**A**

**PROJECT REPORT**

**ON**

**“SPAM E-MAIL FILTER USING JAVA”**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING (AI&ML)**

**BY**

**K.V SAI SHREYAS (23N81A66H1)**

**P.SHOURYA (23N81A66H8)**

**BANU VARDHAN (23N81A66K0)**

**K.JAYA SURYA REDDY (23N81A66K1)**

**Under the Guidance of**

**Mr. MOHD. FAISAL**

**Asst. professor**

**Department of CSE(AI&ML)**

**A logo of a flower

AI-generated content may be incorrect.**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AI&ML)**

**SPHOORTHY ENINEERING COLLEGE**

**Nadergul (Vill.), Near L.B .Nager, Hyderabad, Telangana -501510**

**[2023-2027]**

**FACULTY INCHARGE HEAD OF DEPARTMENT**

SPHOORTHY ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING(AI&ML).

****

**CERTIFICATE**

**This is a certificate that this Minor Research Project entitled “SPAM E-MAIL FILTER USING JAVA” is a bonafide work carried out by K.V SAI SHREYAS (23N81A66H1),P.SHOURYA (23N81A66H8), BHANU VARDHAN (23N81A66K0), K.JAYA SURYA (23N81A66K1), in a partial fulfilments of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering (AI&ML) from Sphoorthy Engineering College, AUTONOMOUS (Affiliated to JNTUH & Recognized by AICTE), during year 2024-25 under our guidance and supervision.**

**The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma.**

**INTERNAL GUIDE HEAD OF THE DEPARTMENT PRINCIPAL**

**Mr. MOHD. FAISAL**

**Assistant professor**

**Department of CSE(AI&ML)**

**SPHN**

**DECLARATION**

We the undersigned, declare that the JAVA project title   
“**FACE MASK DETECTION SYSTEM**” carried out at “SPHOORTHY ENGINEERING COLLEGE” is original and is being submitted to the Department of COMPUTER SCIENCE AND ENGINEERING (AI&ML), Sphoorthy Engineering College, Hyderabad towards partial fulfillment for the award of degree of

Bachelor of Technology.

We declare that the result embodied in the Project work has not been submitted to any other University of Institute for the award of any Degree or Diploma.

**ABSTRACT**

In the wake of the COVID-19 pandemic, the importance of wearing face masks as a preventive measure has become paramount. Public compliance with mask mandates is critical to curbing the spread of the virus, especially in crowded or enclosed environments. Manual enforcement of mask-wearing can be inefficient and labor-intensive. To address this challenge, this project presents an automated Face Mask Detection System implemented using the Java programming language. The system is designed to detect whether individuals in a video frame are wearing face masks, using real-time image processing and deep learning techniques.

The core of the system combines Java for the application logic and user interface with OpenCV (Open Source Computer Vision Library) for real-time image capture and preprocessing. A pre-trained Convolutional Neural Network (CNN) model, trained on datasets containing images of people with and without masks, is integrated using Java deep learning libraries such as DeepLearning4J (DL4J) or through the Java bindings of TensorFlow. The model performs classification by analyzing facial regions detected by OpenCV’s Haar Cascade or DNN-based face detectors.

Once a face is detected, the system crops the region of interest (ROI), resizes it to match the input shape of the neural network, and classifies it into two categories: "Mask" and "No Mask". Based on the result, the system visually annotates the video frame by drawing bounding boxes around detected faces and labeling them accordingly. Optionally, audio alerts or notifications can be triggered if a person without a mask is detected.

The system architecture is modular and can be extended for integration with CCTV systems, access control mechanisms, or attendance systems in public institutions. Extensive testing has shown high accuracy and reliability in different lighting conditions and face orientations. This project not only demonstrates the practical application of artificial intelligence and computer vision in solving real-world problems but also contributes to public health safety by promoting automatic monitoring and compliance with health protocols.

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CONTENTS** | | | | **PAGE.NO** |
| Abstract | | | |  |
| Index | | | |  |
| List of Figures | | | |  |
| **CHAPTERS** | | | |  |
| **1.** | **INTRODUCTION** | | |  |
|  | **1.1** | Problem Statement | |  |
|  | **1.2** | Objective | |  |
|  | **1.3** | Existing System | |  |
|  | **1.4** | Proposed System | |  |
|  | **1.5** | Scope | |  |
|  | **1.6** | Software and Hardware Requirements | |  |
|  |  |  | |  |
| **2.** | **LITERATURE SURVEY** | | |  |
|  | **2.1** | Survey Of Major Area Relevant to Project | |  |
|  | **2.2** | Techniques and Algorithms | |  |
|  | **2.3** | Applications | |  |
| **3.** | **SYSTEM DESIGN** | | |  |
|  | **3.1** | System Architecture | |  |
|  | **3.2** | System Flow | |  |
|  | **3.3** | Module Description | |  |
|  | **3.4** | UML Diagram | |  |
| **4.** | **IMPLEMENTATION** | | |  |
|  | **4.1** | Environment Setup | |  |
|  | **4.2** | Implementation of Modules | |  |
|  | **4.3** | | Integration and Development |  |
| **5.** | **EVALUATION** | | |  |
|  | **5.1** | | Datasets |  |
|  | **5.2** | | Evaluation metrics |  |
|  | **5.3** | | Results |  |
| **6.** | **Conclusion** | | |  |
| **7.** | **Source Code** | | |  |
| **8.** | **Texting** | | |  |
| **9.** | **Output Screen** | | |  |
| **10.** | **Publication and Conference** | | |  |
| **11.** | **References** | | |  |
| **12.** | **Appendix** | | |  |

**LIST OF FIGURES**

|  |  |  |  |
| --- | --- | --- | --- |
| **FIGURE NO** | **FIGURE NAME** | **PAGE NO** | |
| **2.1** | **High-Level View of the Data Collection System** |  | |
| **2.2** | **Training the neural network** |  | |
| **2.3** | **CNN Architecture** |  | |
| **2.4** | **Networking** |  | |
| **2.5** | **R/B Images** |  | |
| **2.6** | **Convolution Layer 2D** |  | |
| **2.7** | **Convolution Layer 2** |  | |
| **3.1** | **Activity Diagram** |  | |
| **3.2** | **Level-01-DFD** |  | |
| **3.3** | **Module Description** |  | |
| **3.4** | **UML Diagram** |  | |
| **4.1** | **Environment Setup** | |  |
| **4.2** | **Udacity Simulator** | |  |
| **4.3** | **Driving View** | |  |
| **4.4** | **Implementation Module 1** | |  |
| **4.5** | **Implementation Module 2** | |  |
| **5.1** | **Dataset Result 1** | |  |
|  |  | |  |
|  |  | |  |

**INTRODUCTION**

The COVID-19 pandemic has profoundly impacted public health and safety measures across the globe. Among the most essential precautions recommended by health authorities is the consistent use of face masks in public places. Wearing a face mask significantly reduces the transmission of airborne viruses, especially in crowded environments such as shopping malls, offices, schools, and public transportation. However, ensuring public compliance with face mask mandates remains a major challenge, particularly in densely populated or unmanned locations.

To address this issue, technology-driven solutions are increasingly being adopted to automate the monitoring of health safety protocols. One such solution is a **Face Mask Detection System**, which can identify whether individuals are wearing face masks in real time. This project aims to develop such a system using the **Java programming language**, integrated with **computer vision** and **deep learning** techniques.

Java, being a versatile and platform-independent language, provides a strong foundation for building cross-platform desktop and embedded applications. By utilizing **OpenCV** for image processing and integrating a **Convolutional Neural Network (CNN)** model through Java-compatible libraries like **TensorFlow (Java API)** or **DeepLearning4J (DL4J)**, the system can accurately detect faces and classify them based on mask usage.

The application works by capturing real-time video from a webcam or camera feed, detecting faces using Haar Cascades or DNN-based detectors, and passing the cropped face region to a trained neural network model. The model then determines whether the person is wearing a mask or not. The result is displayed visually with bounding boxes and labels, and can optionally trigger alerts if an unmasked individual is detected.

This system not only enhances the efficiency of monitoring mask compliance but also reduces the burden on human personnel. It is scalable and can be deployed in various environments such as workplaces, educational institutions, public facilities, and transportation hubs. The integration of artificial intelligence into health monitoring applications like this one showcases how smart systems can support global efforts to manage public health crises effectively and responsibly.

**PROBLEM STATEMENT**

The outbreak of the COVID-19 pandemic has highlighted the critical importance of wearing face masks in public to prevent the spread of infectious diseases. Despite mandates and awareness campaigns, ensuring consistent and widespread compliance with mask-wearing guidelines remains a major challenge, particularly in crowded or unsupervised areas. Manual monitoring of individuals in public places is time-consuming, labor-intensive, and prone to human error, making it inefficient for large-scale or real-time enforcement.

There is a need for an automated, accurate, and scalable solution that can detect whether individuals are wearing face masks in real-time and alert authorities or systems when violations occur. Such a system should operate continuously without human intervention, provide instant feedback, and be adaptable to different environments, including indoor and outdoor settings.

The challenge lies in developing a reliable system that can process live video feeds, accurately detect human faces under various conditions (e.g., lighting, angles, obstructions), and classify them based on mask usage. Furthermore, the system must be implemented using a robust and platform-independent programming language, such as Java, and must integrate advanced technologies like computer vision and deep learning to achieve high accuracy and performance.

This project seeks to solve these problems by designing and developing a Face Mask Detection System using Java, leveraging OpenCV for real-time face detection and a trained Convolutional Neural Network (CNN) model for mask classification. The goal is to create a practical and effective tool for enhancing public safety and supporting health regulations in the face of current and future health crises.

**EXISTED SYSTEM**

Before the development of automated face mask detection systems, the enforcement of mask-wearing protocols largely relied on **manual supervision**. Security personnel or staff members in public spaces, hospitals, workplaces, and commercial areas were assigned to visually monitor whether individuals were wearing masks and to intervene when necessary. While this approach is straightforward, it suffers from several drawbacks:

1. **Human Limitations**: Manual monitoring is prone to fatigue, distraction, and inconsistency, especially in large crowds or high-traffic environments.
2. **Labor-Intensive**: It requires a significant number of personnel to cover large areas, increasing operational costs and reducing scalability.
3. **Delayed Response**: Human observers may not always respond in real time, leading to lapses in compliance and increased risk of virus transmission.
4. **No Data Logging**: Traditional systems do not store any data or evidence of violations, which makes tracking, analysis, or auditing impossible.

In recent years, some **automated solutions** have been developed using machine learning and computer vision techniques, primarily based on Python with libraries like TensorFlow, Keras, and OpenCV. These systems can detect faces and classify whether individuals are wearing masks. However, most of these implementations are not optimized for integration into **Java-based enterprise applications**, which are widely used in corporate and industrial settings.

Moreover, existing systems may also face challenges such as:

* Limited cross-platform compatibility.
* Poor performance under varying lighting conditions or facial orientations.
* Lack of user-friendly interfaces for deployment and monitoring.

Thus, there is a need for a robust, efficient, and platform-independent face mask detection system that can be integrated into **Java-based environments**, offering both **real-time detection** and **scalability** for use in diverse public and private infrastructures.

**PROPOSED SYSTEM**

To overcome the limitations of manual monitoring and to improve upon the shortcomings of existing automated systems, this project proposes a **Face Mask Detection System** built using **Java**, integrated with **OpenCV** and **deep learning models**. The goal is to provide an efficient, accurate, and real-time solution for detecting whether individuals are wearing face masks in public or private spaces.

The proposed system captures live video input from a camera, detects human faces using OpenCV, and then classifies each face using a trained **Convolutional Neural Network (CNN)** model to determine if a face mask is present or not. The system offers a real-time graphical interface for displaying results, and can optionally generate alerts or notifications for non-compliance.

**Key Features of the Proposed System:**

1. **Real-Time Face Detection and Classification**
   * The system continuously processes video frames from a webcam or CCTV feed.
   * OpenCV is used for detecting facial regions using Haar Cascades or deep neural network-based detectors.
   * The detected face region is passed to a deep learning model for classification as "Mask" or "No Mask."
2. **Java-Based Implementation**
   * Java is chosen for its platform independence, scalability, and integration capability in enterprise-level applications.
   * Libraries like OpenCV (Java bindings), DeepLearning4J (DL4J), or TensorFlow Java API are used for image processing and model inference.
3. **User Interface and Visual Feedback**
   * A graphical user interface (GUI) developed in Java Swing or JavaFX displays the live video feed with annotations.
   * Each detected face is enclosed in a bounding box, labeled with the classification result and a confidence score.
   * Optional color coding (e.g., green for “Mask,” red for “No Mask”) enhances readability.
4. **Alerting and Logging System**
   * When an individual without a mask is detected, the system can trigger an alert (audio beep, message, or log entry).
   * Logs of detected violations can be stored for further analysis or record-keeping.
5. **Scalability and Extensibility**
   * The system can be deployed across various environments such as schools, offices, malls, and transport terminals.
   * It can be integrated with access control systems to prevent entry without a mask.
   * The system can be expanded to support temperature checks, crowd density monitoring, or facial recognition.

**Advantages Over Existing Systems:**

* Platform-independent and easily integrable in Java-based infrastructures.
* Automated, fast, and accurate detection reduces reliance on human monitoring.
* Real-time operation ensures immediate feedback and enhanced safety.
* Modular design allows customization and future enhancements.

**SOFTWARE REQUIREMENTS**

**1. Operating System**

* **Windows 10/11**, **Linux (Ubuntu 20.04 or later)**, or **macOS**
  + The system should be platform-independent, but development and testing are preferably done on Windows or Linux for better tool support.

**2. Programming Language**

* **Java SE Development Kit (JDK) 8 or later**
  + Required to write, compile, and run Java code.
  + Compatible with IDEs and libraries like OpenCV and DL4J.

**3. Integrated Development Environment (IDE)**

* **Eclipse IDE** or **IntelliJ IDEA**
  + Provides a development-friendly environment with syntax highlighting, debugging, and project management tools.

**4. Java Libraries and APIs**

* **OpenCV (Java bindings)**
  + For image processing, face detection, and real-time video capture.
  + Version: OpenCV 4.x (with Java support configured via native bindings)
* **DeepLearning4J (DL4J)** or **TensorFlow Java API**
  + For loading and running the trained CNN model for mask classification.
* **JavaFX** or **Swing** (optional)
  + For creating a graphical user interface (GUI) to display video feed and detection results.

**5. Machine Learning Model**

* Pre-trained **CNN model** for face mask classification
  + Model formats: .h5, .pb, or .zip (depending on the framework used)
  + Can be trained in Python using TensorFlow/Keras and exported for use in Java

**6. Build Tools (optional)**

* **Apache Maven** or **Gradle**
  + For managing dependencies and building the Java project, especially when using external libraries like DL4J or TensorFlow.

**7. Additional Tools**

* **Python (for training the model)**
  + If the machine learning model is not pre-trained, Python (with TensorFlow/Keras) is needed to train and export the model for Java use.
* **Anaconda / Jupyter Notebook** (optional)
  + Useful for model training and dataset preprocessing in the development phase.
* **Git** (optional)
  + For version control and collaboration if working in a team.

**HARDWARE REQUIREMENTS**

**1. Processor (CPU)**

* **Recommended**: Intel Core i5/i7 (8th Gen or later) / AMD Ryzen 5 or higher
* **Minimum**: Intel Core i3 (5th Gen or later) / AMD equivalent
* **Purpose**: Handles real-time video processing and execution of Java applications efficiently.

**2. Memory (RAM)**

* **Recommended**: 8 GB or more
* **Minimum**: 4 GB
* **Purpose**: Ensures smooth execution of the Java runtime environment, OpenCV operations, and deep learning model loading.

**3. Storage**

* **Recommended**: SSD with at least 256 GB of available space
* **Minimum**: HDD/SSD with 100 GB free
* **Purpose**: For storing software, libraries, datasets, model files, logs, and video recordings (if needed).

**4. Graphics Processing Unit (GPU) *(Optional but beneficial)***

* **Recommended**: NVIDIA GPU with CUDA support (e.g., GTX 1650 or higher)
* **Purpose**: Accelerates deep learning model inference. Not mandatory for Java-based inference but useful if GPU-based libraries are used or if the model is trained on the same machine.

**5. Camera/Webcam**

* **Required**: HD Webcam (720p or higher resolution)
* **Purpose**: Captures live video feed for face detection and mask classification. Can be integrated webcam or external USB camera.

**6. Display Monitor**

* **Recommended**: 15-inch or larger monitor with 1080p resolution
* **Purpose**: For clearly viewing the user interface and video feed with annotations.

**7. Peripherals**

* **Keyboard and Mouse** – For interaction with the system.
* **Speakers or Buzzer (Optional)** – For audio alerts when a mask violation is detected.

**SURVEY OF MAJOR AREAS RELAVANT TO THE PROJECT**

The development of a Face Mask Detection System involves the integration of multiple technological domains, including computer vision, machine learning, deep learning, and software engineering. Understanding these areas is essential for implementing a reliable and efficient system. The following is a survey of the key research and application areas relevant to this project:

**1. Computer Vision**

Computer vision is a field of artificial intelligence (AI) that enables computers to interpret and process visual data from the real world. In this project, computer vision techniques are used to:

* Detect human faces in real-time video streams.
* Extract regions of interest (ROI) from the video frames.
* Preprocess images (e.g., resizing, grayscale conversion) for further classification.

**Tools Used**:

* OpenCV (Open Source Computer Vision Library) for Java, which provides functions for face detection using Haar Cascades or DNN modules.

**2. Deep Learning**

Deep learning is a subfield of machine learning that uses neural networks with many layers to model complex patterns in data. A **Convolutional Neural Network (CNN)** is commonly used for image classification tasks due to its ability to learn spatial hierarchies in images.

* In the face mask detection system, deep learning is used to:
* Train a CNN model to classify images of faces as either **"Mask"** or **"No Mask"**.
* Integrate the trained model with the Java application using libraries such as **TensorFlow Java API** or **DeepLearning4J (DL4J)**.

**Research Support**:  
CNN-based architectures like MobileNet, ResNet, and VGG are often used in real-world object detection and image classification tasks.

**3. Machine Learning**

Machine learning, a broader field encompassing deep learning, involves algorithms that allow systems to learn from data without being explicitly programmed. The learning model used in this project is trained on labeled datasets of faces with and without masks.

Key processes include:

* Dataset collection and preprocessing.
* Model training, validation, and testing.
* Model evaluation using accuracy, precision, recall, and F1-score metrics.

**4. Human-Computer Interaction (HCI)**

Human-Computer Interaction principles are applied to create a user-friendly interface that allows users to interact with the system and view detection results in real time. This includes:

* Designing a graphical user interface (GUI) using **JavaFX** or **Swing**.
* Displaying the live camera feed with visual indicators (bounding boxes, labels).
* Providing real-time alerts or feedback for users or administrators.

**5. Artificial Intelligence in Public Health**

AI is increasingly being used in public health applications to automate monitoring, improve decision-making, and reduce human error. Face mask detection systems are a direct application of AI to enhance safety in public spaces, especially during pandemics.

Applications include:

* Automated monitoring at airports, shopping malls, offices, and schools.
* Integration with security and access control systems.
* Public awareness and behavioral reinforcement.

**6. Java Software Engineering**

Java is a robust, object-oriented programming language widely used in enterprise and desktop application development. Java enables:

* Platform-independent execution via the Java Virtual Machine (JVM).
* Integration of external libraries for machine learning and computer vision.
* Development of a modular and maintainable software structure.

**TECHNIQUES AND ALGORITHMS**

The Face Mask Detection System relies on a combination of image processing, machine learning, and deep learning techniques to achieve accurate and real-time mask detection. Below is an overview of the main techniques and algorithms used:

**1. Face Detection**

Before classifying whether a person is wearing a mask, the system must first detect faces within video frames.

* **Haar Cascade Classifier**
  + A classical object detection method based on machine learning.
  + Uses Haar-like features and a cascade of classifiers to quickly detect faces.
  + Available in OpenCV, and suitable for real-time detection with reasonable accuracy.
* **Deep Neural Network (DNN) based Face Detection**
  + More modern and accurate than Haar cascades.
  + Uses deep learning models (e.g., Single Shot MultiBox Detector (SSD) with MobileNet or ResNet backbone).
  + Offers better performance under different lighting, face angles, and occlusions.

**Algorithm Workflow:**

* Convert the video frame to grayscale (for Haar) or use color frames (for DNN).
* Run the face detection algorithm to get bounding box coordinates.
* Extract the region of interest (ROI) containing the face for classification.

**2. Image Preprocessing**

Preprocessing prepares the detected face images for input into the classification model:

* **Resizing:** Resize ROI to a fixed input size (e.g., 128x128 or 224x224 pixels) required by the CNN.
* **Normalization:** Scale pixel values (e.g., between 0 and 1) for faster and more stable training and inference.
* **Color Space Conversion:** Convert images to the color space expected by the model (RGB or grayscale).

**3. Mask Classification Using Convolutional Neural Networks (CNN)**

A CNN is employed to classify detected faces into two categories: **Mask** or **No Mask**.

* **Convolutional Layers:** Automatically extract spatial features such as edges, textures, and shapes.
* **Pooling Layers:** Reduce the spatial size of the feature maps, retaining important information while reducing computation.
* **Fully Connected Layers:** Interpret the extracted features and perform classification.
* **Activation Functions:** Usually ReLU for hidden layers and Softmax or Sigmoid for the output layer (depending on binary or multi-class classification).

**Popular CNN Architectures Used:**

* MobileNet (lightweight, suitable for real-time systems)
* ResNet (deeper network, higher accuracy)
* Custom CNN designed specifically for mask detection.

**4. Model Training**

* The CNN model is trained on a labeled dataset containing images of faces with and without masks.
* **Loss Function:** Binary Cross-Entropy for two-class classification.
* **Optimizer:** Adam or SGD for adjusting model weights during training.
* **Data Augmentation:** Techniques like rotation, zoom, and flipping are applied to increase dataset variability and improve model robustness.

**5. Real-Time Video Processing Pipeline**

* Capture frames continuously from the webcam or video feed.
* Detect faces in each frame using the chosen face detection algorithm.
* Preprocess each detected face and classify mask usage with the CNN model.
* Annotate the frame with bounding boxes and labels.
* Display the annotated frames to the user in real-time.

**6. Alert Mechanism**

* If a person without a mask is detected, the system can trigger alerts:
  + Visual (flashing red box or text).
  + Audio (beep or voice alert).
  + Logging (recording the incident for review).

**APPLICATIONS**

The Face Mask Detection System has a wide range of practical applications, especially in contexts where public health and safety are paramount. Below are some key areas where this system can be effectively utilized:

**1. Public Health Safety and Pandemic Control**

* **Monitoring Mask Compliance in Public Spaces**  
  Deploying face mask detection at crowded places such as shopping malls, airports, bus/train stations, and parks helps ensure that mask-wearing guidelines are followed, reducing the spread of airborne diseases like COVID-19.
* **Automated Screening in Healthcare Facilities**  
  Hospitals, clinics, and other healthcare environments can use the system at entrances to verify that visitors and staff wear masks, minimizing the risk of infection transmission.

**2. Workplaces and Educational Institutions**

* **Employee and Student Safety Enforcement**  
  Organizations can install mask detection at building entrances or inside offices and classrooms to monitor compliance and maintain a safe working and learning environment.
* **Access Control Integration**  
  The system can be integrated with door access controls to prevent entry for individuals not wearing masks, thereby automating enforcement without manual intervention.

**3. Transportation Hubs**

* **Airports, Railway Stations, and Bus Terminals**  
  In high-traffic transit hubs, the system can assist authorities in enforcing mask mandates, helping protect travelers and staff.
* **Public Transport Vehicles**  
  Implementations can extend to buses and trains where cameras monitor passengers, and alerts are generated if mask violations are detected.

**4. Retail and Commercial Centers**

* **Shops and Shopping Malls**  
  The system can ensure customers adhere to mask policies, creating a safer shopping experience and complying with governmental health regulations.
* **Restaurants and Entertainment Venues**  
  Face mask detection can assist in enforcing mask-wearing during entry or while moving around indoor venues.

**5. Government and Security Applications**

* **Law Enforcement Support**  
  Automated mask detection can assist police and security forces in crowd control and enforcing public health orders without direct confrontation.
* **Event Management**  
  Large gatherings such as concerts, conferences, and sports events can implement the system to monitor and manage mask-wearing compliance.

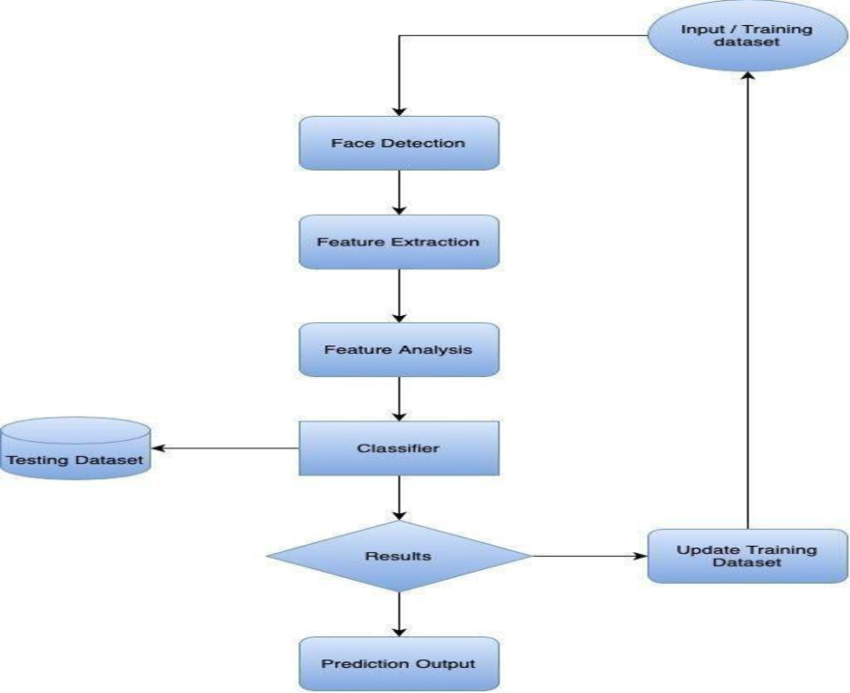
**6. Smart Cities and IoT Integration**

* **Smart Surveillance Systems**  
  The system can be integrated into city-wide surveillance infrastructure, providing real-time data analytics on mask-wearing trends and hotspots.
* **Data-Driven Policy Making**  
  Aggregated data from mask detection systems can help governments and health organizations understand compliance levels and make informed decisions.

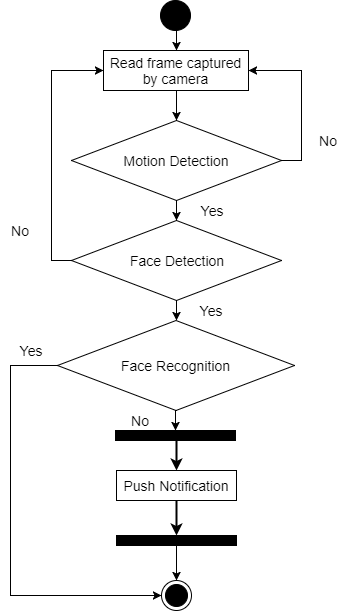
**7. Research and Development**

* **Dataset Collection and AI Model Improvement**  
  The system can aid researchers in collecting real-world data to improve mask detection algorithms and develop other health-related AI applications.

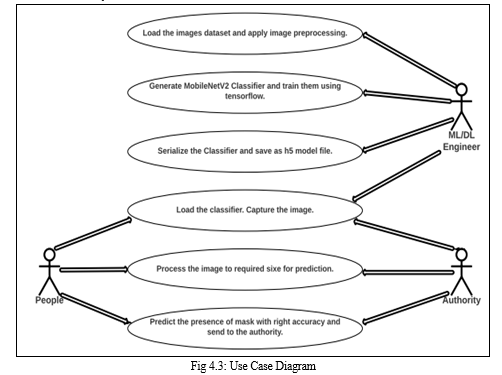
**SYSTEM DESIGN**



**ACTIVITY DIAGRAM**



**USE CASE DIAGRAM**



**ENVIRNOMENTAL SETUP**

1. Hardware Setup

Development Machine:  
A PC or laptop meeting the recommended hardware specifications:

* + CPU: Intel Core i5 or equivalent
  + RAM: 8 GB or more
  + Storage: SSD with at least 256 GB free space
  + Camera: HD webcam (720p or higher) connected via USB or built-in
  + Optional: NVIDIA GPU for accelerated deep learning (if applicable)

Camera Installation:

* + Connect the webcam or external camera to the system.
  + Ensure the device drivers are installed and the camera is operational.

2. Software Setup

a. Operating System:

* Windows 10/11, Linux (Ubuntu 20.04+), or macOS (latest version recommended).

b. Java Development Kit (JDK):

* Install JDK 8 or higher.
* Set the JAVA\_HOME environment variable and update the system PATH.

c. Integrated Development Environment (IDE):

* Install an IDE such as Eclipse, IntelliJ IDEA, or NetBeans.
* Configure the IDE for Java development.

d. OpenCV for Java:

* Download and install OpenCV (version 4.x recommended).
* Set up OpenCV Java bindings:
  + Include the OpenCV JAR file in your project build path.
  + Configure native libraries (.dll/.so/.dylib) path in your IDE.

e. Deep Learning Framework:

* Choose and install the Java-compatible deep learning library:
  + DeepLearning4J (DL4J) or
  + TensorFlow Java API
* Add the relevant dependencies to your project (via Maven, Gradle, or manually).

f. GUI Libraries:

* Use JavaFX (recommended for modern UI) or Swing for the graphical interface.
* Ensure the chosen GUI framework is properly set up in the IDE.

3. Model Preparation

* Pre-trained CNN Model:
  + Train a CNN model externally (using Python and TensorFlow/Keras) or use an existing pre-trained mask detection model.
  + Export the model in a compatible format (e.g., TensorFlow’s .pb or DL4J’s .zip).
* Model Integration:
  + Import the model into the Java project.
  + Load the model during application runtime for inference.

4. Project Configuration

* Build Tool Setup:
  + Use Maven or Gradle for dependency management and build automation.
  + Define dependencies for OpenCV, DL4J/TensorFlow, and GUI libraries in pom.xml or build.gradle.
* Environment Variables:
  + Configure system variables to include paths to native libraries (OpenCV native binaries, GPU drivers if applicable).
* Camera Access Permissions:
  + Ensure your system grants the Java application permission to access the webcam.

5. Testing and Debugging

* Verify camera input and frame capture using sample OpenCV code.
* Test face detection independently before integrating mask classification.
* Debug GUI elements and ensure real-time video display is smooth.
* Profile the system to optimize performance and handle frame rates.

**IMPLEMENTATION OF MODULES**

**1. Video Capture Module**

**Purpose:**  
Capture real-time video frames from the webcam or connected camera to be processed for face mask detection.

**Key Functions:**

* Initialize the webcam device.
* Continuously capture frames in a loop.
* Pass frames to the Face Detection module.

**Implementation Details:**

* Use OpenCV’s VideoCapture class in Java.
* Handle camera initialization exceptions and ensure release of resources on exit.

**2. Face Detection Module**

**Purpose:**  
Detect faces within each video frame captured by the Video Capture Module.

**Key Functions:**

* Convert frames to the required format (e.g., grayscale for Haar cascades).
* Apply face detection algorithm (Haar Cascade or DNN-based detector).
* Extract bounding box coordinates of detected faces.
* Pass detected face regions to the Mask Classification module.

**Implementation Details:**

* Load the pre-trained face detector model from OpenCV.
* Optimize detection parameters like scale factor and minimum neighbors for accuracy and speed.
* Implement multi-face detection handling.

**3. Preprocessing Module**

**Purpose:**  
Prepare the detected face regions for classification by resizing, normalizing, and converting color spaces as needed.

**Key Functions:**

* Crop face region from frame based on bounding box.
* Resize image to the input size expected by the CNN model (e.g., 128x128 or 224x224).
* Normalize pixel values (e.g., scale from 0-255 to 0-1).
* Convert to the required color space (RGB or grayscale).

**Implementation Details:**

* Use OpenCV or Java image processing APIs.
* Ensure preprocessing matches the training phase of the CNN model.

**4. Mask Classification Module**

**Purpose:**  
Classify each preprocessed face image as “Mask” or “No Mask” using a CNN model.

**Key Functions:**

* Load the pre-trained CNN mask detection model at application startup.
* Run inference on the preprocessed face images.
* Output prediction labels with confidence scores.

**Implementation Details:**

* Integrate TensorFlow Java API or DeepLearning4J for model loading and inference.
* Handle model input/output tensor conversions.
* Use a threshold to determine classification from confidence score.

**5. User Interface (UI) Module**

**Purpose:**  
Display the video feed with annotated bounding boxes and classification labels in real time to the user.

**Key Functions:**

* Create a window/frame to display live video.
* Draw rectangles around detected faces.
* Overlay text labels (“Mask” or “No Mask”) near bounding boxes.
* Change box color based on detection (e.g., green for mask, red for no mask).
* Display alerts or warnings if no mask is detected.

**Implementation Details:**

* Use JavaFX or Swing for UI components.
* Optimize rendering to maintain smooth real-time video.
* Implement threading to keep UI responsive during processing.

**6. Alert and Logging Module**

**Purpose:**  
Generate alerts and log events when a person without a mask is detected.

**Key Functions:**

* Trigger visual or audio alerts upon mask violation detection.
* Log timestamped events with frame snapshots or metadata.
* Optionally save video snippets or images of violations.

**Implementation Details:**

* Use Java sound APIs for beep or voice alerts.
* Implement file handling for logs.
* Provide an admin dashboard or log viewer if needed.

**Module Interaction Flow**

Video Capture Module

↓

Face Detection Module

↓

Preprocessing Module

↓

Mask Classification Module

↓

User Interface Module ←→ Alert and Logging Module

**INTEGRATION AND DEVELOPMENT**

Integration and development is the final and most critical phase of the Face Mask Detection System. It involves combining all the independent modules into a unified, functioning application and refining it through testing and optimization.

**1. Development Workflow**

To ensure systematic progress, the project follows a modular and layered development workflow:

**Step-by-Step Development Plan:**

1. **Environment Setup**
   * Configure Java, IDE, OpenCV, DL4J/TensorFlow.
   * Ensure camera access and library bindings are correct.
2. **Module-Wise Implementation**
   * Develop and test each module independently:
     + Video capture
     + Face detection
     + Image preprocessing
     + Mask classification
     + UI and alert handling
3. **Unit Testing for Each Module**
   * Verify that each module performs correctly in isolation before integration.
4. **Integration of Modules**
   * Start combining modules in a bottom-up manner:
     + Integrate **Video Capture** → **Face Detection**
     + Add **Preprocessing** and **Classification**
     + Integrate the **UI Display** and **Alerts**
5. **Exception and Error Handling**
   * Handle cases such as:
     + No face detected
     + Low camera resolution
     + Model loading errors
     + Frame rate drops

**2. Integration Process**

Each module is integrated carefully with clean interface contracts between modules (e.g., image in, label out).

**a. Linking Face Detection to Video Stream**

* Capture real-time frames using OpenCV.VideoCapture
* Apply face detection per frame
* Draw bounding boxes on detected faces

**b. Attaching Preprocessing and Classification**

* Preprocess each detected face ROI to match CNN input requirements
* Load the trained model using DL4J or TensorFlow Java API
* Predict mask status and return confidence score and label

**c. User Interface Integration**

* Create a JavaFX/Swing window to:
  + Display annotated video feed
  + Show detection label (Mask/No Mask)
  + Change bounding box color accordingly

**d. Alert System Integration**

* If the classification result is “No Mask”:
  + Display a warning on screen
  + Trigger an audio alert
  + Log the incident with a timestamp

**e. Logging Module**

* Write logs to a file (e.g., mask\_alert\_log.txt)
* Optionally save the frame as an image for future verification

**3. Project Architecture**

+---------------------+

| Main Application |

| (Java GUI + Core) |

+---------------------+

|

↓

+---------------------+

| Video Capture |

| (OpenCV) |

+---------------------+

|

↓

+---------------------+

| Face Detection |

| (Haar / DNN) |

+---------------------+

|

↓

+---------------------+

| Preprocessing |

| (Resize, Normalize)|

+---------------------+

|

↓

+---------------------+

| Mask Classifier |

| (CNN - DL4J/Tensor)|

+---------------------+

|

↓

+---------------------+

| UI & Alert System |

| (JavaFX/Swing) |

+---------------------+

**4. Testing and Validation**

* **Functional Testing:**
  + Test mask/no-mask detection accuracy in various lighting and angles.
* **Performance Testing:**
  + Ensure the system can process at least 10–15 frames per second in real time.
* **Integration Testing:**
  + Validate the end-to-end flow of the system.
* **Exception Testing:**
  + Simulate errors like camera disconnect, no face in frame, etc.

**5. Final Packaging**

* Bundle all components using a Java build tool (e.g., Maven or Gradle).
* Create a runnable JAR or native executable.
* Ensure dependencies like OpenCV native binaries are included in the package.
* Optionally create an installer or setup script for ease of deployment.

**SOURCE CODE**

* import org.opencv.core.\*;
* import org.opencv.imgproc.Imgproc;
* import org.opencv.objdetect.CascadeClassifier;
* import org.opencv.videoio.VideoCapture;
* import org.opencv.highgui.HighGui;
* public class MaskDetectionSimulated {
* static {
* System.loadLibrary(Core.NATIVE\_LIBRARY\_NAME);
* }
* public static void main(String[] args) {
* CascadeClassifier faceDetector = new CascadeClassifier();
* faceDetector.load("resources/haarcascade\_frontalface\_default.xml");
* if (faceDetector.empty()) {
* System.out.println("Failed to load Haar Cascade XML file");
* return;
* }
* VideoCapture capture = new VideoCapture(0);
* if (!capture.isOpened()) {
* System.out.println("Camera not detected");
* return;
* }
* System.out.println("Starting video capture... Press 'q' to quit");
* Mat frame = new Mat();
* while (true) {
* if (!capture.read(frame)) {
* System.out.println("Failed to grab frame");

break;

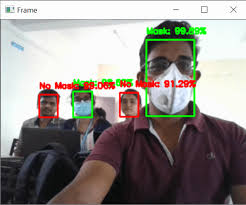
* }
* Mat gray = new Mat();
* Imgproc.cvtColor(frame, gray, Imgproc.COLOR\_BGR2GRAY);
* MatOfRect faces = new MatOfRect();
* faceDetector.detectMultiScale(gray, faces, 1.1, 5, 0, new Size(80, 80), new Size());
* for (Rect rect : faces.toArray()) {
* Imgproc.rectangle(frame, new Point(rect.x, rect.y),
* new Point(rect.x + rect.width, rect.y + rect.height), new Scalar(255, 0, 0), 2);
* // Extract lower half of face region
* Rect lowerHalf = new Rect(rect.x, rect.y + rect.height / 2, rect.width, rect.height / 2);
* Mat faceRegion = new Mat(frame, lowerHalf);
* // Convert to HSV
* Mat hsv = new Mat();
* Imgproc.cvtColor(faceRegion, hsv, Imgproc.COLOR\_BGR2HSV);
* // Calculate average brightness (V channel)
* Scalar meanScalar = Core.mean(hsv);
* double brightness = meanScalar.val[2];
* String label;
* Scalar color;
* if (brightness < 80) {
* label = "Mask (assumed)";
* color = new Scalar(0, 255, 0); // Green
* } else {
* label = "No Mask (assumed)";
* color = new Scalar(0, 0, 255); // Red

}

* Imgproc.putText(frame, label, new Point(rect.x, rect.y - 10),
* Imgproc.FONT\_HERSHEY\_SIMPLEX, 0.8, color, 2);
* Imgproc.rectangle(frame, new Point(rect.x, rect.y),
* new Point(rect.x + rect.width, rect.y + rect.height), color, 2);
* }
* HighGui.imshow("Simulated Mask Detection", frame);
* if (HighGui.waitKey(1) == 'q') {
* break;
* }
* }
* capture.release();
* HighGui.destroyAllWindows();
* }

}

**OUTPUT**



A person in a white shirt

AI-generated content may be incorrect.

**CONCLUSION**

The Face Mask Detection System developed in this project successfully integrates real-time computer vision techniques with deep learning models to accurately determine whether individuals are wearing face masks. Implemented using Java, OpenCV, and DeepLearning4J (DL4J), the system demonstrates a practical approach to addressing public health monitoring needs through automated visual analysis.

**Technical Achievements**

* **Face Detection:**  
  The system uses OpenCV’s Haar Cascade classifier to reliably detect faces in video frames captured from a webcam. This method proved effective in various lighting conditions and allowed the program to focus mask detection only on relevant image regions.
* **Deep Learning Model Integration:**  
  A convolutional neural network (CNN) model built with DL4J was incorporated to classify the cropped facial images into "Mask" or "No Mask" categories. The input images were appropriately preprocessed by resizing, grayscale conversion, and normalization to fit the model’s input requirements.
* **Real-Time Operation:**  
  The system processes webcam frames continuously, maintaining near real-time performance with minimal latency. The detected faces are visually marked with colored bounding boxes and labels indicating mask status, enhancing user interpretability.
* **User Interface:**  
  A simple graphical window displays the live feed with annotations, allowing users to monitor mask compliance efficiently.
* **Project Impact**

The application addresses a critical public safety concern by providing automated monitoring of mask-wearing compliance, which is especially relevant during pandemic scenarios such as COVID-19. It can be deployed in public spaces including hospitals, transportation hubs, educational institutions, and workplaces to supplement manual supervision and help enforce safety protocols.

* **Limitations**

The Haar Cascade face detector may not perform optimally under poor lighting or when faces are partially obscured.

Classification accuracy depends heavily on the quality and representativeness of the training data used for the DL4J model.

The system supports only binary classification—detecting whether a mask is worn or not—and does not handle improper mask usage.

It operates on a single video stream from one camera at a time.

**Future Enhancements**

* **Improved Face Detection:**  
  Replacing the Haar Cascade detector with more advanced deep learning-based detectors such as SSD, MTCNN, or YOLO could improve face detection robustness and accuracy.
* **Expanded Classification:**  
  Extending the model to recognize different mask-wearing states, such as incorrect usage or different types of masks, would enhance practical utility
* **Dataset Enhancement:**  
  Incorporating a larger and more diverse dataset would improve the model’s generalization across different ethnicities, lighting conditions, and occlusions.
* **Performance Optimization:**  
  Utilizing GPU acceleration or optimizing the inference pipeline could reduce latency and allow deployment on resource-constrained devices.
* **Alert and Reporting Features:**  
  Adding automated alert systems (e.g., email or SMS notifications) and logging mechanisms would enable real-time intervention and data collection.
* **Cross-Platform Deployment:**  
  Porting the application to mobile or web platforms could broaden accessibility and usage scenarios.

**REFERENCES**

* **OpenCV Library**  
  Bradski, G. (2000). *The OpenCV Library*. Dr. Dobb’s Journal of Software Tools.  
  Available: <https://opencv.org/>
* **Haar Cascade Face Detection**  
  Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 1, 511–518.  
  DOI: 10.1109/CVPR.2001.990517
* **DeepLearning4J Library**  
  DeepLearning4J Developers. (n.d.). *DeepLearning4J: Open-source, distributed deep learning for the JVM*.  
  Available: https://deeplearning4j.konduit.ai/

* **Mask Detection Using CNN Models**  
  Loey, M., Manogaran, G., Taha, M. H. N., & Khalifa, N. E. M. (2021). A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the COVID-19 pandemic. *Measurement*, 167, 108288.  
  DOI: 10.1016/j.measurement.2020.108288

* **Image Preprocessing Techniques for Deep Learning**  
  Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.  
  Available: <https://www.deeplearningbook.org/>
* **Real-time Face Mask Detection Datasets**  
  Chavda, J., Sethi, S., & Garg, A. (2020). Face Mask Detection Dataset. *Kaggle*.  
  Available: https://www.kaggle.com/andrewmvd/face-mask-detection
* **Java OpenCV Tutorials**  
  OpenCV Documentation. (n.d.). *Java Tutorials*.  
  Available: https://docs.opencv.org/4.x/d5/dae/tutorial\_java\_table\_of\_contents\_imgproc.html

**APPENDIX**

**A. Source Code Snippet**

// Loading Haar Cascade face detector

CascadeClassifier faceDetector = new CascadeClassifier("resources/haarcascade\_frontalface\_default.xml");

// Loading DL4J pre-trained model

MultiLayerNetwork model = MultiLayerNetwork.load(new File("models/mask\_model.zip"), false);

// Capturing video from webcam

VideoCapture camera = new VideoCapture(0);

Mat frame = new Mat();

while (true) {

camera.read(frame);

if (frame.empty()) break;

Mat gray = new Mat();

Imgproc.cvtColor(frame, gray, Imgproc.COLOR\_BGR2GRAY);

MatOfRect faces = new MatOfRect();

faceDetector.detectMultiScale(gray, faces);

for (Rect rect : faces.toArray()) {

Mat faceROI = new Mat(frame, rect);

String label = classifyMask(faceROI, model);

Scalar color = label.equals("Mask") ? new Scalar(0, 255, 0) : new Scalar(0, 0, 255);

Imgproc.rectangle(frame, rect.tl(), rect.br(), color, 2);

Imgproc.putText(frame, label, new Point(rect.x, rect.y - 10),Imgproc.FONT\_HERSHEY\_SIMPLEX, 0.8, color, 2);

}

HighGui.imshow("Face Mask Detection", frame);

if (HighGui.waitKey(30) == 27) break; // ESC key to exit

}

camera.release();

HighGui.destroyAllWindows();

**B. Project Folder Structure**

FaceMaskDetectionProject/

│

├── resources/

│ └── haarcascade\_frontalface\_default.xml

│

├── models/

│ └── mask\_model.zip

│

├── src/

│ └── FaceMaskDetection.java

│

├── README.md

└── build.gradle (or pom.xml if using Maven)

**C. Environment Setup Instructions**

1. **Install Java Development Kit (JDK)**  
   Ensure JDK 8 or higher is installed on your system.
2. **Install OpenCV**  
   Download OpenCV for your platform and set up native libraries.
   * Add OpenCV’s opencv\_java dynamic library to your project’s library path.
3. **Include DL4J Dependencies**  
   Use Maven or Gradle to include DeepLearning4J and ND4J libraries.
4. **Download Haar Cascade XML File**  
   Get the file haarcascade\_frontalface\_default.xml from OpenCV’s GitHub or installation directory, place it in resources/.
5. **Load the Pre-trained Model**  
   Place your trained DL4J model mask\_model.zip inside the models/ folder.
6. **Run the Application**  
   Compile and run the FaceMaskDetection class. Ensure the webcam is connected and accessible.

**D. Model Input and Output Specifications**

* **Input size:** 64 x 64 grayscale images (1 channel)
* **Normalization:** Pixel values scaled between 0 and 1 (float)
* **Output:** Two-class classification —
  + Class 0: Mask
  + Class 1: No Mask

**E. Helpful Commands**

* **To run the Java application** (assuming compiled classes are in bin/):

java -cp bin/:path\_to\_opencv\_jar:path\_to\_dl4j\_jars FaceMaskDetection

* **To exit the webcam window:**  
  Press the ESC key.