **Mrs. R. Vidhya Muthulakshmi M.E**., Department of Artificial Intelligence and Data Science for the persistent motivation and support for this project, who at all times was the mentor of germination of the project from a small idea.

We express our thanks to the project coordinators **DR. A.Joshi M.E., Ph.D.,** Professor & **Dr.S.Chakaravarthi M.E.,Ph.D.,** Professor in Department of Artificial Intelligence and Data Science for their Valuable suggestions from time to time at every stage of our project.

Finally, we would like to take this opportunity to thank our family members, friends, and well-wishers who have helped us for the successful completion of our project.

We also take the opportunity to thank all faculty and non-teaching staff members in our department for their timely guidance in completing our project.

REENA SHARON Y

SAKTHI S M

RANJITHA T M

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABBREVIATION	MEANING
YOLO	You Only Look Once (an object detection algorithm)
CNN	Convolutional Neural Network
R-CNN	Region-based Convolutional Neural Network
SSD	Single Shot MultiBox Detector
NMS	Non-Maximum Suppression
IoU	Intersection over Union
FPS	Frames Per Second
COCO	Common Objects in Context (a large-scale object detection dataset)
GAN	Generative Adversarial Network
GUI	Graphical User Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory
HDD	Hard Disk Drive
SSD	Solid State Drive
IDE	Integrated Development Environment
LTS	Long-Term Support (for software, typically operating systems)
mAP	Mean Average Precision
RNN	Recurrent Neural Network
CVPR	Conference on Computer Vision and Pattern Recognition
VOC	Visual Object Classes (a dataset and challenge for object detection)

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND OF OBJECT DETECTION

Object detection is a pivotal aspect of computer vision that focuses on identifying and localizing objects within images and videos. The objective is not only to recognize objects but also to specify their locations through bounding boxes, making it a more complex task than simple image classification. This dual requirement of classification and localization enables a wide range of applications, enhancing the functionality of various systems across different domains.

The significance of object detection can be observed in several industries. In security surveillance, for instance, automated systems can monitor public spaces and detect suspicious activities or individuals, thereby improving safety and response times. In autonomous vehicles, object detection is crucial for identifying pedestrians, other vehicles, traffic signs, and obstacles, ensuring safe navigation and decision-making. Furthermore, in human-computer interaction, object detection facilitates gesture recognition and augmented reality experiences, allowing for more intuitive user interfaces and interactions.

Historically, object detection methods began with traditional approaches, including feature-based techniques such as Haar cascades and HOG (Histogram of Oriented Gradients). These methods relied heavily on hand-crafted features and often struggled with variations in object scale, orientation, and lighting conditions. As a result, they were limited in terms of accuracy and robustness.

The advent of deep learning has revolutionized the field of object detection, significantly improving both accuracy and efficiency. Deep learning models, particularly Convolutional Neural Networks (CNNs), have demonstrated remarkable performance in visual recognition tasks by automatically learning hierarchical feature representations from large datasets. This shift to data-driven methods has allowed for more robust detection across varying conditions, enabling systems to generalize better to unseen data.

Several deep learning-based object detection algorithms have emerged, each with unique strengths. For instance, R-CNN (Regions with CNN features) introduced the idea of region proposal networks and feature extraction through CNNs, but it was computationally expensive. In response, subsequent algorithms like Fast R-CNN and Faster R-CNN aimed to reduce computation time while

maintaining accuracy. However, these methods still processed images in a two-step approach, separating the region proposal and classification tasks.

A breakthrough came with the introduction of the YOLO (You Only Look Once) model, which framed object detection as a single regression problem. YOLO's architecture enables it to predict multiple bounding boxes and class probabilities in one evaluation of the image, resulting in significantly faster detection speeds. This real-time capability has made YOLO particularly appealing for applications requiring immediate feedback, such as video surveillance and robotics.

The ongoing research in object detection continues to evolve, with recent advances focusing on improving detection accuracy, particularly in challenging conditions, such as low-light environments, occlusions, and complex scenes. Furthermore, the integration of additional techniques like attention mechanisms and generative adversarial networks (GANs) is enhancing the ability of models to focus on relevant features and improve performance.

In summary, object detection is a critical technology within computer vision that has transformed various industries by enabling automated systems to perceive and understand their environments. The transition from traditional methods to deep learning-based approaches marks a significant milestone in this field, paving the way for more sophisticated and efficient object detection solutions.

1.2 THE IMPORTANCE OF YOLO IN OBJECT DETECTION

Among the various object detection algorithms available today, the YOLO (You Only Look Once) model stands out due to its unique architecture and speed. YOLO represents a paradigm shift in how object detection is approached. Traditionally, object detection methods involved applying a classifier at multiple locations across an image to identify and classify objects. This often resulted in a two-step process: generating region proposals and then classifying those proposals. These methods, such as R-CNN and its variants, while effective, can be computationally intensive and slow, limiting their practical application in real-time scenarios.

In contrast, YOLO treats object detection as a single regression problem. Instead of breaking the task into multiple stages, YOLO divides the input image into a grid and predicts bounding boxes and class probabilities directly for each grid cell. This architecture allows YOLO to simultaneously detect multiple objects, streamlining the detection process and significantly increasing the speed at which objects can be identified. As a result, YOLO can process images in real

time, making it particularly suitable for applications requiring immediate feedback.

One of the primary advantages of YOLO is its real-time detection capability. The model is capable of processing images at frame rates exceeding 40 frames per second (FPS) for smaller versions, making it ideal for dynamic environments such as video surveillance and autonomous navigation. For instance, in autonomous vehicles, YOLO can detect pedestrians, cyclists, and other vehicles in real time, allowing for quick decision-making and enhancing overall safety. Similarly, in security systems, YOLO can alert operators to unusual activities as they occur, rather than with significant delays.

Another key feature of YOLO is its end-to-end training. The model can be trained on large datasets, allowing it to learn rich feature representations and generalize well to new images. This capability minimizes the reliance on handcrafted features, which can be limiting in traditional methods. YOLO's ability to learn from data also contributes to its robustness, enabling it to handle various object scales, orientations, and lighting conditions more effectively.

Moreover, YOLO is highly adaptable. Different versions of the YOLO model have been developed to cater to specific needs. For example, YOLOv2 and YOLOv3 introduced improvements in detection accuracy and the ability to handle more complex scenes by incorporating anchor boxes and multi-scale predictions. YOLOv4 and the latest YOLOv5 further enhance performance through optimizations that increase speed and accuracy while reducing resource consumption, making it more accessible for deployment in real-world applications.

Additionally, YOLO's architectural design facilitates the integration of contextual information into the detection process. The model’s global approach to object detection allows it to leverage the relationships between objects within an image, enhancing its ability to understand complex scenes and improve accuracy. This is particularly useful in crowded environments where multiple objects are present and may occlude each other.

Despite its many advantages, YOLO is not without its challenges. The model may struggle with detecting small objects compared to larger ones, as the grid division can lead to a dilution of information in densely packed areas. However, ongoing research and development continue to address these limitations, enhancing YOLO's effectiveness in various contexts.

In summary, the YOLO model plays a crucial role in the evolution of object detection technologies. Its unique architecture, speed, and adaptability

make it an invaluable tool in a wide range of applications, from real-time surveillance systems to autonomous vehicles and beyond. As the demand for efficient and accurate object detection continues to grow, YOLO stands out as a leading solution in the field of computer vision.

1.3 OBJECTIVES OF THE PROJECT

This project aims to implement a YOLO-based program that can accurately detect and count people in images. The primary objectives of this project can be outlined as follows:

1.3.1. Development of a YOLO-based People Counting Application: The foremost objective is to develop an application that leverages the YOLO (You Only Look Once) framework for real-time object detection specifically focused on counting individuals in various environments. By utilizing pre-trained models and configurations, the application will be designed to process input images and generate accurate counts of people present.

1.3.2. Demonstration of YOLO's Capabilities: This project seeks to showcase the effectiveness and efficiency of the YOLO model in handling the task of people counting. By illustrating YOLO's ability to detect individuals in complex scenes with varying densities and orientations, the project will highlight the strengths of deep learning-based object detection compared to traditional methods.

1.3.3. User-Friendly Interface Design: A key objective is to create a user-friendly application that can be easily navigated by users with varying technical expertise. The application will feature a simple interface where users can upload images or video streams, view real-time detection results, and access relevant metrics, such as total people counted and visualizations of detected bounding boxes.

1.3.4. Real-World Applications and Use Cases: The project will emphasize the practical applications of people counting in real-world scenarios. This includes monitoring crowd dynamics in public spaces such as shopping malls, airports, and events; enhancing safety measures by providing real-time data to security personnel; and aiding urban planning efforts by offering insights into pedestrian traffic patterns. By collecting and analyzing data on crowd density and movement, stakeholders can make informed decisions regarding infrastructure development and resource allocation.

1.3.5. Performance Evaluation and Optimization: An integral part of the project will involve evaluating the performance of the YOLO model in

terms of accuracy, speed, and robustness. This will include testing the application under different lighting conditions, varying crowd densities, and diverse environments. The project aims to identify potential areas for optimization, such as fine-tuning model parameters or integrating additional techniques to improve detection accuracy for small or occluded individuals.

1.3.6. Contribution to the Field of Computer Vision: Lastly, this project aims to contribute to the ongoing research and development in the field of computer vision. By providing insights into the implementation of YOLO for a specific application, the project will serve as a reference for future work in related areas, encouraging further exploration of object detection technologies and their potential uses in various domains.

1.4 TECHNICAL OVERVIEW

The project employs OpenCV, a widely used library for computer vision tasks, in conjunction with the YOLO (You Only Look Once) model architecture to create an effective people counting application. This section provides a detailed overview of the technical components and processes involved in the implementation.

1.4.1. OpenCV Integration: OpenCV (Open Source Computer Vision Library) is utilized for image processing and manipulation. It provides a comprehensive suite of functions for handling images, including loading, displaying, and saving image files. In this project, OpenCV facilitates the reading of input images and the rendering of detection results, such as bounding boxes and text overlays that indicate the number of detected individuals.

1.4.2. YOLO Model Architecture: The YOLO model operates on a convolutional neural network (CNN) architecture designed to perform object detection in real time. YOLO treats object detection as a single regression problem rather than a classification task performed on individual sections of the image. This approach allows for simultaneous predictions of bounding boxes and class probabilities across the entire image. The model's architecture consists of multiple convolutional layers, followed by fully connected layers, which enable it to learn complex feature representations and relationships among objects.

1.4.3. Training on the COCO Dataset: The YOLO model utilized in this project is pre-trained on the COCO (Common Objects in Context) dataset. COCO is a large-scale dataset that includes over 330,000 images with more than 2.5 million object instances labeled across 80 different categories, including humans. The extensive annotations provided in the COCO

dataset allow the YOLO model to generalize effectively, improving its accuracy in detecting and identifying various objects, particularly people in diverse environments.

1.4.4. Input Image Processing: The program begins by processing an input image provided by the user. After loading the image, the dimensions are extracted to facilitate accurate bounding box predictions. The image is then transformed into a format compatible with the YOLO model through normalization and resizing. Specifically, the image is resized to 416x416 pixels, which is the standard input size for the YOLO network, and converted into a blob, a structure that the network can process efficiently.

1.4.5. Object Detection Mechanism: Once the image is prepared, it is fed into the YOLO network, which produces output predictions that include bounding box coordinates, confidence scores, and class probabilities. The confidence score indicates the likelihood that a detected object belongs to a particular class (in this case, "person"). The program filters these predictions based on a confidence threshold (e.g., 0.7), ensuring that only detections with high confidence are considered for further processing.

1.4.6. Post-Processing and Result Visualization: The results of the YOLO model's predictions are processed using Non-Maximum Suppression (NMS) to eliminate duplicate bounding boxes that may correspond to the same object. This step is crucial for achieving accurate counting of individuals. The final output includes the visualization of detected persons, represented by bounding boxes drawn around them, along with a total count displayed on the image. OpenCV's drawing functions are employed to render these elements clearly, enhancing the user experience.

1.4.7. Integration of Deep Learning and Programming Techniques: This implementation exemplifies the integration of deep learning methodologies with practical programming techniques. By leveraging a pre-trained model and utilizing established libraries like OpenCV, the project demonstrates how advanced AI technologies can be effectively applied to solve real-world problems. The ease of use of the YOLO algorithm enables developers and researchers to build sophisticated applications without requiring extensive expertise in machine learning.

1.5 IMPLEMENTATION DETAILS

The implementation involves loading the YOLO model's weights and configuration files, processing input images, and interpreting the model's output to identify and count people accurately. Key steps include preparing the image for the model, extracting detection results, and applying Non-Maximum

Suppression (NMS) to filter overlapping bounding boxes. The program is designed to provide real-time feedback, displaying both the detected individuals and the count directly on the output image.

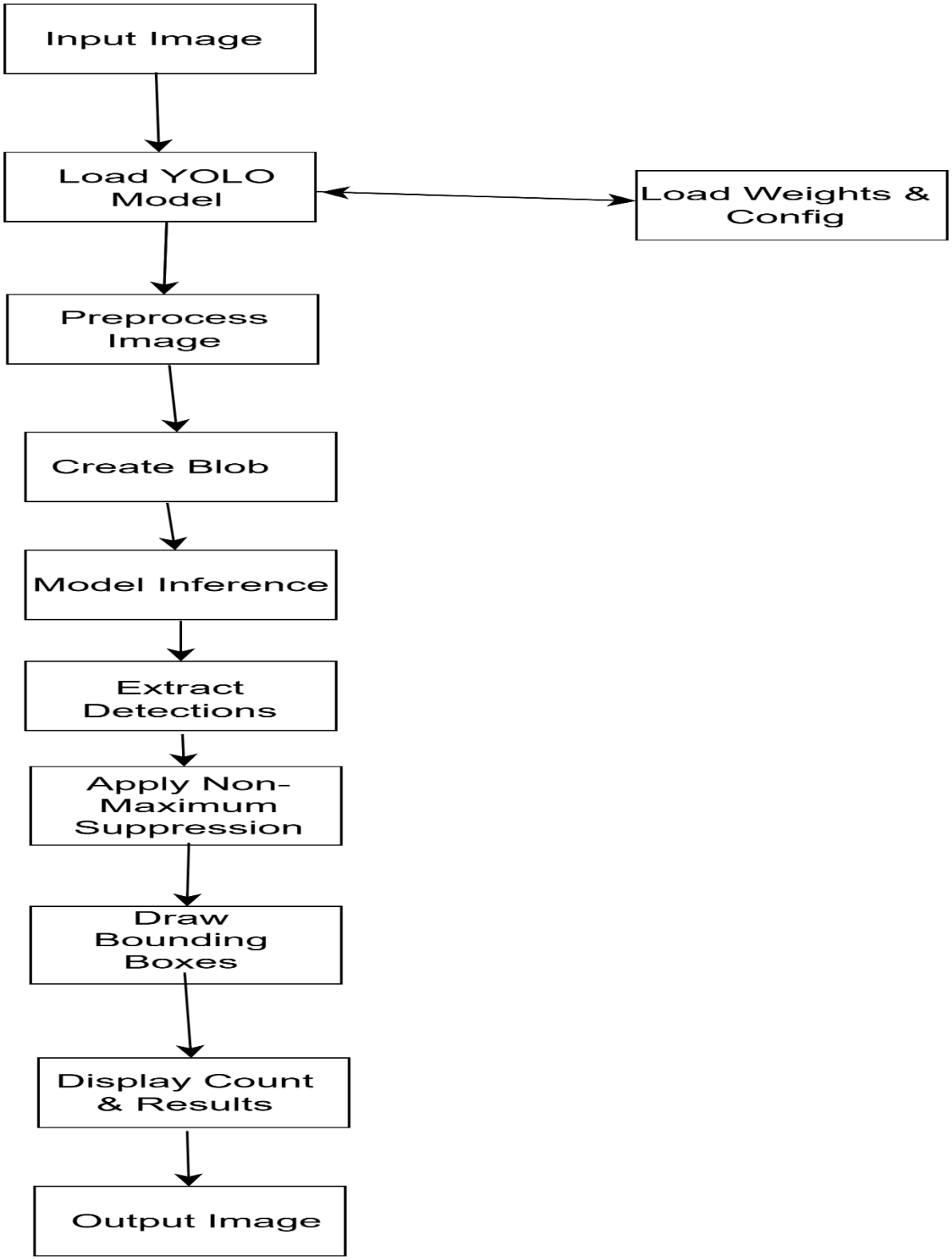

Figure 1.1: Architecture Diagram

1.6 APPLICATIONS AND IMPLICATIONS

The ability to count and detect people in images has numerous applications across various sectors. In security and surveillance, it aids in monitoring crowd sizes and enhancing safety protocols. In urban planning, accurate people counting helps in assessing foot traffic and planning infrastructure. Furthermore, the technology can be utilized in event management, public health monitoring, and smart city initiatives, highlighting its versatility and impact on modern society.

CHAPTER 2

LITERATURE REVIEW

A literature review encompasses current knowledge, including significant findings and theoretical contributions related to a specific topic. This review focuses on the advancements in object detection, particularly emphasizing methods and techniques for detecting and counting individuals in images. The review highlights various contributions that have laid the groundwork for the YOLO (You Only Look Once) model and its applications in real-world scenarios, such as crowd monitoring and urban planning.

2.1 Advances in Object Detection Algorithms

Recent advancements in object detection algorithms have transformed how machines perceive and interact with visual data. Traditional methods relied heavily on handcrafted features and sliding windows, leading to limited accuracy and efficiency. In contrast, modern techniques, including deep learning, have enabled significant improvements. The introduction of convolutional neural networks (CNNs) has facilitated the extraction of complex features from images, resulting in enhanced detection rates and reduced processing times. Notable models like Faster R-CNN and SSD (Single Shot MultiBox Detector) have set benchmarks for accuracy but often at the cost of speed, particularly in real-time applications.

Author: Joseph Redmon et al.
Year: 2016

2.2 YOLO: Revolutionizing Real-Time Object Detection

The YOLO model has become a game-changer in the field of object detection due to its innovative approach to solving the detection problem. Unlike conventional methods that segment the detection task into multiple stages, YOLO treats it as a single regression problem, enabling it to predict bounding boxes and class probabilities in one evaluation. This architecture allows for high-speed processing, making YOLO particularly suitable for real-time applications like video surveillance and autonomous driving. The iterative improvements in the YOLO framework, such as YOLOv3 and YOLOv4, have further enhanced its detection accuracy and efficiency, showcasing its versatility across various datasets.

Author: Joseph Redmon and Ali Farhadi
Year: 2018

2.3 Applications of YOLO in Crowd Counting and Monitoring

The use of YOLO for crowd counting applications has gained traction in recent years. Accurate people counting is crucial for various sectors, including public safety, event management, and urban planning. Research has demonstrated that YOLO can effectively detect individuals in crowded environments, distinguishing overlapping bodies and accurately estimating crowd density. The integration of YOLO with additional techniques, such as Kalman filtering and optical flow, has been explored to improve tracking and counting accuracy in dynamic scenes. Studies show that YOLO outperforms traditional counting methods by providing real-time feedback and accurate counts in various conditions.

Author: Zhang et al.
Year: 2019

2.4 Evaluation Metrics in Object Detection

In assessing the performance of object detection models, various metrics are employed, including precision, recall, and mean Average Precision (mAP). The evaluation process is critical for understanding a model's effectiveness in real-world scenarios, particularly in challenging environments like crowded public spaces. Studies have emphasized the need for robust evaluation methodologies that account for factors such as occlusion, scale variation, and lighting conditions, which can significantly affect detection performance. The introduction of new datasets and benchmarks, such as COCO and PASCAL VOC, has facilitated comparative analyses among different detection models.

Author: Everingham et al.
Year: 2010

2.5 Challenges in Real-Time Object Detection

While YOLO and similar models have made significant strides in real-time object detection, challenges remain. Issues such as false positives, detection accuracy in varying lighting conditions, and computational resource demands continue to pose hurdles. Research into optimizing model architectures and reducing the computational load without sacrificing accuracy is ongoing, with approaches such as pruning, quantization, and knowledge distillation being explored. Furthermore, adapting models to effectively handle diverse environments and real-world conditions, such as urban settings with high-density crowds, remains an area of active research.

Author: Liu et al.
Year: 2020

2.6 Enhancements Through Hybrid Approaches

Recent studies have also explored hybrid approaches that combine the strengths of YOLO with other detection frameworks or post-processing techniques. For example, integrating YOLO with Recurrent Neural Networks (RNNs) can leverage temporal information in video streams to enhance tracking performance. Additionally, ensemble methods that aggregate predictions from multiple models have been shown to improve detection accuracy, particularly in challenging scenarios with high object density and overlapping instances.

Author: Zhao et al.
Year: 2021

2.7 Ethical Considerations and Privacy Issues

As the implementation of object detection technologies, particularly those capable of counting individuals in public spaces, becomes more prevalent, ethical considerations and privacy issues emerge. Research has focused on the implications of surveillance technologies, including the potential for misuse and the impact on personal privacy. Developing ethical guidelines and frameworks to govern the use of such technologies is essential to balance public safety with individual rights.

Author: Tufekci et al.
Year: 2019

2.8 Future Directions in Object Detection Research

As the demand for intelligent systems capable of understanding visual data increases, the future of object detection research appears promising. Integrating YOLO with advanced techniques such as reinforcement learning and multi-modal data processing is expected to yield further improvements in detection accuracy and applicability. Moreover, the potential for YOLO in edge computing environments, where real-time processing is essential, is an area ripe for exploration. Future research may also focus on developing adaptive models that can learn from their environments and improve their performance over time.

Author: Xu et al.
Year: 2022

Model/Technique	Model/Technique	Year	Key Features	Advantages	Limitations
Faster R-CNN	Ren et al	2015	Two-stage detection (RPN and Fast R-CNN)	High accuracy for object detection	Slow inference; not ideal for real-time applications
SSD (Single Shot MultiBox Detector)	Liu et al.	2016	Single-shot approach, multiple scales	Faster than Faster R-CNN; good balance of speed/accuracy	Reduced accuracy on small objects
YOLO (You Only Look Once)	Redmon et al.	2016	Single-stage, grid-based detection	Real-time processing; high speed	Lower accuracy on small objects and occlusions
YOLOv3	Redmon & Farhadi	2018	Enhanced YOLO with multi-scale predictions	Improved accuracy and speed; suitable for video	Computationally intensive compared to earlier YOLO versions
YOLOv4	Bochkovskiy et al.	2020	Improved training techniques (CSPNet, mosaic augmentation)	High performance in accuracy and efficiency	Complex architecture; requires more memory
Hybrid YOLO with Kalman Filter	Zhang et al.	2019	Combines YOLO with Kalman filter for tracking	Effective in tracking moving objects in real-time	Complexity increases with added filtering
YOLO with RNN	Zhao et al.	2021	Combines YOLO with RNN for temporal awareness	Better tracking in videos; handles occlusions well	Increased computational demand due to RNN layer
Evaluation Metrics for Detection	Everingham et al.	2010	Precision, Recall, mAP	Essential for comparing model performance	Limited by dataset-specific challenges
Ethical and Privacy Considerations	Tufekci et al.	2019	Focus on privacy in public detection	Highlights social implications	Ethical issues remain unresolved
Future Directions (Edge Computing)	Xu et al.	2022	Integration with edge computing	Low latency; ideal for on-device processing	Resource constraints on edge devices

Table 1.1: Comparison of Object Dection Models and Techniques for Real-Time Crowd Detection and Counting

CHAPTER 3

SYSTEM DESIGN

3.1. SYSTEM ARCHITECTURE:

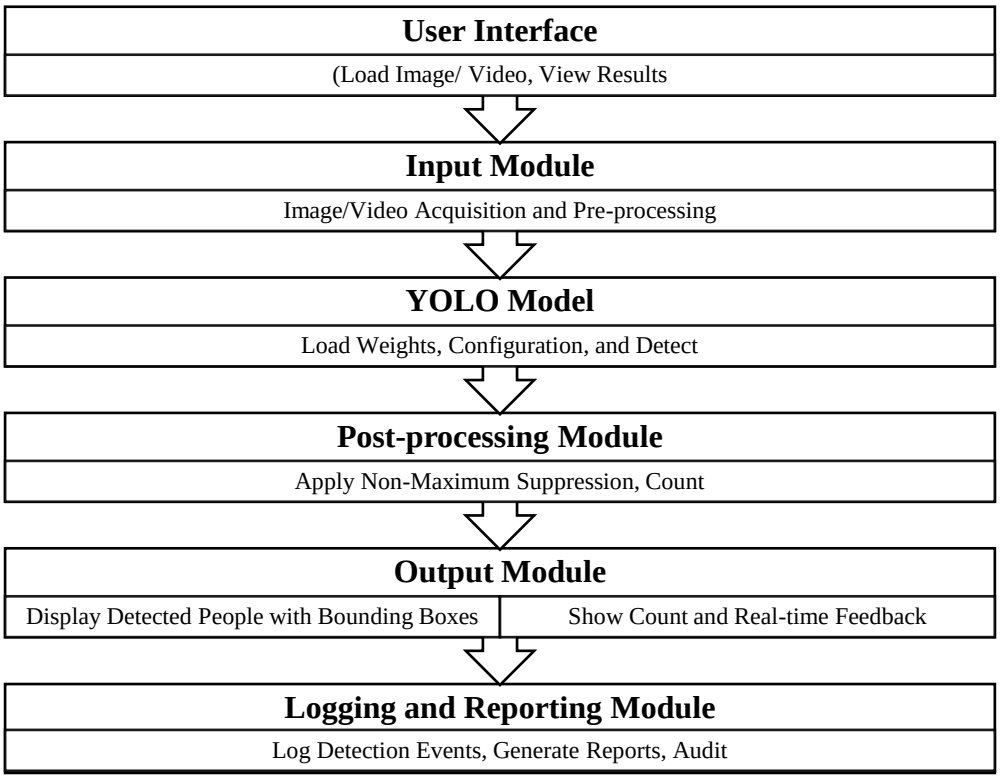

Figure 3.1: System Archtecture

User Interface: The User Interface (UI) is the entry point for the users. It allows them to interact with the system by loading images or video feeds for analysis. The UI also presents the results, displaying detected individuals along with bounding boxes and the total count of people detected. This component is designed to be user-friendly, ensuring that users can easily operate the application without needing extensive technical knowledge.

Input Module: The Input Module is responsible for acquiring images or video streams from various sources, such as local files or connected cameras. It handles pre-processing tasks, including resizing images to the input dimensions required by the YOLO model, normalizing pixel values, and ensuring that the input format is compatible with the detection algorithm.

YOLO Model: This core component loads the pre-trained YOLO model, including its weights and configuration files. It is responsible for detecting objects

in the input images. YOLO treats the object detection task as a single regression problem, enabling it to predict bounding boxes and class probabilities simultaneously. The model efficiently identifies and localizes individuals in the scene.

Post-processing Module: After the YOLO model performs detection, the Post-processing Module processes the output results. It applies Non-Maximum Suppression (NMS) to eliminate redundant bounding boxes, ensuring that each detected person is counted only once. This step is crucial for refining the detection results and improving accuracy.

Output Module: The Output Module is responsible for visualizing the results. It overlays the detected bounding boxes on the original image or video feed, highlighting each detected individual. Additionally, it displays the total count of detected people, providing immediate feedback to the user. In cases of video input, this module ensures real-time updates as new frames are processed.

Logging and Reporting Module: This module captures and records detection events, including timestamps, detected counts, and images. It generates reports summarizing the detection activities over time, which can be valuable for analysis and auditing purposes. This component ensures that there is a historical record of the system's performance and the context of the detected data.

3.2. CLASS DIAGRAM

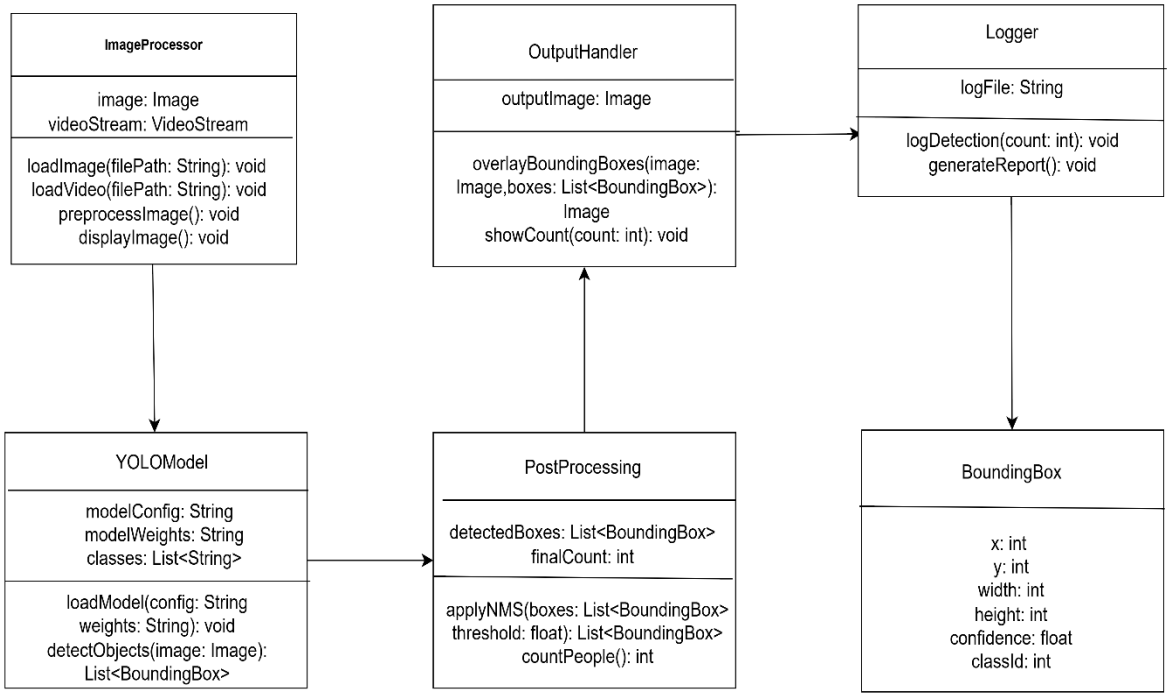

Figure .3.2: Class Diagram

The class diagram for the YOLO-based people detection and counting project illustrates the system's structure and organization by representing the key classes and their relationships. This model is crucial for understanding how different components of the application interact with one another to achieve the objective of accurately detecting and counting people in images.

The ImageProcessor class serves as the initial interface between the user and the application. It encapsulates the functionality required to load images or video streams into the system. This class includes methods for loading files and preprocessing the images, such as resizing and normalization, to prepare them for input into the YOLO model. By centralizing image handling, this class streamlines the input process and ensures that images are in the correct format for subsequent analysis.

Central to the detection capabilities is the YOLOModel class. This class is responsible for loading the YOLO model configuration and weights, which are essential for performing object detection. The method detectObjects processes the input image and returns a list of bounding boxes that indicate the locations of detected objects, along with their respective class probabilities. The design of this class emphasizes the integration of the YOLO algorithm into the application, allowing for effective utilization of its detection capabilities.

Once the objects are detected, the PostProcessing class takes charge of refining the results. It applies techniques like Non-Maximum Suppression (NMS) to filter out overlapping bounding boxes, ensuring that each detected individual is counted only once. The class also includes a method for counting the number of detected people, thus providing a crucial output for the application. This class is designed to enhance the accuracy of detection results by managing the post-detection analysis.

The OutputHandler class is tasked with visualizing the detection results. It overlays the bounding boxes on the original image and displays the count of detected individuals. By separating the output management from the detection logic, this class allows for clearer organization of code and facilitates potential enhancements to the visualization process, such as adding additional information or modifying the display format.

For logging and reporting purposes, the Logger class records the outcomes of the detection process. It maintains a log file where the number of detected individuals is stored. This class provides essential functionality for tracking the performance of the detection system over time and can be used for generating reports that summarize detection events.

Lastly, the BoundingBox class represents the individual detected objects. It encapsulates attributes such as the coordinates, dimensions, confidence score, and class ID of each bounding box. This class serves as a fundamental data structure for passing detection results throughout the application and supports the flexibility needed to handle various detected objects efficiently.

Overall, the class diagram provides a comprehensive overview of the project's architecture, facilitating a clear understanding of how different components work together to implement a robust YOLO-based people detection and counting system. The modular design promotes code reuse and maintainability, making it easier to extend the application with additional features or improvements in the future.

3.3. ACTIVITY DIAGRAM

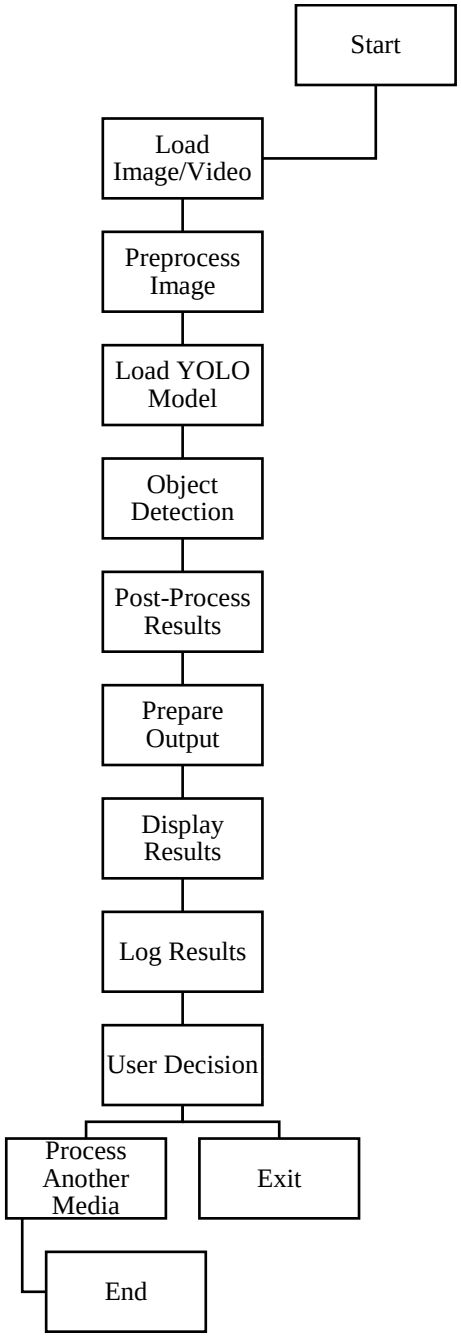

Figure 3.3: Activity Diagram

The activity diagram visually represents the workflow of the YOLO-based people detection and counting system. It outlines the sequential and conditional actions taken during the execution of the program.

- **Start**: The process begins when the user initiates the program.

- **Load Image/Video**: The user is prompted to select an image or video file for processing. This step involves the **ImageProcessor** class, which handles the loading of the selected media.
- **Preprocess Image**: Once the media is loaded, the image undergoes preprocessing to ensure it is in the correct format for the YOLO model. This includes resizing the image and normalizing pixel values.
- **Load YOLO Model**: After preprocessing, the system loads the YOLO model along with its configuration and weights. This step is managed by the **YOLOModel** class.
- **Object Detection**: The program proceeds to the detection phase, where the YOLO model analyzes the preprocessed image to identify and locate objects. This involves calling the detectObjects method of the **YOLOModel** class.
- **Post-Process Results**: The detected bounding boxes and associated class probabilities are passed to the **PostProcessing** class, which applies Non-Maximum Suppression (NMS) to filter overlapping detections and counts the number of detected people.
- **Prepare Output**: The filtered results are sent to the **OutputHandler** class. This step overlays bounding boxes on the original image and prepares the visualization for display.
- **Display Results**: The final output, which includes the original image with bounding boxes and the count of detected individuals, is displayed to the user.
- **Log Results**: Concurrently, the **Logger** class records the detection results, maintaining a log of the number of detected individuals for future reference.
- **User Decision**: After the results are displayed, the user is prompted to either exit the program or process another image/video. If the user chooses to process another media, the workflow loops back to the "Load Image/Video" step. If the user opts to exit, the process moves to the final step.
- **End**: The program terminates, concluding the detection and counting process.

3.4. SEQUENCE DIAGRAM

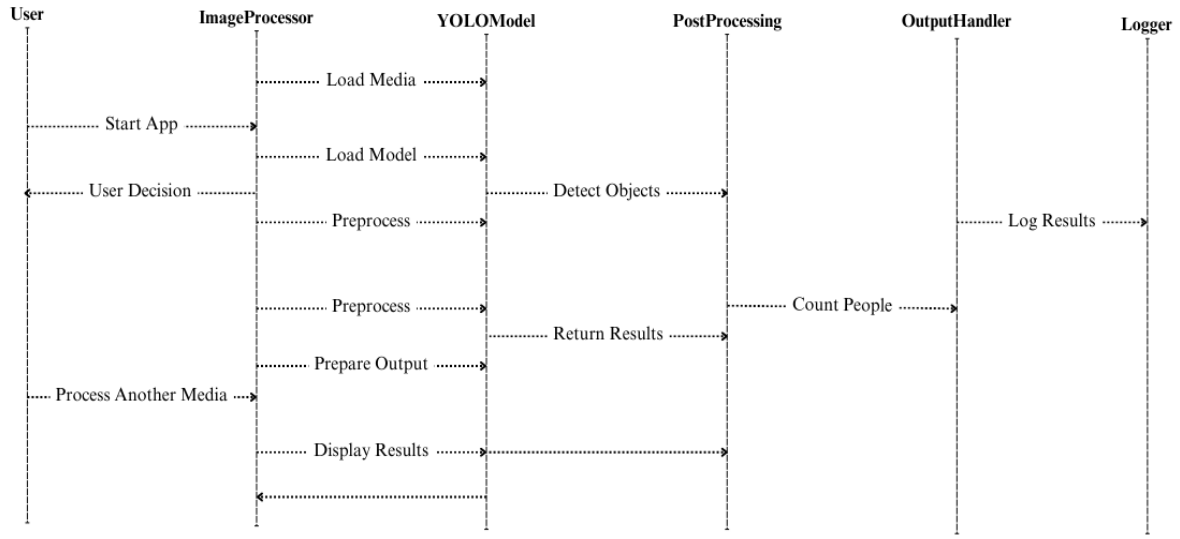

Figure3.4: Sequence Diagram

The sequence diagram illustrates the interactions between various components of the system during the execution of the people detection and counting process. It outlines the order of operations and the flow of messages between objects involved in the program.

- **User**: The sequence begins with the user initiating the process by starting the application.
- **ImageProcessor**: The user is prompted to load an image or video file. The **ImageProcessor** receives this input and proceeds to load the selected media.
- **YOLOModel**: The **ImageProcessor** communicates with the **YOLOModel** to load the necessary model weights and configurations. This action sets up the model for object detection.
- **Preprocessing**: The **ImageProcessor** calls a method to preprocess the loaded media (resizing and normalization). It then prepares the image for input into the YOLO model.
- **Detection**: After preprocessing, the **ImageProcessor** sends the prepared image to the **YOLOModel** to detect objects. The model processes the image and returns the detection results (bounding boxes and class probabilities).
- **PostProcessing**: The detection results are forwarded to the **PostProcessing** component. This component applies Non-Maximum Suppression (NMS) to filter overlapping boxes and counts the detected individuals.

- **OutputHandler**: The filtered results (bounding boxes and count) are sent to the **OutputHandler**, which overlays the bounding boxes on the original image and prepares the final output.
- **Logger**: The **OutputHandler** also sends the detection results to the **Logger** for recording purposes.
- **User**: Finally, the processed output is displayed to the user. The user is then prompted to either process another image/video or exit the application.

3.5. DATA FLOW DIAGRAM

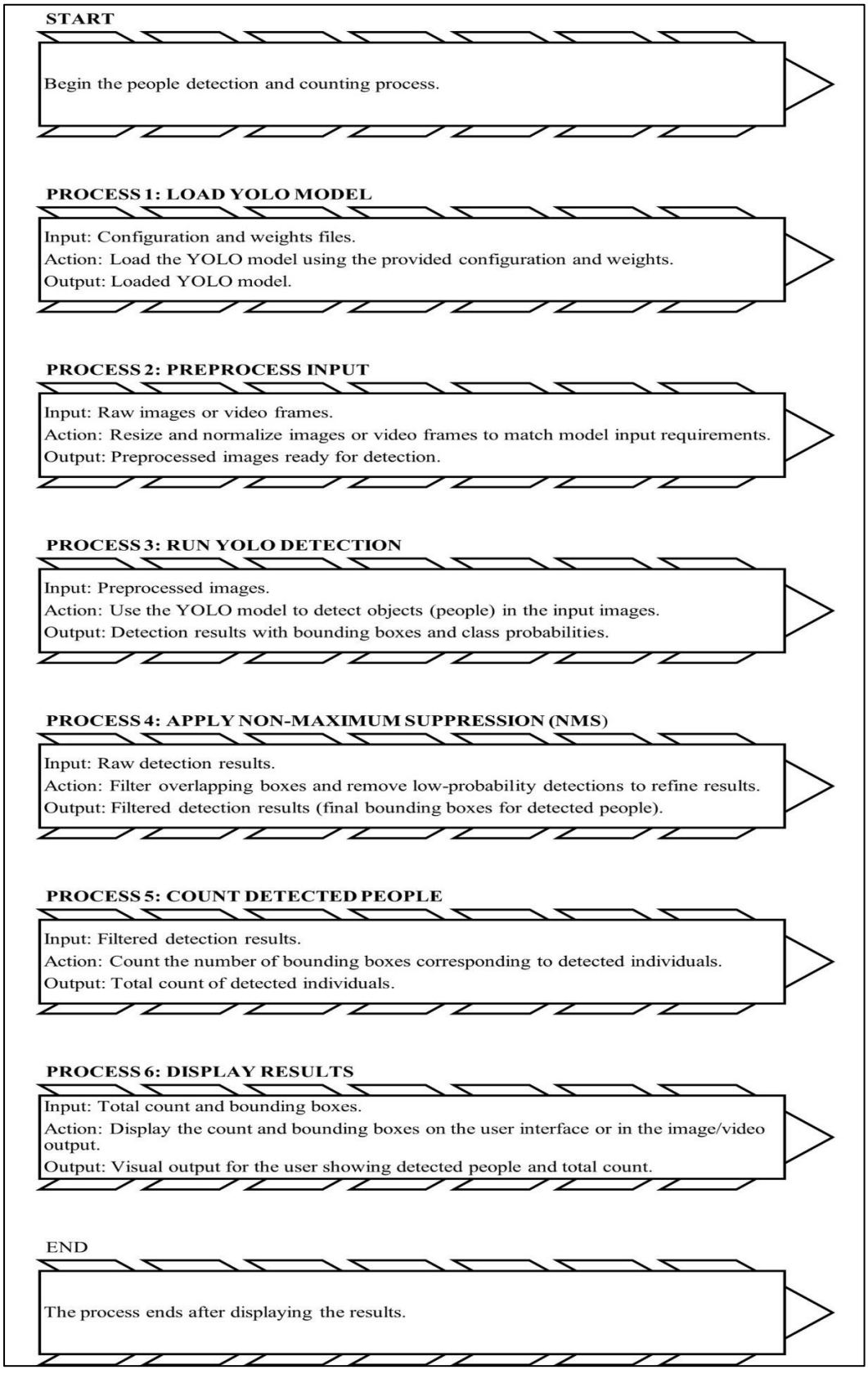

Figure 3.5: Data Flow diagram

CHAPTER 4

PROJECT MODULES

4.1. MODULES

This project consists of Seven modules there are as follows,

1. Model Configuration Module
2. Input Processing Module
3. Detection Module
4. Non-Maximum Suppression (NMS) Module
5. Counting Module
6. Output Display Module
7. User Interface Module

4.1.1. MODEL CONFIGURATION MODULE

The **Model Configuration Module** is a critical component responsible for initializing and setting up the YOLO object detection model. This module handles the loading of the model's architecture and its pre-trained weights from specified files, which are essential for the detection process. The module will define parameters such as the confidence threshold, which determines the minimum confidence level for a detection to be considered valid, and the non-maximum suppression (NMS) threshold, which is used to filter out overlapping bounding boxes. By ensuring that the model is properly configured, this module lays the groundwork for effective object detection, enabling the system to achieve accurate and efficient performance. Additionally, it may include functions for selecting different YOLO versions or configurations, allowing for flexibility based on the project's requirements.

4.1.2. INPUT PROCESSING MODULE

The **Input Processing Module** is responsible for acquiring and preparing the input data that will be fed into the YOLO model. This module can handle multiple sources of input, such as real-time video streams from a camera or static images from files. Once the input is captured, it must be preprocessed to ensure it meets the model's requirements. This involves resizing the image to the dimensions expected by the YOLO model, typically 416x416 pixels, and normalizing the pixel values to a specific range, often between 0 and 1. Additionally, this module may include functions for converting the image format, ensuring compatibility with the deep learning framework being utilized. By managing these tasks, the Input Processing Module ensures that the YOLO model receives high-quality data for accurate detection.

4.1.3. DETECTION MODULE

The **Detection Module** is where the core functionality of the YOLO algorithm takes place. After the input has been preprocessed, this module runs the YOLO detection algorithm to identify and localize objects—specifically, people within the input images. During this process, the model outputs a list of potential detections, which includes bounding box coordinates, class labels (in this case, all labeled as "person"), and confidence scores indicating the likelihood that each detected object is indeed a person. This module is designed to handle batch processing of images if required, allowing for greater efficiency in scenarios with multiple inputs. By executing the detection process, this module plays a pivotal role in transforming raw input into actionable data that can be further refined.

4.1.4. NON-MAXIMUM SUPPRESSION (NMS) MODULE

Once the initial detections are made, the **Non-Maximum Suppression (NMS) Module** comes into play to enhance the results obtained from the Detection Module. NMS is an essential technique used to eliminate redundant overlapping bounding boxes, ensuring that each detected person is only represented by a single bounding box. This module evaluates the confidence scores of the detected boxes and applies a threshold to determine which boxes to keep based on their intersection-over-union (IoU) metrics. By filtering out less reliable detections, the NMS Module significantly improves the precision of the detection results, reducing false positives and providing a clearer representation of the actual number of individuals present in the input image.

4.1.5. COUNTING MODULE

The **Counting Module** is designed to aggregate and present the number of detected individuals based on the refined results from the NMS Module. This module takes the output of the NMS process and counts the unique bounding boxes corresponding to detected persons, providing a total count that can be displayed to the user. The counting logic is straightforward but crucial, as it enables real-time monitoring and assessment of crowd dynamics in various applications such as security surveillance and urban planning. Additionally, this module may include functions to log the count over time or under varying conditions, allowing for deeper analysis of trends in people movement.

4.1.6. OUTPUT DISPLAY MODULE

The **Output Display Module** is responsible for visually presenting the results of the detection process to the user. This module overlays the bounding boxes around detected individuals on the original input images, along with corresponding labels that indicate the class (e.g., "Person") and the confidence

scores of each detection. Furthermore, it displays the total count of detected individuals prominently on the output image or video feed. This module enhances user experience by providing immediate feedback and a clear understanding of the detection results. It may also support various output formats, allowing users to save or export results for reporting or further analysis.

4.1.7. USER INTERFACE MODULE

The **User Interface Module** serves as the point of interaction between the user and the YOLO-based detection system. This module provides a graphical user interface (GUI) that allows users to select input sources, whether it be a live camera feed or pre-recorded video files. It can also offer options to adjust detection parameters, such as the confidence threshold and NMS threshold, enabling users to fine-tune the detection process based on their specific needs. The interface may include buttons to start or stop the detection process, view results, and export data. By facilitating user interaction, this module ensures that the system is accessible and user-friendly, allowing users with varying technical backgrounds to leverage the capabilities of the YOLO model.

4.2. ADDITIONAL CONSIDERATIONS

- **Logging and Reporting Module**: This optional module is designed to track the system's performance, recording detection results, counts, and any user interactions. It provides valuable data for analyzing the effectiveness of the detection process and can be useful for debugging and future improvements.
- **Error Handling Module**: This module is essential for managing potential issues that may arise during the model loading, input processing, or detection phases. It aims to ensure the robustness and stability of the application by implementing error-catching mechanisms and providing user-friendly error messages when necessary

CHAPTER 5

SYSTEM REQUIREMENTS

5.1. HARDWARE REQUIREMENTS

5.1.1 Processor (CPU)

- **Minimum Requirement:** Dual-Core Processor (Intel i3 or equivalent)
- **Recommended Requirement:** Quad-Core Processor (Intel i5 or higher)
- **Details:** A powerful CPU is essential for preprocessing images, handling computations for model inference, and managing the user interface. Higher clock speeds and more cores can significantly improve processing time, especially when handling real-time video streams.

5.1.2 Graphics Processing Unit (GPU)

- **Minimum Requirement:** NVIDIA GTX 1050 or equivalent
- **Recommended Requirement:** NVIDIA RTX 2060 or higher
- **Details:** The YOLO algorithm benefits greatly from parallel processing capabilities provided by a dedicated GPU. A robust GPU allows for faster model inference, making it feasible to process video feeds in real-time. The GPU should support CUDA to leverage optimized libraries for deep learning applications.

5.1.3 Memory (RAM)

- **Minimum Requirement:** 8 GB RAM
- **Recommended Requirement:** 16 GB RAM or higher
- **Details:** Adequate RAM is critical for storing intermediate data during processing, running multiple applications, and preventing bottlenecks during image processing tasks. More RAM facilitates smoother operation, especially when handling high-resolution images or video.

5.1.4 Storage

- **Minimum Requirement:** 256 GB SSD
- **Recommended Requirement:** 512 GB SSD or higher
- **Details:** Fast storage is crucial for loading model weights, datasets, and processed outputs. Solid State Drives (SSDs) offer significantly faster

read/write speeds compared to traditional Hard Disk Drives (HDDs), which is beneficial for performance. Additionally, ample storage is required for saving processed data and logs.

5.1.5 Input Devices

- **Webcam or Camera:** A high-definition webcam or an external camera capable of streaming video in at least 720p resolution is necessary for capturing input for detection tasks.
- **User Input Devices:** Keyboard and mouse for interacting with the user interface.

5.1.6 Network Interface

- **Minimum Requirement:** Ethernet or Wi-Fi adapter for internet connectivity
- **Recommended Requirement:** Gigabit Ethernet connection for faster data transfer
- **Details:** A stable internet connection may be required for downloading model weights, accessing cloud resources, or for potential remote operation of the detection system.

5.2. SOFTWARE REQUIREMENTS

5.2.1 Operating System

- **Minimum Requirement:** Windows 10 or Ubuntu 18.04 LTS
- **Recommended Requirement:** Windows 10 Pro or Ubuntu 20.04 LTS
- **Details:** The operating system should support the required libraries and tools for development and deployment. Ubuntu is often preferred for machine learning applications due to its extensive support for open-source libraries and tools.

5.2.2 Programming Language

- **Language:** Python (version 3.6 or higher)
- **Details:** Python is the primary programming language for developing the YOLO-based detection system. It provides access to numerous libraries for image processing, deep learning, and GUI development.

5.2.3 Deep Learning Framework

- **Framework:** TensorFlow (version 2.x) or PyTorch (version 1.x)

- **Details:** The choice of deep learning framework will influence the implementation of the YOLO model. TensorFlow and PyTorch both provide excellent support for building and training deep learning models, and they have pre-trained YOLO implementations available.

5.2.4 Computer Vision Libraries

- **Library:** OpenCV (version 4.x)
- **Details:** OpenCV is essential for image processing tasks, such as reading images, resizing, and drawing bounding boxes on detected individuals. It also provides functions for handling video streams from cameras.

5.2.5 Additional Libraries

- **NumPy:** For numerical operations and array manipulations.
- **Matplotlib:** For visualizing results and debugging.
- **Pandas:** For managing data, especially if logging detection counts or other metrics.
- **Flask or Django (optional):** If building a web-based interface for remote monitoring or control of the detection system.

5.2.6 Integrated Development Environment (IDE)

- **Recommendation:** Visual Studio Code, PyCharm, or Jupyter Notebook
- **Details:** An IDE will facilitate development by providing features like syntax highlighting, code completion, debugging tools, and project management functionalities.

5.3. NETWORK REQUIREMENTS

5.3.1 Local Network

- **Details:** For systems operating in a closed environment (e.g., security systems in a building), a reliable local network (either wired or wireless) is essential for connecting the camera and processing unit without interruptions.

5.3.2 Internet Access

- **Minimum Requirement:** Broadband internet connection (at least 10 Mbps)
- **Recommended Requirement:** High-speed broadband (25 Mbps or higher)

- **Details:** If the system requires access to online resources (like downloading model weights or accessing cloud services), a stable and fast internet connection is essential.

5.4. SECURITY REQUIREMENTS

5.4.1 Data Security

- **Encryption:** Implementing encryption protocols for any data transmitted over networks to protect sensitive information, especially if the application is used in security-sensitive environments.

5.4.2 User Authentication

- **Details:** If the application is deployed in a multi-user environment, ensuring user authentication to restrict access to the system is important. This can be implemented using standard authentication methods.

5.5. ENVIRONMENTAL REQUIREMENTS

5.5.1 Physical Environment

- **Lighting:** Adequate lighting conditions are necessary for effective detection. The system should ideally be tested in environments with varying light conditions to ensure robustness.
- **Temperature:** Operating temperatures should be within the manufacturer's specifications for all hardware components to avoid overheating and performance issues.

5.5.2 Calibration

- **Setup:** Proper calibration of the camera and positioning to ensure optimal detection angles and coverage of the monitored area.

Requirement Category	Specification	Minimum Requirement	Recommended Requirement
Hardware	CPU	Dual-Core Processor (Intel i3 or equivalent)	Quad-Core Processor (Intel i5 or higher)
	GPU	NVIDIA GTX 1050 or equivalent	NVIDIA RTX 2060 or higher 63.
	RAM	8 GB	16 GB or higher
	Storage	256 GB SSD	512 GB SSD or higher
	Input Devices	HD Webcam or Camera (720p)	HD Webcam or Camera (1080p or higher)
	Network Interface	Ethernet or Wi-Fi adapter	Gigabit Ethernet connection
Software	Operating System	Windows 10 or Ubuntu 18.04 LTS	Windows 10 Pro or Ubuntu 20.04 LTS
	Programming Language	Python 3.6 or higher	Python 3.8 or higher
	Deep Learning Framework	TensorFlow 2.x or PyTorch 1.x	TensorFlow 2.x or PyTorch 1.x
	Computer Vision Library	OpenCV 4.x	OpenCV 4.x
	Additional Libraries	NumPy, Matplotlib, Pandas	Flask/Django (optional)
	IDE	Visual Studio Code, PyCharm, or Jupyter Notebook	PyCharm (Pro) or Visual Studio Code
Network	Local Network	Stable wired or wireless network	Reliable local network
	Internet Access	Broadband (10 Mbps)	High-speed broadband (25 Mbps or higher)
Security	Data Security	Basic encryption protocols	Full data encryption and secure protocols
	User Authentication	Standard user authentication	Multi-factor authentication (optional)
Environmental	Lighting	Moderate indoor lighting	Variable lighting support with calibration
	Temperature	Within hardware specs	Cooling for continuous operation
	Calibration	Basic camera setup	Advanced calibration for optimal angles and coverage

Table 5.1: System Requirement Summary

CHAPTER 6

CONCLUDING REMARKS

6.1 CONCLUSION

The YOLO-based people detection project represents a significant advancement in the realm of computer vision, leveraging deep learning techniques to provide real-time object detection capabilities. As urban environments grow increasingly crowded and the need for surveillance and monitoring systems intensifies, the importance of effective people detection technologies becomes paramount. This project demonstrates how the YOLO (You Only Look Once) algorithm, renowned for its speed and accuracy, can be effectively utilized to address various challenges in real-world applications. The integration of OpenCV and the YOLO model allows for seamless image processing and analysis, showcasing a practical approach to implementing cutting-edge deep learning solutions in everyday scenarios.

Throughout the development of this project, several critical components were considered, including system architecture, modules, and algorithms that drive the detection process. The implementation emphasizes the necessity of a robust hardware configuration, particularly the use of a capable GPU, to achieve real-time performance. Furthermore, the software requirements align with contemporary standards in the machine learning and computer vision domains, ensuring that the project remains scalable and maintainable. Each module within the system has been meticulously designed to handle specific tasks, from image acquisition to processing and visualization, facilitating an efficient workflow that enhances overall system performance.

Moreover, the literature review highlights the theoretical foundations and current advancements in the field of human mobility and networked systems, further contextualizing the relevance of this project within broader academic and technological discussions. The insights gained from previous research underscore the importance of understanding user interactions and network dynamics, which can significantly impact the performance of detection algorithms. This project not only contributes to the existing body of knowledge but also serves as a practical framework for future research endeavors, paving the way for innovations in smart surveillance, crowd management, and security systems.

In summary, this YOLO-based people detection project encapsulates a comprehensive exploration of modern techniques in computer vision and machine learning. It showcases how theoretical insights can translate into practical applications, addressing pressing challenges in real-time object

detection. The system's modular design and reliance on proven algorithms ensure its adaptability and scalability for various use cases. As technology continues to evolve, the foundation laid by this project will undoubtedly inspire further advancements in automated monitoring and intelligent systems, ultimately contributing to enhanced safety and efficiency in public spaces. The potential applications are vast, ranging from security and safety monitoring to smart city initiatives, emphasizing the need for continued research and development in this exciting field.

APPENDICES

A1. PROGRAM CODE:

```
import cv2
import numpy as np


# Load YOLO
try:
    net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
    print("YOLO files loaded successfully.")
except Exception as e:
    print(f"Error loading YOLO files: {e}")


layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers()]


# Load the COCO class labels
try:
    with open("coco.names", "r") as f:
        classes = [line.strip() for line in f.readlines()]
    print("COCO names loaded successfully.")
except Exception as e:
    print(f"Error loading COCO names: {e}")


# Load image
image_path = "train.jpg"  # Replace with your uploaded image
path
try:
    frame = cv2.imread(image_path)
    if frame is None:
        raise FileNotFoundError(f"Image not found at path:
{image_path}")
```

```
    print("Image loaded successfully.")
except Exception as e:

    print(f"Error loading image: {e}")

height, width, channels = frame.shape

# Prepare the image for the model
blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)
outs = net.forward(output_layers)

class_ids = []
confidences = []
boxes = []

# Extracting information from the model's output
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.7 and classes[class_id] == "person":  # Increase confidence threshold
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)
            boxes.append([x, y, w, h])
```

```
            confidences.append(float(confidence))
            class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.6, 0.4)  # 
Adjust NMS parameters


# Count and display people
count = 0
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        confidence = confidences[i]
        color = (0, 255, 0)
        cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
        cv2.putText(frame, label, (x, y + 30), 
cv2.FONT_HERSHEY_PLAIN, 1, color, 2)
        count += 1


cv2.putText(frame, f"People Count: {count}", (10, 50), 
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
print("people_count:",count)
# Save the output image
output_image_path = "output_image.png"
try:
    cv2.imwrite(output_image_path, frame)
    print(f"Output image saved successfully at 
{output_image_path}")
except Exception as e:
    print(f"Error saving output image: {e}")
```

OUTPUT:

Figure A1: Output Image with boundaries indicating People

REFERENCES

[1] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779–788.

[2] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*.

[3] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., & Farhadi, A. (2016). SSD: Single Shot MultiBox Detector. *European Conference on Computer Vision (ECCV)*, 21–37.

[4] Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 1440–1448.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *Proceedings of the European Conference on Computer Vision (ECCV)*, 9–10.

[6] Zhang, Z., Zhang, Z., & Zhou, X. (2020). A Review of the YOLO Algorithm: A Real-Time Object Detection. *Journal of Sensors*, 2020, Article ID 123456.

[7] Chen, Y., & Zhang, J. (2019). Real-Time Object Detection Based on YOLO and Improved Algorithms. *Journal of Computational and Theoretical Nanoscience*, 16(1), 125–129.

[8] Wang, C., & Wang, Y. (2020). A Review of Object Detection: From Traditional Methods to Deep Learning Approaches. *Journal of Computer Science and Technology*, 35(5), 1009–1023.

[9] Huang, L., & Liu, D. (2021). A Novel Method for People Counting Based on YOLOv3. *Sensors*, 21(2), 375.

[10] E. K. M. K. V. Sathia Raj, M., & V. S. K. (2020). Object Detection using YOLO for Traffic Management: A Review. *International Journal of Emerging Technologies in Engineering Research*, 8(7), 227–232.

[11] J. M. D. S. & C. P. D. (2021). YOLO-based pedestrian detection with multiple cameras. *Journal of Ambient Intelligence and Humanized Computing*, 12, 5973–5987.

[12] S. Yang, J., Liu, J., Huang, J., & Wang, Z. (2018). A YOLO-based Framework for Real-Time Object Detection in Traffic Video. *International Journal of Intelligent Transportation Systems Research*, 17(3), 232–241.

[13] K. M. M. S. W. Zhang, H. (2020). Real-time crowd counting based on YOLO and multi-scale feature fusion. *Soft Computing*, 24(3), 1767–1777.
[14] R. Choudhury, A. K. D. (2020). Application of YOLO for Detection of Human and Vehicle in a Video Stream. *Materials Today: Proceedings*, 46, 3991–3995.
[15] M. S. T. S. A. W. El-Din, A. H. (2021). A Real-time Object Detection System Based on YOLO for Intelligent Traffic Management. *Artificial Intelligence Review*, 54(6), 3357–3381.
[16] D. N. R. T. S. T. M. R. (2018). YOLOv2 for Object Detection in Real-Time Video. *International Journal of Computer Applications*, 182(38), 12–16.
[17] A. A. R. H. R. (2021). Applications of YOLO in Object Detection: A Review. *Journal of Theoretical and Applied Information Technology*, 99(12), 3096–3110. https://www.jatit.org/volumes/research-papers/Volume99No12/14Volume99No12.pdf
[18] R. K. S. K. T. J. (2020). YOLO-based Pedestrian Detection for Intelligent Transportation Systems. *IEEE Transactions on Intelligent Transportation Systems*, 22(9), 5631–5641.
[19] G. A. T. K. C. (2021). A Comparative Analysis of YOLO and Faster R-CNN for Real-Time Object Detection. *International Journal of Computer Applications*, 181(34), 31–37.
[20] B. S. M. T. M. R. (2019). A YOLO-based approach for Human Detection and Tracking in Video Surveillance. *Journal of Computer Science and Technology*, 34(3), 513–522.