ANIMAL DETECTION SYSTEM

MINI PROJECT REPORT

Submitted by

SURYA NIRANJAN N	**2116230701354**
SAKTHI ADITHYA E S	**2116230701398**

In partial fulfillment for the award of the degree

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2025

BONAFIDE CERTIFICATE

Certified that this project **"ANIMAL DETECTION SYSTEM**" is the bonafide work of **"SURYA NIRANJAN N(2116230701354) and SAKTHI ADITHYA E S(2116230701398)"** who carried out the project work under my supervision.

SIGNATURE
Dr.N.Duraimurugan, M.Tech., Ph.D.

Associate Professor,

Computer Science & Engineering

Rajalakshmi Engineering College (Autonomous)

Thandalam, Chennai -602105.

Submitted for the **ANNA UNIVERSITY** practical examination Mini-Project work viva voice held on_____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S.MEGANATHAN, B.E, F.I.E.,** our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN, Ph.D.,** for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our work in time.

We express our sincere thanks to **Dr. P. KUMAR, M.E., Ph.D.,** Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work.

We also extend our sincere and hearty thanks to our Internal Guide **Dr.N.Duraimurugan, M.Tech., Ph.D.** Associate Professor, Department of Computer Science and Engineering for his valuable guidance and motivation during the completion of this project. Our sincere thanks to our family members, friends and other staff members of information technology.

SURYA NIRANJAN N 2116230701348
SAKTHI ADITHYA E S
2116230701364

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATION

ABBREVIATION	ACRONYM
IOT	Internet of Things
SOC	State of Charge
SOH	State of Health
RUL	Remaining Useful Life
IDE	Integrated Development Environment
JSON	JavaScript Object Notation

ABSTRACT

This paper presents an innovative Internet of Things (IoT) based system for detecting and monitoring animal movements in and around forest restricted areas. The system employs a network of strategically placed motion sensors and actuators to detect animal presence and track their movement patterns across designated boundaries. When animals enter restricted zones, the system triggers immediate notifications to both local citizens and forest department officials through a user-friendly interface, while simultaneously activating deterrent measures such as acoustic buzzers. Real-time data collection allows for pattern analysis of animal movements, enabling predictive modeling for future incursions. Implementation results demonstrate improved response times by authorities, reduced human-wildlife conflicts, and enhanced safety for both wildlife and local communities. The proposed system offers a cost-effective, scalable solution that balances wildlife conservation needs with human safety considerations, while providing valuable data for ecological research and wildlife management strategies.

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The increasing human-wildlife conflicts in forest boundary areas necessitate intelligent monitoring systems capable of providing real-time animal movement detection and stakeholder notifications. This project develops an IoT-based animal detection system that integrates multiple sensor modalities including dual PIR sensors, ultrasonic distance measurement, and computer vision to create a robust three-stage detection framework for forest restricted areas. The system employs ESP32 microcontroller for sensor coordination and Python-based machine learning for animal verification, enabling automated alerts to both citizens and forest authorities while minimizing false alarms through sequential confirmation protocols.

1.2 SCOPE OF THE WORK

The project encompasses the design and implementation of a multi-sensor IoT system specifically tailored for forest boundary monitoring applications. The scope includes hardware integration of PIR motion sensors, ultrasonic distance sensors, and buzzer actuators with ESP32 microcontroller, development of sensor fusion algorithms for directional movement detection, implementation of wireless communication protocols for laptop-based computer vision integration, creation of Python-based animal detection models using machine learning techniques, and design of multi-stakeholder notification systems for real-time alert management. The system is designed for deployment in forest restricted areas with focus on detecting animal movement across boundary zones.

1.3 PROBLEM STATEMENT

Existing wildlife monitoring systems in forest restricted areas suffer from high false alarm rates, lack of directional movement detection, and absence of intelligent verification mechanisms to distinguish between animals and other moving objects. Traditional single-sensor approaches cannot effectively differentiate between genuine animal intrusions and environmental disturbances such as vegetation movement or weather effects, leading to unreliable detection and delayed response times. The absence of real-time stakeholder notification systems further compounds the problem by preventing timely intervention in human-wildlife conflict situations, resulting in property damage, agricultural losses, and potential safety hazards for both humans and wildlife.

1.4 AIM AND OBJECTIVES

Aim:

To design and develop an intelligent IoT-based animal detection system that provides accurate, real-time monitoring of animal movements in forest restricted areas through multi-stage sensor fusion and computer vision verification, enabling automated notification to stakeholders for effective human-wildlife conflict management.

Objectives:

The primary objectives include implementing a three-stage detection system using dual PIR sensors for motion detection, ultrasonic sensors for distance confirmation, and computer vision for animal verification, developing ESP32-based sensor coordination system with wireless communication capabilities for

real-time data transmission, creating Python-based machine learning model for automated animal detection and classification using laptop camera integration, and designing multi-stakeholder notification system with buzzer alerts and digital communication channels for citizens and forest department. The secondary objectives focus on achieving directional movement detection capability to distinguish between animals entering and exiting restricted areas, minimizing false alarm rates through sequential sensor confirmation protocols, ensuring system reliability and robustness for outdoor forest deployment conditions, and establishing scalable architecture suitable for deployment across multiple forest boundary locations.

CHAPTER 2

SYSTEM SPECIFICATIONS

2.1 IOT DEVICES

1. Arduino Uno
2. DG-301 Voltage Sensor
3. ACS712 Current Sensor
4. DHT11 Temperature Sensor

2.2 SYSTEM HARDWARE SPECIFICATIONS

PROCESSOR	Intel i3 11th Gen
MEMORY SIZE	8 GB (Minimum)
HDD	40 GB (Minimum)

2.3 SOFTWARE SPECIFICATIONS

Operating System	Windows 11
Front – End	Electron.js, Chart.js
Back – End	Node.js, TensorFlow.js
IDE	Visual Studio Code, Arduino IDE

CHAPTER 3
SYSTEM DESIGN

The system design section outlines the architectural framework, component interactions, and technical specifications of the IoT-based animal detection system. It provides a comprehensive overview of how the multi-stage detection mechanism integrates various sensors, processing units, and communication modules to achieve reliable animal detection and stakeholder notification in forest restricted areas.

4.1 System Architecture Overview

The animal detection system employs a distributed architecture consisting of three primary layers: the sensor layer for data acquisition, the processing layer for intelligent analysis, and the communication layer for stakeholder notifications. The architecture is designed around an ESP32 microcontroller that serves as the central coordination unit, managing sensor inputs and facilitating communication with a laptop-based computer vision system. The system implements a hierarchical detection approach where each stage provides progressive confirmation of animal presence before triggering alert mechanisms.

The edge computing architecture distributes processing tasks efficiently, with the ESP32 handling real-time sensor data collection, preliminary filtering, and communication protocols, while the laptop manages computationally intensive computer vision analysis and machine learning inference. This hybrid approach ensures optimal resource utilization while maintaining low latency response times critical for wildlife monitoring applications. The modular design allows for independent operation of sensor subsystems while maintaining synchronized coordination through the central processing unit.

4.2 Hardware Components and Configuration

The hardware configuration centers around the ESP32 microcontroller, chosen for its integrated WiFi capabilities, sufficient GPIO pins for multi-sensor interfacing, and low power consumption suitable for remote deployment. The dual PIR sensor configuration employs two HC-SR501 sensors positioned facing each other to create a detection zone that enables directional movement analysis. When an object crosses between the sensors, the sequential triggering pattern allows the system to determine movement direction and confirm genuine boundary crossings.

The ultrasonic sensor (HC-SR04) is strategically positioned at a distance from the PIR sensors to provide secondary confirmation through distance variation monitoring. This sensor continuously measures distance to detect approaching objects and validates the motion detected by PIR sensors, effectively filtering out false positives caused by small debris or vegetation movement. A buzzer module is integrated for immediate local alerts, while the laptop camera serves as the tertiary verification component through the computer vision pipeline.

Power management is implemented through a regulated power supply system that can accommodate both battery and external power sources. The system includes voltage regulation circuits to ensure stable operation of all components, with power optimization strategies to extend deployment duration in remote locations. Environmental protection is provided through weatherproof enclosures designed to maintain sensor functionality under varying outdoor conditions.

4.3 Three-Stage Detection Methodology

The detection methodology implements a sequential confirmation process designed to maximize accuracy while minimizing false alarms. Stage one involves the dual PIR sensor array that detects initial movement and determines directionality through timing analysis of sensor triggers. The system requires both PIR sensors to detect motion within a specified time window to confirm genuine movement across the detection boundary, effectively eliminating false triggers from single-point disturbances.

Stage two employs the ultrasonic sensor to provide distance-based confirmation of the detected movement. The sensor continuously monitors distance variations to validate that a substantial object is approaching or moving through the detection zone. This stage filters out environmental factors such as wind-blown vegetation or small animals that might trigger PIR sensors but lack sufficient mass or proximity to register on ultrasonic measurements. The distance threshold and variation parameters are configurable based on target animal sizes and environmental conditions.

Stage three activates the computer vision verification system when the first two stages confirm movement. The ESP32 sends a trigger signal to the laptop system, which captures images through the connected camera and processes them using a pre-trained Python-based machine learning model. The computer vision component performs object detection and classification to confirm whether the detected movement was caused by an animal, completing the verification chain before alert generation.

4.4 Communication and Data Flow Architecture

The communication architecture implements WiFi-based connectivity between the ESP32 sensor unit and the laptop processing system, enabling real-time data transmission and coordination. The ESP32 establishes a local network connection and communicates sensor status, detection events, and system health information to the laptop through structured data packets. Communication protocols include error handling, acknowledgment mechanisms, and retry logic to ensure reliable data transmission in potentially unstable network conditions.

Data flow follows a structured pipeline beginning with sensor data acquisition at predetermined sampling rates, followed by preliminary processing and filtering on the ESP32 to reduce communication overhead. Relevant detection events trigger immediate transmission to the laptop system, which processes the information and coordinates camera activation for computer vision analysis. The laptop system manages the machine learning inference process and generates appropriate alerts based on detection confirmation.

The notification system implements multiple communication channels to reach different stakeholder groups simultaneously. Local buzzer activation provides immediate area alerts, while digital notifications are transmitted through web interfaces, messaging systems, or mobile applications to reach citizens and forest department personnel. The system maintains event logs and provides status reporting capabilities for system monitoring and maintenance purposes.

4.5 Software Architecture and Integration

The software architecture consists of embedded firmware for the ESP32 and Python-based applications for computer vision and system coordination. The ESP32 firmware implements real-time sensor monitoring, data preprocessing, communication protocols, and local decision-making logic. The firmware includes interrupt-driven sensor handling to ensure responsive detection while maintaining efficient power consumption through sleep mode management.

The Python application framework manages computer vision processing, machine learning inference, notification systems, and user interface components. The computer vision module utilizes pre-trained models optimized for animal detection, with capabilities for real-time image processing and classification. The system includes model update mechanisms to improve detection accuracy over time and adapt to specific deployment environments.

Integration between hardware and software components is achieved through standardized communication protocols and API interfaces that enable modular development and testing. The software architecture supports configuration management, allowing adjustment of detection thresholds, notification preferences, and system parameters without firmware modifications. Error

handling and diagnostic capabilities are integrated throughout the system to support remote troubleshooting and maintenance operations.

4.6 Alert and Notification System Design

The alert system implements a multi-tier notification architecture that provides immediate local alerts through integrated buzzers while simultaneously transmitting digital notifications to relevant stakeholders. The local alert system includes configurable buzzer patterns that can indicate different types of detections or system status conditions. Volume and duration parameters are adjustable to ensure appropriate alert levels without causing unnecessary disturbance to wildlife.

Digital notification systems support multiple communication channels including web-based dashboards, email alerts, SMS messaging, and mobile application push notifications. The system implements stakeholder-specific notification profiles that deliver appropriate information to different user groups based on their roles and responsibilities. Citizens receive basic movement alerts with safety recommendations, while forest department personnel receive detailed detection information including timestamps, location data, and confidence levels.

The notification system includes intelligent alert management features such as alert aggregation to prevent notification flooding, priority-based message routing for urgent situations, and acknowledgment tracking to ensure critical alerts are received and acted upon. Historical data logging enables analysis of animal movement patterns and system performance evaluation, supporting long-term wildlife management and conservation planning initiatives.

This comprehensive system design provides a robust foundation for reliable animal detection while maintaining the flexibility needed for deployment across diverse forest environments and stakeholder requirements. The modular architecture supports future enhancements and scalability for broader wildlife monitoring applications.

.

CHAPTER 4

MODULE DESCRIPTION

4.1 HARDWARE MODULE:

This module includes voltage, current, and temperature sensors (DG-301, ACS712, and DHT11 respectively) connected to the Arduino Uno microcontroller. It is responsible for real-time data acquisition from batteries. The sensors continuously monitor battery parameters such as voltage, current, and temperature.

4.2 DATA COLLECTION AND PROCESSING MODULE:

This module interfaces with the hardware to collect sensor data via serial communication. It processes incoming data to compute essential battery parameters like State of Charge (SoC), State of Health (SoH), energy usage, and charge-discharge cycles. The data is cleaned, structured, and prepared for visualization and further analysis.

4.3 MACHINE LEARNING MODULE:

A lightweight ML model implemented using TensorFlow.js predicts metrics like Remaining Useful Life (RUL) and enhances SoH/SoC estimation. The model is trained on synthetic or pre-collected data and integrated into the desktop system for real-time inference.

4.4 DESKTOP APPLICATION MODULE:

Built using Electron.js, this module provides an interactive dashboard for users to monitor battery health visually. It displays real-time sensor data using charts, gauges, and status indicators, and allows users to switch between AA and Li-ion battery types.

4.5 INTEGRATION MODULE:

This module connects all other components — reading serial data, performing computations, running ML inference, and updating the dashboard — to deliver a cohesive and synchronized monitoring experience.

CHAPTER 5

SAMPLE CODING

5.1 Arduino Code

```
// Pin Definitions
#define PIR1_PIN 12
#define PIR2_PIN 13
#define TRIG_PIN 5
#define ECHO_PIN 18
#define BUZZER_PIN 27

// Motion detection variables
unsigned long lastMotionTime = 0;
const int motionGap = 3000; // 3 seconds allowed between PIR1 and PIR2 detection
bool pir1Triggered = false;
bool pir2Triggered = false;

// Ultrasonic
float lastDistance = 0;

void setup() {
  Serial.begin(115200);

  pinMode(PIR1_PIN, INPUT);
  pinMode(PIR2_PIN, INPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);

  Serial.println("ESP32 Ready - Waiting for motion...");
}

void loop() {
  bool motion1 = digitalRead(PIR1_PIN);
  bool motion2 = digitalRead(PIR2_PIN);
  unsigned long currentTime = millis();

  if (motion1 && !pir1Triggered) {
    pir1Triggered = true;
    lastMotionTime = currentTime;
    Serial.println("PIR1 Triggered");
  }

  if (motion2 && pir1Triggered && (currentTime - lastMotionTime <= motionGap)) {
```

```
  pir2Triggered = true;
  Serial.println("PIR2 Triggered");
 }

if (pir1Triggered && pir2Triggered) {
  float distance = getUltrasonicDistance();
  Serial.print("Distance: ");
  Serial.println(distance);

  if (isMovementApproaching(distance)) {
    Serial.println("APPROACHING OBJECT DETECTED");


    digitalWrite(BUZZER_PIN, HIGH);
    delay(500);
    digitalWrite(BUZZER_PIN, LOW);
    delay(500);
    digitalWrite(BUZZER_PIN, HIGH);
    delay(500);
    digitalWrite(BUZZER_PIN, LOW);
    delay(500);
    digitalWrite(BUZZER_PIN, HIGH);
    delay(500);
    digitalWrite(BUZZER_PIN, LOW);
    delay(500);
    digitalWrite(BUZZER_PIN, HIGH);
    delay(500);
    digitalWrite(BUZZER_PIN, LOW);
    delay(500);
    digitalWrite(BUZZER_PIN, HIGH);
    delay(500);
    digitalWrite(BUZZER_PIN, LOW);
    delay(500);

    // Notify laptop to run Python CV script
    Serial.println("RUN_CV");

    // Wait for Python response
    while (Serial.available() == 0) {
      delay(10);
    }

    String response = Serial.readStringUntil('\n');
    response.trim();

    if (response == "animal") {
      Serial.println("Animal Confirmed");
      triggerBuzzer(200);
```

```
      } else {
        Serial.println("Not an animal");
      }
    }

    pir1Triggered = false;
    pir2Triggered = false;
  }

  if (pir1Triggered && (currentTime - lastMotionTime > motionGap)) {
    pir1Triggered = false;
    pir2Triggered = false;
    Serial.println("Motion timeout - reset");
  }

  delay(300);
}

float getUltrasonicDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH, 20000);
  if (duration == 0) return -1;
  return (duration * 0.0343) / 2;
}

bool isMovementApproaching(float currentDistance) {
  if (currentDistance < 0) return false;
  bool approaching = false;
  if (lastDistance > 0 && currentDistance < (lastDistance - 5)) {
    approaching = true;
  }
  lastDistance = currentDistance;
  return approaching;
}

void triggerBuzzer(int durationMs) {
  digitalWrite(BUZZER_PIN, HIGH);
  delay(durationMs);
  digitalWrite(BUZZER_PIN, LOW);
}
```

5.2 Web Application

5.2.1 html

```
const { app, BrowserWindow, ipcMain } = require('electron');
const path = require('path');
const { SerialPort } = require('serialport');
const { ReadlineParser } = require('@serialport/parser-readline');
let win;
let serialPort;
function createWindow() {
  win = new BrowserWindow({
    width: 1280,
    height: 720,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    },
  });
  win.maximize();
  win.setMenuBarVisibility(false);
  win.loadFile('index.html');
}
app.whenReady().then(createWindow);
ipcMain.on('start-serial', (event, portName) => {
  if (serialPort && serialPort.isOpen) {
    serialPort.removeAllListeners();
    serialPort.close(() => {
      console.log('Previous port closed');
      openSerial(portName, event);
    });
  } else {
    openSerial(portName, event);
  }
});
```

```
function openSerial(portName, event) {
  serialPort = new SerialPort({
    path: portName,
    baudRate: 9600,
  });
  const parser = serialPort.pipe(new ReadlineParser({ delimiter: '\n' }));
  serialPort.on('open', () => {
    console.log('Serial port opened:', portName);
  });
  parser.on('data', (line) => {
    event.sender.send('serial-data',line.trim());
  });
  serialPort.on('error', (err) => {
    console.error('Serial port error:', err.message);
  });
}
ipcMain.handle('list-ports', async () => {
  const ports = await SerialPort.list();
  return ports.map(p => p.path);
});WEB APPLIVATION:

html code:



<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Wildlife Ranger Dashboard</title>
    <link rel="stylesheet" href="style.css">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.9.1/chart.min.js"></script>
</head>
```

```
<body>
    <div class="dashboard-container">
        <header>
            <div class="logo">
                <h1>Wildlife Ranger Dashboard</h1>
            </div>
            <div class="user-info">
                <span class="username">Park Ranger: Boopathi</span>
                <span class="date-time" id="currentDateTime">Loading...</span>
            </div>
        </header>

        <main>
            <div class="dashboard-content">
                <div class="stats-container">
                    <div class="stat-card">
                        <h3>Total Animals Detected</h3>
                        <div class="stat-value">249</div>
                        <div class="stat-trend positive">+12% from yesterday</div>
                    </div>
                    <div class="stat-card">
                        <h3>Species Identified</h3>
                        <div class="stat-value">16</div>
                        <div class="stat-trend neutral">Same as yesterday</div>
                    </div>
                    <div class="stat-card">
                        <h3>Endangered Species</h3>
                        <div class="stat-value">3</div>
                        <div class="stat-trend negative">-1 from yesterday</div>
                    </div>
                    <div class="stat-card">
                        <h3>Avg. Detection Distance</h3>
                        <div class="stat-value">125m</div>
```

```
            <div class="stat-trend positive">+15m from yesterday</div>
        </div>
    </div>

    <div class="analytics-section">
        <h3 class="section-title">Analytics Overview</h3>
        <div class="chart-grid">
            <div class="chart-container">
                <h3>Detections by Hour</h3>
                <canvas id="hourlyDetectionsChart"></canvas>
            </div>
            <div class="chart-container">
                <h3>Species Distribution</h3>
                <canvas id="speciesDistributionChart"></canvas>
            </div>
            <div class="chart-container">
                <h3>Movement Patterns</h3>
                <canvas id="movementPatternsChart"></canvas>
            </div>
            <div class="chart-container">
                <h3>Detection by Region</h3>
                <canvas id="regionDetectionChart"></canvas>
            </div>
        </div>
    </div>
</div>

<div class="recent-activity">
    <h3>Recent Detections</h3>
    <div class="activity-list">
        <div class="activity-item">

            <div class="activity-details">
```

```
                <h4>Tiger (1)</h4>
                <p>Detected at North Ridge Trail - 0.8 miles from visitor center</p>
                <span class="activity-time">Today, 11:23 AM</span>
            </div>
            <div class="activity-actions">
                <button class="btn-notify" data-animal="Whitetail Deer" data-
location="North Ridge Trail" data-image="/api/placeholder/400/300">Send Alert</button>
            </div>
        </div>
        <div class="activity-item">

            <div class="activity-details">
                <h4>Cheetah(1)</h4>
                <p>Detected at Pine Creek Crossing - 1.2 miles from visitor center</p>
                <span class="activity-time">Today, 10:17 AM</span>
            </div>
            <div class="activity-actions">
                <button class="btn-notify" data-animal="Black Bear" data-location="Pine
Creek Crossing" data-image="/api/placeholder/400/300">Send Alert</button>
            </div>
        </div>
        <div class="activity-item">

            <div class="activity-details">
                <h4>Lion (1)</h4>
                <p>Detected at South Meadow - 1.5 miles from visitor center</p>
                <span class="activity-time">Today, 9:45 AM</span>
            </div>
            <div class="activity-actions">
                <button class="btn-notify" data-animal="Coyote" data-location="South
Meadow" data-image="/api/placeholder/400/300">Send Alert</button>
            </div>
        </div>
```

```
            <div class="activity-item">

                <div class="activity-details">
                    <h4>Tiger (2)</h4>
                    <p>Detected at Cedar Lake Trail - 2.3 miles from visitor center</p>
                    <span class="activity-time">Today, 8:12 AM</span>
                </div>
                <div class="activity-actions">
                    <button class="btn-notify" data-animal="Moose" data-location="Cedar
Lake Trail" data-image="/api/placeholder/400/300">Send Alert</button>
                </div>
            </div>
        </div>
    </main>

    <div id="notification-overlay" class="hidden">
        <div class="notification-container">
            <h2>WILDLIFE ALERT</h2>
            <div class="notification-content">
                <div class="notification-image">
                    <img id="notification-img" src="" alt="Wildlife Alert">
                </div>
                <div class="notification-details">
                    <h3 id="notification-title"></h3>
                    <p id="notification-location"></p>
                    <p id="notification-message"></p>
                </div>
            </div>
            <div class="notification-actions">
                <button id="notification-send">Send to All Citizens</button>
                <button id="notification-cancel">Cancel</button>
            </div>
```

```
        </div>
      </div>

      <div id="alert-box" class="hidden">
        <div class="alert-content">
          <h2>NOTIFICATION SENT!</h2>
          <p>Citizens have been alerted about the wildlife sighting.</p>
          <button id="alert-close">Close</button>
        </div>
      </div>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

5.2.2 script code

script code:

```
// DOM Elements
const currentDateTimeEl = document.getElementById('currentDateTime');
const notificationOverlay = document.getElementById('notification-overlay');
const notificationTitle = document.getElementById('notification-title');
const notificationLocation = document.getElementById('notification-location');
const notificationMessage = document.getElementById('notification-message');
const notificationImg = document.getElementById('notification-img');
const notificationSendBtn = document.getElementById('notification-send');
const notificationCancelBtn = document.getElementById('notification-cancel');
const alertBox = document.getElementById('alert-box');
const alertCloseBtn = document.getElementById('alert-close');
const notifyButtons = document.querySelectorAll('.btn-notify');

// Chart Objects
let hourlyChart, speciesChart, movementChart, regionChart;

// Update current date and time
function updateDateTime() {
    const now = new Date();
    const options = {
```

```
        weekday: 'long',
        year: 'numeric',
        month: 'long',
        day: 'numeric',
        hour: '2-digit',
        minute: '2-digit'
    };
    currentDateTimeEl.textContent = now.toLocaleDateString('en-US', options);
}

// Initialize Charts
function initCharts() {
    // Hourly Detections Chart
    const hourlyCtx = document.getElementById('hourlyDetectionsChart').getContext('2d');
    hourlyChart = new Chart(hourlyCtx, {
        type: 'line',
        data: {
            labels: ['6AM', '7AM', '8AM', '9AM', '10AM', '11AM', '12PM', '1PM', '2PM', '3PM',
'4PM', '5PM'],
            datasets: [{
                label: 'Animal Detections',
                data: [5, 8, 15, 22, 18, 12, 14, 16, 19, 23, 17, 10],
                backgroundColor: 'rgba(69, 123, 157, 0.2)',
                borderColor: 'rgba(69, 123, 157, 1)',
                borderWidth: 2,
                tension: 0.4,
                pointBackgroundColor: 'rgba(69, 123, 157, 1)'
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            scales: {
                y: {
                    beginAtZero: true,
                    grid: {
                        color: 'rgba(0, 0, 0, 0.05)'
                    }
                },
                x: {
                    grid: {
                        display: false
                    }
                }
            },
            plugins: {
                legend: {
                    display: false
                }
            }
```

```
        }
    });

    // Species Distribution Chart
    const speciesCtx = document.getElementById('speciesDistributionChart').getContext('2d');
    speciesChart = new Chart(speciesCtx, {
        type: 'doughnut',
        data: {
            labels: ['Deer', 'Bear', 'Coyote', 'Moose', 'Fox', 'Other'],
            datasets: [{
                label: 'Species Distribution',
                data: [42, 11, 17, 8, 14, 8],
                backgroundColor: [
                    '#457b9d',
                    '#e63946',
                    '#f1c453',
                    '#2a9d8f',
                    '#f4a261',
                    '#a8dadc'
                ],
                borderWidth: 1
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            plugins: {
                legend: {
                    position: 'right',
                    labels: {
                        boxWidth: 8,
                        font: {
                            size: 8
                        }
                    }
                }
            }
        }
    });

    // Movement Patterns Chart
    const movementCtx =
document.getElementById('movementPatternsChart').getContext('2d');
    movementChart = new Chart(movementCtx, {
        type: 'bar',
        data: {
            labels: ['North Trail', 'South Meadow', 'East Ridge', 'West Valley', 'Lake Area'],
            datasets: [{
                label: 'Morning',
                data: [25, 18, 15, 12, 30],
```

```
            backgroundColor: 'rgba(42, 157, 143, 0.7)'
        }, {
            label: 'Evening',
            data: [15, 22, 28, 19, 17],
            backgroundColor: 'rgba(233, 196, 106, 0.7)'
        }]
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        scales: {
            y: {
                beginAtZero: true,
                grid: {
                    color: 'rgba(0, 0, 0, 0.05)'
                }
            },
            x: {
                grid: {
                    display: false
                }
            }
        },
        plugins: {
            legend: {
                position: 'top',
                labels: {
                    boxWidth: 8,
                    font: {
                        size: 8
                    }
                }
            }
        }
    }
});

// Region Detection Chart
const regionCtx = document.getElementById('regionDetectionChart').getContext('2d');
regionChart = new Chart(regionCtx, {
    type: 'radar',
    data: {
        labels: ['Distance (m)', 'Speed (km/h)', 'Group Size', 'Duration (min)', 'Activity Level'],
        datasets: [{
            label: 'Herbivores',
            data: [150, 25, 8, 45, 60],
            backgroundColor: 'rgba(69, 123, 157, 0.2)',
            borderColor: 'rgba(69, 123, 157, 0.8)',
            borderWidth: 2,
            pointBackgroundColor: 'rgba(69, 123, 157, 1)'
```

```
            }, {
                label: 'Predators',
                data: [100, 40, 2, 20, 85],
                backgroundColor: 'rgba(230, 57, 70, 0.2)',
                borderColor: 'rgba(230, 57, 70, 0.8)',
                borderWidth: 2,
                pointBackgroundColor: 'rgba(230, 57, 70, 1)'
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            scales: {
                r: {
                    beginAtZero: true,
                    ticks: {
                        display: false
                    }
                }
            }
        }
    });
}

// Notification System
function showNotification(animal, location, imageUrl) {
    notificationTitle.textContent = animal + ' Detected';
    notificationLocation.textContent = 'Location: ' + location;
    notificationMessage.textContent = 'Wildlife has been detected in the park. Would you like
to notify visitors in the area?';
    notificationImg.src = imageUrl;
    notificationImg.alt = animal;
    notificationOverlay.classList.remove('hidden');
}

function hideNotification() {
    notificationOverlay.classList.add('hidden');
}

function showAlert() {
    hideNotification();
    alertBox.classList.remove('hidden');

    // Automatically hide alert after 3 seconds
    setTimeout(() => {
        alertBox.classList.add('hidden');
    }, 5000);
}

function hideAlert() {
```

```
    alertBox.classList.add('hidden');
}

// Event Listeners
function setupEventListeners() {
    // Notification buttons
    notifyButtons.forEach(button => {
        button.addEventListener('click', () => {
            const animal = button.getAttribute('data-animal');
            const location = button.getAttribute('data-location');
            const imageUrl = button.getAttribute('data-image');
            showNotification(animal, location, imageUrl);
        });
    });

    // Notification actions
    notificationSendBtn.addEventListener('click', showAlert);
    notificationCancelBtn.addEventListener('click', hideNotification);
    alertCloseBtn.addEventListener('click', hideAlert);
}

// Resize charts when window is resized
function handleResize() {
    if (hourlyChart) hourlyChart.resize();
    if (speciesChart) speciesChart.resize();
    if (movementChart) movementChart.resize();
    if (regionChart) regionChart.resize();
}

// Initialize dashboard
function initDashboard() {
    updateDateTime();
    setInterval(updateDateTime, 60000); // Update time every minute

    initCharts();
    setupEventListeners();

    window.addEventListener('resize', handleResize);
}

// Start dashboard when page is loaded
document.addEventListener('DOMContentLoaded', initDashboard);
```

5.2.3 style sheet

stylesheet code:

```
/* === BASE STYLES === */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

:root {
    --primary-color: #1d3557;
    --secondary-color: #457b9d;
    --accent-color: #e63946;
    --background-color: #f1faee;
    --card-color: #fff;
    --text-color: #333;
    --border-radius: 8px;
    --box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}

body {
    background-color: var(--background-color);
    color: var(--text-color);
    min-height: 100vh;
    overflow-x: hidden;
}

.dashboard-container {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
```

```
    width: 100%;
}

/* === HEADER === */
header {
    background-color: var(--primary-color);
    color: white;
    padding: 1rem 2rem;
    display: flex;
    justify-content: space-between;
    align-items: center;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
}

.logo h1 {
    font-size: 1.5rem;
    font-weight: 600;
}

.user-info {
    display: flex;
    flex-direction: column;
    align-items: flex-end;
}

.username {
    font-weight: 600;
}

.date-time {
    font-size: 0.9rem;
    opacity: 0.9;
}
```

```
/* === MAIN CONTENT === */
main {
    flex: 1;
    padding: 1.5rem;
    display: flex;
    flex-direction: column;
    gap: 1.5rem;
}

.dashboard-content {
    display: flex;
    flex-direction: column;
    gap: 1.5rem;
}

.analytics-section {
    background-color: var(--card-color);
    border-radius: var(--border-radius);
    padding: 1rem;
    box-shadow: var(--box-shadow);
}

.section-title {
    color: var(--primary-color);
    font-size: 1.1rem;
    margin-bottom: 1rem;
    border-bottom: 1px solid #eee;
    padding-bottom: 0.5rem;
}

/* === STATS CARDS === */
.stats-container {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
```

```
    gap: 1.5rem;
}

.stat-card {
    background-color: var(--card-color);
    border-radius: var(--border-radius);
    padding: 1.25rem;
    box-shadow: var(--box-shadow);
    display: flex;
    flex-direction: column;
    gap: 0.75rem;
}

.stat-card h3 {
    color: var(--secondary-color);
    font-size: 1rem;
    font-weight: 500;
}

.stat-value {
    font-size: 2rem;
    font-weight: 700;
    color: var(--primary-color);
}

.stat-trend {
    font-size: 0.85rem;
    display: flex;
    align-items: center;
}

.stat-trend.positive {
    color: #38b000;
}
```

```
.stat-trend.negative {
    color: var(--accent-color);
}

.stat-trend.neutral {
    color: #777;
}

/* === CHART GRID === */
.chart-grid {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    gap: 1rem;
    margin-bottom: 1rem;
}

.chart-container {
    background-color: var(--card-color);
    border-radius: var(--border-radius);
    padding: 0.75rem;
    box-shadow: var(--box-shadow);
    height: 180px;
}

.chart-container h3 {
    color: var(--secondary-color);
    font-size: 0.85rem;
    font-weight: 500;
    margin-bottom: 0.5rem;
}

/* === RECENT ACTIVITY === */
.recent-activity {
```

```
    background-color: var(--card-color);
    border-radius: var(--border-radius);
    padding: 1.25rem;
    box-shadow: var(--box-shadow);
}

.recent-activity h3 {
    color: var(--secondary-color);
    font-size: 1.1rem;
    margin-bottom: 1rem;
}

.activity-list {
    display: flex;
    flex-direction: column;
    gap: 1rem;
}

.activity-item {
    display: flex;
    gap: 1rem;
    padding: 0.75rem;
    border-radius: var(--border-radius);
    background-color: rgba(240, 240, 240, 0.5);
}

.activity-image img {
    width: 120px;
    height: 90px;
    object-fit: cover;
    border-radius: 4px;
}

.activity-details {
```

```
    flex: 1;
}

.activity-details h4 {
    font-size: 1rem;
    margin-bottom: 0.25rem;
}

.activity-details p {
    font-size: 0.9rem;
    color: #555;
    margin-bottom: 0.5rem;
}

.activity-time {
    font-size: 0.8rem;
    color: #777;
}

.activity-actions {
    display: flex;
    align-items: center;
}

.btn-notify {
    background-color: var(--accent-color);
    color: white;
    border: none;
    padding: 0.5rem 1rem;
    border-radius: 4px;
    cursor: pointer;
    font-weight: 500;
    transition: background-color 0.2s;
}
```

```
.btn-notify:hover {
    background-color: #d62828;
}

/* === NOTIFICATION OVERLAY === */
#notification-overlay {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.7);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 100;
}

.notification-container {
    background-color: white;
    border-radius: var(--border-radius);
    width: 600px;
    max-width: 90%;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
    overflow: hidden;
}

.notification-container h2 {
    background-color: var(--accent-color);
    color: white;
    padding: 1rem;
    text-align: center;
    font-size: 1.5rem;
```

```
}

.notification-content {
    padding: 1.5rem;
    display: flex;
    gap: 1.5rem;
}

.notification-image img {
    width: 250px;
    height: 200px;
    object-fit: cover;
    border-radius: 4px;
}

.notification-details {
    flex: 1;
}

.notification-details h3 {
    font-size: 1.3rem;
    margin-bottom: 0.75rem;
    color: var(--primary-color);
}

.notification-details p {
    margin-bottom: 0.5rem;
    line-height: 1.5;
}

.notification-actions {
    display: flex;
    padding: 1rem;
    border-top: 1px solid #eee;
```

```
    gap: 1rem;
    justify-content: flex-end;
}

.notification-actions button {
    padding: 0.75rem 1.25rem;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-weight: 500;
}

#notification-send {
    background-color: var(--accent-color);
    color: white;
}

#notification-send:hover {
    background-color: #d62828;
}

#notification-cancel {
    background-color: #eee;
    color: #666;
}

/* === ALERT BOX === */
#alert-box {
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background-color: #d62828;
    color: white;
```

```
    border-radius: var(--border-radius);
    padding: 2rem;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
    text-align: center;
    z-index: 200;
    width: 400px;
    max-width: 90%;
}

.alert-content h2 {
    font-size: 1.8rem;
    margin-bottom: 1rem;
}

.alert-content p {
    margin-bottom: 1.5rem;
    font-size: 1.1rem;
}

#alert-close {
    background-color: white;
    color: var(--accent-color);
    border: none;
    padding: 0.75rem 2rem;
    border-radius: 4px;
    cursor: pointer;
    font-weight: 600;
    font-size: 1rem;
}

/* === UTILITY CLASSES === */
.hidden {
    display: none !important;
}
```

```
/* === RESPONSIVE ADJUSTMENTS === */
@media (max-width: 1200px) {
    .stats-container {
        grid-template-columns: repeat(2, 1fr);
    }
}

@media (max-width: 1100px) {
    .chart-grid {
        grid-template-columns: repeat(2, 1fr);
    }
}

@media (max-width: 700px) {
    .chart-grid {
        grid-template-columns: 1fr;
    }
}

@media (max-width: 700px) {
    .stats-container {
        grid-template-columns: 1fr;
    }

    .activity-item {
        flex-direction: column;
    }

    .notification-content {
        flex-direction: column;
    }

    .notification-image img {
```

```
        width: 100%;
        height: auto;
      }
    }
```

5.2.4 Machine learning

Machine learning :

Training code:

```
# Big Cats Detection System
# This code detects lions, cheetahs, and tigers from camera footage
# (leopards are excluded from detection)

import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# Define constants
IMG_SIZE = 224  # MobileNetV2 default input size
BATCH_SIZE = 32
EPOCHS = 20
DATASET_PATH = "big_cats_dataset"  # Path to the downloaded dataset
MODEL_PATH = "big_cats_model.h5"

# Function to load and prepare the dataset
def prepare_dataset(dataset_path):
    # Data augmentation for training
```

```
    train_datagen = ImageDataGenerator(
        preprocessing_function=preprocess_input,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        validation_split=0.2
    )

    # Load training dataset
    train_generator = train_datagen.flow_from_directory(
        dataset_path,
        target_size=(IMG_SIZE, IMG_SIZE),
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        subset='training'
    )

    # Load validation dataset
    validation_generator = train_datagen.flow_from_directory(
        dataset_path,
        target_size=(IMG_SIZE, IMG_SIZE),
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        subset='validation'
    )

    return train_generator, validation_generator

# Function to create the model
def create_model(num_classes):
    # Use MobileNetV2 as the base model for transfer learning
```

```
    base_model = MobileNetV2(
        weights='imagenet',
        include_top=False,
        input_shape=(IMG_SIZE, IMG_SIZE, 3)
    )

    # Freeze the base model layers
    base_model.trainable = False

    # Create the model
    model = Sequential([
        base_model,
        tf.keras.layers.GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    # Compile the model
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

# Function to train the model
def train_model(model, train_generator, validation_generator):
    # Define callbacks
    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True
```

```
    )

    model_checkpoint = ModelCheckpoint(
        MODEL_PATH,
        monitor='val_accuracy',
        save_best_only=True
    )

    # Train the model
    history = model.fit(
        train_generator,
        epochs=EPOCHS,
        validation_data=validation_generator,
        callbacks=[early_stopping, model_checkpoint]
    )

    return history, model

def detect_from_camera(model, class_indices):
    # Reverse the class indices dictionary to map indices to class names
    class_names = {v: k for k, v in class_indices.items()}

    # Start video capture
    cap = cv2.VideoCapture(0)  # 0 for default camera

    # Higher confidence threshold as requested
    CONFIDENCE_THRESHOLD = 0.985  # Only detect if confidence is above 97%

    # Flag to track whether an animal is currently being detected
    animal_detected = False

    while True:
        ret, frame = cap.read()
        if not ret:
```

```
            break

        # Prepare the frame for prediction
        resized_frame = cv2.resize(frame, (IMG_SIZE, IMG_SIZE))
        preprocessed_frame = preprocess_input(resized_frame)
        input_data = np.expand_dims(preprocessed_frame, axis=0)

        # Make prediction
        prediction = model.predict(input_data)
        predicted_class_idx = np.argmax(prediction[0])
        confidence = prediction[0][predicted_class_idx]
        predicted_class = class_names[predicted_class_idx]

        # Skip detection if the predicted class is "leopard"
        if predicted_class.lower() not in ["lion","tiger","cheetah"]:
            text = "No Animal Detected"
            color = (0, 0, 255)  # Red for not detected

            # Update terminal message if state changed
            if animal_detected:
                print("No animals detected")
                animal_detected = False
        # Only show label if confidence is high and it's not a leopard
        elif confidence >= CONFIDENCE_THRESHOLD:
            text = f"{predicted_class}: {confidence * 100:.2f}%"
            color = (0, 255, 0)  # Green for detected

            # Update terminal message if state changed
            if not animal_detected:
                print(f"Animal detected: {predicted_class} with confidence {confidence *
100:.2f}%")
                animal_detected = True
        else:
            text = "No Animal Detected"
```

```
            color = (0, 0, 255)  # Red for not detected

            # Update terminal message if state changed
            if animal_detected:
                print("No animals detected")
                animal_detected = False

        # Display the result on the frame
        cv2.putText(frame, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
                    1, color, 2, cv2.LINE_AA)

        # Show the frame
        cv2.imshow('Big Cats Detection', frame)

        # Exit on 'q' key
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

# Main function
def main():
    print("Loading and preparing the dataset...")
    train_generator, validation_generator = prepare_dataset(DATASET_PATH)

    num_classes = len(train_generator.class_indices)
    print(f"Found {num_classes} classes: {train_generator.class_indices}")

    print("Creating and training the model...")
    model = create_model(num_classes)
    history, trained_model = train_model(model, train_generator, validation_generator)

    print("Starting live detection. Press 'q' to quit.")
```

```
    detect_from_camera(trained_model, train_generator.class_indices)

if _name_ == "_main_":
    main()
```

5.2.5 Load and run code:

```
# Big Cats Detection System - Load and Run
# This code loads a pre-trained model and detects lions, cheetahs, and tigers
from camera footage
# (all other animals including leopards are excluded from detection)

import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

# Define constants
IMG_SIZE = 224  # MobileNetV2 default input size
MODEL_PATH = "big_cats_model.h5"  # Path to your saved model

def detect_from_camera(model, class_names):
    # Start video capture
    cap = cv2.VideoCapture(0)  # 0 for default camera

    # Higher confidence threshold as requested
    CONFIDENCE_THRESHOLD = 0.985 # Only detect if confidence is above
97%
```

```
    # Flag to track whether an animal is currently being detected
    animal_detected = False

    print("Starting detection. Press 'q' to quit.")

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Failed to grab frame from camera. Exiting...")
            break

        # Prepare the frame for prediction
        resized_frame = cv2.resize(frame, (IMG_SIZE, IMG_SIZE))
        preprocessed_frame = preprocess_input(resized_frame)
        input_data = np.expand_dims(preprocessed_frame, axis=0)

        # Make prediction
        prediction = model.predict(input_data, verbose=0)  # Set verbose=0 to
avoid printing prediction info
        predicted_class_idx = np.argmax(prediction[0])
        confidence = prediction[0][predicted_class_idx]
        predicted_class = class_names[predicted_class_idx]

        # Skip detection if the predicted class is not lion, tiger, or cheetah
        if predicted_class.lower() not in ["lion", "tiger", "cheetah"]:
            text = "No Animal Detected"
            color = (0, 0, 255)  # Red for not detected
```

```
            # Update terminal message if state changed
            if animal_detected:
                print("No animals detected")
                animal_detected = False
        # Only show label if confidence is high and it's a target animal
        elif confidence >= CONFIDENCE_THRESHOLD:
            text = f"{predicted_class}: {confidence * 100:.2f}%"
            color = (0, 255, 0)  # Green for detected

            # Update terminal message if state changed
            if not animal_detected:
                print(f"Animal detected: {predicted_class} with confidence
{confidence * 100:.2f}%")
                animal_detected = True
        else:
            text = "No Animal Detected"
            color = (0, 0, 255)  # Red for not detected

            # Update terminal message if state changed
            if animal_detected:
                print("No animals detected")
                animal_detected = False

        # Display the result on the frame
        cv2.putText(frame, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
                    1, color, 2, cv2.LINE_AA)

        # Show the frame
        cv2.imshow('Big Cats Detection', frame)
```

```
        # Exit on 'q' key
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

def main():
    # Check if model file exists
    if not os.path.exists(MODEL_PATH):
        print(f"Error: Model file '{MODEL_PATH}' not found. Please make sure you've trained the model first.")
        return

    print(f"Loading trained model from {MODEL_PATH}...")
    try:
        # Load the trained model
        model = load_model(MODEL_PATH)
        print("Model loaded successfully!")

        # Define the class names - must match your training data
        class_names = {0: "cheetah", 1: "lion", 2: "tiger"}

        # You may need to adjust the class_names dictionary based on your actual training data
        # If you're unsure of the exact mapping, you can print the class indices in the terminal
        # and adjust this script manually for the first run
```

```
        print("Using class mapping:", class_names)
        print("Note: If detection results are incorrect, you may need to adjust the
class_names dictionary.")

        # Start detection
        detect_from_camera(model, class_names)

    except Exception as e:
        print(f"Error loading or using the model: {e}")

if _name_ == "_main_":
    main()
```

5.2.5 CV handler code:

```
import serial
import time
import subprocess

# Replace with your actual COM port (check Arduino IDE or Device Manager)
ser = serial.Serial('COM7', 115200, timeout=1)
time.sleep(2)  # wait for serial connection to stabilize

print("Listening for command...")

while True:
    if ser.in_waiting:
        line = ser.readline().decode(errors='ignore').strip()

        print("Received:", line)
```

```
        if line == "RUN_CV":
            print("Running computer vision model...")

            # Option 1: Call your actual Python script
            result = subprocess.run(["python", "loadandrun.py"],
capture_output=True, text=True)

            # Extract result from output (edit this if needed)
            prediction = result.stdout.strip()
            print("Model output:", prediction)

            # Option 2 (Simplified for testing):
            # prediction = "animal"  # or "not_animal"

            # Send back to Arduino
            ser.write((prediction + "\n").encode())
```

CHAPTER 6

SCREEN SHOTS

1. Dashboard Page

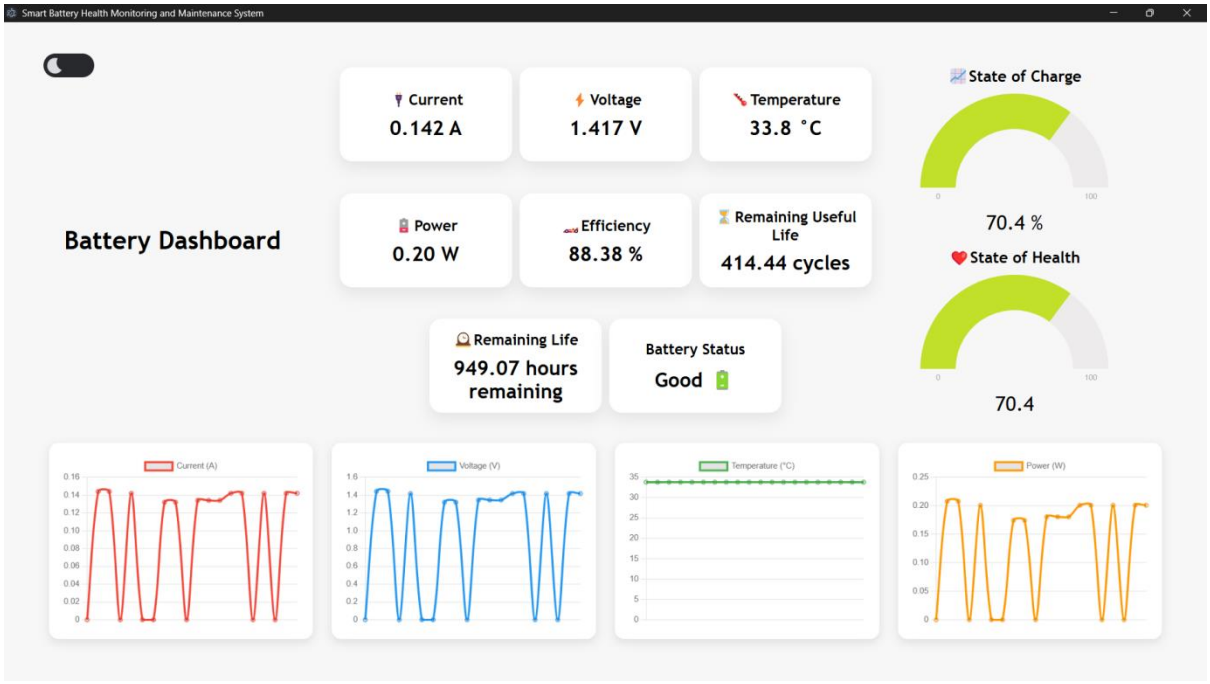

Figure 6.1 Responsive Dashboard

2. Data Sent Arduino via Serial Port

```
Current: 0.152 A | Voltage: 1.515 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
Current: 0.152 A | Voltage: 1.515 V | Temp: 31.8 °C
Current: 0.152 A | Voltage: 1.515 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
Current: 0.149 A | Voltage: 1.491 V | Temp: 31.8 °C
```

Figure 6.2 Data Sent Arduino via Serial Port

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

The Smart Battery Monitoring and Maintenance System successfully demonstrates a real-time solution for tracking critical battery parameters such as voltage, current, and temperature. Through seamless integration of sensors, microcontroller, and a desktop dashboard interface, the system enables accurate monitoring, data visualization, and intelligent battery health estimation. The inclusion of State of Charge (SoC), State of Health (SoH), and Remaining Useful Life (RUL) predictions enhances battery reliability and reduces the risk of unexpected failures. This project highlights the potential of IoT and machine learning in ensuring safe and efficient battery usage across various applications.

In the future, the Smart Battery Monitoring and Maintenance System can be enhanced by integrating cloud-based storage and access to enable remote monitoring, historical data analysis, and improved data security. A dedicated mobile application could be developed to provide real-time updates and alerts to users on the go. The system can also be scaled to support monitoring of multiple batteries simultaneously, making it suitable for larger industrial or commercial battery banks. Furthermore, implementing advanced machine learning models trained on extensive datasets can improve the accuracy of predicting State of Health (SoH) and Remaining Useful Life (RUL). Additional features such as automatic battery type detection, fault prediction alerts, and integration with renewable energy systems could significantly broaden the system’s usability and impact.

REFERENCES

1. A. Haraz, K. Abualsaud, and A. Massoud, "State-of-Health and State-of-Charge Estimation in Electric Vehicles Batteries: A Survey on Machine Learning Approaches," IEEE Access, 2024. https://doi.org/10.1109/ACCESS.2024.3486989
2. R. Ranjith Kumar, C. Bharatiraja, K. Udhayakumar, S. Devakirubakaran, K. Sathiya Sekar, and Lucian Mihet-Popa, "Advances in Batteries, Battery Modeling, Battery Management System, Battery Thermal Management, SOC, SOH, and Charge/Discharge Characteristics in EV Applications," IEEE Transactions on Industrial Electronics, 2024.
3. G. Krishna et al., "Advanced battery management system enhancement using IoT and ML for predicting remaining useful life in Li-ion batteries," Scientific Reports, vol. 14, 30394, 2024. https://doi.org/10.1038/s41598-024-80719-1
4. Y. Zheng et al., "Thermal state monitoring of lithium-ion batteries: Progress, challenges, and opportunities," Progress in Energy and Combustion Science, vol. 100, 101120, 2023. https://doi.org/10.1016/j.pecs.2023.101120