INDUSTRIAL WEB ROVER

PROJECT REPORT

Submitted by

K. V. Nihal Reddy (CB.SC.U4AIE23239)

V. Sai Ved (CB.SC.U4AIE23262)

U. Aditya Sandeep (CB.SC.U4AIE23257)

Hari Vaarthan (CB.SC.U4AIE23228)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

(ARTIFICIAL INTELLIGENCE)

COMPUTER SCIENCE ENGINEERING- ARTIFICIAL INTELIIGENCE

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE- 641 112 (INDIA)

APRIL- 2025

BONAFIDE CERTIFICATE

This is to certify that the thesis entitled "**Industrial Web Rover"** submitted by **Nihal Reddy (CB.SC.U4AIE23239)**, **Sai Ved(CB.SC.U4AIE23262)**, **Aditya Sandeep (CB.SC.U4AIE23257)**, **Hari Vaarthan(CB.SC.U4AIE23228)** for the award of the Degree of Bachelor of Technology in the **"COMPUTER SCIENCE ENGINEERING - ARTIFICIAL INTELLIGENCE"** for courses **22MAT230** and **22AIE214** is a Bonafide record of the work carried out by them under my guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

Dr. D. Palmani Sir

Project Guide

Dr. Milton Mondal Sir

Project Guide

Sign of Faculty

Sign of Faculty

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE- 641 112

DECLARATION

We, **Nihal Reddy (CB.SC.U4AIE23239)**, **Sai Ved(CB.SC.U4AIE23262)**, **Aditya Sandeep (CB.SC.U4AIE23257)**, **Hari Vaarthan(CB.SC.U4AIE23228)** hereby declare that this thesis entitled **Industrial Web Rover**, is the record of the original work done by us under the guidance of **Dr. D. Palmani sir** and **Dr. Milton Mondal**, Amrita School of Artificial Intelligence, Coimbatore. To the best of our knowledge, this work has not formed the basis for the award of any degree/ diploma/ associateship/ fellowship/ or a similar award to any candidate in any university.

Place: Coimbatore Signature of the Student

Date: 21/04/2025

COUNTERSIGNED

Dr. K. P. Soman

Professor and Dean

Amrita School of Artificial Intelligence

Amrita Vishwa Vidyapeetham

Table of Contents

Acknowledgement

We would like to express our sincere appreciation to all those who have helped and guided us step by step throughout this project. Most importantly, we are significantly grateful to our project guide, **Dr. D. Palmani Sir** and **Dr. Milton Mondal**, for their constant encouragement, expert advice, and constructive criticism. Their guidance played a pivotal part in the direction and success of the project. We would also like to appreciate the staff and faculty at Amrita School of Artificial Intelligence, Amrita Vishwa Vidyapeetham, Coimbatore, for providing us with the facilities, materials, and learning environment to deliver our research went without a hitch to completion. Second, we appreciate our peer and teammate for their cooperation, motivation, and zeal. Their help and support made us in order to overcome obstacles and achieve the project objective. Lastly, we would like to thank our families for supporting us throughout, for being patient with us, and for believing in us along the way during our learning process. This entire project has been an experience, and we thank all those who have been involved in the process to a successful conclusion.

Abstract

Industrial Web Rover is a multi-functional robotic system designed for real time industrial monitoring in hazardous and dangerous environment. The rover is remotely controlled through a web interface, which enables directional travel and live video feed for visual inspection. In order to provide better environment safety, the rover is equipped with a gas sensor that detects toxic gases and displays real time concentrations on the web interface.

One of the major features of the system is that it can detect structural cracks on industrial equipment using a deep learning model. For this purpose, a Convolutional Neural Network (CNN) was optimized and trained using the ADAM algorithm to obtain fast convergence and high accuracy. The trained model was then compiled into TensorFlow Lite format for efficient deployment on a Raspberry Pi, which can carry out real-time inference on the rover itself.

Chapter 1

1.Introduction

In the modern industrial era, it is very crucial to maintain equipment in healthy conditions to promote safety and smooth operations of machinary. However, manual examination - particularly in hazardous places - is time consuming, expensive, and even risky. That is where this project enters, an Industrial Web Rover, a robot that remotely inspects and monitors industrial places.

One of the features of this rover that is good is that it can identify cracks on the surface of machines via deep learning. At its core is a Convolutional Neural Network (CNN) that is trained to differentiate between images that have cracks and those that don't - really a binary classification problem. The model is trained with the ADAM optimizer, which accelerates and stabilizes training by intelligently scaling learning rates from gradient information. After training, the model is converted to TensorFlow Lite format and deployed on a Raspberry Pi, enabling real-time crack detection where it's needed.

In the background, optimization methods such as gradient descent, stochastic gradient descent, and ADAM are responsible for how the model learns. These processes assist the network in reducing its loss function and becoming more precise with time. Through the union of theory in optimization and hands-on application, this project demonstrates how classroom theory can directly drive real-world applications in industrial automation and safety.

Table 1.1: Literature Review Summary

Paper Title and Authors	***Key Contributions***	***Methodology and Relevance***
Automatic Crack Detection on Concrete Surfaces Using Convolutional Neural Networks	Proposed a CNN model to automatically detect cracks in concrete structures with high accuracy.	Used a deep CNN architecture trained on labeled crack images. Relevant to this project as it establishes CNN effectiveness in structural defect detection.
Real-Time Surface Crack Detection Using MobileNet and TensorFlow Lite	Demonstrated deployment of a lightweight CNN on embedded devices using TensorFlow Lite.	Applied transfer learning and optimized MobileNet with TensorFlow Lite for deployment on mobile devices. Highly relevant for real-time Raspberry Pi inference.
A Review on Deep Learning Techniques for Surface Crack Detection	Surveyed multiple deep learning approaches and their performance for crack detection.	Highlights the evolution from traditional image processing to CNNs, reinforcing the decision to use CNNs in this project.
ADAM: A Method for Stochastic Optimization	Introduced the ADAM optimizer for training deep neural networks, combining advantages of RMSProp and momentum.	Core to your CNN model training; ADAM improves convergence speed and accuracy. Highly relevant for training models on limited datasets efficiently.
Optimizing Deep Learning Models for Edge Devices	Discussed strategies for compressing and optimizing deep learning models for deployment on edge hardware.	Covered quantization and model pruning techniques, directly supporting the use of TensorFlow Lite for Raspberry Pi in your project.

- **Automatic Crack Detection on Concrete Surfaces Using Convolutional Neural Networks (Zhang)** applied deep CNN to effectively detect cracks on concrete but did not consider deploying on resource-limited devices such as Raspberry Pi.
- **Real-Time Surface Crack Detection with MobileNet and TensorFlow Lite (Sharma)** demonstrated a light-weight MobileNet powered model optimized using TensorFlow Lite for embedded applications but less robotic application than mobile one.
- **Review on Deep Learning Techniques for Surface Crack Detection (Kumar)** gave an in-depth review of CNN-based techniques used in crack detection, but without details of implementation for usage in real-time field applications.
- **A Method for Stochastic Optimization (Kingma and J.Ba)** proposed an adaptive optimizer to train deep networks efficiently, but the paper isn't directly focused on embedded deployment issues.
- **Optimization of Deep Learning Models for Edge Devices (Nguyen and Zhou)** talked about compression and quantization methods to deploy DL models on edge devices such as Raspberry Pi but didn't address domain-specific tasks such as crack detection in the industrial sector.

1.1 Problem Statement

Industries often face challenges in performing regular inspections of machinery and infrastructure, especially in environments that are dangerous and difficult to access. Manual inspection methods are time consuming and risks and may lead to delayed detection of structural faults such as surface cracks.

There is a critical need for an automated, real time, and remote monitoring system that can identify such defects efficiently. Additionally, deploying deep learning models for detection in real-time environments requires optimization techniques to ensure lightweight performance and high accuracy, especially on resource-constrained devices like the Raspberry Pi.

1.2 Objectives

• To implement and deploy a web-based rover to be driven remotely and live video streamed for industrial inspection.

• To incorporate a Convolutional Neural Network (CNN) to detect cracks in real-time on industrial surfaces.

• To tune the CNN with methods like Stochastic Gradient Descent (SGD) and ADAM and plot their performances against training metrics.

• To implement the optimized CNN model in TensorFlow Lite model on Raspberry Pi to facilitate real-time inference in the field.

1.3. Organization of Report

The structure of this report is outlined below:

• **Chapter 1:** Introduction Provides the background, literature survey, problem statement, and objectives of the project.

• **Chapter 2:** Proposed Work Describes the architecture, methodology, system design, sensing and execution strategies.

• **Chapter 3:** Results and Discussion Presents the experimental setup, performance analysis, and observations from various tests.

• **Chapter 4:** Conclusion and Future Work Summarizes the achievements, highlights contributions, and suggests directions for future improvement.

Chapter 2

2. Proposed Work

2.1. System Architecture

The Industrial Web Rover system integrates mechanical control, real-time video streaming, environmental sensing, and deep learning-based defect detection into a Raspberry Pi-powered module. It consists of:

- A **Raspberry Pi** that acts as the central controller.
- A **motor driver (MDD10A)** to control left and right motors for rover mobility.
- A **USB camera** or Pi camera for live video feed.
- **Gas sensors** to monitor hazardous gases.
- A **trained CNN model in TensorFlow Lite format** for crack detection.
- A **Flask-based web server** that hosts the user interface for both video feed and directional control.

2.2. Execution Strategy

- **Startup:** Raspberry Pi initializes Flask server, camera stream, and motor control scripts.
- **Login Interface:** Authenticates the user before showing the control panel.
- **User Control:** Receives movement commands via web Buttons.
- **Streaming:** Displays real-time video and processes occasional frames through TFLite crack detection.
- **Sensing:** Reads gas sensor values periodically and displays them on the interface.
- **Feedback Loop:** Logs command feedback, anomalies, and sensor alerts.

2.3. Methodology

2.3.1. Web Interface & Control

A responsive web interface (via Flask) allows authenticated users to:

- View live video from the rover camera.
- Control the rover's movement (forward, backward, left, right, stop) through buttons or keyboard input.
- Receive sensor data and crack detection results.

2.3.2. Motor Movement

- MDD10A.py handles PWM control of two motors using GPIO pins.
- Motor direction and speed are adjusted via route-based commands (ex: left, right) received from the web interface.

2.3.3. Live Streaming

- A camera stream is accessed via OpenCV and transmitted to the web interface using Flask's Response object with a multipart MJPEG stream (video_feed endpoint).

2.3.4. Gas Sensing & Crack Detection

- A gas sensor is connected to the Raspberry Pi via GPIO to fetch gas concentration.
- A pre-trained CNN model converted to .tflite is deployed on the Raspberry Pi for detecting cracks from the video stream or captured images.

2.3.5. Integration with Linear Algebra & Optimization

- The crack detection CNN leverages ADAM optimizer during training for efficient convergence.
- Feature extraction and pre-processing are inspired by Toeplitz filters (convolution-like operations in CNNs).
- Spectral clustering techniques can be optionally applied for grouping detected crack patterns.
- Potential future integration includes compressed sensing for efficient video transmission and low-rank matrix completion for reconstructing noisy sensor data.

2.4. Workflow

2.4.1. Web interface and User design

- A user accesses the rover system through a web browser.
- The homepage presents a **login screen** (login.html) for authentication.
- Only users with correct credentials (admin/password) are allowed to access the control panel (stream.html).

2.4.2. Live Video Streaming

- Upon successful login, a **live video feed** from the USB camera is displayed using OpenCV.
- The video feed route continuously captures frames using cv2.VideoCapture, encodes them as JPEG, and streams them using **MJPEG** format to the browser.

2.4.3. Rover Movement Control

- The control panel has **directional buttons** (Forward, Backward, Left, Right, Stop).
- When a button is clicked (or a keyboard arrow key is pressed), a **GET request** is sent to the server (ex: /down).
- Based on the route (direction), the Flask app calculates motor speeds and sends commands to the **MDD10A motor driver** using GPIO.

2.4.4. Motor Driver Control

- GPIO pins on the Raspberry Pi are configured to control direction and speed of **left and right motors**.
- **PWM signals** (0 to 100%) are generated for motor speed control.
- Direction is determined based on the sign of the input power (positive = forward, negative = reverse).

2.4.5. Crack Detection using CNN

- A CNN model is trained on 120x120 grayscale images of concrete surfaces to detect cracks.
- The model uses layers like Conv2D, MaxPooling2D, GlobalAveragePooling2D, and is converted to TensorFlow Lite (TFLite) for edge deployment.
- The model is integrated into the Raspberry Pi and will analyze captured frames to detect cracks on industrial equipment.

2.4.6. Sensor Data Monitoring

- The rover will include **gas and temperature sensors**.
- Sensor readings will be displayed on the same webpage in real-time using Flask.
- Future scope includes **data fusion techniques** (like Compressed Sensing, Low-Rank Matrix Completion) for noise reduction and better signal interpretation.

2.5. Codes

2.5.1. app.py:

```
from flask import Flask, render_template, request, Response
import cv2
import socket
import MDD10A as HBridge
import time

app = Flask(__name__)

# User authentication
USERNAME = "admin"
PASSWORD = "password"

# Motor speed variables
speedleft = 0
speedright = 0

# Set camera FPS and resolution
FPS = 30
FRAME_DELAY = 1 / FPS
FRAME_WIDTH = 420    # Lower resolution for faster streaming
FRAME_HEIGHT = 240

# Generate video frames for streaming
def gen_frames():
    cap = cv2.VideoCapture(0, cv2.CAP_V4L2)  # Force Video4Linux2
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, FRAME_WIDTH)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT)
    cap.set(cv2.CAP_PROP_FPS, FPS)
    cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)

    while cap.isOpened():
        success, frame = cap.read()
        if not success:
            break
        else:
            ret, buffer = cv2.imencode('.jpg', frame, [cv2.IMWRITE_JPEG_QUALITY, 70])
            frame = buffer.tobytes()
            yield (
                b'--frame\r\n'
                b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n'
            )
            time.sleep(FRAME_DELAY)

    cap.release()
```

```python
# Login route
@app.route('/', methods=['GET', 'POST'])
def index():
    error = None
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username == USERNAME and password == PASSWORD:
            return render_template('stream.html')
        else:
            error = 'Invalid credentials. Please try again.'
    return render_template('login.html', error=error)

# Video feed route
@app.route('/video_feed')
def video_feed():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

# Motor control route
@app.route('/<direction>')
def move(direction):
    global speedleft, speedright

    speeds = {
        "up": (0.02, -0.02),
        "down": (-0.02, 0.02),
        "stop": (0, 0),
        "right": (0.02, 0.02),
        "left": (-0.02, -0.02)
    }

    speedleft, speedright = speeds.get(direction, (0, 0))
    HBridge.setMotorLeft(speedleft)
    HBridge.setMotorRight(speedright)

    print(f"Command Received: {direction}")
    print(f"Command Sent to MDD10A.py - Left: {speedleft}, Right: {speedright}")
    return f"Key pressed: {direction}"

# Main entry point
if __name__ == '__main__':
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(("8.8.8.8", 80))
    ip_address = s.getsockname()[0]
    s.close()

    print(f"Access the Web Rover at: http://{ip_address}:5000")
    app.run(host=ip_address, port=5000, debug=True)
```

2.5.2. MDD10A.py

```python
import RPi.GPIO as io

# Use Broadcom pin numbering
io.setmode(io.BCM)
io.setwarnings(False)

# Constants
PWM_MAX = 100  # Maximum PWM value

# Motor control GPIO pins
leftMotor_DIR_pin = 22
rightMotor_DIR_pin = 23
leftMotor_PWM_pin = 17
rightMotor_PWM_pin = 18

# GPIO setup
io.setup(leftMotor_DIR_pin, io.OUT)
io.setup(rightMotor_DIR_pin, io.OUT)
io.setup(leftMotor_PWM_pin, io.OUT)
io.setup(rightMotor_PWM_pin, io.OUT)

# Set initial direction to False (neutral)
io.output(leftMotor_DIR_pin, False)
io.output(rightMotor_DIR_pin, False)

# Initialize PWM at 1kHz frequency
leftMotorPWM = io.PWM(leftMotor_PWM_pin, 1000)
rightMotorPWM = io.PWM(rightMotor_PWM_pin, 1000)

# Start PWM with 0% duty cycle
leftMotorPWM.start(0)
rightMotorPWM.start(0)

# Initial motor power
leftMotorPower = 0
rightMotorPower = 0

# Get current motor power levels
def getMotorPowers():
    return (leftMotorPower, rightMotorPower)
```

```
# Set power for the left motor
def setMotorLeft(power):
    global leftMotorPower
    io.output(leftMotor_DIR_pin, power > 0)
    pwm = min(abs(int(PWM_MAX * power)), PWM_MAX)
    leftMotorPower = pwm
    leftMotorPWM.ChangeDutyCycle(pwm)

# Set power for the right motor
def setMotorRight(power):
    global rightMotorPower
    io.output(rightMotor_DIR_pin, power < 0)  # Adjusted logic for right direction
    pwm = min(abs(int(PWM_MAX * power)), PWM_MAX)
    rightMotorPower = pwm
    rightMotorPWM.ChangeDutyCycle(pwm)

# Stop motors and clean up GPIO
def exit():
    io.output(leftMotor_DIR_pin, False)
    io.output(rightMotor_DIR_pin, False)
    io.cleanup()
```

2.5.3. stream.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Web Rover Control</title>
    <style>
        body {
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            min-height: 100vh;
            margin: 0;
            font-family: 'Verdana', sans-serif;
            background-color: #1e1e2e;
            color: #ffffff;
        }
        h1 {
            margin-bottom: 20px;
        }
        .container {
            display: flex;
            width: 90%;
            max-width: 1000px;
            background: #282a36;
            border-radius: 12px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.3);
            padding: 20px;
            align-items: center;
            gap: 20px;
        }
        .video-feed {
            flex: 1;
            border-radius: 10px;
            border: 3px solid #ff79c6;
            overflow: hidden;
        }
        .video-feed img {
            width: 100%;
            display: block;
            border-radius: 10px;
        }
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Web Rover Control</title>
    <style>
        body {
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            min-height: 100vh;
            margin: 0;
            font-family: 'Verdana', sans-serif;
            background-color: #1e1e2e;
            color: #ffffff;
        }
        h1 {
            margin-bottom: 20px;
        }
        .container {
            display: flex;
            width: 90%;
            max-width: 1000px;
            background: #282a36;
            border-radius: 12px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.3);
            padding: 20px;
            align-items: center;
            gap: 20px;
        }
        .video-feed {
            flex: 1;
            border-radius: 10px;
            border: 3px solid #ff79c6;
            overflow: hidden;
        }
        .video-feed img {
            width: 100%;
            display: block;
            border-radius: 10px;
        }

<body>
    <h1>Web Rover Control</h1>
    <div class="container">
        <div class="video-feed">
            <img id="video" src="{{ url_for('video_feed') }}" alt="Live Stream">
        </div>
        <div class="controls">
            <button id="up" onclick="sendCommand('up')">Move Forward</button>
            <div class="direction-controls">
                <button id="left" onclick="sendCommand('left')">Left</button>
                <button id="stop" onclick="sendCommand('stop')">Stop</button>
                <button id="right" onclick="sendCommand('right')">Right</button>
            </div>
            <button id="down" onclick="sendCommand('down')">Move Backward</button>
        </div>
    </div>
</body>
</html>
```

2.5.4. login.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            background: linear-gradient(to right, #4facfe, rgb(185, 22, 226));
            margin: 0;
            font-family: 'Poppins', sans-serif;
        }
        .login-container {
            background: rgba(255, 255, 255, 0.9);
            padding: 40px;
            border-radius: 12px;
            box-shadow: 5px 5px 15px rgba(0, 0, 0, 0.2);
            text-align: center;
            width: 320px;
        }
        .login-container h2 {
            color: #333;
            margin-bottom: 20px;
        }
        .login-container label {
            display: block;
            margin: 10px 0 5px;
            font-weight: 500;
            color: #555;
        }
        .login-container input[type="text"],
        .login-container input[type="password"] {
            width: 100%;
            padding: 12px;
            margin-bottom: 15px;
            border-radius: 8px;
            border: 1px solid #ccc;
            outline: none;
            font-size: 14px;
        }
        .login-container input[type="text"]:focus,
        .login-container input[type="password"]:focus {
            border-color: #4facfe;
            box-shadow: 0 0 5px rgba(79, 172, 254, 0.5);
        }
```

```
        .login-container input[type="submit"] {
            width: 100%;
            padding: 12px;
            border: none;
            border-radius: 8px;
            background: #4facfe;
            color: #fff;
            font-size: 16px;
            font-weight: bold;
            cursor: pointer;
            transition: background 0.3s;
        }
        .login-container input[type="submit"]:hover {
            background: rgb(0, 38, 255);
        }
    </style>
</head>
<body>
    <div class="login-container">
        <h2>Login</h2>
        <form method="post">
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" required>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>
            <input type="submit" value="Login">
        </form>
    </div>
</body>
</html>
```

2.5.5. crack detection.py:

```python
import os
import cv2
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from google.colab import drive
```

```python
# Mount Drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```python
# Set dataset paths
dataset_path = "/content/drive/MyDrive/Concrete Crack Images for Classification"
positive_dir = f"{dataset_path}/Positive"
negative_dir = f"{dataset_path}/Negative"
toeplitz_path = "/content/processed_cracks"
toeplitz_pos = f"{toeplitz_path}/Positive"
toeplitz_neg = f"{toeplitz_path}/Negative"
os.makedirs(toeplitz_pos, exist_ok=True)
os.makedirs(toeplitz_neg, exist_ok=True)
```

```python
# Toeplitz preprocessing
def preprocess_images(source_dir, dest_dir, max_images=200):
    files = os.listdir(source_dir)[:max_images]
    for file in files:
        img_path = os.path.join(source_dir, file)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (120, 120))

        # Toeplitz filter kernel
        kernel_1d = np.array([1, 0, -1]) / 2
        toeplitz_matrix = np.outer(kernel_1d, kernel_1d)
        filtered_img = cv2.filter2D(img, -1, toeplitz_matrix)

        norm_img = cv2.normalize(filtered_img, None, 0, 255, cv2.NORM_MINMAX)
        cv2.imwrite(os.path.join(dest_dir, file), norm_img.astype(np.uint8))
```

```
[5]
# Apply to both classes
preprocess_images(positive_dir, toeplitz_pos)
preprocess_images(negative_dir, toeplitz_neg)
```

```
[6]
# Create DataFrame
def generate_df(image_dir, label):
    filepaths = pd.Series(list(Path(image_dir).glob("*.jpg")), name="Filepath").astype(str)
    labels = pd.Series(label, name="Label", index=filepaths.index)
    return pd.concat([filepaths, labels], axis=1)

positive_df = generate_df(toeplitz_pos, "POSITIVE")
negative_df = generate_df(toeplitz_neg, "NEGATIVE")
all_df = pd.concat([positive_df, negative_df], axis=0).sample(frac=1.0, random_state=1).reset_index(drop=True)
```

```
[7]
# Train/Test Split
train_df, test_df = train_test_split(all_df, train_size=0.7, random_state=1)
```

```
[8]
# Image Data Generators
train_gen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
test_gen = ImageDataGenerator(rescale=1./255)

train_data = train_gen.flow_from_dataframe(
    train_df, x_col="Filepath", y_col="Label", target_size=(120, 120),
    color_mode='grayscale', class_mode="binary", batch_size=32, shuffle=True, subset='training'
)
val_data = train_gen.flow_from_dataframe(
    train_df, x_col="Filepath", y_col="Label", target_size=(120, 120),
    color_mode='grayscale', class_mode="binary", batch_size=32, shuffle=True, subset='validation'
)
test_data = test_gen.flow_from_dataframe(
    test_df, x_col="Filepath", y_col="Label", target_size=(120, 120),
    color_mode='grayscale', class_mode="binary", batch_size=32, shuffle=False
)
```

```
Found 1632 validated image filenames belonging to 2 classes.
Found 408 validated image filenames belonging to 2 classes.
Found 875 validated image filenames belonging to 2 classes.
```

```
[9]
# CNN Model
inputs = tf.keras.Input(shape=(120, 120, 1))
x = tf.keras.layers.Conv2D(16, (3, 3), activation='relu')(inputs)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs, outputs)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

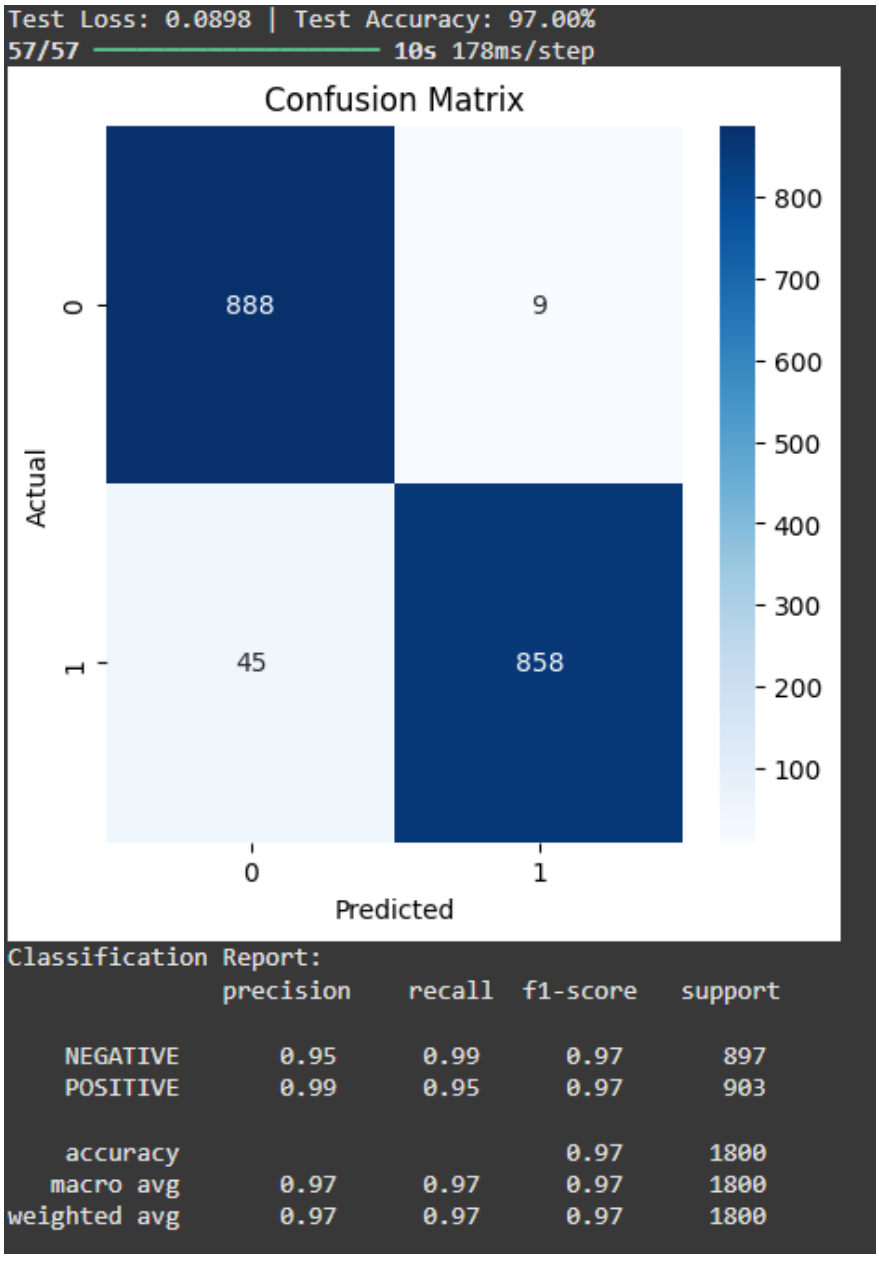
```python
# Train the Model
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=20,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)]
)
```

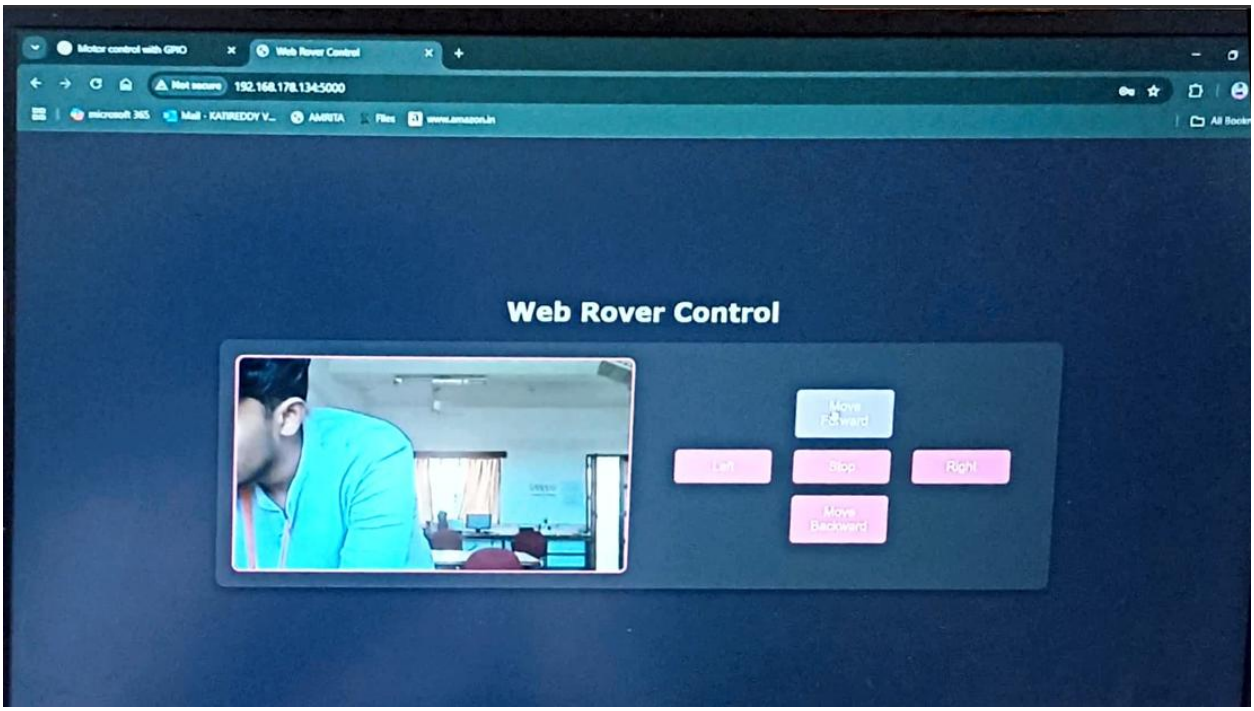
```python
# Confusion Matrix Plot
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

print("Classification Report:\n", clr)
```

Chapter – 3

3. Results and Discussions

3.1. Outputs





3.2. Discussions

- The integration of real-time video, motor control, and crack detection provides a robust platform for remote industrial inspection.
- Web-based architecture ensures platform independence, enabling rover control from any internet-connected device.
- Future work will focus on:
    - Enhancing CNN with multi-class defect detection.
    - Deploying sensor fusion algorithms (ex: Kalman Filter, ADMM, Low-Rank Matrix Completion).
    - Adding alerts and logs for abnormal gas/temperature readings or crack detection events.

Chapter – 4

Conclusion

The Industrial Web Rover built under this project has live remote control, live video streaming, and intelligent crack detection in one rugged package for dangerous conditions. With a Raspberry Pi, motor driver, and web-based Flask interface, the user can easily control the rover from any browser while observing the surroundings with a video feed.

Utilizing a light CNN model for crack detection has shown good performance in identifying structural flaws with high accuracy and thus is a dependent tool for inspection work. Gas sensor, the current deployment itself demonstrates potential to merge IoT, machine learning, and robotics for industrial use.

This project builds a way for intelligent, more secure inspection systems and leaves much more space for future enhancement such as advanced sensor data processing and real-time anomaly notification.

References

- Rustem Galiev, YOLOv8, EfficientDet, Faster R-CNN, or YOLOv5 for Remote Sensing?, *Medium*, 2023. https://medium.com/@rustemgal/yolov8-efficientdet-faster-r-cnn-or-yolov5-for-remote-sensing-12487c40ef68

- Adrian Rosebrock, Building a basic motion detection and tracking system with Python and OpenCV, *PyImageSearch*, 2015. https://pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/

- Raspberry Pi Foundation, Getting started with Raspberry Pi, *Raspberry Pi Documentation*, 2024. https://www.raspberrypi.com/documentation/

- TensorFlow Team, Save and load models, *TensorFlow Documentation*, 2023. https://www.tensorflow.org/tutorials/keras/save_and_load

- Flask Documentation Team, Flask Web Framework, *Flask Docs*, 2023. https://flask.palletsprojects.com/en/2.3.x/

- Murtaza Hassan, Real-time face detection and object tracking using OpenCV and Flask, *YouTube*, 2022. https://www.youtube.com/watch?v=wcW8xNIRGVc

- Jason Brownlee, How to Develop a Convolutional Neural Network to Classify Photos of Dogs and Cats (Crack Classification Example Used Here), *Machine Learning Mastery*, 2019. https://machinelearningmastery.com/how-to-develop-a-cnn-to-classify-photos-of-dogs-and-cats/

- W3Schools, HTML & CSS Styling References, *W3Schools.com*, 2024. https://www.w3schools.com/css/css_intro.asp

- IEEE Spectrum, Robots in Hazardous Environments, *IEEE Spectrum*, 2023. https://spectrum.ieee.org/robotics-in-danger-zones

- Pranav Dar, An Introduction to Compressed Sensing, *Analytics Vidhya*, 2021. https://www.analyticsvidhya.com/blog/2021/06/introduction-to-compressed-sensing/