FACE MASK DETECTION

1st Nathiya A

B.Sc. Computer Science with Cognitive Systems

Dr. N. G. P. Arts and Science College

Coimbatore, India

nathiyaarumugam512@gmail.com

**ABSTRACT**

The Face Mask Detection System is an AI-driven solution designed to ensure mask compliance in public and private spaces. Using a combination of computer vision and deep learning, the system detects human faces in real-time and classifies them as "masked" or "unmasked." The project employs OpenCV for face detection, a Convolutional Neural Network (CNN) for mask classification, and Flask for the web interface. When an unmasked individual is detected, an alert system, integrated with SMTP, sends email notifications. The system achieves 98.5% accuracy in mask classification and 95% accuracy in face detection, operating at 15-20 FPS to provide real-time feedback. Key features include a real-time video feed, a configurable alert mechanism, and a user-friendly web interface. This system has potential applications in healthcare facilities, workplaces, public transport, and other high-traffic areas to enhance health and safety measures. Future enhancements may include multi-camera support, cloud-based deployment, and improved model performance through transfer learning.

**Keywords**: Face Mask Detection, Convolutional Neural Network (CNN), Deep Learning, Computer Vision, OpenCV, Real-Time Detection, COVID-19 Safety, Image Classification, Artificial Intelligence (AI), Object Detection.

**I. INTRODUCTION**

The advent of the COVID-19 pandemic significantly altered the global landscape, bringing with it an urgent need for effective preventive measures. Among the most effective and widely adopted strategies to reduce transmission was the wearing of face masks. As a result, various technologies were developed to ensure compliance with mask-wearing protocols, especially in public spaces and institutions. One such technology is the Face Mask Detection System, which uses artificial intelligence (AI) and computer vision to monitor individuals and ensure that masks are being worn appropriately. The

2nd Mrs. S. Vishnu Priya,

M.Sc., B.Ed.

B.Sc. Computer Science with Cognitive Systems

 Dr. N. G. P Arts and Science College

Coimbatore, India

vishnupriya.s@drngpasc.ac.in

Face Mask Detection System leverages Convolutional Neural Networks (CNNs) and advanced machine learning techniques to identify whether a person is wearing a mask or not. The system utilizes real-time video processing to detect

faces and classify them as either masked or unmasked. In addition to this core functionality, the system is enhanced with features like real-time alerts, email notifications, and session management, making it a comprehensive tool for monitoring mask compliance. This journal discusses the design, development, and implementation of a Face Mask Detection System that uses Flask as the web framework, OpenCV for real-time face and mask detection, and TensorFlow for AI-based classification. The system integrates with SMTP for email alerts and allows for customizations such as threshold settings and cooldowns for alert triggers. The key objectives of this project are Real-time Face and Mask Detection: Identifying faces in video feeds and determining if a mask is worn, Email Alert System: Automatically sending email alerts when unmasked individuals are detected, User Interface: Providing an interactive web-based dashboard to manage detection settings and alerts. By automating mask detection and providing real-time notifications, this system offers a practical solution to ensure public health safety during the ongoing global health crisis and beyond.

**II. LITERATURE REVIEW**

Hybrid Deep Learning Approaches: Loey et al. (2020) introduced a hybrid model combining deep transfer learning with machine learning techniques for face mask detection, achieving high accuracy rates. ​ Real-Time Detection Systems: Nagrath et al. (2020) developed SSDMNV2, a real-time face mask detection system utilizing the Single Shot Multibox Detector (SSD) and MobileNetV2, demonstrating efficient performance suitable for live applications. Transfer Learning Applications: Sethi et al. (2021) employed transfer learning with InceptionV3 to enhance face mask detection accuracy, highlighting the effectiveness of leveraging pre-trained models for this task. Comparative Analyses: Singh et al. (2021) compared YOLOv3 and Faster R-CNN models for face mask detection, providing insights into their performance within COVID-19 environments. ​ These studies collectively advance the development of efficient and accurate face mask detection systems, crucial for monitoring and enforcing health protocols during the pandemic.

**III. METHODOLOGY**

The development of the Face Mask Detection System follows a structured methodology to ensure that it can accurately detect whether a person is wearing a mask and respond accordingly. The system integrates computer vision techniques, machine learning models, and a web-based interface to achieve the desired functionality. The methodology can be divided into the following phases:

**1. System Design and Architecture**

The architecture of the Face Mask Detection System is designed to process video feeds in real-time, detect faces, and classify them as either "masked" or "unmasked." The system is built on several modules that work together seamlessly:

* **Face Detection Module**: This module detects faces within the video stream using the OpenCV library, which allows for real-time face tracking.
* **Mask Classification Module:** Once faces are detected, the system classifies them using a pre-trained Convolutional Neural Network (CNN) model. This model has been trained to differentiate between faces with masks and faces without masks.
* **Alert System:** The system is configured to send an email alert when an unmasked face is detected, ensuring timely action can be taken to address the situation.
* **User Interface (UI):** The Flask-based web interface allows users to interact with the system, view detection results, and manage alert settings.

**2. Data Collection and Preprocessing**

To train the mask detection model, a dataset consisting of images of people with and without masks is required. Publicly available datasets, such as the "Face Mask Detection Dataset," are commonly used for training the model. These datasets contain labelled images of faces in various environments, with different lighting conditions, and from various angles.

* **Data Augmentation:** To improve model performance and prevent overfitting, data augmentation techniques like rotation, zooming, and flipping are applied to increase the variety of input images.
* **Image Preprocessing:** Images are preprocessed to resize them to a uniform size and normalize pixel values before feeding them into the neural network. This helps the model learn better and faster.

**3. Model Selection and Training**

The mask detection model is based on a Convolutional Neural Network (CNN), which is well-suited for image classification tasks. The architecture of the CNN includes several convolutional layers followed by pooling layers, which help the model extract important features from the images.

* **CNN Architecture:** The model consists of multiple layers of convolutions and pooling followed by fully connected layers for classification. The output layer uses a sigmoid activation function to output the binary classification result: mask or no mask.
* **Training**: The model is trained using labeled data (masked and unmasked images). The training process involves minimizing a binary cross-entropy loss function, which measures the difference between the predicted and actual class labels.

**4. Real-Time Face and Mask Detection**

Once the model is trained, it is integrated into the real-time video feed processing pipeline. Using OpenCV, the system captures video frames from a webcam or IP camera. The face detection module identifies faces in each frame, and the mask classification model is applied to each detected face.

* **Face Detection with OpenCV:** The Haar Cascade Classifier, a feature-based classifier, is used to detect faces in real-time. It is a lightweight and fast method, making it suitable for real-time applications.
* **Mask Classification**: After detecting a face, the image of the face is passed through the trained CNN model, which outputs whether the person is wearing a mask or not.

**5. Alert System**

In situations where an unmasked person is detected, the system triggers an alert. The alert is handled through an SMTP (Simple Mail Transfer Protocol) integration, which sends email notifications to predefined recipients.

* **Threshold Settings**: Users can configure the threshold for detection, such as the number of unmasked faces in a given time period before an alert is triggered. This customization allows for flexibility in the system’s operation.
* **Cooldown Period**: A cooldown period is implemented to prevent multiple alerts for the same unmasked individual within a short period. This reduces alert fatigue and ensures the system is not overreacting to brief lapses.

**6. Web Interface (Flask)**

The system’s web interface is built using Flask, a lightweight web framework for Python. The interface includes several key features:

* **Login and Session Management**: Users can log in to the system, ensuring secure access to the dashboard and control settings.
* **Dashboard:** The dashboard displays real-time video feed and provides options to enable or disable email alerts, adjust threshold settings, and view the history of email alerts sent.
* **Real-Time Detection:** The video stream is shown on the dashboard, and the system indicates whether a detected face is wearing a mask or not. If a person is unmasked, an alert is triggered.

**IV. ARCHITECTURE**

The architecture of the Face Mask Detection System is designed to integrate computer vision, machine learning, and web technologies into a cohesive solution for real-time mask detection and alert management. The system's modular design allows for efficient communication between various components, ensuring that each part performs its function optimally.

**1. Overall System Architecture**

The system architecture can be broken down into the following key components:

* **Input (Video Feed):** The system receives live video input from a camera (webcam or IP camera) that captures images or video streams for face detection.
* **Face Detection Module**: This module detects faces in the video feed using OpenCV’s Haar Cascade Classifier or other face detection algorithms.

 • **Mask Classification Module:** After detecting faces, this module uses a pre-trained Convolutional Neural Network (CNN) to classify each face as either "masked" or "unmasked."

* **Alert System:** If the system detects an unmasked individual, it triggers an alert, which is sent via email using SMTP integration.
* **Web Interface (Flask):** The web interface displays the real-time video feed, mask detection results, and system settings for the user. It also allows the user to interact with the system, such as enabling or disabling alerts or adjusting threshold settings.
* **Database (Optional):** Although not implemented in this project, a database could be used to store images of unmasked individuals, email alert history, and user preferences.

**2. Component Breakdown**

**a. Camera/Video Capture**: The first component of the system is the video feed input, which can be a webcam, an IP camera, or any other video capture device. The camera streams frames continuously, which are processed by the system for face detection. This component is crucial for the real-time functionality of the mask detection system.

**b. Face Detection:** Once a video frame is captured, the face detection algorithm is applied to locate faces in the image. OpenCV’s Haar Cascade Classifier is commonly used in real-time applications due to its efficiency and fast detection capabilities. It operates by scanning the image at multiple scales and detecting face-like patterns based on pre-trained data. The detected faces are then passed to the next stage for mask classification.

**c. Mask Classification:** This module is the core of the system, where Convolutional Neural Networks (CNN) come into play. The CNN model, trained on a dataset of faces with and without masks, classifies the detected faces as either masked or unmasked. The CNN model architecture typically consists of several convolutional layers followed by fully connected layers that output a binary result (masked or unmasked).

**d. Email Alert System:** When an unmasked individual is detected, the system triggers an email alert. This component integrates the Simple Mail Transfer Protocol (SMTP) to send an email to a predefined list of recipients. The email includes information such as the detected time and the status of the face (unmasked). The alert system also allows users to configure thresholds (e.g., the number of unmasked faces in a given time) and cooldown periods to avoid repetitive notifications.

**e. Flask Web Interface:** The Flask web framework serves as the backbone of the user interface, providing a dashboard where users can:

* View the real-time video feed.
* Monitor the mask detection results for each individual.
* Adjust system settings such as threshold levels for alert triggers and cooldown periods.
* Enable or disable email notifications.
* View the history of email alerts sent.
* Configure camera settings or other system parameters.

Flask routes manage the communication between the front-end user interface and the back-end Python code, ensuring smooth interaction with the system. The web interface is accessible through any browser, making it convenient for users to monitor the mask detection process.

**3. Communication Between Components**

 The components communicate as follows:

* **Video Capture to Face Detection:** The video feed is processed frame by frame, passing each frame to the face detection module.
* **Face Detection to Mask Classification:** Once faces are detected, their corresponding images are passed to the CNN model for classification.
* **Mask Classification to Web Interface:** The classification result (masked or unmasked) is sent to the web interface for real-time display.
* **Mask Classification to Alert System:** If the result is "unmasked," the email alert system is triggered.
* **Web Interface to Flask Backend:** The web interface communicates with the Flask backend, which manages user interactions and handles settings adjustments.

**4. Technology Stack**

* **Python:** The programming language used to implement the core functionality, including face detection, mask classification, and email alert system.
* **OpenCV:** A library for computer vision tasks, such as video capture and face detection.
* **TensorFlow/Keras:** Deep learning frameworks used to train and deploy the CNN model for mask classification.
* **Flask:** A lightweight web framework used for the user interface, which allows users to interact with the system via a browser.
* **SMTP:** Protocol used for sending email notifications when unmasked individuals are detected.

**V. IMPLEMENTATION**

The implementation of the Face Mask Detection System involves several key steps, from setting up the development environment to coding the core functionalities and deploying the system. The following section outlines the detailed process of implementing the system, including the technical stack, coding process, integration of modules, and testing.



Figure 5.1

**1. Development Environment Setup**

The first step in the implementation process is to set up the development environment. For this project, the following tools and libraries were used:

* **Python 3.x:** The programming language for implementing the machine learning model, video processing, and Flask-based web interface.
* **TensorFlow/Keras:** These deep learning libraries are used to implement and train the Convolutional Neural Network (CNN) model for mask detection.
* **OpenCV:** A computer vision library that facilitates real-time face detection from video feeds.
* **Flask:** A web framework for creating the user interface and handling interactions between the user and the backend.
* **SMTP (Simple Mail Transfer Protocol):** Used to send email alerts when an unmasked individual is detected.

All libraries and dependencies are installed using pip or conda to ensure proper version management. The environment is created using either virtual environments or Docker for consistent deployment across different machines.

**2. Face Detection Module**

The first core module developed is the face detection module. OpenCV provides several algorithms for detecting faces in an image or video feed. The Haar Cascade Classifier was used in this project due to its efficiency and relatively low computational cost for real-time processing.

1. **Face Detection Flow:**
* **Capture video Feed:** The webcam or an IP camera is used to capture continuous video frames.
* **Convert Frames to Grayscale:** Since the Haar Cascade Classifier works on grayscale images, each captured frame is converted to grayscale.
* **Apply Haar Cascade Classifier:** The classifier is applied to detect faces in the grayscale frame.
* **Draw Rectangle Around Detected Faces:** Once faces are detected, rectangles are drawn around them to visualize the detection.

This is done continuously for each frame captured by the camera, enabling real-time face detection.



Figure 5.2

**3. Mask Classification Module**

The mask classification module leverages a Convolutional Neural Network (CNN), trained on a dataset of images of people wearing masks and without masks. The CNN model is responsible for classifying detected faces as "masked" or "unmasked."

1. **Training the CNN Model:**
* **Dataset:** A publicly available dataset of images with and without masks was used for training. The dataset contains multiple images of people in different environments, lighting conditions, and angles.
* **Preprocessing:** The images are resized to a fixed size (e.g., 128x128 pixels) and normalized to have pixel values between 0 and 1. This ensures the model learns effectively from the data.
* **CNN Architecture:** The CNN model consists of several layers, including convolutional layers for feature extraction, pooling layers to reduce the spatial dimensions, and fully connected layers for final classification. The output layer uses a sigmoid activation function to classify faces into two categories: masked or unmasked.
* **Training:** The model is trained using binary cross-entropy as the loss function and an optimizer like Adam to minimize the error. The model's accuracy is evaluated using a separate test set to ensure it generalizes well.
* **Model Saving:** Once trained, the model is saved in a format that can be loaded later for real-time classification in the system.



Figure 5.3

**b. Real-Time Classification:** Once faces are detected, the image of each detected face is passed to the CNN model for classification. The model returns a probability score that is thresholded to classify the face as either "masked" or "unmasked."



Figure 5.4

**4. Email Alert System**

The email alert system is implemented to notify users when an unmasked individual is detected. This is achieved using SMTP, which is configured to send emails to a predefined recipient list.

**a. Alert Flow:**

* When an unmasked face is detected, the system triggers the alert by calling the SMTP server.
* The system constructs an email that contains details such as the detected individual’s status (unmasked), timestamp, and an optional image.
* The email is sent to the user’s email address using SMTP, informing them of the violation.
* **Thresholds:** The alert is only triggered after a certain threshold of unmasked detections in a specified time period, to prevent repeated alerts for brief lapses.
* **Cooldown Period:** A cooldown is implemented to ensure that the system does not send multiple alerts for the same unmasked individual in quick succession.
* **Flask Web Interface:** The Flask web interface serves as the front end for the system. It is used to display the real-time video feed, allow the user to interact with the system, and manage configurations such as alert settings and system preferences.



Figure 5.5

**b. Key Features of the Web Interface:**

* **Login System:** Users need to log in with their credentials to access the dashboard and configure the system settings.
* **Real-Time Video Feed:** The web interface shows the video feed from the camera, with rectangles drawn around detected faces. The classification result (masked or unmasked) is displayed for each detected face.
* **Alert Settings:** Users can configure settings such as the number of unmasked faces before triggering an alert and set the cooldown period to manage alert frequency.
* **Email Alert Configuration:** The system allows users to enable or disable email alerts, and configure the email server settings.
* **Detection History:** A section can be included to show the history of detected unmasked individuals, the email alerts sent, and logs of system activity.
* **Flask Routing:** The backend logic is implemented in Flask routes, where each route corresponds to a specific action or page. For example:
* **/ login:** Displays the login page.
* / **dashboard:** Displays the dashboard with the live feed and mask detection results.
* **/ configure:** Allows users to set their preferences for alerts and email configuration.

**6. Testing and Evaluation**

* Once the core modules were implemented, the system underwent rigorous testing to evaluate its performance.
* **Face Detection Accuracy:** The face detection module was tested in various lighting conditions and camera angles to ensure that it accurately detects faces.
* **Mask Detection Accuracy:** The CNN model was evaluated on a separate test dataset to measure its accuracy, precision, recall, and F1 score in detecting whether a face was masked or unmasked.
* **Real-Time Performance:** The system’s real-time performance was tested to ensure that it can detect faces and classify them without significant delay.
* **Email Alert Testing:** The email alert system was tested by manually triggering unmasked detections and verifying that the correct alerts were sent.
* **User Interface Testing:** The Flask-based web interface was tested to ensure that it displayed the video feed, mask detection results, and alert settings correctly.

**VI. RESULT**

The Face Mask Detection System was evaluated based on detection accuracy, real-time performance, and the email alert system functionality.

**1. Detection Accuracy**

**a. Face Detection:** The Haar Cascade Classifier detected faces with 95% accuracy, with minor issues in extreme lighting or partial obstructions.

**b. Mask Classification:** The CNN model achieved 98.5% accuracy, with precision of 98.3%, recall of 98.6%, and F1 score of 98.4%, proving it robust in varied real-world scenarios.

**2. Real-Time Performance**

a. The system processed 15-20 frames per second (FPS) on a standard laptop, sufficient for real-time detection. Latency was around 100-200 milliseconds, ensuring responsive feedback.

**3. Email Alert System**

a. Email alerts were triggered promptly upon detecting unmasked individuals. The system allowed configuration of alert thresholds and cooldown periods, functioning as expected.

**4. Flask Web Interface**

a. The web interface displayed real-time video with face detection results and allowed users to configure settings like alerts and thresholds. It also showed alert history.

**5. System Robustness**

a. The system adapted well to different lighting conditions and handled partial obstructions. It also performed well in multi-face scenarios.

 **VII. DISCUSSION**

The Face Mask Detection System demonstrated promising results in both accuracy and real-time performance.

**1. System Performance**

**a. Strengths:**

* Real-time detection with 15-20 FPS.High accuracy: 98.5% in mask classification and 95% in face detection.
* Adaptability to different lighting conditions and face obstructions.

**b. Limitations:**

* Challenges with highly obstructed faces and high-resolution feeds, where performance dropped slightly.

**c. Real-World Applications**

* Potential for use in public health monitoring (airports, hospitals), offices, and crowded spaces (malls, public transport).
* Email alerts provide real-time notifications for unmasked individuals.

**d. Challenges During Development**

* **Data Collection:** Difficulty in obtaining a diverse dataset with varied lighting and mask types.
* **Performance Optimization:** Issues with high-resolution video feeds, requiring further optimization.
* **False Positives/Negatives:** Occasional misclassifications due to dynamic environments.

**e. System Scalability and Flexibility**

* **Future Enhancements:** Multi-camera support, cloud integration, and mobile device compatibility could improve scalability and accessibility.

**f. Ethical Considerations**

* **Privacy Concerns**: Need to ensure data is processed according to privacy laws, using anonymized data when necessary.
* **Bias:** Ensuring diverse training datasets to prevent performance issues across different demographics.

**g. Future Directions**

* **Model Enhancement:** Incorporating Transfer Learning and adaptive learning for continuous improvement.
* **Advanced Alerts**: Integrating voice notifications and automated actions for enhanced safety.

**VIII. CONCLUSION**

The Face Mask Detection System effectively addresses the need for automated mask compliance monitoring in public and private spaces. Using Haar Cascade Classifier for face detection and CNNs for mask classification, the system achieved high accuracy and real-time performance.

**Key Achievements:**

* **High Accuracy:** 98.5% mask classification and 95% face detection.
* **Real-Time Performance:** 15-20 FPS for immediate feedback and alerts.
* **Email Alerts:** Automated notifications for unmasked individuals.
* **Scalability:** Ability to extend to multi-camera setups, cloud integration, and mobile support.
* Challenges:
* **Face Obstruction:** Difficulty with heavily obstructed faces.
* **High-Resolution Feeds**: Performance issues with higher-resolution videos.
* **Privacy Concerns**: Ethical considerations regarding face detection.
* **Model Enhancement:** Using transfer learning and advanced techniques for improved accuracy.
* **Adaptive Learning**: Continuous system improvement with new data.
* **Scalability:** Expanding to multi-camera and IoT integration for large-scale deployment.

In conclusion, the system offers a strong solution for enforcing health protocols, with potential for further improvements in accuracy, scalability, and integration.

**IX. REFERENCE**

1. Loey, M., Manogaran, G., Taha, M. H. N., & Khalifa, N. E. M. (2020). 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. https://doi.org/10.1016/j.measurement.2020.108288

2. Nagrath, P., Jain, R., Madan, A., Arora, R., Kataria, P., & Hemanth, J. (2020). SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2. Sustainable Cities and Society, 66, 102692. https://doi.org/10.1016/j.scs.2020.102692

3. Sethi, S., Kathuria, M., & Kaushik, T. (2021). Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread. Journal of Biomedical Informatics, 120, 103848. https://doi.org/10.1016/j.jbi.2021.103848

4. Singh, S., Ahuja, U., Kumar, M., Kumar, K., & Sachdeva, M. (2021). Face mask detection using YOLOv3 and faster R-CNN models: COVID-19 environment. Multimedia Tools and Applications, 80(13), 19753–19768.

https://doi.org/10.1007/s11042-021-10711-8

5. Wu, P., Li, H., Zeng, N., & Li, F. (2021). FMD-Yolo: An efficient face mask detection method for COVID-19 prevention and control in public. Image and Vision Computing, 117, 104341. https://doi.org/10.1016/j.imavis.2021.104341