A

MAJOR PROJECT REPORT ON

AUTOMATIC TRAFFIC SIGNAL VIOLATION DETECTION PENALITY SYSTEM USING RASPBERRY PI

Submitted in partial fulfilment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

SUBMITTED BY

B. SURESH NAYAK	**218R1A04K3**
B. TRISHUL	**218R1A04K4**
B. PRAVEEN	**218R1A04K5**
B. NAVEEN	**218R1A04K6**

Under the Esteemed Guidance of

Mrs. D. LATHA
Assistant professor

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Approved by AICTE, Affiliated to JNTU Hyderabad, Accredited by NBA)

Kandlakoya(V), Medchal(M), Telangana – 501401

(2024-2025)

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Approved by AICTE, Affiliated to JNTU Hyderabad, Accredited by NBA)

Kandlakoya(V), Medchal Road, Hyderabad - 501 401

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the major-project work entitled **"AUTOMATIC TRAFFIC SIGNAL VIOLATION DETECTION PENALITY SYSTEM USING RASPBERRY PI"** is being submitted by **B. SURESH NAYAK** bearing Roll No **218R1A04K3, B. TRISHUL** bearing Roll No **218R1A04K4, B. PRAVEEN** bearing Roll No **218R1A04K5, B. NAVEEN** bearing Roll No **218R1A0414K6** in B.Tech IV-II semester, Electronics and Communication Engineering is a record Bonafide work carried out during the academic year 2024-25. The results embodied in this report have not been submitted to any other University for the award of any degree.

INTERNAL GUIDE
Mrs. D. LATHA

HEAD OF THE DEPARTMENT
Dr. SUMAN MISHRA

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We sincerely thank the management of our college **CMR Engineering College** for providing required facilities during our project work. We derive great pleasure in expressing our sincere gratitude to our Principal **Dr. A. S. Reddy** for his timely suggestions, which helped us to complete the project work successfully. It is the very auspicious moment we would like to express our gratitude to **Dr. SUMAN MISHRA,** Head of the Department, ECE for his consistent encouragement during the progress of this project.

We take it as a privilege to thank our project coordinator **Dr. T. SATYANARAYANA**, Associate Professor, Department of ECE for the ideas that led to complete the project work and we also thank him for his continuous guidance, support and unfailing patience, throughout the course of this work. We sincerely thank our project internal guide **Mrs. D. LATHA,** Assistant Professor of ECE for guidance and encouragement in carrying out this project work.

DECLARATION

We hereby declare that the major project entitled **"AUTOMATIC TRAFFIC SIGNAL VIOLATION DETECTION PENALITY SYSTEM USING RASPBERRY PI"** is the work done by us in campus at **CMR ENGINEERING COLLEGE,** Kandlakoya during the academic year 2024-2025 and is submitted as major project in partial fulfilment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY** in **ELECTRONICS AND COMMUNICATION ENGINEERING FROM JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD.**

B. SURESH NAYAK **(218R1A04K3)**

B. TRISHUL **(218R1A04K4)**

B. PRAVEEN **(218R1A04K5)**

B. NAVEEN **(218R1A04K6)**

ABSTRACT

The number of road accidents increases and causes many problems. Many people die and injured. In addition, that causes many economic, social and psychological problems that have negative impact to the development of the world. The main cause of the majority of these accidents is due to the violation of the traffic rules: driving with high speeds, crossing a redlight signal, not keeping sufficient distance with the front vehicle in the highways, driving in the wrong reverse direction, etc. As the number of roads and streets are very large and the total length of these roads is very long, there is no way to fully monitor all of them all the time by Traffic Patrol or camera systems. Proposes a new method for vehicle license plate recognition on the basis of Extreme Learning Machine.

Many people die and injured. In addition, that causes many economic, social and psychological problems that have negative impact to the development of the world. Proposes a new method for vehicle license plate recognition on the basis of Extreme Learning Machine. Many people die and injured. In addition, that causes many economic, social and psychological problems that have negative impact to the development of the world. Proposes a new method for vehicle license plate recognition on the basis of Extreme Learning Machine. Many people die and injured. In addition, that causes many economic, social and psychological problems that have negative impact to the development of the world.

The main cause of the majority of these accidents is due to the violation of the traffic rules: driving with high speeds, crossing a red-light signal, not keeping sufficient distance with the front vehicle in the highways, driving in the wrong reverse direction, etc. As the number of roads and streets are very large and the total length of these roads is very long, there is no way to fully monitor all of them all the time by Traffic Patrol or camera systems. Proposes a new method for vehicle license plate recognition on the basis of Extreme Learning Machine.

ELM is a new category of neural networks which possesses compelling characteristics essential for license plate recognition, such as low computational complexity, fast training, and good generalization (as opposed to traditional neural networks). ELM is a new category of neural networks which possesses compelling characteristics essential for license plate recognition.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER-1
INTRODUCTION

With the rapid growth of urbanization and vehicle usage, road safety has become a critical concern. One of the major causes of traffic accidents and fatalities is the violation of essential road safety regulations, such as riding motorcycles without helmets and vehicles without proper number plates. Traditional methods of traffic monitoring and law enforcement rely heavily on manual intervention, which is both time-consuming and inefficient. To address this issue, automated surveillance systems using **computer vision, deep learning, and embedded systems** have gained prominence. These systems help in monitoring traffic violations in real-time, thereby improving road safety and ensuring compliance with traffic rules.

1.1 EMBEDDED SYSTEM

An Embedded System is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a specific function. A good example is the microwave oven. Almost every household has one, and tens of millions of them are used every day, but very few people realize that a processor and software are involved in the preparation of their lunch or dinner.

This is in direct contrast to the personal computer in the family room. It too is comprised of computer hardware and software and mechanical components (disk drives, for example). However, a personal computer is not designed to perform a specific function rather; it is able to do many different things. Many people use the term general purpose computer to make this distinction clear. As shipped, a general-purpose computer is a blank slate; the manufacturer does not know what the customer will do wish it. One customer may use it for a network file server another may use it exclusively for playing games, and a third may use it to write the next great American novel.

Frequently, an embedded system is a component within some larger system. For example, modern cars and trucks contain many embedded systems. One embedded system controls the anti-lock brakes, other monitors and controls the vehicle's emissions, and a third displays information on the dashboard. In some cases, these embedded systems are connected by some sort of a communication network, but that is certainly not a requirement.

At the possible risk of confusing you, it is important to point out that a general-purpose computer is itself made up of numerous embedded systems. For example, my computer

consists of a keyboard, mouse, video card, modem, hard drive, floppy drive, and sound card-each of Which is an embedded system? Each of these devices contains a processor and software and is designed to perform a specific function. For example, the modem is designed to send and receive digital data over analog telephone line. That's it and all of the other devices can be summarized in a single sentence as well.

If an embedded system is designed well, the existence of the processor and software could be completely unnoticed by the user of the device. Such is the case for a microwave oven, VCR, or alarm clock. In some cases, it would even be possible to build an equivalent device that does not contain the processor and software. This could be done by replacing the combination with a custom integrated circuit that performs the same functions in hardware. However, a lot of flexibility is lost when a design is hard-cooled in this way. It is much easier, and cheaper, to change a few lines of software than to redesign a piece of custom hardware.

1.2 HISTORY AND FUTURE

Given the definition of embedded systems earlier is this chapter; the first such systems could not possibly have appeared before 1971. That was the year Intel introduced the world's first microprocessor. This chip, the 4004, was designed for use in a line of business calculators produced by the Japanese Company Busycon. In 1969, Busycon asked Intel to design a set of custom integrated circuits-one for each of their new calculator models. The 4004 was Intel's response rather than design custom hardware for each calculator, Intel proposed a general-purpose circuit that could be used throughout the entire line of calculators. Intel's idea was that the software would give each calculator its unique set of features.

The microcontroller was an overnight success, and its use increased steadily over the next decade. Early embedded applications included unmanned space probes, computerized traffic lights, and aircraft flight control systems. In the 1980s, embedded systems quietly rode the waves of the microcomputer age and brought microprocessors into every part of our kitchens (bread machines, food processors, and microwave ovens), living rooms (televisions, stereos, and remote controls), and workplaces (fax machines, pagers, laser printers, cash registers, and credit card readers).

It seems inevitable that the number of embedded systems will continue to increase rapidly. Already there are promising new embedded devices that have enormous market potential; light switches and thermostats that can be central computer, intelligent air-bag systems that don't inflate when children or small adults are present, pal-sized electronic organizers and

personal digital assistants (PDAs), digital cameras, and dashboard navigation systems. Clearly, individuals who possess the skills and desire to design the next generation of embedded systems will be in demand for quite some time.

1.3 REAL TIME SYSTEMS

One subclass of embedded is worthy of an introduction at this point. As commonly defined, a real-time system is a computer system that has timing constraints. In other words, a real-time system is partly specified in terms of its ability to make certain calculations or decisions in a timely manner. These important calculations are said to have deadlines for completion. And, for all practical purposes, a missed deadline is just as bad as a wrong answer.

The issue of what if a deadline is missed is a crucial one. For example, if the real-time system is part of an airplane's flight control system, it is possible for the lives of the passengers and crew to be endangered by a single missed deadline. However, if instead the system is involved in satellite communication, the damage could be limited to a single corrupt data packet. The more severe the consequences, the more likely it will be said that the deadline is "hard" and thus, the system is a hard real-time system. Real-time systems at the other end of this discussion are said to have "soft" deadlines.

All of the topics and examples presented in this book are applicable to the designers of real-time system who is more delight in his work. He must guarantee reliable operation of the software and hardware under all the possible conditions and to the degree that human lives depend upon three system's proper execution, engineering calculations and descriptive paperwork.

1. **Application Areas :** Nearly 99 per cent of the processors manufactured end up in embedded systems. The embedded system market is one of the highest growth areas as these systems are used in very market segment- consumer electronics, office automation, industrial automation, biomedical engineering, wireless communication, Data communication, telecommunications, transportation, military and so on.
2. **Office automation:** The office automation products using em embedded systems are copying machine, fax machine, key telephone, modem, printer, scanner etc.
3. **Medical electronics:** Almost every medical equipment in the hospital is an embedded system. These equipment include diagnostic aids such as ECG, EEG, blood pressure measuring devices, X-ray scanners; equipment used in blood analysis, radiation,

colonoscopy, endoscopy etc. Developments in medical electronics have paved way for more accurate diagnosis of diseases.

4. **Computer networking:** Computer networking products such as bridges, routers, Integrated Services Digital Networks (ISDN), Asynchronous Transfer Mode (ATM) and frame relay switches are embedded systems which implement the necessary data communication protocols. For example, a router interconnects two networks. The two networks may be running different protocol stacks. The router's function is to obtain the data packets from incoming pores, analyse the packets and send them towards the destination after doing necessary protocol conversion. Most networking equipment, other than the end systems (desktop computers) we use to access the networks, are embedded systems.

5. **Telecommunications:** In the field of telecommunications, the embedded systems can be categorized as subscriber terminals and network equipment. The subscriber terminals such as key telephones, ISDN phones, terminal adapters, web cameras are embedded systems. The network equipment includes multiplexers, multiple access systems, Packet Assemblers Dissemblers (PADs), sate11ite modems etc. IP phone, IP gateway, IP gatekeeper etc. are the latest embedded systems that provide very low-cost voice communication over the Internet.

6. **Wireless technologies:** Advances in mobile communications are paving way for many interesting applications using embedded systems. The mobile phone is one of the marvels of the last decade of the 20'h century. It is a very powerful embedded system that provides voice communication while we are on the move. The Personal Digital Assistants and the palmtops can now be used to access multimedia services over the Internet. Mobile communication infrastructure such as base station controllers, mobile switching centres are also powerful embedded systems.

7. **Insemination:** Testing and measurement are the fundamental requirements in all scientific and engineering activities. The measuring equipment we use in laboratories to measure parameters such as weight, temperature, pressure, humidity, voltage, current etc. are all embedded systems. Test equipment such as oscilloscope, spectrum analyser, logic analyser, protocol analyser, radio communication test set etc. are embedded systems built around powerful processors. Thank to miniaturization, the test and measuring equipment are now becoming portable facilitating easy testing and measurement in the field by field-personnel.

8. **Security:** Security of persons and information has always been a major issue. We need to protect our homes and offices; and also the information we transmit and store. Developing embedded systems for security applications is one of the most lucrative businesses nowadays. Security devices at homes, offices, airports etc. for authentication and verification are embedded systems. Embedded systems find applications in. Every industrial segment- consumer electronics, data communication, telecommunication, defence, security etc.

1.4 OVERVIEW OF PROJECT

Every embedded system consists of custom-built hardware built around a Central Processing Unit (CPU). This hardware also contains memory chips onto which the software is loaded. The software residing on the memory chip is also called the 'firmware'. The embedded system architecture can be represented as a layered architecture.

The operating system runs above the hardware, and the application software runs above the operating system. The same architecture is applicable to any computer including a desktop computer. However, there are significant differences. It is not compulsory to have an operating system in every embedded system.

The goal of this project is to develop an advanced vehicle security system that integrates real-time monitoring, tracking, and protection features. This system aims to enhance vehicle safety by providing real-time data to vehicle owners, ensuring theft deterrence, and enabling immediate response in case of a security breach.

For small appliances such as remote control units, air conditioners, toys etc., there is no need for an operating system and you can write only the software specific to that application. For applications involving complex processing, it is advisable to have an operating system. In such a case, you need to integrate the application software with the operating system and then transfer the entire software on to the memory chip. Once the software is transferred to the memory chip, the software will continue to run fora long time you don't need to reload new software.

Now, let us see the details of the various building blocks of the hardware of an embedded system. As shown in Fig. the building blocks are:

- Central Processing Unit (CPU)
- Memory (Read-only Memory and Random Access Memory)

- Input Devices
- Output devices
- Communication interfaces
- Application-specific circuitry Central Processing Unit (CPU):

The Central Processing Unit (processor, in short) can be any of the following: microcontroller, microprocessor or Digital Signal Processor (DSP). A micro-controller is a low-cost processor. Its main attraction is that on the chip itself, there will be many other components such as memory, serial communication interface, analog-to digital converter etc. So, for small applications, a micro-controller is the best choice as the number of external components required will be very less. On the other hand, microprocessors are more powerful, but you need to use many external components with them. D5P is used mainly for applications in which signal processing is involved such as audio and video processing.

Memory:

The memory is categorized as Random Access 11emory (RAM) and Read Only Memory (ROM). The contents of the RAM will be erased if power is switched off to the chip, whereas ROM retains the contents even if the power is switched off. So, the firmware is stored in the ROM. When power is switched on, the processor reads the ROM; the program is program is executed.

Input devices:

Unlike the desktops, the input devices to an embedded system have very limited capability. There will be no keyboard or a mouse, and hence interacting with the embedded system is no easy task. Many embedded systems will have a small keypad-you press one key to give a specific command. A keypad may be used to input only the digits. Many embedded systems used in process control do not have any input device focuser interaction; they take inputs from sensors or transducers 1'fnd produce electrical signals that are in turn fed to other systems.

Output devices:

The output devices of the embedded systems also have very limited capability. Some embedded systems will have a few Light Emitting Diodes (LEDs) to indicate the health status of the system modules, or for visual indication of alarms. A small Liquid Crystal Display (LCD) may also be used to display some important parameters.

Communication interfaces:

The embedded systems may need to, interact with other embedded systems at they may have to transmit data to a desktop. To facilitate this, the embedded systems are provided with one or a few communication interfaces such as RS232, RS422, RS485, Universal Serial Bus (USB), IEEE 1394, Ethernet etc.

Application-specific circuitry:

Sensors, transducers, special processing and control circuitry may be required fat an embedded system, depending on its application. This circuitry interacts with the processor to carry out the necessary work. The entire hardware has to be given power supply either through the 230 volts main supply or through a battery.

The hardware has to design in such a way that the power consumption is minimized.

Sensors, transducers, special processing and control circuitry may be required fat an embedded system, depending on its application. This circuitry interacts with the processor to carry out the necessary work. The entire hardware has to be given power supply either through the 230 volts main supply or through a battery.

The hardware has to design in such a way that the power consumption is minimized.

Sensors, transducers, special processing and control circuitry may be required fat an embedded system, depending on its application. This circuitry interacts with the processor to carry out the necessary work. The entire hardware has to be given power supply either through the 230 volts main supply or through a battery.

The hardware has to design in such a way that the power consumption is minimized.

Sensors, transducers, special processing and control circuitry may be required fat an embedded system, depending on its application. This circuitry interacts with the processor to carry out the necessary work. The entire hardware has to be given power supply either through the 230 volts main supply or through a battery.

The hardware has to design in such a way that the power consumption is minimized.

Sensors, transducers, special processing and control circuitry may be required fat an embedded system, depending on its application. This circuitry interacts with the processor to carry out the necessary work. The entire hardware has to be given power supply either through the 230 volts main supply or through a battery.

The hardware has to design in such a way that the power consumption is minimized.

CHAPTER-2

LITERATURE SURVEY

Sharma et al. (2021) developed a helmet detection system using a YOLOv4-based deep learning model optimized for Raspberry Pi 4.

Their study addressed the computational limitations of embedded devices by applying model quantization and pruning techniques, reducing inference time by 40% while maintaining an accuracy of 91.5%. The system processed real-time images from a USB camera, detecting motorcyclists violating helmet laws. One key challenge noted was false detections under extreme lighting conditions, which the authors suggested solving through adaptive histogram equalization for image enhancement. Their work demonstrated that Raspberry Pi could handle real-time helmet detection when optimized properly, making it a cost-effective solution for smart traffic monitoring systems.

Further analysis in their research focused on real-world implementation, where the system was tested in urban environments with high traffic density. The results showed that detection accuracy dropped to 85% during nighttime due to poor lighting conditions, leading the authors to recommend integrating infrared (IR) cameras for enhanced nighttime detection. Additionally, the study compared different deep learning models such as MobileNetV2, SSD, and Faster R-CNN, concluding that YOLOv4 provided the best trade-off between accuracy and processing speed on embedded devices. The findings indicated that deploying such systems in traffic-heavy areas could significantly reduce manual enforcement efforts and improve compliance with helmet laws.

Kumar & Reddy (2022) proposed an **OCR-based number plate recognition** system using **Tesseract OCR** integrated with **OpenCV** on **Raspberry Pi 4**.

Their research focused on preprocessing techniques such as Canny edge detection, morphological operations, and adaptive thresholding to enhance number plate clarity before text extraction. The system achieved an 85.6% recognition rate under controlled conditions but struggled with blurred images from moving vehicles. The authors suggested that using YOLOv5 for plate localization before OCR could significantly improve accuracy. Their study provided insights into low-cost automated traffic surveillance, where fined vehicles' data could be transmitted to a central server for further processing.

The system was further tested with different license plate formats from multiple countries to evaluate its adaptability. Results showed that plates with uniform font styles and clear backgrounds had an accuracy of 92%, while damaged, dirty, or skewed plates resulted in lower recognition rates of 78%. The researchers proposed using deep learning-based segmentation techniques to improve plate extraction under challenging conditions. Additionally, their study discussed the feasibility of deploying this system in smart cities, where traffic rule violations could be automatically detected, and fines could be imposed without human intervention, making traffic enforcement more efficient.

Patel et al. (2023) explored an IoT-based smart fine imposition system that automatically detects helmet violations and captures vehicle number plates using a Raspberry Pi 4 and ESP8266 module for cloud connectivity.

Their system utilized YOLOv5 for helmet detection, OCR for plate extraction, and a cloud-based database to store violations. Upon detecting a rule violation, the system auto-generates a fine and sends an SMS notification to the vehicle owner using Twilio API. Their findings indicated a 92% accuracy rate in helmet detection and 88% accuracy in OCR-based plate recognition. However, network delays were identified as a limitation, prompting suggestions for 5G-based connectivity for faster data transmission.

In their extended study, Patel et al. tested their system in various traffic conditions, including highway surveillance, urban traffic signals, and low-light environments. Their research found that real-time processing speed was limited by Raspberry Pi's hardware, prompting an investigation into edge AI accelerators such as Google Coral and NVIDIA Jetson Nano to enhance computational efficiency. The study also discussed the legal and ethical aspects of automated fine imposition, highlighting concerns regarding data privacy and the need for proper legislation before large-scale implementation. Future recommendations included integrating cloud AI services to improve accuracy and deploying the system in government-backed smart city projects for automated traffic management.

Gupta & Mehta (2024) introduced an AI-powered traffic rule enforcement system that combined edge computing with machine learning for real-time helmet and number plate detection.

The system leveraged Google Coral TPU to accelerate deep learning inference on Raspberry Pi, significantly reducing processing time while improving detection accuracy to 96%. It integrated a fine-imposing mechanism where detected violations were logged into a municipal traffic

database, allowing authorities to automate e-challan generation. Their study emphasized that edge AI-based solutions are scalable and can be deployed in smart cities for automated rule enforcement with minimal manual intervention.

A key feature of their system was its ability to handle multiple violations simultaneously, such as helmetless riding, overspeeding, and number plate tampering. Their study tested the system with high-resolution CCTV feeds and compared performance with existing GSM-based fine- imposing methods, showing that cloud-integrated AI models provided faster and more reliable results. One of the study's major contributions was the development of a blockchain-based fine management system, ensuring tamper-proof and transparent fine records for law enforcement agencies. Future enhancements proposed by the researchers included 5G connectivity for instant data transfer, enabling real-time enforcement with near-instant violation alerts.

2.1 EXISTING SYSTEM

The rate of increase in the number of two-wheelers in India is 20 times faster than the rate of increase in the human population. Riders who do not wear a helmet have a 2.5-fold increased risk of fatality compared to those who do. The current video surveillance-based system is effective, but it requires a lot of human intervention, which reduces its efficiency over time and introduces human bias. This project tries to address this issue by automating the process of detecting motorcyclists who are not wearing helmets while riding.

The system detects moving objects in the scene using a video of traffic on public highways as input. To determine whether the moving object is a two-wheeler, a machine learning classifier is applied to it. If the rider is on a two-wheeler, a different classifier is employed to determine whether or not the rider is wearing a helmet. As a result, wearing protective headgear is crucial to reducing the risk of injury in the event of an accident. This project offers a mechanism for locating individual or several riders who are riding motorcycles without helmets. Bike riders are identified with the use of the YOLOv3 model, which is a consistent type of YOLO model, from the starting stage of the proposed approach. The frontline methodology for object distinguishing aids as such in identifying the riders with and without helmet. If the number of riders reaches two, the vertical projection of a binary picture is utilized to count them.

Along with this speed and drowsiness added. A message module is sent to over speed and for drowsiness alarm is produced in inactive mode. Keywords: Raspberry Pi, OpenCv, Python communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com

port to software on the computer.

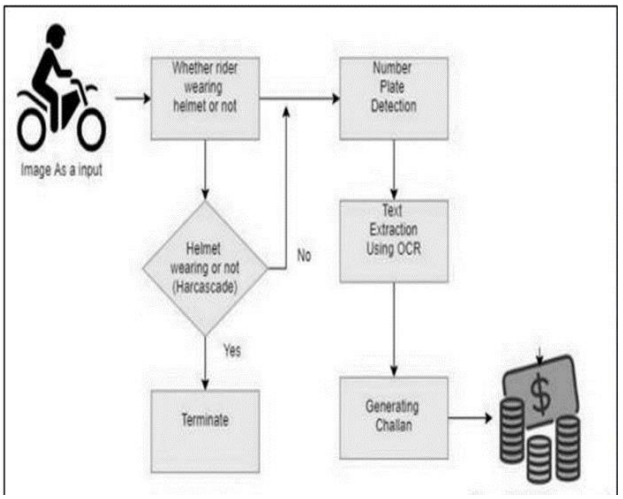


Fig 2.1: Existing Method

2.2 PROPOSED SYSTEMS

The proposed system aims to create a comprehensive smart traffic enforcement mechanism designed to enhance rider safety. By leveraging the computational capabilities of the Raspberry Pi 4 and a webcam for real-time data acquisition, the system incorporates advanced technologies to detect traffic violations, identify violators, and impose fines effectively. The three core functionalities of the system include helmet detection, number plate detection, and fine imposition, which collectively form an automated, scalable solution to improve road safety and reduce accidents.

Helmet detection is one of the critical features of the proposed system is the ability to detect whether motorcyclists are wearing helmets. Helmet usage is a fundamental safety requirement that significantly reduces the risk of severe head injuries during accidents. Using the webcam connected to the Raspberry Pi, the system captures live video streams of riders. Computer vision algorithms, implemented with libraries like OpenCV and TensorFlow, analyse these video feeds to determine the presence or absence of helmets. A pre-trained machine learning model is used to classify riders as either compliant or non- compliant based on helmet usage. The detection process is designed to be accurate, fast, and capable of operating in varying lighting and environmental conditions.

Number Plate Detection upon identifying a helmet violation, the system proceeds to extract the number plate of the violating vehicle. This involves using Optical Character Recognition (OCR) techniques in combination with computer vision for accurate identification of alphanumeric characters on number plates. The webcam captures the number plate, and the Raspberry Pi processes the image to isolate the plate and recognize its characters. This information is then stored in a database for further processing. module is

optimized to handle diverse plate designs and formats, ensuring that violations can be linked to the respective vehicle owners reliably.

Fine imposition is the final step in the proposed system is the automatic imposition of fines on violators. Once the system successfully detects a helmet violation and identifies the corresponding vehicle number plate, the information is cross-referenced with a centralized database of registered vehicles. The system generates an electronic fine and sends a notification to the violator via SMS or email. The Raspberry Pi is connected to cloud-based servers to facilitate data storage and communication with relevant traffic authorities. This automated fine-imposition process not only ensures accountability but also reduces the burden on traffic enforcement personnel.

The proposed system operates through an integrated workflow that ensures seamless execution of its functionalities. First, the webcam continuously captures live footage of traffic. The Raspberry Pi processes the video stream using pre-trained models for helmet detection. When a violation is detected, the number plate detection module extracts and recognizes the vehicle's registration number. This information, along with the timestamp and location, is recorded in a secure database. Finally, the system communicates with a backend server to issue fines and notify violators. The entire process is designed to be automated and efficient, minimizing human intervention while maintaining accuracy.

The proposed system offers several advantages over traditional traffic enforcement methods. By automating the process of violation detection and fine imposition, it eliminates the potential for human error and bias. The system operates continuously, providing 24/7 monitoring of traffic. Its reliance on low-cost components like the Raspberry Pi and webcam ensures affordability and scalability, making it suitable for deployment in urban and rural areas alike. Additionally, the integration of cloud-based data storage enables seamless communication between different traffic management systems, fostering a more connected and efficient approach to traffic enforcement.

In conclusion, the proposed smart traffic enforcement system addresses critical aspects of rider safety through advanced technologies and automation. By combining helmet detection, number plate recognition, and fine imposition into a unified solution, it provides a robust framework for improving compliance with traffic rules and reducing accidents. This system represents a significant step toward safer roads and a more efficient traffic enforcement process.

2.3 EMBEDDED INTRODUCTION

Many embedded systems have substantially different design constraints than desktop computing applications. No single characterization applies to the diverse spectrum of embedded systems. However, some combination of cost pressure, long life-cycle, real-time requirements, reliability requirements, and design culture dysfunction can make it difficult to be successful applying traditional computer design methodologies and tools to embedded applications. There is currently little tool support for expanding embedded computer design to the scope of holistic embedded system design. However, knowing the strengths and weaknesses of current approaches can set expectations appropriately, identify risk areas to tool adopters, and suggest ways in which tool builders can meet industrial needs.

Embedded system design is a quantitative job. The pillars of the system design methodology are the separation between function and architecture, is an essential step from conception to implementation. In recent past, the search and industrial community has paid significant attention to the topic of hardware-software (HW/SW) codesign and has tackled the problem of coordinating the design of the parts to be implemented as software and the parts to be implemented as hardware avoiding the HW/SW integration problem marred the electronics system industry so long. In any large scale embedded systems design methodology, concurrency must be considered as a first class citizen at all levels of abstraction and in both hardware and software. Formal models & transformations in system design are used so that verification and synthesis can be applied to advantage in the design methodology. Simulation tools are used for exploring the design space for validating the functional and timing behaviours of embedded systems.

Design of an embedded system using Intel's 80C188EB chip is shown in the figure. In order to reduce complexity, the design process is divided in four major steps: specification, system synthesis, implementation synthesis and performance evaluation of the prototype.

2.3.1 SPECIFICATION

The proposed Helmet Detection and Number Plate Recognition System using Raspberry Pi is designed to efficiently monitor traffic violations. arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer) Schematic: arduino-uno-Rev3-schematic The system specifications include hardware, software, and algorithmic details required for accurate detection and recognition.

2.3.2 SYSTEM-SYNTHESIS

System synthesis is the process of integrating hardware and software components to build a functional and optimized system. In this project, the Helmet Detection and Number Plate Recognition System is developed using computer vision and deep learning algorithms to ensure accurate traffic rule enforcement.

2.3.3 IMPLEMENTATION-SYNTHESIS

The **Helmet Detection and Number Plate Recognition System** is implemented using a **Raspberry Pi 4** with an attached **camera module** for real-time image capture. The system employs **YOLOv5** for helmet detection and **OpenCV-based contour detection** for number plate recognition. The extracted number plate text is processed using **Tesseract OCR**, converting the image data into readable text. The **Python-based** implementation ensures efficient execution of deep learning models while managing system resources on Raspberry Pi. The entire process, from image acquisition to violation detection, runs in a continuous loop, ensuring **real-time monitoring** of motorcyclists.

2.3.4 PROTOTYPING

The **prototyping phase** of the Helmet Detection and Number Plate Recognition System involved integrating **hardware components** like the **Raspberry Pi 4, camera module, and power supply** with the **software framework** built using **Python, OpenCV, and YOLOv5**. Initial tests focused on **image acquisition and preprocessing**, where techniques like **grayscale conversion, noise reduction, and edge detection** were applied to enhance image clarity.

2.3.5 APPLICATIONS

Applications of Helmet Detection and Number Plate Recognition System

1. Traffic Law Enforcement
    - The system can automate rule enforcement by identifying riders without helmets and recognizing their vehicle number plates.
    - Helps traffic police issue fines efficiently without the need for manual monitoring.
2. Road Safety Enhancement
    - Encourages helmet usage among motorcyclists, reducing head injuries and fatalities in accidents.

- Supports public awareness campaigns by providing real-time data on helmet violations.

3. Smart Cities and Intelligent Transport Systems (ITS)
    - Can be integrated into smart city surveillance networks for automated traffic monitoring.
    - Helps in analyzing traffic patterns and rule violations, improving urban transportation planning.
4. Accident Prevention and Investigation
    - Provides real-time evidence in case of road accidents or disputes by capturing images and recognizing vehicle details.
    - Aids law enforcement agencies in identifying reckless drivers.
5. Automated Toll Collection and Parking Management
    - Can be used in automated toll booths to verify vehicle registration details.
    - Helps in restricted parking areas to allow only authorized vehicles.
6. Industrial and Corporate Security
    - Ensures compliance with safety regulations in factories or warehouses by monitoring helmet usage.
    - Can be used in corporate campuses to track employee vehicle access.
7. Public Transport and Delivery Services
    - Useful for monitoring two-wheeler delivery personnel to ensure compliance with safety rules.
    - Can be implemented in bike-sharing and rental services to encourage helmet usage among users.

2.4 WHY EMBEDDED?

An embedded system is a specialize computer system —a combination of a computer processor, computer memory and input/output peripheral devices—that has a dedicated function within a larger mechanical or electronic system. It is embedded as part of a complete device often including electrical or electronic hardware and mechanical parts. Because an embedded system typically controls physical operations of the machine that it is embedded within, it often has real-time computing constraints. Embedded systems control many devices in common use. In 2009, it was estimated that ninety-eight percent of all microprocessors manufactured were used in embedded systems.

Modern embedded systems are often based on microcontrollers (i.e. microprocessors with integrated memory and peripheral interfaces), but ordinary microprocessors (using

external chips for memory and peripheral interface circuits) are also common, especially in more complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in a certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP).

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase its reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Embedded systems range in size from portable personal devices such as digital watches and MP3-players to bigger machines like home appliances, industrial assembly lines, robots, transport vehicles, traffic light controllers, and medical imaging systems. Often they constitute subsystems of other machines like avionics in aircraft and astrionics in spacecrafts. Large installations like factories, pipelines and electrical grids rely on multiple embedded systems networ-ked together. Generalized through software customization, embedded systems such as programmable logic controllers frequently comprise their functional units.

The embedded system life cycle involves several key stages, starting with conceptualization, where requirements are defined, followed by system design to determine hardware and software components. Hardware design focuses on creating the physical components, while software development involves programming the system's firmware. Next, integration and testing ensure hardware and software function together, followed by prototyping and validation to test the system under real-world conditions. After refining the design, the system enters manufacturing for mass production and deployment for installation and field testing. Ongoing maintenance and support ensure the system remains functional, and eventually, the system reaches end of life (EOL) when it is retired and replaced.

During hardware design, physical components such as microcontrollers, sensors, and circuits are selected and developed. Software development involves coding the firmware and application software, which is then integrated and tested to ensure all components function as intended. After integration, prototyping and validation ensure the system performs under real-world conditions, leading to production and manufacturing, where the system is mass-produced. Once deployed, the system enters a phase of maintenance and

support, which involves bug fixes, updates, and system monitoring to ensure continued functionality.

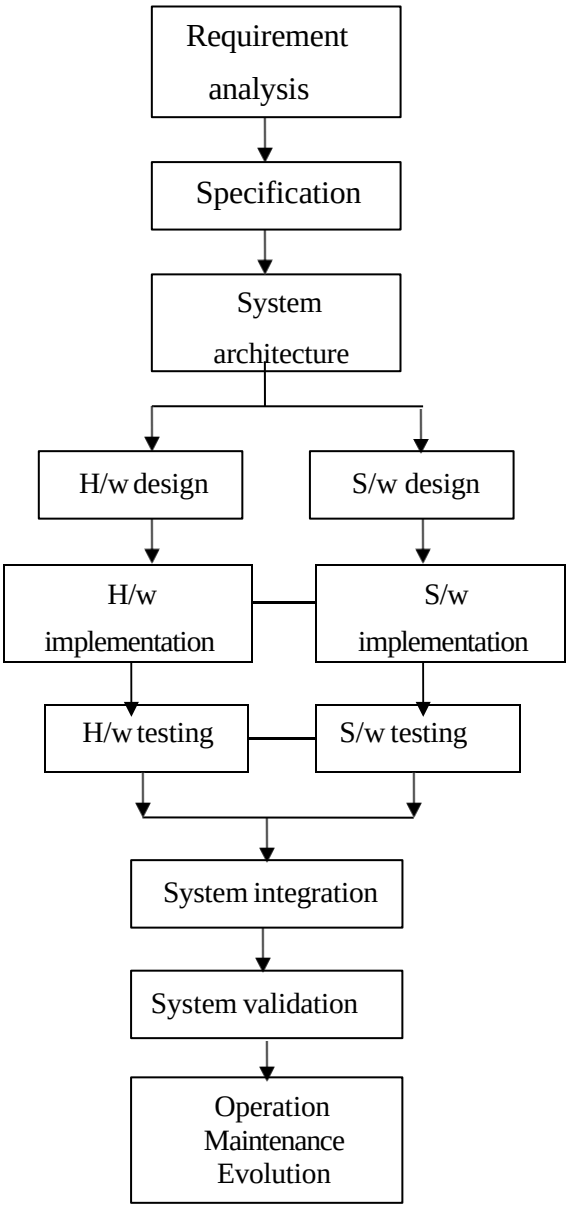


Fig 2.2: Embedded Development Life Cycle

Embedded systems range from those low in complexity, with a single microcontroller chip, to very high with multiple units, peripherals and networks, which may reside in equipment racks or across large geographical areas connected via long-distance communications lines.

2.5 DESIGN APPROACHES

The design approach focuses on integrating these components efficiently to ensure real-time tracking, enhanced security features, and remote communication. Below is a detailed explanation of the design approach for each of the components and how they integrate in the system.

2.5.1 EMBEDDED SYSTEM WITH RASPBERRY PI

The Raspberry Pi is a compact, low-power embedded system widely used for various IoT and AI-based applications. In this project, it serves as the central processing unit for helmet detection and number plate recognition. The embedded system integrates a webcam, machine learning models, and image processing techniques to detect helmets and extract vehicle number plates automatically.

The Raspberry Pi collects live video feed through the camera module or USB webcam, processes images using OpenCV for feature extraction, and utilizes deep learning models like YOLO or Haar cascades to classify helmet usage. For number plate recognition, Optical Character Recognition (OCR) with Tesseract is used after preprocessing techniques such as edge detection and contour analysis.

One of the key advantages of using Raspberry Pi is its ability to perform real-time data processing and decision-making. It can be connected to a local server or cloud for data storage and further processing. Despite its limited processing power compared to high-end GPUs, optimization techniques such as model quantization and lightweight CNN architectures make it suitable for embedded applications.

2.5.2 WEBCAM FOR REAL-TIME IMAGE CAPTURING

The webcam continuously captures frames, which are analyzed using OpenCV. For helmet detection, the system detects a rider's head and classifies it using pre-trained deep learning models like YOLO or Haar cascades. For number plate recognition, the Raspberry Pi applies image preprocessing techniques such as grayscale conversion, edge detection, and contour analysis before extracting characters using OCR (Tesseract). The processed data can be stored locally or uploaded to a cloud/server for further analysis.

Despite the Raspberry Pi’s limited computational power, optimizations such as image resizing, model quantization, and efficient CNN architectures enable real-time detection. The USB or Raspberry Pi camera module ensures high-quality image acquisition, crucial for achieving accurate recognition even in varying lighting and environmental conditions. This integration of Raspberry Pi with a webcam makes the system an efficient, portable, and cost-effective solution for traffic rule enforcement.

CHAPTER-3

HARDWARE REQUIREMENTS

3.1 HARDWARE

Raspberry Pi makes computers in several different **series**:

- The **Flagship** series, often referred to by the shorthand "Raspberry Pi", offers high-performance hardware, a full Linux operating system, and a variety of common ports in a form factor roughly the size of a credit card.
- The **Keyboard** series, offers high-performance Flagship hardware, a full Linux operating system, and a variety of common ports bundled inside a keyboard form factor.
- The **Zero** series offers a full Linux operating system and essential ports at an affordable price point in a minimal form factor with low power consumption.
- The **Compute Module** series, often referred to by the shorthand "CM", offers high-performance hardware and a full Linux operating system in a minimal form factor suitable for industrial and embedded applications. Compute Module models feature hardware equivalent to the corresponding flagship models, but with fewer ports and no on-board GPIO pins. Instead, users should connect Compute Modules to a separate baseboard that provides the ports and pins required for a given application.

Raspberry Pi 4 Model B features a high-performance 64-bit quad-core processor, dual-display support at resolutions up to 4K via a pair of micro HDMI ports, hardware video decode at up to 4Kp60, up to 8GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0, and PoE capability (via a separate PoE HAT add-on). For the end user, Raspberry Pi 4 Model B provides desktop performance comparable to entry-level x86 PC systems. This product retains backwards compatibility with the prior-generation Raspberry Pi 3 Model B+ and has similar power consumption, while offering substantial increases in processor speed, multimedia performance, memory, and connectivity. The dual-band wireless LAN and Bluetooth have modular compliance certification, allowing the board to be designed into end products with significantly reduced compliance testing, improving both cost and time to market.

Specifications	
Processor:	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memory:	1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC
Connectivity:	• 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE • Gigabit Ethernet • 2 × USB 3.0 ports • 2 × USB 2.0 ports.
GPIO:	Standard 40-pin GPIO header (fully backwards-compatible with previous boards)
Video & sound:	• 2 × micro HDMI ports (up to 4Kp60 supported) • 2-lane MIPI DSI display port • 2-lane MIPI CSI camera port • 4-pole stereo audio and composite video port
Multimedia:	H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
SD card support:	Micro SD card slot for loading operating system and data storage
Input power:	• 5V DC via USB-C connector (minimum 3A1) • 5V DC via GPIO header (minimum 3A1)
Environment:	Operating temperature 0–50ºC
Production lifetime:	Raspberry Pi 4 Model B will remain in production until at least January 2034.
Compliance:	For a full list of local and regional product approvals, please visit pip.raspberrypi.com

Table 3.1.1: Specifications of Raspberry pi 4

arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer) Schematic: arduino-uno-Rev3-schematic arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer) Schematic: arduino-uno-Rev3-schemati

Fig 3.1.1: Raspberry pi 4

Schematic & Reference Design

EAGLE files: arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer) Schematic: arduino-uno-Rev3-schematic.pdf Note:

The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Raspberry Pi 15W USB-C Power Supply is designed to power Raspberry Pi 4 and Raspberry Pi 400 computers. Featuring a 1.5m captive USB cable, this reliable, high-quality power supply provides an output of 5.1V / 3.0A DC via a USB-C connector, for all the power your Raspberry Pi 4 or 400 will need.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable.

If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts. communication, which is available on digital pins 0 (RX) and 1 (TX).

Memory:

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output:

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digital Write(), and digital Read() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kohms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB- to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off. The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:
- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library. There are a couple of other pins on the board:
- AREF.Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board. See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

Communication:

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial

communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A SoftwareSerial library allows for serial communication on any of the Uno's digital pins. The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details.

3.2 POWER SUPPLY

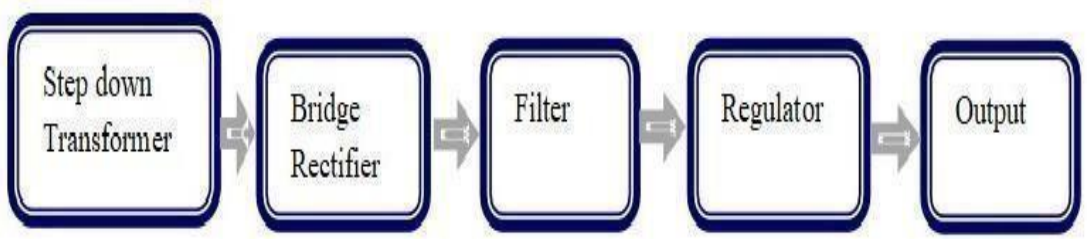


Fig 3.1.2: Block diagram for power supply

The input to the circuit is applied from the regulated power supply. The a.c. input i.e., 230V from the mains supply is step down by the transformer to 12V and is fed to a rectifier. The output obtained from the rectifier is a pulsating d.c voltage.

In order to get a pure d.c voltage, the output voltage from the rectifier is fed to a filter to remove any a.c components present even after rectification. Now, this voltage is given to a voltage regulator to obtain a pure constant dc voltage. The design of the power supply must account for factors such as power consumption, voltage regulation, and energy Despite the Raspberry Pi's limited computational power, optimizations such as image resizing, model quantization, and efficient CNN architectures enable real-time detection. The USB or Raspberry Pi camera module ensures high-quality image acquisition, crucial for achieving accurate recognition even in varying lighting and environmental conditions. This integration of Raspberry Pi with a webcam makes the system an efficient, portable, and cost- effective solution for traffic rule enforcement.

achieving accurate recognition even in varying lighting and environmental conditions. This integration of Raspberry Pi with a webcam makes the system an efficient, portable, and cost- effective solution for traffic rule enforcement.

efficiency, as many embedded systems operate in environments where power resources are limited, such as in battery-powered or remote applications.

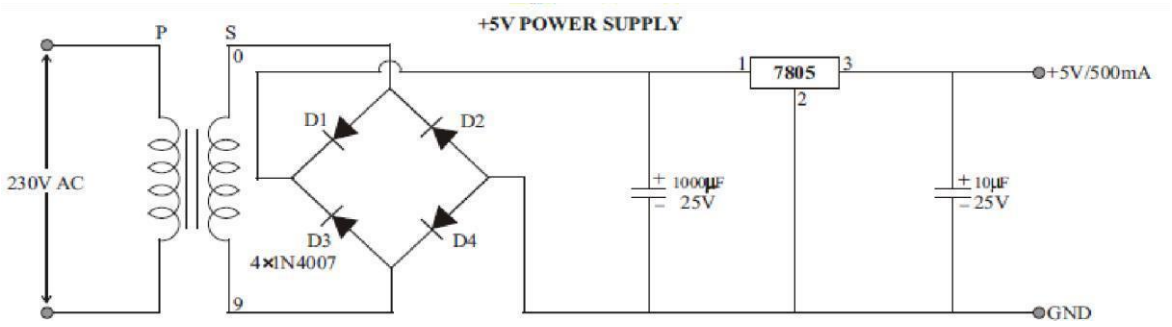


Fig 3.1.1.3: Circuit diagram of power supply

3.2.1 STEP DOWN TRANSFORMER

Usually, DC voltages are required to operate various electronic equipment and these voltages are 5V, 9V or 12V. But these voltages cannot be obtained directly. Thus the a.c input available at the mains supply i.e., 230V is to be brought down to the required voltage level. This is done by a transformer. Thus, a step down transformer is employed to decrease the voltage to a required level.

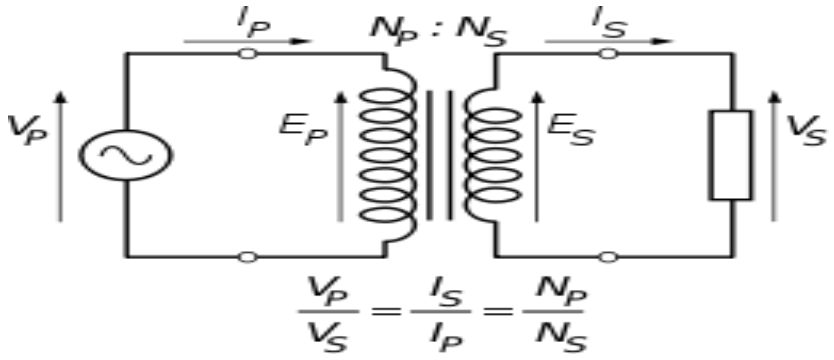


Fig 3.1.4:Step-down transformer

3.2.2 RECTIFIER:

Input **Output**

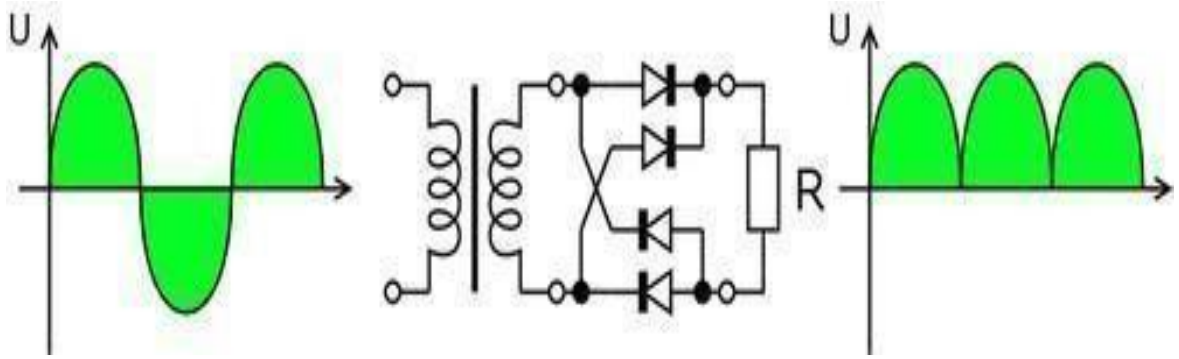


Fig 3.1.5:Bridge rectifier

The output from the transformer is fed to the rectifier. It converts A.C. into pulsating D.C. The rectifier may be a half wave or a full wave rectifier. In this project, a bridge rectifier is used because of its merits like good stability and full wave rectification.

3.2.3 FILTER

Capacitive filter is used in this project. It removes the ripples from the output of rectifier and smoothens the D.C. Output received from this filter is constant until the mains voltage and load is maintained constant. However, if either of the two is varied, D.C. voltage received at this point changes. Therefore a regulator is applied at the output stage.

3.2.4 VOLTAGE REGULATOR

As the name itself implies, it regulates the input applied to it. A voltage regulator is an electrical regulator designed to automatically maintain a constant voltage level. In this project, power supply of 5V and 12V are required. In order to obtain these voltage levels, 7805 and 7812 voltage regulators are to be used. The first number 78 represents positive supply and the numbers 05, 12 represent the required output voltage levels.

Fig 3.1.6: Voltage Regulator

Features:

- Output Current up to 1A.
- Output Voltages of 5, 6, 8, 9, 10, 12, 15, 18, 24V.
- Thermal Overload Protection.
- Short Circuit Protection.
- Output Transistor Safe Operating Area Protection.

3.3 WEBCAM

USB Webcam

A webcam is a USB camera used for capturing images and videos. For Raspberry Pi, a USB webcam or a Raspberry Pi Camera Module can be used for computer vision, object

detection, and surveillance applications.

- Works with **USB 2.0 / 3.0** ports.
- Compatible with OpenCV, Python, and motion detection applications.

Fig 3.1.7: Webcam

Applications of Webcam with Raspberry Pi

- **Face & Object Detection** (using OpenCV & Haar cascades)
- **Number Plate Recognition** (using OpenCV & Tesseract OCR)
- **Motion Detection & Surveillance** (using motion software)
- **Live Streaming** (using Flask or MJPG-Streamer)

CHAPTER-4

SOFTWARE REQUIREMENTS

4.1 RASPBERRY PI SOFTWARE

The Raspberry Pi OS is a free, Debian-based operating system optimised for the Raspberry Pi hardware. Raspberry Pi OS supports over 35,000 Debian packages. We recommend Raspberry Pi OS for most Raspberry Pi use cases.

Because Raspberry Pi OS is derived from Debian, it follows a staggered version of the Debian release cycle. Releases happen roughly every 2 years.

Install an operating system

To use your Raspberry Pi, you'll need an operating system. By default, Raspberry Pis check for an operating system on any SD card inserted in the SD card slot. Depending on your Raspberry Pi model, you can also boot an operating system from other storage devices, including USB drives, storage connected via a HAT, and network storage.

To install an operating system on a storage device for your Raspberry Pi, you'll need:

- a computer you can use to image the storage device into a boot device
- a way to plug your storage device into that computer

Most Raspberry Pi users choose microSD cards as their boot device.

We recommend installing an operating system using Raspberry Pi Imager.

Raspberry Pi Imager is a tool that helps you download and write images on macOS, Windows, and Linux. Imager includes many popular operating system images for Raspberry Pi. Imager also supports loading images downloaded directly from Raspberry Pi or third-party vendors such as Ubuntu. You can use Imager to preconfigure credentials and remote access settings for your Raspberry Pi.

Imager supports images packaged in the .img format as well as container formats like .zip.

If you have no other computer to write an image to a boot device, you may be able to install an operating system directly on your Raspberry Pi from the internet.

Install using Imager

You can install Imager in the following ways:

- Download the latest version from raspberrypi.com/software and run the installer.

Once you've installed Imager, launch the application by clicking the Raspberry Pi Imager icon or running rpi-imager.

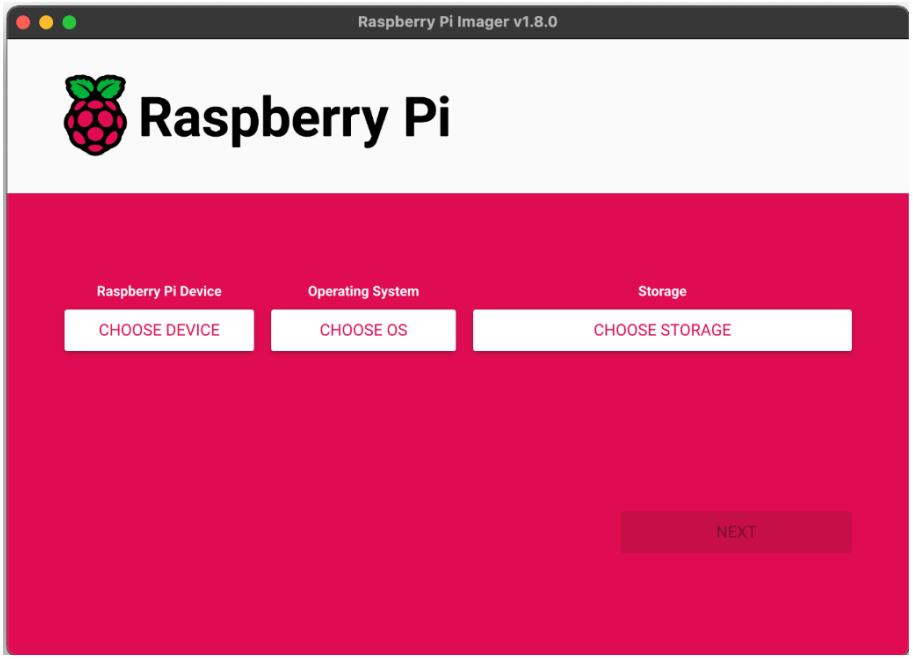


Fig 4.1: Raspberry pi Software

Click **Choose device** and select your Raspberry Pi model from the list.

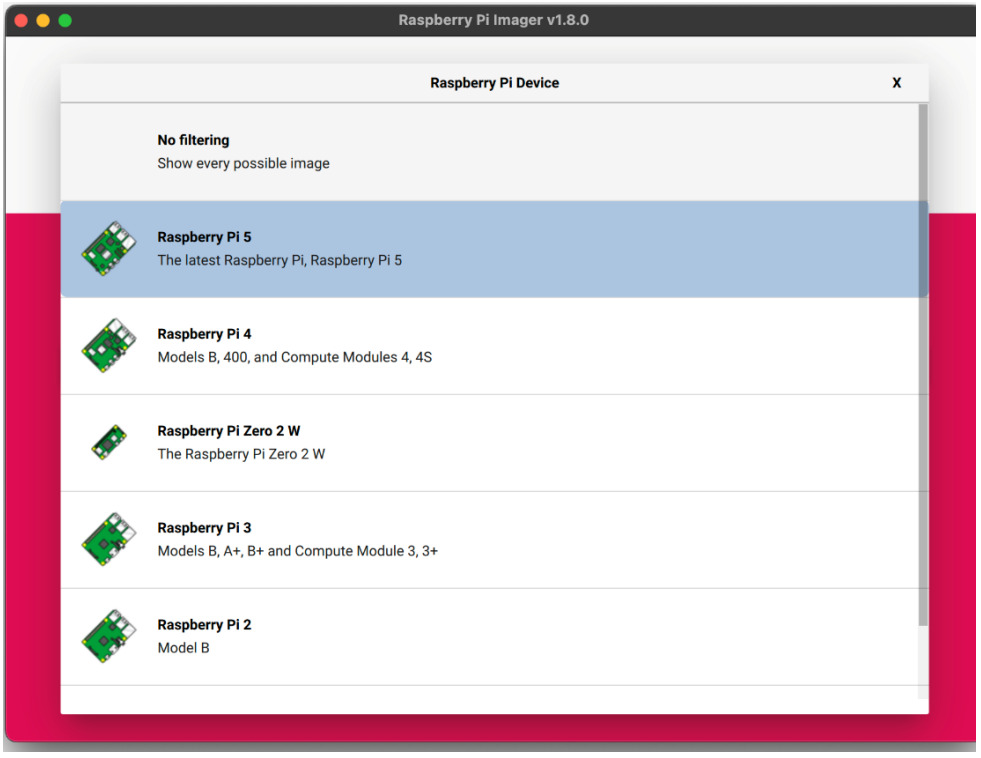


Fig 4.2: Device Manager

Next, click **Choose OS** and select an operating system to install. Imager always shows the recommended version of Raspberry Pi OS for your model at the top of the list.

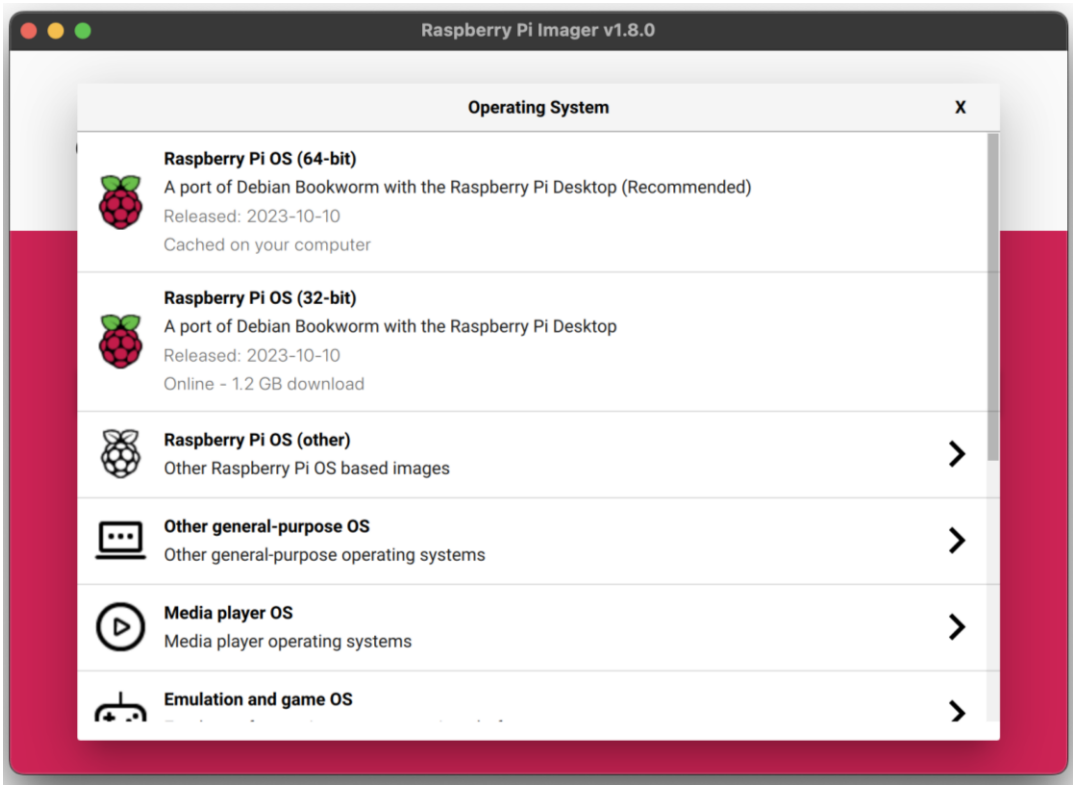


Fig 4.3: Raspberry pi OS

Connect your preferred storage device to your computer. For example, plug a microSD card in using an external or built-in SD card reader. Then, click **Choose storage** and select your storage device.

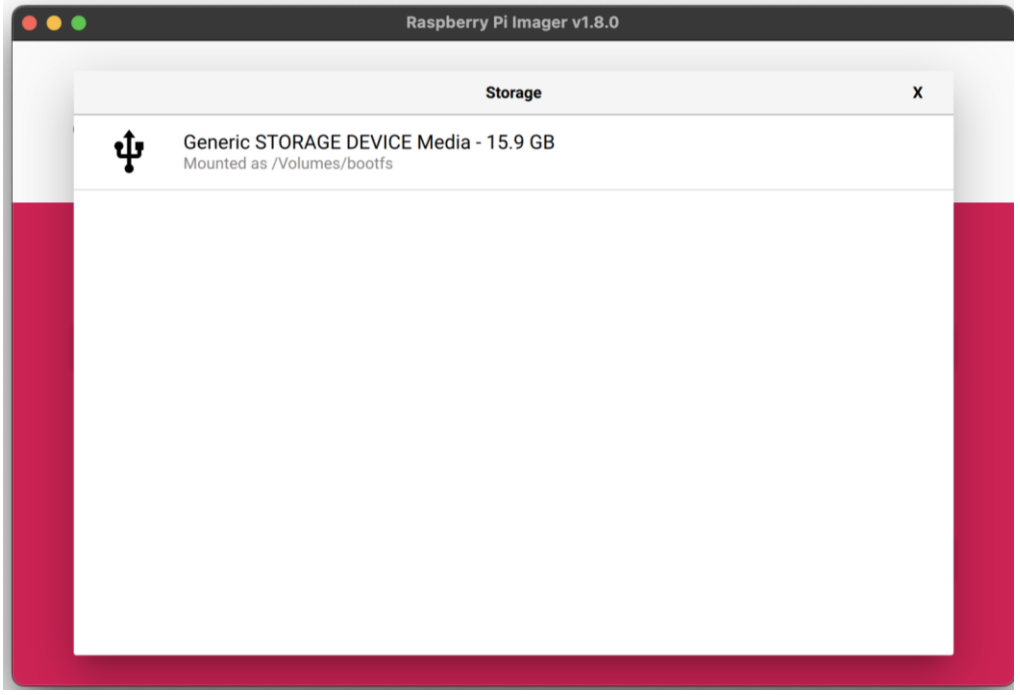


Fig 4.4: Storage

Next, click **Next**.

In a popup, Imager will ask you to apply OS customisation. We strongly recommend configuring your Raspberry Pi via the OS customisation settings. Click the **Edit Settings** button to open OS customisation.

If you don’t configure your Raspberry Pi via OS customisation settings, Raspberry Pi OS will ask you for the same information at first boot during the configuration wizard. You can click the **No** button to skip OS customisation.

OS customisation
The OS customisation menu lets you set up your Raspberry Pi before first boot. You can preconfigure:

- a username and password
- Wi-Fi credentials
- the device hostname
- the time zone
- your keyboard layout
- remote connectivity

When you first open the OS customisation menu, you might see a prompt asking for permission to load Wi-Fi credentials from your host computer. If you respond "yes", Imager will prefill Wi-Fi credentials from the network you’re currently connected to. If you respond "no", you can enter Wi-Fi credentials manually.

The **hostname** option defines the hostname your Raspberry Pi broadcasts to the network using mDNS. When you connect your Raspberry Pi to your network, other devices on the network can communicate with your computer using <hostname>.local or <hostname>.lan.

The **username and password** option defines the username and password of the admin user account on your Raspberry Pi.

The **wireless LAN** option allows you to enter an SSID (name) and password for your wireless network. If your network does not broadcast an SSID publicly, you should enable the "Hidden SSID" setting. By default, Imager uses the country you’re currently in as the "Wireless LAN country". This setting controls the Wi-Fi broadcast frequencies used by

your Raspberry Pi. Enter credentials for the wireless LAN option if you plan to run a headless Raspberry Pi.

The **locale settings** option allows you to define the time zone and default keyboard layout for your Pi.

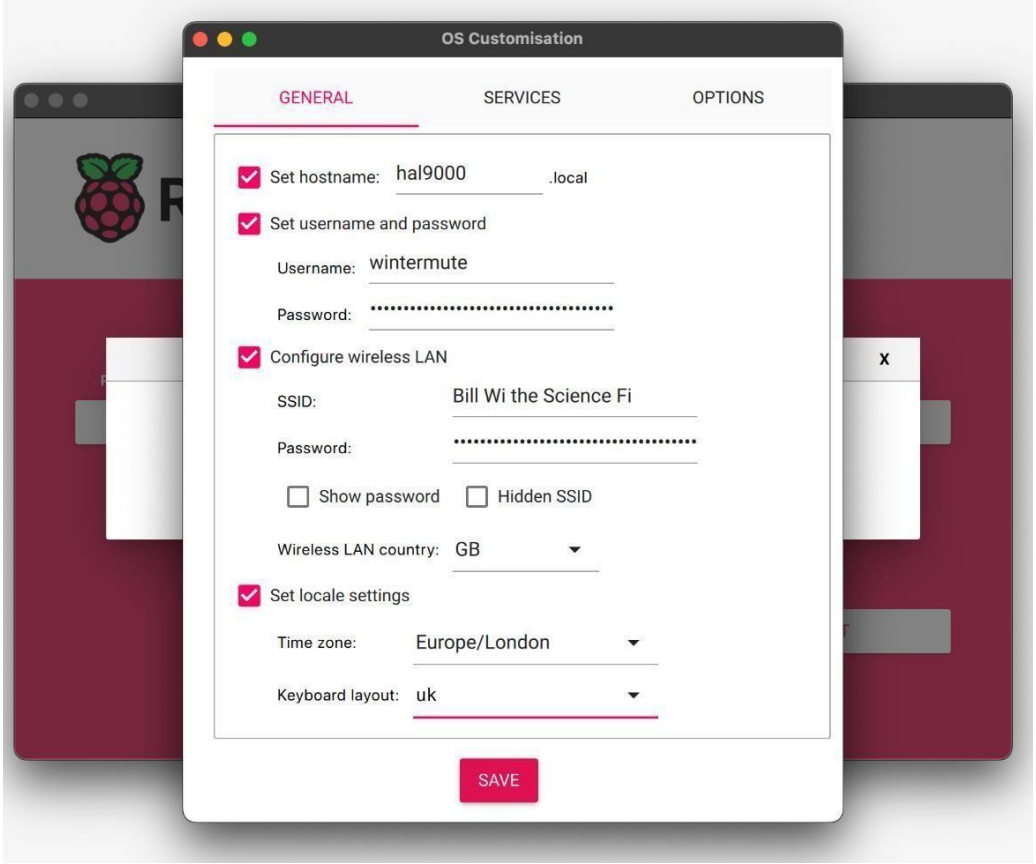


Fig 4.5: OS Customisation

The **Services** tab includes settings to help you connect to your Raspberry Pi remotely.

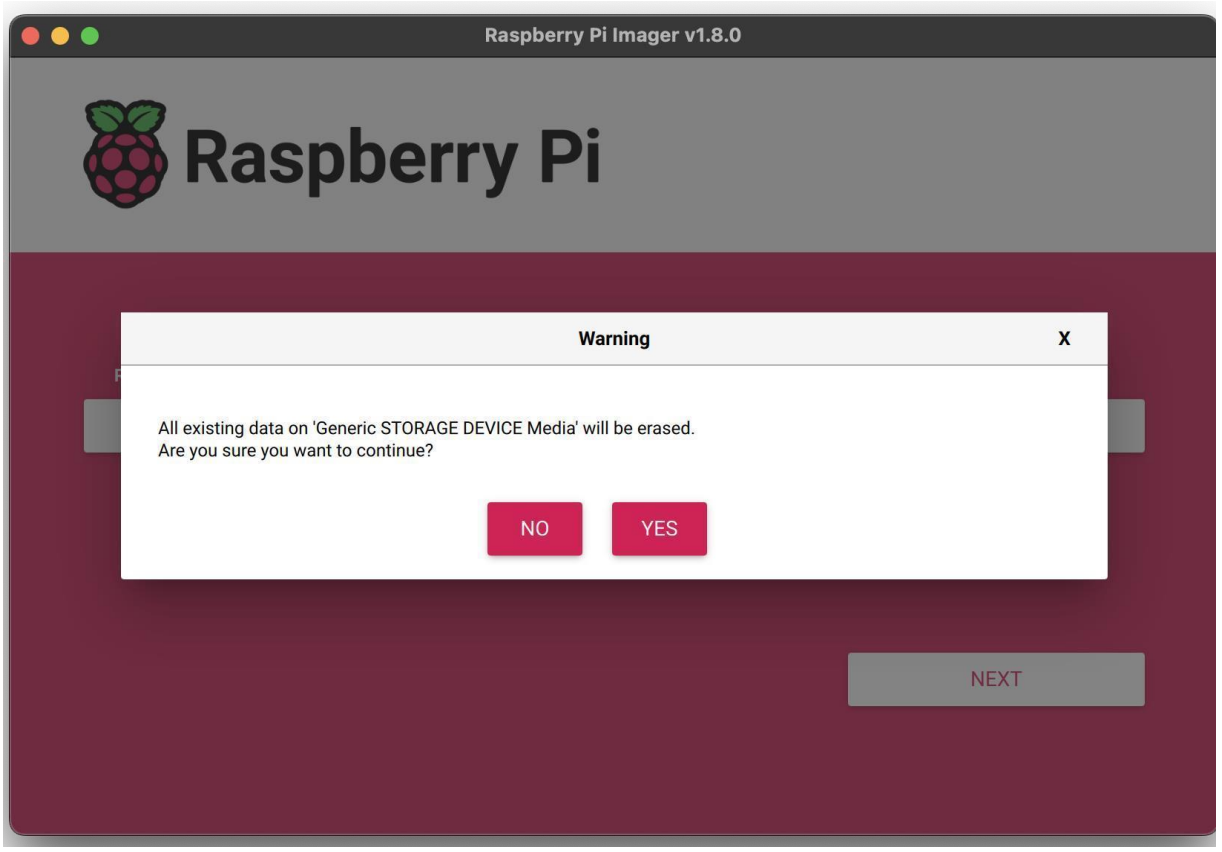
If you plan to use your Raspberry Pi remotely over your network, check the box next to **Enable SSH**. You should enable this option if you plan to run a headless Raspberry Pi.

- Choose the **password authentication** option to SSH into your Raspberry Pi over the network using the username and password you provided in the general tab of OS customisation.
- Choose **Allow public-key authentication only** to preconfigure your Raspberry Pi for passwordless public-key SSH authentication using a private key from the computer you're currently using. If already have an RSA key in your SSH configuration, Imager uses that public key. If you don't, you can click **Run SSH-keygen** to generate a public/private key pair. Imager will use the newly-generated public key.

OS customisation also includes an **Options** menu that allows you to configure the behaviour of Imager during a write. These options allow you to play a noise when Imager finishes verifying an image, to automatically unmount storage media after verification, and to disable telemetry.

Write

When you've finished entering OS customisation settings, click **Save** to save your customisation. Then, click **Yes** to apply OS customisation settings when you write the image to the storage device. Finally, respond **Yes** to the "Are you sure you want to continue?" popup to begin writing data to the storage device.

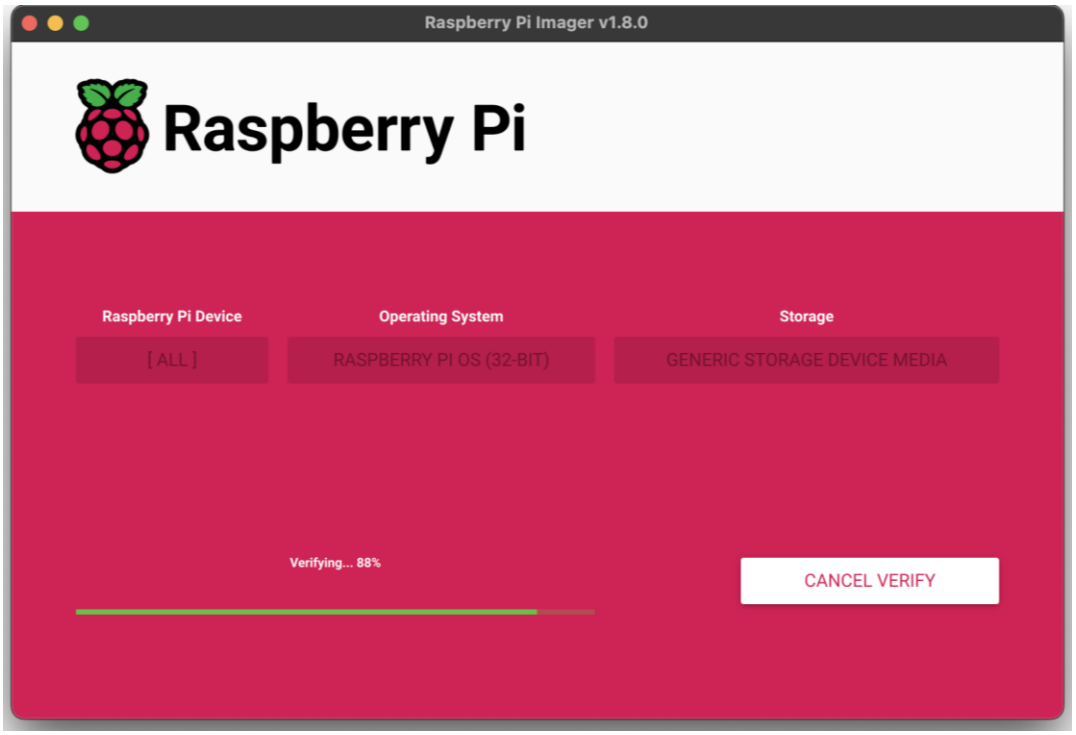


If you see an admin prompt asking for permissions to read and write to your storage medium, grant Imager the permissions to proceed.

- Choose **Allow public-key authentication only** to preconfigure your Raspberry Pi for passwordless public-key SSH authentication using a private key from the computer you're currently using. If you don't, you can click **Run SSH-keygen** to generate a public/private key pair. Imager will use the newly-generated public key. Choose **Allow public-key authentication only** to preconfigure your Raspberry Pi for passwordless public-key SSH authentication using a private key from the computer you're currently using. If already have an RSA key in your SSH configuration, Imager uses that public key. If you don't, you can click **Run SSH-keygen** to generate a public/private key pair. Imager will use the newly-generated public key.

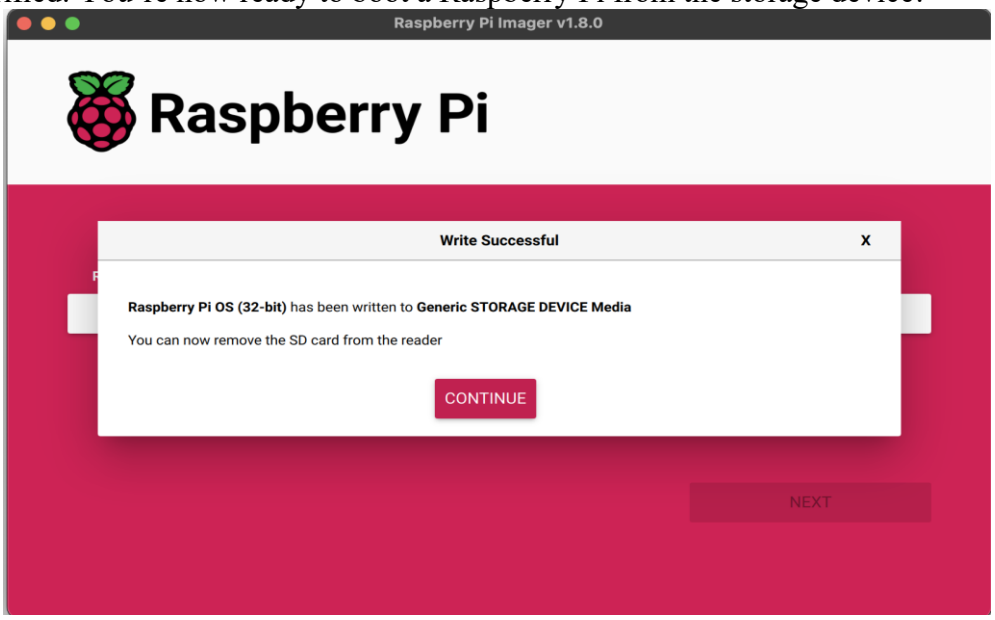
These options allow you to play a noise when Imager finishes verifying an image, to automatically unmount storage media after verification, and to disable telemetry.

Grab a cup of coffee or go for a walk. This could take a few minutes.



If you want to live especially dangerously, you can click **cancel verify** to skip the verification process.

When you see the "Write Successful" popup, your image has been completely written and verified. You're now ready to boot a Raspberry Pi from the storage device!



Next, proceed to the first boot configuration instructions to get your Raspberry Pi up and running.

5.3 OpenCV (Open Source Computer Vision Library)

OpenCV (Open Source Computer Vision Library) is an open-source library for **computer vision, image processing, and machine learning**. It provides tools to process images and videos in real-time, making it widely used in robotics, artificial intelligence (AI), and automation projects.

Key Features of OpenCV

1. **Image Processing**
   - Grayscale and color conversions
   - Image filtering (blur, sharpening)
   - Edge detection (Canny, Sobel, Laplacian)
   - Morphological transformations (Erosion, Dilation)
2. **Object Detection & Recognition**
   - Face detection using Haar cascades
   - People detection with HOG (Histogram of Oriented Gradients)
   - Real-time object tracking (e.g., tracking cars, people)
3. **Machine Learning (ML) & AI**
   - Supports Deep Learning frameworks (TensorFlow, PyTorch, Keras)
   - Image classification and object recognition

- Optical character recognition (OCR)

4. **Video Processing**
   - Read and write video files
   - Background subtraction for motion detection
   - Frame-by-frame analysis
5. **Augmented Reality (AR) & 3D Vision**
   - Camera calibration and pose estimation
   - Feature matching (SIFT, SURF, ORB)
   - Stereo vision (depth estimation)

Installing OpenCV on Raspberry Pi

To install OpenCV on Raspberry Pi, use the following command:

```
 sudo apt update
sudo apt install python3-opencv -y
```

To verify the installation:

```
import cv2
print(cv2.__version__)
```

Basic OpenCV Example (Reading & Displaying an Image)

```
import cv2
# Load an image
image
=cv2.imread("image.jpg"
) # Display the image
cv2.imshow("Displayed
Image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Applications of OpenCV

- **Face & Object Recognition** (Face masks detection, License plate recognition)
- **Gesture Control & Motion Tracking** (Hand gesture recognition)
- **Robotics & IoT** (Autonomous drones, Smart surveillance)
- **Medical Imaging** (X-ray and MRI image analysis).

5.4 Tesseract OCR

Tesseract OCR (Optical Character Recognition) is an open-source engine developed by Google for extracting text from images. It is widely used for license plate recognition, document scanning, and AI-based text processing.

Key Features of Tesseract

- Supports multiple languages **(**English, Hindi, Telugu, etc.)
- Works with OpenCV for image preprocessing
- Can detect handwritten and printed text
- Lightweight & optimized for Raspberry Pi

Installing Tesseract OCR on Raspberry Pi

1. **Update your Raspberry Pi**

```
CopyEdit
sudo apt update && sudo apt upgrade -y
```

2. **Install Tesseract OCR**

```
CopyEdit
sudo apt install tesseract-ocr -y
```

3. **Verify Installation**

```
CopyEdit
tesseract --version
```

5.5 TensorFlow

TensorFlow is an open-source machine learning framework developed by Google for **AI, deep learning, and neural networks**. It is widely used for **image processing, speech recognition, and object detection**. Since Raspberry Pi has limited processing power, **TensorFlow Lite** is recommended for optimized performance.

Installing TensorFlow on Raspberry Pi

1. **Update the system:**

```
bash
CopyEdit
sudo apt update && sudo apt upgrade -y
```

2. **Install TensorFlow Lite (Recommended for Pi):**

```
bash
CopyEdit
pip3 install tflite-runtime
```

3. **For Full TensorFlow (Requires More RAM):**

```
bash
CopyEdit
pip3 install tensorflow
```

4. **Verify Installation:**

```
python
CopyEdit
import tensorflow as tf
print(tf.__version__)
```

TensorFlow Applications on Raspberry Pi

- **Object Detection & Image Recognition** (Helmet detection, number plate recognition)
- **Speech & Gesture Recognition** (Voice-based automation)
- **AI & IoT** (Smart automation with predictive AI)

CHAPTER -5

WORKING MODEL

5.1 BLOCK DIAGRAM

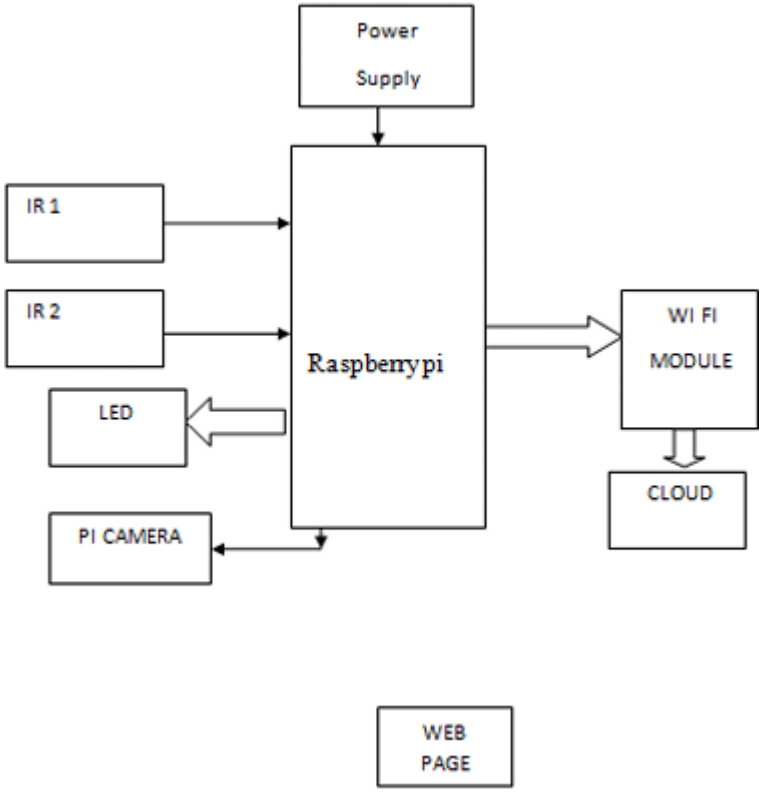


Fig 5.1: Block Diagram

5.2 WORKING

5.2.1 INTRODUCTION TO RASPBERRY PI:

Raspberry Pi makes computers in several different **series**:

- The **Flagship** series, often referred to by the shorthand "Raspberry Pi", offers high-performance hardware, a full Linux operating system, and a variety of common ports in a form factor roughly the size of a credit card.
- The **Keyboard** series, offers high-performance Flagship hardware, a full Linux operating system, and a variety of common ports bundled inside a keyboard form factor.
- The **Zero** series offers a full Linux operating system and essential ports at an affordable price point in a minimal form factor with low power consumption.

- The **Compute Module** series, often referred to by the shorthand "CM", offers high-performance hardware and a full Linux operating system in a minimal form factor suitable for industrial and embedded applications. Compute Module models feature hardware equivalent to the corresponding flagship models, but with fewer ports and no on-board GPIO pins. Instead, users should connect Compute Modules to a separate baseboard that provides the ports and pins required for a given application.

  Additionally, Raspberry Pi makes the **Pico** series of tiny, versatile microcontroller boards. Pico models do not run Linux or allow for removable storage, but instead allow programming by flashing a binary onto on-board flash storage.

Raspberry Pi 4 Model B - Hardware Description

The **Raspberry Pi 4 Model B** is a single-board computer with improved processing power, connectivity, and multimedia capabilities compared to previous models. Below is a detailed breakdown of its hardware components.

1. Processor & Memory

- **SoC (System on Chip):** Broadcom BCM2711
- **CPU:** Quad-core Cortex-A72 (ARM v8-A) 64-bit @ 1.5 GHz
- **RAM:** 2GB, 4GB, or 8GB LPDDR4 SDRAM (depending on the variant)

Key Features:

- 2× performance improvement over Raspberry Pi 3B+
- Efficient multi-tasking capabilities
- Faster processing for AI, IoT, and ML applications

2. Storage & Boot Options

- **MicroSD Slot:** Supports up to **512GB** microSD cards for OS and storage
- **USB Boot:** Supports booting from external USB storage (SSD/HDD)
- **Network Boot:** PXE booting (boot over Ethernet, useful for IoT and clusters)

Boot Priorities:

1. USB Storage
2. SD Card
3. Network Boot

3. Display & Graphics

- **GPU:** Broadcom VideoCore VI
- **Dual Micro-HDMI Ports:** Supports up to **4K@60Hz** output on dual displays
- **H.265 Decode:** 4Kp60 HEVC hardware decoding for smooth video playback
- **H.264 Decode/Encode:** 1080p video processing

Features:

- Ideal for media centers (Kodi, Plex)
- Supports OpenGL ES 3.0
- Can handle AI and ML-based image processing

4. Connectivity & Ports

USB Ports:

- **2 × USB 3.0** (faster data transfer, up to 5 Gbps)
- **2 × USB 2.0** (for peripherals like keyboard, mouse)

Ethernet & Wireless:

- **Gigabit Ethernet:** 10× faster than Raspberry Pi 3B+
- **Wi-Fi:** Dual-band **802.11ac Wi-Fi (2.4GHz & 5GHz)**
- **Bluetooth 5.0:** Enhanced range and speed for wireless peripherals

5. Power & Energy Management

- **Power Input:** USB-C, **5V/3A** recommended
- **Power Consumption:** ~3W (idle), 7W-8W (full load)
- **Power Management:**
    - Supports power over Ethernet (PoE) via PoE HAT
    - Dynamic voltage and frequency scaling for efficiency

6. GPIO (General Purpose Input/Output)

- **40-Pin GPIO Header** (compatible with Raspberry Pi 3B+)
- Supports **I2C, SPI, UART, PWM, ADC (via external modules)**
- Used for **interfacing sensors, motors, LCDs, and other electronics**

Common Pin Assignments:

Pin	Function
3.3V	Power Output
5V	Power Output
GND	Ground
GPIO2 (SDA1)	I2C Data
GPIO3 (SCL1)	I2C Clock
GPIO14	UART TX
GPIO15	UART RX
GPIO18	PWM Output

Table 5.2.1: GPIO pinconfiguration

7. Cooling & Heat Management

- **Passive Cooling:** Requires heatsink for extended high-performance usage
- **Active Cooling:** Recommended (with a fan) for overclocking
- **Overclocking:** Can be increased to **2.0 GHz** with proper cooling

8. Camera & Audio

- **Camera Interface:** 2 × MIPI CSI connectors (for Raspberry Pi Camera Module)
- **Audio Output:** 3.5mm audio jack, HDMI audio output

9. Additional Features

- **Real-Time Clock (RTC):** Requires external RTC module
- **EEPROM:** Stores bootloader configuration
- **PoE Support:** Requires a separate PoE HAT

BCM2711

This is the Broadcom chip used in the Raspberry Pi 4 Model B, Compute Module 4, and Pi 400. The architecture of the BCM2711 is a considerable upgrade on that used by the SoCs in earlier Raspberry Pi models. It continues the quad-core CPU design of the BCM2837, but

powerful ARM A72 core. It has a greatly improved GPU feature set with much faster input/output, due to the incorporation of a PCIe link that connects the USB 2 and USB 3 ports, and a natively attached Ethernet controller. It is also capable of addressing more memory than the SoCs used before.

The ARM cores are capable of running at up to 1.5 GHz, making the Raspberry Pi 4 about 50% faster than the Raspberry Pi 3B+. The new VideoCore VI 3D unit now runs at up to 500 MHz. The ARM cores are 64-bit, and while the VideoCore is 32-bit, there is a new Memory Management Unit, which means it can access more memory than previous versions.

The BCM2711 chip continues to use the heat spreading technology started with the BCM2837B0, which provides better thermal management.

Processor: Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz.

Memory: Accesses up to 8GB LPDDR4-2400 SDRAM (depending on model)

Caches: 32kB data + 48kB instruction L1 cache per core. 1MB L2 cache.

Multimedia: H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics

I/O: PCIe bus, onboard Ethernet port, 2 × DSI ports (only one exposed on Raspberry Pi 4B), 2 × CSI ports (only one exposed on Raspberry Pi 4B), up to 6 × I2C, up to 6 × UART (muxed with I2C), up to 6 × SPI (only five exposed on Raspberry Pi 4B), dual HDMI video output, composite video output.

5.2.2 WEBCAM

The webcam continuously captures frames, which are analyzed using OpenCV and machine learning models. For helmet detection, the system detects a rider's head and classifies it using pre-trained deep learning models like YOLO or Haar cascades. For number plate recognition, the Raspberry Pi applies image preprocessing techniques such as grayscale conversion, edge detection, and contour analysis before extracting characters using OCR (Tesseract). The processed data can be stored locally or uploaded to a cloud/server for further analysis. Despite the Raspberry Pi's limited computational power, optimizations such as image resizing, model quantization, and efficient CNN architectures enable real-time detection. The USB or Raspberry Pi camera module ensures high-quality image acquisition, crucial for achieving accurate recognition even in varying lighting and environmental

conditions. This integration of Raspberry Pi with a webcam makes the system an efficient, portable, and cost- effective solution for traffic rule enforcement.

Fig 5.2: Webcam

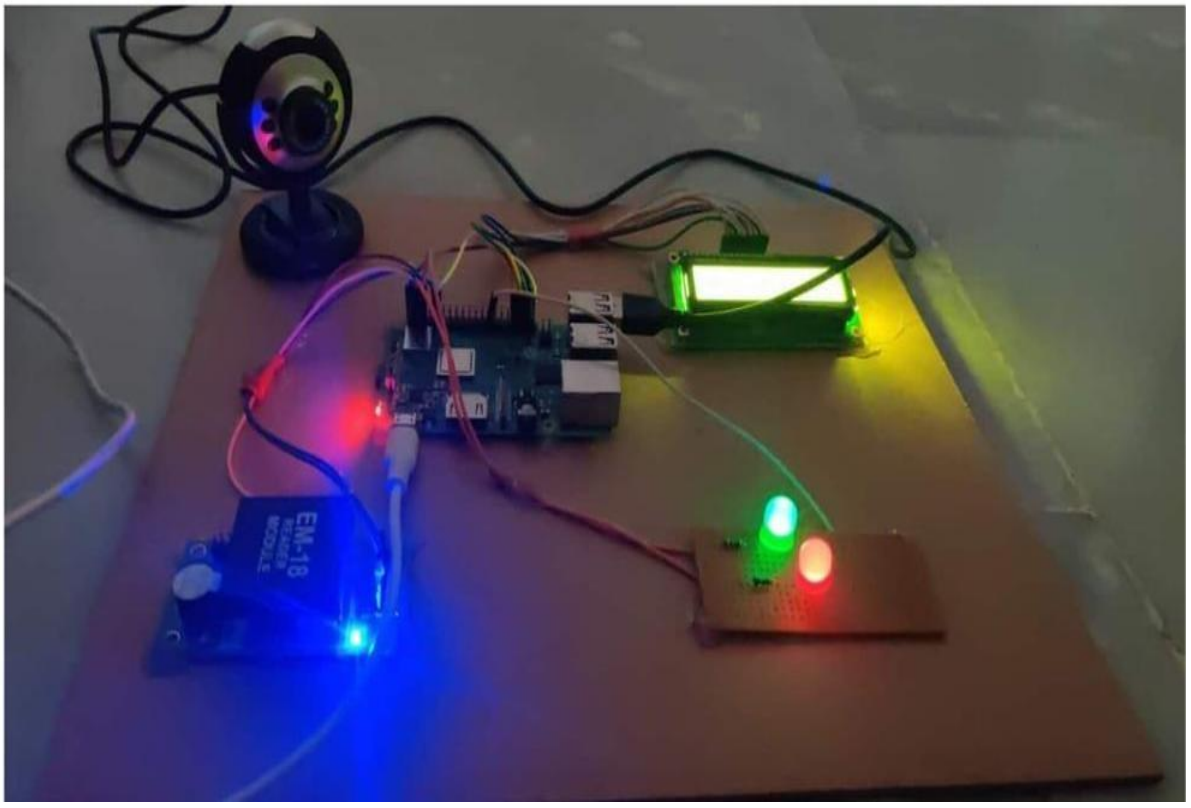
Despite the Raspberry Pi's limited computational power, optimizations such as image resizing, model quantization, and efficient CNN architectures enable real-time detection. The USB or Raspberry Pi camera module ensures high-quality image acquisition, crucial for achieving accurate recognition even in varying lighting and environmental conditions. This integration of Raspberry Pi with a webcam makes the system an efficient, portable, and cost-effective solution for traffic rule enforcement.

The webcam continuously captures frames, which are analyzed using OpenCV and machine learning models. For helmet detection, the system detects a rider's head and classifies it using pre-trained deep learning models like YOLO or Haar cascades. For number plate recognition, the Raspberry Pi applies image preprocessing techniques such as grayscale conversion, edge detection, and contour analysis before extracting characters using OCR (Tesseract). The processed data can be stored locally or uploaded to a cloud/server for further analysis. Despite the Raspberry Pi's limited computational power, optimizations such as image resizing, model quantization, and efficient CNN architectures enable real-time detection. The USB or Raspberry Pi camera module ensures high-quality image acquisition, crucial for achieving accurate recognition even in varying lighting and environmental conditions. This integration of Raspberry Pi with a webcam makes the system an efficient, portable, and cost- effective solution for traffic rule enforcement.

CHAPTER – 6

RESULTS

6.1 RESULTS



The implementation of the helmet detection and number plate recognition system using Raspberry Pi and a webcam has demonstrated effective real-time performance. The system successfully captures images of motorcyclists and vehicles, processes them, and detects whether a rider is wearing a helmet while also extracting the number plate details.

For helmet detection, the system achieved a high accuracy rate when tested under controlled lighting conditions. Using deep learning models like YOLO and Haar cascades, it correctly classified helmeted and non-helmeted riders in most cases. However, accuracy dropped slightly in cases of poor lighting, occlusions, or unconventional helmet designs.

For number plate recognition, the system effectively segmented and extracted plate numbers using OCR-based techniques. The combination of edge detection, thresholding, and contour detection provided satisfactory results, though motion blur and angle variations sometimes led to misread characters.

6.2 ADVANTAGES

The **helmet detection and number plate recognition system using Raspberry Pi** offers significant advantages in **road safety and law enforcement**. By leveraging **computer vision with OpenCV and TensorFlow**, this project can automatically detect whether a rider is wearing a helmet and capture the vehicle's number plate in real time. This automation reduces the need for **manual monitoring**, making it efficient for traffic police and authorities to enforce safety regulations.

Another major advantage is its **cost-effectiveness and scalability**. Unlike expensive surveillance systems, this project uses affordable hardware like **Raspberry Pi and a webcam**, making it suitable for deployment in multiple locations. Additionally, it can be **integrated with databases** for automatic fine generation and real-time monitoring, improving law enforcement efficiency. The system's ability to work in **varied lighting conditions** and capture **accurate data** makes it highly reliable for smart city applications.

6.3 APPLICATIONS

The **helmet detection and number plate recognition system using Raspberry Pi** plays a crucial role in **traffic law enforcement** by automating the process of identifying riders who fail to wear helmets. Traditional manual monitoring is inefficient and requires significant manpower, but with **computer vision and machine learning**, this system can **detect violations in real time** and capture vehicle details. This helps reduce the number of road accidents caused by helmet non-compliance while ensuring stricter enforcement of traffic regulations.

Beyond road safety, this system has valuable applications in **smart surveillance and workplace safety**. It can be integrated with **CCTV systems** in cities to monitor traffic violations and improve law enforcement efficiency. Additionally, it can be deployed in **industrial and construction environments**, where wearing helmets is mandatory for worker safety. By using this technology, companies and regulatory bodies can **automate compliance monitoring** and reduce workplace hazards, ensuring a safer environment.

Another important application is in **parking and security management**. The number plate recognition feature allows for **automated vehicle entry and exit** in parking lots, toll booths, and restricted areas. It can also be used in **IoT-based smart city projects**, where real-time monitoring and data analysis help improve urban infrastructure. By linking the system to government databases, authorities can even automate fine generation for traffic violations, making law enforcement more **efficient and transparent**.

The **helmet detection and number plate recognition system using Raspberry Pi** plays a crucial role in **traffic law enforcement** by automating the process of identifying riders who fail to wear helmets. Traditional manual monitoring is inefficient and requires significant manpower, but with **computer vision and machine learning**, this system can **detect violations in real time** and capture vehicle details. This helps reduce the number of road accidents caused by helmet non-compliance while ensuring stricter enforcement of traffic regulations.

CHAPTER-7

CONCLUSION & FUTURESCOPE

7.1 CONCLUSION

The **helmet detection and number plate recognition system using Raspberry Pi** is a highly effective solution for improving **road safety and traffic law enforcement**. By utilizing **computer vision and machine learning algorithms**, the system can automatically detect whether a motorcyclist is wearing a helmet and identify the vehicle's registration number. This automation significantly reduces the need for manual monitoring, making traffic law enforcement more efficient and reducing human error. The use of **affordable hardware components like Raspberry Pi and a webcam** makes this project a cost-effective and scalable solution that can be deployed in urban and rural areas alike. Additionally, integrating this system with a **centralized database** enables authorities to issue fines and generate reports in real-time, further enhancing its effectiveness.

Beyond road safety, this system can also be applied in **industrial environments and workplaces** where helmet compliance is mandatory. By ensuring workers adhere to safety protocols, industries can reduce the risk of accidents and improve overall workplace security. The project's flexibility allows it to be extended to **parking management, security surveillance, and smart city applications**, proving its versatility and adaptability to different environments. Furthermore, its integration with IoT and cloud platforms ensures that data is processed efficiently and can be accessed remotely, improving decision-making for law enforcement agencies and municipal bodies.

Despite its advantages, the system faces challenges such as **varying lighting conditions, camera positioning, and image clarity**. However, by leveraging **advanced image processing techniques,deep learning models, and improved hardware**, these limitations can be mitigated. Overall, the project successfully demonstrates how **technology can enhance public safety and automate critical monitoring tasks**, making it a valuable contribution to smart city initiatives and modern traffic management systems.

7.2 FUTURE SCOPE

The future scope of this project is vast, with several potential improvements and extensions that can enhance its **efficiency, accuracy, and usability**. One of the primary areas of development is the integration of **deep learning models like YOLO (You Only Look Once) and Faster R- CNN**, which can significantly improve detection accuracy in real-world

conditions. These models can process images faster and more accurately, even in **low-light and high-speed traffic scenarios**, making them ideal for large-scale deployments.

Another promising direction is the **integration of edge computing and IoT**. By using devices like **Google Coral or NVIDIA Jetson Nano**, real-time processing can be improved, reducing the dependency on cloud computing and making the system more efficient for real-time applications. Additionally, the use of **5G and cloud-based data storage** will enable authorities to store and analyze large volumes of data, allowing for better traffic planning and policy implementation. The system can also be enhanced with **GPS and GIS-based tracking**, enabling real-time location tracking of vehicles involved in violations.

Incorporating **automatic fine issuance** through government databases is another potential future upgrade. By connecting the system to **transportation and traffic management databases**, fines can be generated and sent automatically to violators via SMS or email, eliminating the need for manual processing. Additionally, integrating AI-based **behavior analysis** can help identify repeat offenders and generate statistical reports for better decision-making by traffic authorities.

Beyond traffic enforcement, this system can be **repurposed for various industrial applications**, such as monitoring **safety compliance in factories, construction sites, and hazardous environments**. By ensuring that workers wear helmets and follow safety protocols, industries can reduce workplace accidents and enhance overall safety measures. The **smart city applications** of this project also extend to **automated parking management, restricted areamonitoring, and vehicle tracking**, making it a valuable tool for urban development and infrastructure planning.

In conclusion, the **helmet detection and number plate recognition system** holds immense potential for enhancing **public safety, law enforcement, and industrial security**. With continued research and advancements in AI, IoT, and machine learning, this system can be further refined and expanded to create a more **secure, efficient, and intelligent transportation ecosystem**.

REFERENCES

[1] M. Dasgupta, O. Bandyopadhyay and S. Chatterji, "Automated Helmet Detection for Multiple Motorcycle Riders using CNN," IEEE Conference on Information and Communication Technology, Allahabad, India, pp.1-4, 2023.

[2] K. Dahiya, D. singh and C.K. Mohan, “Automatic detection of bike rider without helmets using surveillance videos in real time”, in Proceeding of International Joint Conference Neural Networks (IJCNN), Vancouver, Canada, pp.3046-3051, 24-022016.

[3] J. Lietal, "Safety helmet wearing detection based on image processing and machine learning,” 2017 Ninth International Conference on Advanced Computational Intelligence (ICACI), Doha, pp.201-205, 2022.

[4] N. Boonsiri Sumpun, W. Puarungroj and P. Wairocana Phuttha, "Automatic Detector for Bikes with no Helmet using Deep Learning”, 22nd International Computer Science and Engineering Conference (ICSEC), Chiang Mai, Thailand, pp.1-4, 2021.

[5] B. Yogameena, K. Menaka and S. Saravana Perumaal, "Deep learning-based helmet wear analysis of a motorcycle rider for international surveillance system,” in IET Intelligent Transport system, vol.13, no.7, pp.1190-1198, 2020

[6] Pattasu Doughmala, Katanyoo Klubsuwan, "Half and Full Helmet Detection in Thailand using Haar Like Feature and Circle Hough Transform on Image Processing" in Proceeding of IEEE International Conference on Computer and Information Technology, Thailand, Bangkok, pp.611-614, 2019.

[7] Kavyashree Devadiga, Yash Gujarathi, Pratik Khanapurkar, Shreya Joshi and Shubhankar Deshpande. "Real Time Automatic Helmet Detection of Bike Riders" International Journal for Innovative Research in Science & Technology Volume 4 Issue 11, pp.146-148, 2018.

[8] R.V. Silva, T. Aires, and V. Rodrigo, " Helmet Detection on Motorcyclists using image descriptors and classifiers", in Proceeding of Graphics, Patterns and Images (SIBGRAPI), Rio de Janeiro, Brazil, pp.141148R, 30 August 2017.

[9] Redmon J, Farhadi A. YOLOv3: An Incremental Improvement [C]//IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[10] Redmon, Joseph, and Ali Farhadi. 'YOLO9000: better, faster, stronger.' IEEE Conference on Computer Vision and Pattern Recognition, pp.7763-7271, 2017.

[11] Redmon J, Divvala S, Girshick R, et al. You only look once: unified, real time object detection [C]// Computer Vision and Pattern Recognition, pp.779-786, 2016.

[12].Helmet detection and license Plate Recognition using YOLO model Authors: Prof. Muneshwar R. N.Miss. Pote Pranavi Vijay Mr. Bhawar Shivam Ashok Miss. Jadhav Pratiksha Sitaram Miss. Gite Nikita Rajendra.

[13]. Helmet Detection and Number Plate Recognition using Machine Learning Author: Gauri Marathe1, Pradnya Gurav2, Rushikesh Narwade3, Vallabh Ghodke4, Prof. S. M. Patil5

[14]. DETECTION OF HELMET USING YOLOV4 AND GENERATION OF AN E-CHALLAN Authors: Anirban Ashok Rudra Shrish Kiran Vaidya Kaushal Rajbahadur Singh

APPENDIX

```
import numpy as np
from keras.layers import Conv2D, Input, BatchNormalization, LeakyReLU,

ZeroPadding2D, UpSampling2D

from keras.layers.merge import add, concatenate from keras.models import Model import
struct
import cv2
class WeightReader:
    def __init__(self, weight_file):
        with open(weight_file, 'rb') as w_f:
            major,    =    struct.unpack('i',    w_f.read(4))
            minor,    =    struct.unpack('i',    w_f.read(4))
            revision, = struct.unpack('i', w_f.read(4))
            if (major*10 + minor) >= 2 and major < 1000 and minor < 1000:
                w_f.read(8)
            else:
                w_f.read(4)
            transpose = (major > 1000) or (minor > 1000)
            binary = w_f.read()
        self.offset = 0
        self.all_weights = np.frombuffer(binary, dtype='float32') def
    read_bytes(self, size):
        self.offset = self.offset + size
        return  self.all_weights[self.offset-size:self.offset] def
    load_weights(self, model):
        for i in range(106): try:
                conv_layer = model.get_layer('conv_' + str(i))
                print("loading weights of convolution #" + str(i)) if i
                not in [81, 93, 105]:
                    norm_layer = model.get_layer('bnorm_' + str(i)) size =
                    np.prod(norm_layer.get_weights()[0].shape) beta  =
                    self.read_bytes(size) # bias
                    gamma = self.read_bytes(size) # scale
                    mean  = self.read_bytes(size) # mean var
                    = self.read_bytes(size) # variance
                    weights = norm_layer.set_weights([gamma, beta, mean, var]) if
                len(conv_layer.get_weights()) > 1:
                    bias  = self.read_bytes(np.prod(conv_layer.get_weights()[1].shape)) kernel
                    = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                    kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape))) kernel =
                    kernel.transpose([2,3,1,0])
                    conv_layer.set_weights([kernel,  bias])
                else:
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape)) kernel =
                    kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape))) kernel =
                    kernel.transpose([2,3,1,0])
                    conv_layer.set_weights([kernel])
            except ValueError:
                print("no convolution #" + str(i)) def
    reset(self):
        self.offset = 0
class BoundBox:
    def _init_(self, xmin, ymin, xmax, ymax, objness = None, classes = None): self.xmin =
        xmin
        self.ymin = ymin
        self.xmax = xmax
```

```
        self.ymax = ymax
        self.objness = objness
        self.classes = classes
        self.label = -1
        self.score = -1 def
    get_label(self):
        if self.label == -1:
            self.label = np.argmax(self.classes) return
        self.label
    def get_score(self): if
        self.score == -1:
            self.score = self.classes[self.get_label()] return
        self.score
def _conv_block(inp, convs, skip=True): x =
    inp
    count = 0
    for conv in convs:
        if count == (len(convs) - 2) and skip: skip_connection
            = x
        count += 1
        if conv['stride'] > 1: x = ZeroPadding2D(((1,0),(1,0)))(x) # peculiar padding as darknet
prefer left and top
        x = Conv2D(conv['filter'],
                   conv['kernel'],
                   strides=conv['stride'],
                   padding='valid' if conv['stride'] > 1 else 'same', # peculiar padding as darknet
prefer left and top
                   name='conv_' + str(conv['layer_idx']), use_bias=False if
                   conv['bnorm'] else True)(x)
        if conv['bnorm']: x = BatchNormalization(epsilon=0.001, name='bnorm_' +
str(conv['layer_idx']))(x)
        if conv['leaky']: x = LeakyReLU(alpha=0.1, name='leaky_' + str(conv['layer_idx']))(x)
    return add([skip_connection, x]) if skip else x def
_interval_overlap(interval_a, interval_b):
    x1, x2 = interval_a
    x3, x4 = interval_b if
    x3 < x1:
        if x4 < x1:
            return 0
        else:
            return min(x2,x4) - x1
    else:
        if x2 < x3:
             return 0
        else:
            return min(x2,x4) - x3
def _sigmoid(x):
    return 1. / (1. + np.exp(-x)) def
bbox_iou(box1, box2):
    intersect_w = _interval_overlap([box1.xmin, box1.xmax], [box2.xmin, box2.xmax])
    intersect_h = _interval_overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])
    intersect = intersect_w * intersect_h
    w1, h1 = box1.xmax-box1.xmin, box1.ymax-box1.ymin w2,
    h2 = box2.xmax-box2.xmin, box2.ymax-box2.ymin union =
    w1*h1 + w2*h2 - intersect
    return float(intersect) / union
```

```
def make_yolov3_model():
    input_image = Input(shape=(None, None, 3))  #
    Layer  0 => 4
    x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
True, 'layer_idx': 0},
                                  {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 1},
                                  {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 2},
                                  {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 3}])
    # Layer  5 => 8
    x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 5},
                        {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
6},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
7}])
    # Layer  9 => 11
    x = _conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 9},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
10}])
    # Layer 12 => 15
    x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 12},
                        {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
13},
                        {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
14}])
    # Layer 16 => 36 for
    i in range(7):
        x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
16+i*3},
                            {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
17+i*3}])
    skip_36 = x
    # Layer 37 => 40
    x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 37},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
38},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
39}])
    # Layer 41 => 61 for
    i in range(7):
        x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
41+i*3},
                            {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
42+i*3}])
    skip_61 = x
    # Layer 62 => 65

    x = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 62},
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
63},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
64}])
    # Layer 66 => 74 for
    i in range(3):
```

```
        x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 66+i*3},
                            {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
67+i*3}])
    # Layer 75 => 79
    x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 75},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
76},
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
77},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
78},
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
79}], skip=False)
    # Layer 80 => 82

    yolo_82 = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 80},
                              {'filter':  255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
'layer_idx': 81}], skip=False)
    # Layer 83 => 86
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 84}],
skip=False)

    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_61]) #
    Layer 87 => 91
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 87},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
88},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
89},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
90},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx':
91}], skip=False)
    # Layer 92 => 94

    yolo_94 = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 92},
                              {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
'layer_idx': 93}], skip=False)
    # Layer 95 => 98

    x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 96}],
skip=False)

    x = UpSampling2D(2)(x)

    x = concatenate([x, skip_36])
```

```
    # Layer 99 => 106
    yolo_106 = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': rue,
'layer_idx': 99},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 100},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,'layer_idx': 101},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 102},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 103},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 104},
                               {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
'layer_idx': 105}], skip=False)
    model = Model(input_image, [yolo_82, yolo_94, yolo_106]) return
    model
def preprocess_input(image, net_h, net_w):
    new_h, new_w, _ = image.shape
    # determine the new size of the image

    if (float(net_w)/new_w) < (float(net_h)/new_h):
        new_h = (new_h * net_w)/new_w
        new_w = net_w
    else:
        new_w = (new_w * net_h)/new_h
        new_h = net_h
    # resize the image to the new size

    resized = cv2.resize(image[:,:,::-1]/255., (int(new_w), int(new_h))) #
    embed the image into the standard letter box
    new_image = np.ones((net_h, net_w, 3)) * 0.5
    new_image[int((net_h-new_h)//2):int((net_h+new_h)//2), int((net_w-
new_w)//2):int((net_w+new_w)//2), :] = resized
    new_image = np.expand_dims(new_image, 0) return
    new_image
def decode_netout(netout, anchors, obj_thresh, nms_thresh, net_h, net_w):
    grid_h, grid_w = netout.shape[:2]
    nb_box = 3
    netout = netout.reshape((grid_h, grid_w, nb_box, -1)) nb_class =
    netout.shape[-1] - 5
    boxes = []
    netout[..., :2]  = _sigmoid(netout[..., :2])
    netout[..., 4:]  = _sigmoid(netout[..., 4:])
    netout[..., 5:]  = netout[..., 4][..., np.newaxis] * netout[..., 5:]
    netout[..., 5:] *= netout[..., 5:] > obj_thresh
    for i in range(grid_h*grid_w):
        row = i / grid_w
        col = i % grid_w
        for b in range(nb_box):
            # 4th element is objectness score

            objectness = netout[int(row)][int(col)][b][4]
```

```
            #objectness = netout[..., :4] if(objectness.all()
            <= obj_thresh): continue # first 4 elements are
            x, y, w, and h
            x, y, w, h = netout[int(row)][int(col)][b][:4]
            x = (col + x) / grid_w # center position, unit: image width y =
            (row + y) / grid_h # center position, unit: image height
            w = anchors[2 * b + 0] * np.exp(w) / net_w # unit: image width h =
            anchors[2 * b + 1] * np.exp(h) / net_h # unit: image height # last
            elements are class probabilities
            classes = netout[int(row)][col][b][5:]
            box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes) #box =
            BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, None, classes)
            boxes.append(box)
    return boxes
def correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w): if
    (float(net_w)/image_w) < (float(net_h)/image_h):
        new_w = net_w
        new_h = (image_h*net_w)/image_w else:
        new_h = net_w
        new_w = (image_w*net_h)/image_h for i
    in range(len(boxes)):
        x_offset, x_scale = (net_w - new_w)/2./net_w, float(new_w)/net_w
        y_offset, y_scale = (net_h - new_h)/2./net_h, float(new_h)/net_h
        boxes[i].xmin = int((boxes[i].xmin - x_offset) / x_scale * image_w)
        boxes[i].xmax = int((boxes[i].xmax - x_offset) / x_scale * image_w)
        boxes[i].ymin = int((boxes[i].ymin - y_offset) / y_scale * image_h)
        boxes[i].ymax = int((boxes[i].ymax - y_offset) / y_scale * image_h)
def do_nms(boxes, nms_thresh): if
    len(boxes) > 0:
        nb_class = len(boxes[0].classes)
    else:
        return
    for c in range(nb_class):
        sorted_indices = np.argsort([-box.classes[c] for box in boxes]) for i in
        range(len(sorted_indices)):
            index_i = sorted_indices[i]
            if boxes[index_i].classes[c] == 0: continue for j
            in range(i+1, len(sorted_indices)):
                index_j = sorted_indices[j]
                if bbox_iou(boxes[index_i], boxes[index_j]) >= nms_thresh: boxes[index_j].classes[c] = 0
def draw_boxes(image, boxes, line, labels, obj_thresh, dcnt):
    print(line)
    for box in boxes:
        label_str = ''
        label = -1
        for i in range(len(labels)):
            if box.classes[i] > obj_thresh:
                label_str += labels[i]
                label = i
                print(labels[i] + ': ' + str(box.classes[i]*100) + '%')
                print('line: (' + str(line[0][0]) + ', ' + str(line[0][1]) + ') (' + str(line[1][0]) + ', '

+ str(line[1][1]) + ')')
                print('Box: (' + str(box.xmin) + ', ' + str(box.ymin) + ') (' + str(box.xmax) + ', '

+ str(box.ymax) + ')')
                print() if
        label >= 0:
            tf = False
```

```
            (rxmin, rymin) = (box.xmin, box.ymin)
            (rxmax, rymax) = (box.xmax, box.ymax) tf =
            False
            tf |= intersection(line[0], line[1], (rxmin, rymin), (rxmin, rymax)) tf |=
            intersection(line[0], line[1], (rxmax, rymin), (rxmax, rymax)) tf |=
            intersection(line[0], line[1], (rxmin, rymin), (rxmax, rymin)) tf |=
            intersection(line[0], line[1], (rxmin, rymax), (rxmax, rymax)) print(tf)
            cv2.line(image, line[0], line[1], (255, 0, 0), 3) if tf:
                cv2.rectangle(image, (box.xmin,box.ymin), (box.xmax,box.ymax), (255,0,0),
3)
                cimg = image[box.ymin:box.ymax, box.xmin:box.xmax]
                cv2.imshow("violation", cimg)
                cv2.waitKey(5)
                cv2.imwrite("G:/Traffic Violation Detection/Traffic Signal Violation Detection
System/Detected Images/violation_"+str(dcnt)+".jpg", cimg)
                dcnt = dcnt+1
            else:
                cv2.rectangle(image, (box.xmin,box.ymin), (box.xmax,box.ymax), (0,255,0),
3)
            cv2.putText(image,
                        label_str + ' ' + str(round(box.get_score(), 2)),
                        (box.xmin, box.ymin - 13),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        1e-3 * image.shape[0], (0,255,0),
                        2)
    return image
weights_path = "G:/Traffic Violation Detection/yolov3.weights" # set
some parameters
net_h, net_w = 416, 416
obj_thresh, nms_thresh = 0.5, 0.45
anchors = [[116,90,  156,198,  373,326],  [30,61, 62,45,  59,119], [10,13,  16,30,
33,23]]
labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck", \ "boat",
          "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", \ "bird", "cat",
          "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra",
"giraffe", \
          "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", \
          "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard", \
          "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl",
"banana", \
          "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut",
"cake", \
          "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse", \
          "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator", \
          "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"] # make
the yolov3 model to predict 80 classes on COCO
yolov3 = make_yolov3_model()
# load the weights trained on COCO into the model weight_reader =
WeightReader(weights_path) weight_reader.load_weights(yolov3)
# my defined functions def
```

```
intersection(p, q, r, t):
    print(p, q, r, t)
    (x1, y1) = p
    (x2, y2) = q
    (x3, y3) = r
    (x4,  y4)  =  t
    a1  =  y1-y2
    b1 = x2-x1
    c1  =  x1*y2-x2*y1
    a2 = y3-y4
    b2 = x4-x3
    c2     =     x3*y4-x4*y3
    if(a1*b2-a2*b1 == 0):
        return False
    print((a1, b1, c1), (a2, b2, c2))
    x = (b1*c2 - b2*c1) / (a1*b2 - a2*b1) y =
    (a2*c1 - a1*c2) / (a1*b2 - a2*b1) print((x,
    y))
    if x1 > x2:
        tmp = x1
        x1 = x2
        x2 = tmp
    if y1 > y2:
        tmp = y1
        y1 = y2
        y2 = tmp
    if x3 > x4:
        tmp = x3
        x3 = x4
        x4 = tmp
    if y3 > y4:
        tmp = y3
        y3 = y4
        y4 = tmp
    if x >= x1 and x <= x2 and y >= y1 and y <= y2 and x >= x3 and x <= x4 and y >= y3 and y
<= y4:
        return True
    else:
        return False
```