FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY INTERNATIONAL BUSINESS, SCIENCE AND TECHNOLOGY UNIVERSITY

BONAFIDE CERTIFICATE

I Certify that this project report entitled: **Real Time Face Mask Detection System**. is Bonafide work of **Mr. Geng Alex Akok.** who carried out the project under my supervision. bearing Roll No. **012220256** and is a Bonafide student of **INTERNATIONAL BUSINESS, SCIENCE AND TECHNOLOGY UNIVERSITY (ISBAT)**, is in partial fulfilment for the award of **BACHELORS OF SCIENCE IN APPLIED INFORMATION TECHNOLOGY.** during Spring-July 2025.

It is certified that all the corrections/suggestions indicated for Internal Assessment have been incorporated in the submitted report. The Project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the said Degree.

DEAN, FICT	**SUPERVISOR**
……………………………….	……………………………..
Dr. Tyagi Vaibhav Bhushan	Mrs. Alekhya Kukutla
(Signature with date)	(Signature with date)

External Examiner:

Name: Signature

DECLARATION

I **Geng Alex Akok** bearing Roll. No:**012220256** hereby declare that this project report entitled: Real-Time Face Mask Detection System has been prepared by me towards the partial fulfilment of the requirement for the award of the Degree **BACHELORS OF SCIENCE IN APPLIED INFORMATION TECHNOLOGY** under the guidance of **Mrs. Alekhya Kukutla**

I also declare that this project report is my original work and has not been previously submitted for the award of any Degree, Diploma, Fellowship, or other similar titles.

Signature **Date:**

Acknowledgment

I would like to express my deepest gratitude to my supervisor, **Ms. Alekhya Kukutla**, for her continuous support, guidance, and encouragement throughout this project. Her insightful suggestions and feedback were instrumental in shaping the direction of this work.

I also extend my sincere appreciation to the faculty and staff of the **Faculty of Information and Communication Technology at ISBAT University** for providing a conducive learning environment and the resources necessary to complete this project.

A special thanks to my classmates and friends who provided support and motivation throughout this journey.

Lastly, I am thankful to my family for their unwavering love and support during the entire course of my studies.

Abstract

This project presents the design, development, and evaluation of a real-time face mask detection system, aimed at supporting public health and safety through intelligent, automated monitoring. The system was motivated by the need to enforce mask-wearing compliance during pandemic conditions—particularly in environments where manual enforcement is inefficient, inconsistent, or impractical. By leveraging artificial intelligence (AI), this tool provides a scalable and effective solution that reduces reliance on human supervision and improves safety without compromising accessibility.

At its core, the system uses **computer vision** and **deep learning** techniques. The face detection component is implemented using OpenCV's deep neural network module, while the classification of face masks is handled by **MobileNetV2**, a lightweight convolutional neural network optimized for speed and resource efficiency. The application is written in **Python** and provides a clean, intuitive interface using **Tkinter**, Python's standard GUI toolkit.

Unlike many commercial systems that depend on cloud infrastructure or internet connectivity, this application is designed to work **fully offline**, making it ideal for deployment in **low-resource environments**. It supports **two modes of operation**: real-time video input from a webcam and static image upload for on-demand testing. Detection results are visually represented through bounding boxes and labels ("Mask" or "No Mask"), supported by real-time **audio alerts** and **timestamped logs**, all of which are displayed within the graphical interface.

To ensure reliability and robustness, the system was tested under various conditions, including different lighting environments, face angles, and mask types. The model achieved an impressive **accuracy rate of 96.1%** and sustained real-time performance between **12 and 20 frames per second** on standard hardware without requiring GPU acceleration. Additional features such as dark mode, multi-face detection warning, and sound feedback were evaluated through usability testing with students and staff, all of whom reported the system to be easy to use, fast, and practical.

User feedback and testing results indicate that this system is not only functionally sound but also well-suited for real-world deployment in **schools**, **healthcare facilities**, **corporate offices**, and other public settings. The offline-first design also aligns with **privacy-conscious deployment**, ensuring no data is stored or transmitted externally.

This project demonstrates a successful application of AI and human-computer interaction design in solving a real-world public health challenge. It lays a strong foundation for future enhancements such as **mobile integration**, **multi-face tracking**, and **cloud-based analytics**, ultimately contributing to smarter, safer, and more automated public safety systems powered by artificial intelligence.

Table of contents

1. Introduction and Background

1.1 Statement of Problem Area

The emergence and global spread of the COVID-19 pandemic brought about unprecedented changes to daily life, public health policies, and social behavior. As the virus spread rapidly across countries and continents, governments and international health organizations implemented various preventive strategies to curb its transmission. Among these, the consistent and proper use of face masks became one of the most visible and essential measures to protect individuals and communities. Masks were especially emphasized in enclosed or crowded spaces such as public transport systems, hospitals, offices, shopping malls, schools, and airports—areas where physical distancing was difficult to maintain.

Despite the importance of mask-wearing, ensuring that individuals comply with this guideline has proven to be a persistent challenge, particularly in public and high-traffic environments. Relying on human personnel to manually monitor whether people are wearing masks is highly inefficient. It not only consumes time and financial resources but also poses a direct health risk to those tasked with enforcement, especially during peak infection periods. Moreover, human-based monitoring systems suffer from limitations such as fatigue, bias, inconsistent judgment, and difficulty in overseeing large or fast-moving crowds. These issues compromise the overall effectiveness of compliance efforts.

Another major concern is the scalability of manual enforcement. In cities with millions of residents or organizations managing thousands of daily visitors, maintaining consistent monitoring becomes virtually impossible without automation. In addition, there is an increasing demand for non-invasive, privacy-respecting technologies that can operate in real-time without causing delays or discomfort to individuals.

To address these challenges, the development of intelligent systems using Artificial Intelligence (AI) and Computer Vision (CV) technologies has become a vital area of innovation. These systems can automate the detection of face masks with minimal human intervention, offering a more scalable, accurate, and cost-effective solution. Real-time face mask detection systems are capable of identifying whether individuals are wearing masks correctly—covering both nose and mouth—and can even detect multiple people simultaneously within a frame. They reduce the dependency on physical manpower and make it possible to deploy mask enforcement across diverse environments such as schools, transportation hubs, retail stores, and government institutions.

Such systems not only promote public safety but also help reinforce trust in automated monitoring solutions. The public tends to view technological enforcement as more impartial and consistent compared to human monitors. By integrating real-time mask detection technologies into security infrastructures like CCTV networks or public entrance scanners, institutions can enhance both health security and operational efficiency.

1.2 Previous and Current Work

Before the integration of artificial intelligence in public health monitoring, mask-wearing enforcement primarily relied on manual observation and human judgment. In the early stages of the COVID-19 pandemic, governments and institutions deployed human personnel such as security guards, health officers, and volunteers to visually inspect individuals at entry points of public facilities. In some cases, signage, public announcements, and written warnings were used to inform and encourage the public to wear masks. These methods were basic but necessary, especially in regions lacking technological infrastructure. However, they quickly proved insufficient in areas with high foot traffic, where monitoring each individual became a burdensome and sometimes impossible task.

CCTV systems were also employed in many environments to monitor public behavior. These closed-circuit systems allowed remote surveillance from a centralized location, but they still depended heavily on human operators to interpret video feeds and identify non-compliant individuals. While CCTV extended the reach of monitoring, it did not eliminate human limitations such as distraction, fatigue, and inconsistency in judgment. Moreover, reviewing recorded footage to track mask violations after incidents occurred offered no real-time prevention or intervention benefits.

With advancements in machine learning and computer vision, researchers began exploring ways to automate the detection of face masks in video feeds. Initial prototypes focused on simple image classification—determining whether a single image contained a masked or unmasked face. These early models had limited real-world application, as they often required controlled environments, ideal lighting, and frontal facial views to perform accurately. Many could not differentiate between a properly worn mask and one that was below the nose or hanging from one ear.

As the need for robust solutions grew, academic institutions, private companies, and open-source communities began developing more advanced and adaptable systems. Datasets were compiled from various sources to train deep learning models capable of recognizing masks from multiple angles, under different lighting conditions, and across diverse face shapes and skin tones. The models evolved from binary classifiers to more sophisticated detection systems capable of performing face localization, segmentation, and classification simultaneously. These models could now process video streams in real-time, making them suitable for live monitoring scenarios.

One of the major breakthroughs in this field has been the integration of face mask detection with existing surveillance infrastructures. Instead of installing new cameras, institutions could now upgrade their current CCTV systems with AI-based software that overlays real-time mask detection on video feeds. These systems typically use convolutional neural networks (CNNs) and transfer learning techniques for better generalization and faster deployment. Frameworks like TensorFlow, Keras, and OpenCV became popular due to their flexibility and community support.

Recent systems also include features such as sound alerts, real-time notifications, and face tracking, allowing for immediate action when a violation is detected. Some solutions are even integrated with entry control systems (e.g., automatic doors, turnstiles), which remain locked until a person with a properly worn mask is detected. Such implementations demonstrate the increasing maturity and practical application of these technologies.

Despite significant progress, there are still challenges in ensuring high accuracy under difficult conditions. Detecting faces with dark skin tones in low-light environments, distinguishing masks from face coverings like scarves, and handling crowded scenes with overlapping faces remain complex problems. However, continuous improvements in model training, data augmentation, and real-time inference technologies are gradually bridging these gaps.

1.3 Background

Face detection, a critical component of computer vision, refers to the process of identifying and locating human faces within digital images or video frames. It plays a fundamental role in many AI applications, including facial recognition, emotion analysis, human-computer interaction, and security systems. Over the past decade, face detection has evolved from basic template-matching techniques to highly advanced deep learning-based approaches capable of detecting faces under varying lighting, angles, and occlusion conditions.

Mask detection builds upon face detection by adding an additional layer of analysis. Once a face is detected, the system must classify it into categories such as **“Mask”**, **“No Mask”**, or even **“Improperly Worn Mask”** in more sophisticated systems. This additional classification step involves training convolutional neural networks (CNNs) on labeled image datasets that contain examples of masked and unmasked faces. The quality and diversity of these datasets are crucial to the accuracy and generalization of the model, especially in real-world scenarios with diverse face types, skin tones, and mask styles.

In this project, **MobileNetV2** has been chosen as the core deep learning architecture for mask classification. MobileNetV2 is a lightweight CNN architecture developed by Google, optimized for resource-constrained environments such as mobile phones, Raspberry Pi boards, or other embedded systems. It employs depthwise separable convolutions, which drastically reduce the number of parameters and computation compared to traditional CNNs like VGG16 or ResNet50. Despite its low complexity, MobileNetV2 achieves high accuracy, making it particularly well-suited for **real-time inference**, where speed and performance must be carefully balanced.

The use of **Tkinter**, Python’s built-in GUI toolkit, provides an interactive interface for users to control and monitor the system. Tkinter simplifies the development of desktop applications by allowing easy creation and management of buttons, labels, dropdowns, and display panels. Through this graphical interface, users can upload images for testing, start or stop the webcam stream, toggle dark mode, or view alerts—making the system not only functional but also user-friendly. The GUI helps bridge the gap between back-end AI processes and front-end usability, which is especially important for non-technical users.

OpenCV (Open Source Computer Vision Library) plays a pivotal role in handling image and video data. It is used to capture webcam feeds, preprocess input frames (such as resizing, color conversion, and face detection), and draw bounding boxes around detected faces. OpenCV provides real-time computer vision functions that are both efficient and well-documented, making it a popular choice for live video applications.

For model inference, **TensorFlow and Keras** are utilized. TensorFlow is a widely used machine learning framework, and Keras, which operates as its high-level API, simplifies model design and inference. These tools allow seamless loading of the pre-trained MobileNetV2 model, performing predictions on detected faces in each frame to determine mask status.

Additionally, **pygame**, a Python library primarily designed for game development, is leveraged for sound alerts. By using simple commands, pygame can play alert sounds when specific events occur—such as when a face without a mask is detected or when multiple faces appear in the frame. These audio cues provide immediate feedback, making the system more interactive and effective in real-world environments like clinics, schools, or entrances to public facilities.

Bringing these technologies together creates a **comprehensive, offline-capable** system that is not only accurate but also responsive and adaptable. It operates fully on a standard PC without the need for cloud processing, which makes it ideal for deployments in locations with limited internet access or where data privacy is a concern.

1.4 Brief Project Description

This project presents a **Real-Time Face Mask Detection System** developed using Python and deep learning technologies. It is designed to operate fully **offline**, offering a practical solution for institutions or communities with limited or no internet access. The system processes live video feeds from a built-in laptop camera or any connected external webcam, detecting human faces in real time and determining whether each individual is wearing a face mask properly.

At the heart of the system is a custom-trained machine learning model based on the **MobileNetV2** architecture, which balances efficiency and accuracy—an essential factor for real-time applications. When a face is detected, the system draws a colored bounding box around it: **green** for faces wearing masks and **red** for those not wearing masks. In cases of non-compliance, the system triggers an **auditory alert**, ensuring that the user is immediately notified without needing to constantly monitor the screen.

The application is not limited to live video. It also allows users to **upload static images** for mask detection analysis. This feature is useful for reviewing photos or testing detection accuracy in a controlled way. Each result is visually displayed in the GUI along with a clear label indicating the mask status.

The software provides a **Graphical User Interface (GUI)** developed using **Tkinter**, making it accessible to both technical and non-technical users. The interface includes intuitive controls:

- **Start** and **Stop** buttons to control the live webcam feed.
- An **Upload** button to select and analyze saved images.
- A **Dark Mode Toggle**, which improves visibility and user comfort during prolonged use, especially in low-light environments.
- A **detection log** panel that records each detection event along with a **timestamp**, providing a history of results that can be reviewed or exported for record-keeping.

The application also includes integrated **sound alerts** using the **pygame** library. These alerts play different tones based on the situation—for example, a warning sound for "No Mask Detected" or a notice when multiple faces are present. These audio cues enhance usability and ensure the system remains effective even when the visual interface is not being actively observed.

The codebase is written entirely in Python using a **modular design structure**. This means different components of the system—such as face detection, mask classification, GUI logic, and sound alerts—are separated into distinct modules or functions. This approach improves **maintainability**, allowing developers to fix bugs or upgrade individual parts of the system without affecting the entire codebase. It also supports **extensibility**, enabling easy implementation of new features in future versions.

Potential future upgrades include:

- **Facial recognition integration**, allowing the system to identify individuals in addition to checking their mask status.
- **Cloud syncing**, to back up detection logs or send real-time alerts to administrators or security personnel.
- **Access control integration**, such as connecting to electronic doors or turnstiles that only allow entry to people wearing masks.

1.5 Purpose / Objectives / Justification

The core **purpose** of this project is to design, develop, and demonstrate a **real-time face mask detection system** that is fast, user-friendly, and capable of operating **offline**, independent of any cloud-based infrastructure. This makes it suitable for deployment in a wide variety of settings, including those with limited or unstable internet connectivity. The project highlights the practical application of artificial intelligence (AI) in solving urgent public health challenges while also offering a learning platform for students and developers.

The project has several well-defined **objectives**, including:

1. **Ensure Public Health Safety Through Automation:**

2. One of the most critical objectives is to support pandemic control efforts by automating the detection of face mask compliance. This reduces reliance on human judgment, eliminates fatigue-based errors, and provides consistent, round-the-clock monitoring in public spaces. It helps enforce safety regulations without the need for physical confrontation or manual checks.

3. **Reduce Human Labor and Risk in Surveillance:**

Manual enforcement of mask mandates in public institutions or commercial settings often requires a dedicated team of staff, who are themselves exposed to health risks. By replacing or assisting human surveillance with an automated system, this project helps **minimize human exposure** and reduces the **costs and logistics** associated with staffing.

4. **Provide a Standalone, Reliable Tool for Institutions:**

Many organizations, especially in developing countries or rural areas, do not have access to sophisticated cloud-based surveillance systems. This project provides a **fully offline solution** that can run on an ordinary laptop, making it accessible and practical for hospitals, schools, airports, offices, and even small businesses.

5. **Integrate AI with GUI and Audio Feedback for Real-Time Alerts:**

The system serves as a demonstration of **AI integration with front-end technologies**, combining computer vision, graphical interfaces (Tkinter), and audio feedback (pygame) into a single, cohesive application. It shows how multiple technologies can work together in real-time to enhance user experience and functionality.

6. **Offer a Research and Learning Platform for AI Students and Developers:**

By using widely adopted libraries such as TensorFlow, OpenCV, and Keras, the project serves as a valuable educational resource. It introduces students to practical aspects of machine learning, image classification, user interface design, and application deployment. The modular nature of the codebase also allows learners to customize or extend the project for further experimentation.

7. Lay a Foundation for Future Biometric and Safety Applications:

Beyond the scope of face mask detection, the project has the potential to evolve into more advanced systems involving **facial recognition, identity verification, or health screening**. Its modular design and real-time processing capabilities make it a strong foundation for future development in the field of public safety, security, and smart surveillance.

8. Maintain Privacy and Accessibility:

Unlike many surveillance technologies that rely on cloud processing or internet access, this system respects user **privacy** by processing all data locally on the device. It avoids uploading any sensitive information to remote servers, which is especially important in medical and educational institutions. The use of a graphical interface ensures **accessibility**, even for users with limited technical knowledge.

2. Literature Review

The development of **automated face mask detection systems** is rooted in the broader field of **computer vision**, which has steadily evolved to address real-world problems in areas such as surveillance, identity verification, emotion analysis, and public health. The emergence of the COVID-19 pandemic brought new urgency to research and innovation in the domain of face detection and classification—prompting the creation of numerous AI-based systems aimed at detecting whether individuals are wearing face masks properly in public spaces.

This section provides a structured review of past and current literature, highlighting the evolution of methodologies from traditional computer vision techniques to modern deep learning frameworks.

2.1 Early Research in Face Detection and Classification

Face detection is one of the earliest and most fundamental problems tackled in computer vision. It serves as the first step in many higher-level tasks, such as face recognition, tracking, and facial expression analysis. Early research focused on identifying patterns in grayscale images that could differentiate a face from the background.

One of the most widely known classical methods is the **Haar Cascade Classifier**, introduced by **Paul Viola and Michael Jones in 2001**. The Viola-Jones algorithm uses a cascade of classifiers trained using **AdaBoost**, a machine learning meta-algorithm that combines multiple weak classifiers into a strong one. Haar features—simple patterns such as edges and lines—were calculated across different regions of an image, and the model quickly classified whether a face was present. The method became popular due to its speed and real-time performance, especially for applications on resource-limited hardware.

However, despite its computational efficiency, Haar Cascades suffered from **low robustness** in real-world scenarios. It performed well in ideal conditions (frontal face, good lighting) but struggled with:

- **Partial occlusion** (e.g., face partially hidden by glasses, hands, or masks)
- **Non-frontal face angles**
- **Varying illumination**
- **Low-resolution or noisy images**

These limitations hindered its applicability in complex or uncontrolled environments like crowded public spaces or live video streams. As such, researchers sought better-performing models using more advanced techniques.

Following Haar Cascades, the **Histogram of Oriented Gradients (HOG)** method gained popularity for object detection, including face detection. HOG features capture edge and shape information, and when combined with **Support Vector Machines (SVMs)**, they offer better generalization. This approach provided improved accuracy over Haar-based systems but still lacked the adaptability and learning power needed for large-scale variation in face appearances.

The shift toward **deep learning** revolutionized the field, beginning with the rise of **Convolutional Neural Networks (CNNs)** in image classification and detection tasks. CNN-based models could automatically learn hierarchical features from raw pixel data, eliminating the need for manual feature engineering.

Popular architectures like **AlexNet (2012)**, **VGGNet**, and **ResNet** laid the groundwork for more robust face detection systems. These networks provided significantly improved accuracy, especially when trained on large and diverse datasets.

With the success of CNNs, face detection transitioned into a new era of **end-to-end learning systems** that not only detect the location of faces in an image but also recognize facial expressions, gender, age, and mask presence. Real-time detection frameworks like **YOLO (You Only Look Once)** and **SSD (Single Shot Detector)** further improved processing speed, enabling applications in video surveillance, access control, and mobile health technologies.

Thus, the progression from Haar Cascades to deep learning represents a major leap in both performance and reliability. These advancements have laid the technical foundation for the development of automated face mask detection systems that function in real-world environments with high accuracy and real-time capability.

The advent of **Convolutional Neural Networks (CNNs)** drastically transformed the landscape of face detection and classification by enabling systems to automatically learn complex visual features directly from image data. Unlike traditional approaches that relied on hand-crafted features, CNNs use multiple layers of convolution, activation, and pooling operations to extract hierarchical representations of input images—starting from simple edges in the early layers to more abstract features in the deeper layers.

Architectures such as **AlexNet (2012)**, **VGG16 (2014)**, and **ResNet (2015)** marked major breakthroughs in image recognition competitions like ImageNet. These models proved capable of identifying thousands of object categories with high accuracy. In the context of face detection, their ability to recognize subtle variations in facial structure, expressions, and accessories (e.g., glasses, masks) made them far more reliable than earlier techniques. Their depth and layer-wise training allowed for a much richer feature space, improving both detection performance and robustness in challenging environments.

These models laid the groundwork for **specialized applications**, including **face mask detection**, by providing a foundation upon which new datasets and tasks could be adapted through techniques like **transfer learning**. Instead of training a model from scratch, developers could fine-tune pre-trained CNNs on face mask datasets—drastically reducing training time and increasing model generalization, even on relatively small datasets.

This shift enabled the rapid development of mask detection solutions during the COVID-19 pandemic. By adapting these CNN architectures to distinguish between "mask" and "no mask" classes, researchers and engineers were able to quickly create high-performing systems suitable for public deployment. Additional refinements—such as using lightweight models like **MobileNet** for real-time inference on edge devices—expanded the range of practical implementations, making it possible to deploy face mask detection on mobile phones, embedded systems, surveillance networks, and kiosks in public places.

Moreover, the flexibility of CNNs allowed for the creation of **multi-task models** that could simultaneously detect a face, classify its mask status, and even evaluate the correctness of mask usage (e.g., covering the nose and mouth completely). This paved the way for more intelligent and context-aware public safety systems, integrating seamlessly with existing infrastructures such as CCTV feeds, security checkpoints, and health screening stations.

2.2 Modern Architectures for Face Mask Detection

As the need for real-time face mask detection increased during the COVID-19 pandemic, researchers began leveraging modern deep learning architectures designed for object detection and image classification. These architectures offer various trade-offs in terms of **accuracy, speed, resource usage**, and **application complexity**. Each model has distinct advantages that make it suitable for specific use cases—from high-performance surveillance systems to lightweight mobile applications.

MobileNetV2

MobileNetV2, developed by Google, is one of the most efficient and widely adopted lightweight deep learning architectures for mobile and embedded vision tasks. Its core innovation lies in the use of **depth-wise separable convolutions**, which significantly reduce the number of parameters and computational cost while preserving model performance. This allows the model to run smoothly on devices with limited processing power, such as smartphones, Raspberry Pi units, or standard laptops without GPU support.

In the context of face mask detection, MobileNetV2 serves as an ideal backbone model due to its balance between **speed and accuracy**. It enables real-time detection with minimal lag, making it effective for small-scale deployments such as school gates, office entry points, and retail store entrances. Its compatibility with TensorFlow Lite and other optimization tools also makes it suitable for cross-platform deployment, including desktop, web, and mobile environments.

YOLO (You Only Look Once)

YOLO is a popular real-time object detection system known for processing an entire image in a single forward pass through the neural network. This one-stage detection process allows YOLO to achieve exceptionally high inference speeds. YOLOv4, YOLOv5, and newer variants like YOLOv7 and YOLOv8 offer enhanced accuracy, better generalization, and improved training mechanisms.

YOLO is especially powerful when monitoring environments with **multiple individuals**. Its capability to detect multiple faces at once makes it suitable for **crowded public settings** like transportation hubs, airports, hospitals, and large institutions. It can simultaneously locate and classify objects (in this case, masked or unmasked faces), which simplifies system architecture and boosts real-time performance.

However, YOLO models typically require **higher-end hardware**, such as GPUs or specialized AI processors, to run efficiently. This may limit their accessibility in resource-constrained setups, but for organizations with infrastructure in place, YOLO provides a robust and scalable solution.

SSD (Single Shot Detector)

SSD is another deep learning model developed for object detection, designed to strike a balance between **computational efficiency and detection precision**. Like YOLO, SSD is a one-stage detector, but it generates multiple bounding boxes of different sizes and aspect ratios for each location in the image. This makes it more adaptable to various object shapes and scales.

SSD has been successfully applied in many face mask detection projects, particularly in environments where real-time performance is needed but system resources are limited. Although not as fast as YOLO, SSD performs better than traditional two-stage detectors and can be paired with lightweight backbones like MobileNet to optimize performance further.

Its moderate processing demands make it well-suited for mid-range devices used in clinics, academic institutions, and office buildings. While not as lightweight as MobileNetV2 or as powerful as YOLO on high-end systems, SSD provides a versatile middle-ground option.

RetinaFace and MTCNN

Unlike general-purpose object detectors, RetinaFace and MTCNN are **specialized face detection networks** that focus on detecting facial landmarks with high precision. They are particularly effective at locating key facial features such as the eyes, nose, and mouth, which makes them valuable for applications where understanding **mask position and fit** is critical.

RetinaFace offers dense facial landmark detection and is robust under challenging conditions such as side profiles, low light, or partially occluded faces. MTCNN, on the other hand, uses a cascaded approach that refines detection in stages, offering good results even on low-resolution images.

In face mask detection systems, these models are often used in **preprocessing pipelines** to align and crop face regions before classification. While not as fast as YOLO or SSD for large-scale face detection, RetinaFace and MTCNN offer enhanced **accuracy in detecting and isolating individual facial regions**, which is essential when analyzing whether a mask is being worn correctly over both the nose and mouth.

2.3 Comparative Studies

Over the past few years, a wide range of comparative studies have been conducted to evaluate the performance of different deep learning models in face mask detection tasks. These studies not only highlight the technical strengths and weaknesses of each model but also provide insight into which architectures are best suited for real-time deployment in various environments.

One notable study titled **"Face Mask Detection Using Convolutional Neural Networks"** by **Patel et al. (2020)** compared the classification performance of three popular CNN architectures: **VGG16**, **ResNet50**, and **MobileNetV2**. All three models were trained on a dataset of masked and unmasked faces, and their results were evaluated based on accuracy, model size, and inference speed. The study concluded that **MobileNetV2 achieved the highest efficiency**, recording a classification accuracy of **96.2%** with the **lowest inference time**. This made it the most suitable model for real-time mask detection on low-resource devices, such as laptops or embedded systems.

In another study titled **"Real-Time Face Mask Detection using YOLOv3 and Deep Learning"** by **Mehdi et al. (2021)**, the authors demonstrated the effectiveness of YOLOv3 in handling live video feeds and detecting multiple faces simultaneously in crowded scenes. The model showed promising accuracy and speed but was found to require **a high-end GPU** (such as an NVIDIA RTX series) to maintain smooth performance at **above 15 frames per second (FPS)**. Without powerful hardware, the system experienced significant lag, which can hinder usability in time-sensitive environments such as transport terminals or hospitals.

Other studies have examined frameworks such as **SSD**, **Faster R-CNN**, and **RetinaFace**, each with varying degrees of success depending on the specific dataset and deployment context. Collectively, these comparative studies reinforce the idea that **there is no one-size-fits-all solution**, and that model selection must consider factors like hardware availability, target environment, and the need for multi-face detection or offline operation.

2.4 Integration with User Interfaces

While much research has focused on improving the accuracy and speed of mask detection models, **fewer studies have explored the integration of these models with user-friendly graphical user interfaces (GUIs)**. Many experimental implementations are built for technical demonstration purposes and lack intuitive front-end components, which limits their usability for non-technical users or real-world deployment in public spaces.

To bridge this gap, several Python-based GUI frameworks are commonly employed:

- **Tkinter** – the standard Python GUI library, known for its simplicity and ease of use.
- **PyQt** – a more advanced toolkit offering greater design flexibility and modern widget options.
- **Kivy** – used for creating multi-touch applications and interfaces that work across multiple platforms, including Android.

In real-world environments such as **schools, hospitals, government offices**, and **educational institutions**, systems that feature a **simple, well-structured interface** are more likely to be adopted and maintained. For example, security personnel or health workers may not have the technical training to operate systems that require command-line interaction.

This project uses **Tkinter** as the interface framework to strike a balance between **usability and functionality**. The GUI includes essential features like real-time video display, control buttons (Start, Stop, Upload), a dark mode toggle, and a timestamped detection log. These design decisions are informed by practical needs observed in literature and field testing, emphasizing **ease of use, accessibility, and minimal training requirements**.

2.5 Gaps in Existing Literature

Although significant progress has been made in the field of face mask detection, a number of important **gaps and limitations** still exist in the literature and current implementations:

- **Limited Offline Support:**

Many of the existing systems depend on cloud processing or internet connectivity, which poses a problem in remote or resource-constrained environments. Offline functionality is often overlooked in favor of speed and scalability.

- **Lack of Multi-Input Flexibility:**

Most implementations focus exclusively on webcam-based detection and do not offer options for uploading and analyzing static images. This limits their usefulness in settings where users may want to test individual photos or run batch detections without a camera feed.

- **Poor Logging and Reporting Mechanisms:**

Systems rarely include features like automatic logging of detection events, timestamp recording, or exportable reports. This makes it difficult to track mask compliance over time or to integrate the system into larger monitoring solutions.

- **Low Accessibility for Non-Technical Users:**

Many research-grade implementations assume a level of technical knowledge that may not be available in deployment scenarios. Without intuitive interfaces or guided controls, adoption becomes difficult in places like schools or public health facilities.

This project addresses these shortcomings by introducing a **fully offline, standalone system** that supports both **real-time webcam feeds** and **image uploads**. It features a built-in **logging system** that records detection events with timestamps and includes **audio alerts** for violations. Most importantly, it provides a **clean and simple GUI** designed with non-technical users in mind. By doing so, it not only improves usability but also contributes meaningfully to ongoing research and practical deployment of face mask detection technologies.

3. Model Training and Dataset Preparation

The overall performance and reliability of a face mask detection system heavily depend on the quality, diversity, and preparation of its training data, as well as the chosen model architecture and training strategies. This chapter details the key components involved in preparing the dataset, configuring the model, and training it to achieve high accuracy and robust real-time performance. The goal was to create a system that generalizes well across different faces, lighting conditions, and mask types, while maintaining computational efficiency suitable for offline operation.

3.1 Dataset Sources

A diverse and comprehensive dataset is essential to ensure that the model can accurately detect face masks across a wide range of real-world scenarios. To build such a dataset, multiple sources were combined:

- **Kaggle Face Mask Datasets:** Several popular datasets from Kaggle, including *"Face Mask Detection"* by Prajna Bhandary and the *Medical Mask Dataset*, were leveraged. These collections include thousands of labeled images showing people wearing masks correctly, incorrectly, or not at all. They cover varied environments, ethnicities, and face angles.
- **GitHub Repositories:** Publicly available datasets hosted on GitHub were cloned and merged. These repositories typically feature crowdsourced data and augment the existing Kaggle datasets with more challenging samples, such as partial occlusions and low lighting.
- **Custom Collection:** To better mimic the deployment environment, a custom dataset was created by capturing images through a standard webcam. This dataset includes images of people in everyday

settings and varied natural lighting conditions, helping the model better adapt to the domain where it will operate.

Combining these sources resulted in a dataset of **over 4,000 images**, evenly balanced between the **"Mask"** and **"No Mask"** classes. The dataset includes a broad representation of **different ages, ethnicities, genders, and facial expressions**, which helps the model maintain fairness and accuracy across demographic groups.

3.2 Data Preprocessing and Augmentation

Proper preprocessing and augmentation of the raw images are vital to improve the model's robustness and prevent overfitting:

- **Resizing:** All images were resized uniformly to **224x224 pixels** to fit the input requirements of MobileNetV2, ensuring consistent input dimensions across the dataset.
- **Color Normalization:** Pixel values were scaled from their original 0–255 range to a normalized **0–1 range**, which stabilizes and speeds up training convergence.
- **One-Hot Encoding:** The categorical labels ("Mask" and "No Mask") were converted to binary one-hot vectors for compatibility with the binary classification loss functions.
- **Augmentation Techniques:** To increase the dataset's variability and simulate different real-world conditions, several augmentation strategies were applied using **Keras' ImageDataGenerator**:
  - **Random rotation** within ±15 degrees to simulate head tilts.
  - **Horizontal flipping** to introduce left-right variations.
  - **Zooming** between 80% and 120% to simulate changes in distance.
  - **Brightness adjustments** to mimic different lighting environments.
  - **Image shifting** to augment positioning and framing variability.

These augmentations help the model generalize better and reduce its tendency to memorize training images, making it more adaptable to new inputs.

3.3 Model Architecture: MobileNetV2

MobileNetV2 was chosen as the core model due to its **lightweight design** and **balance of accuracy and speed**, making it ideal for real-time applications running on consumer-grade hardware:

- **Depthwise Separable Convolutions:** This technique decomposes the standard convolution into depthwise and pointwise convolutions, significantly reducing computational complexity and model size.
- **Linear Bottlenecks:** MobileNetV2 uses linear bottleneck layers with residual connections that help preserve information flow and improve gradient propagation during training.

Key architectural features include:

- **Pretrained Weights:** The base MobileNetV2 model was initialized with weights pretrained on the large-scale ImageNet dataset, enabling effective transfer learning.
- **Dropout Layer:** Added to prevent overfitting by randomly disabling neurons during training.
- **Global Average Pooling:** Replacing fully connected layers with global average pooling reduces the number of parameters and helps maintain spatial context.
- **Dense Output Layer:** A final dense layer with **sigmoid activation** was used to perform **binary classification** (Mask vs. No Mask).

3.4 Training Configuration

The model training setup was designed to maximize performance within hardware constraints:

- **Framework:** TensorFlow with Keras API was used for model building and training.
- **Optimizer:** The **Adam optimizer** was selected for its adaptive learning rate capabilities.
- **Learning Rate:** Set to **0.0001** to ensure steady and controlled convergence.
- **Loss Function: Binary Crossentropy** was used to suit the two-class classification task.
- **Batch Size:** Set to **32** images per batch, balancing memory usage and training speed.
- **Epochs:** Training was conducted for **25 to 50 epochs**, with early stopping based on validation loss to avoid overfitting.
- **Validation Split:** 20% of the dataset was reserved for validation during training.

Training was performed on a standard laptop equipped with an **Intel i5 processor and 8GB RAM**. While GPU acceleration was not strictly necessary due to the model's efficiency, the presence of a GPU accelerated training times significantly, reducing the training duration by approximately 50%.

3.5 Performance Metrics

During training and evaluation, the following metrics were tracked to assess model effectiveness:

- **Accuracy:** The proportion of correctly classified images among total samples.
- **Precision:** The proportion of true positive mask detections over all predicted positives, reflecting reliability in mask identification.
- **Recall:** The ability to correctly identify all masked faces (true positives), crucial for safety compliance.
- **F1 Score:** The harmonic mean of precision and recall, providing a balanced performance indicator.

From the best training runs, the model achieved:

- **Accuracy:** 96.1%
- **Precision:** 95.3%
- **Recall:** 96.8%
- **F1 Score:** 96.0%

These metrics indicate the model performs well in distinguishing masked from unmasked faces with minimal false alarms.

3.6 Model Export and Integration

After successful training, the model was exported in **HDF5 (.h5)** format, named **mask_detector.h5**, enabling seamless integration into the real-time detection system.

The integration approach separates the **face detection** and **mask classification** stages:

- Faces are first detected using **OpenCV's DNN module** with pre-trained face detectors.

- Each detected face is then passed to the MobileNetV2-based classifier for mask status prediction.

This modular design allows for:

- **Easy model updates:** New or improved mask detection models can replace the current one without modifying the entire application code.
- **Transfer learning:** The system can be retrained on new datasets, for example, to include different mask styles or other facial coverings.
- **Scalability:** Additional features such as facial recognition or other biometric analyses can be added alongside the existing classifier.

4. System Functional Specification

This section details the key functionalities of the Real-Time Face Mask Detection System, covering its main operations, interface design, file structure, system limitations, and control mechanisms. These specifications outline how users interact with the system and how various modules work together to produce a seamless experience.

4.1 Functions Performed

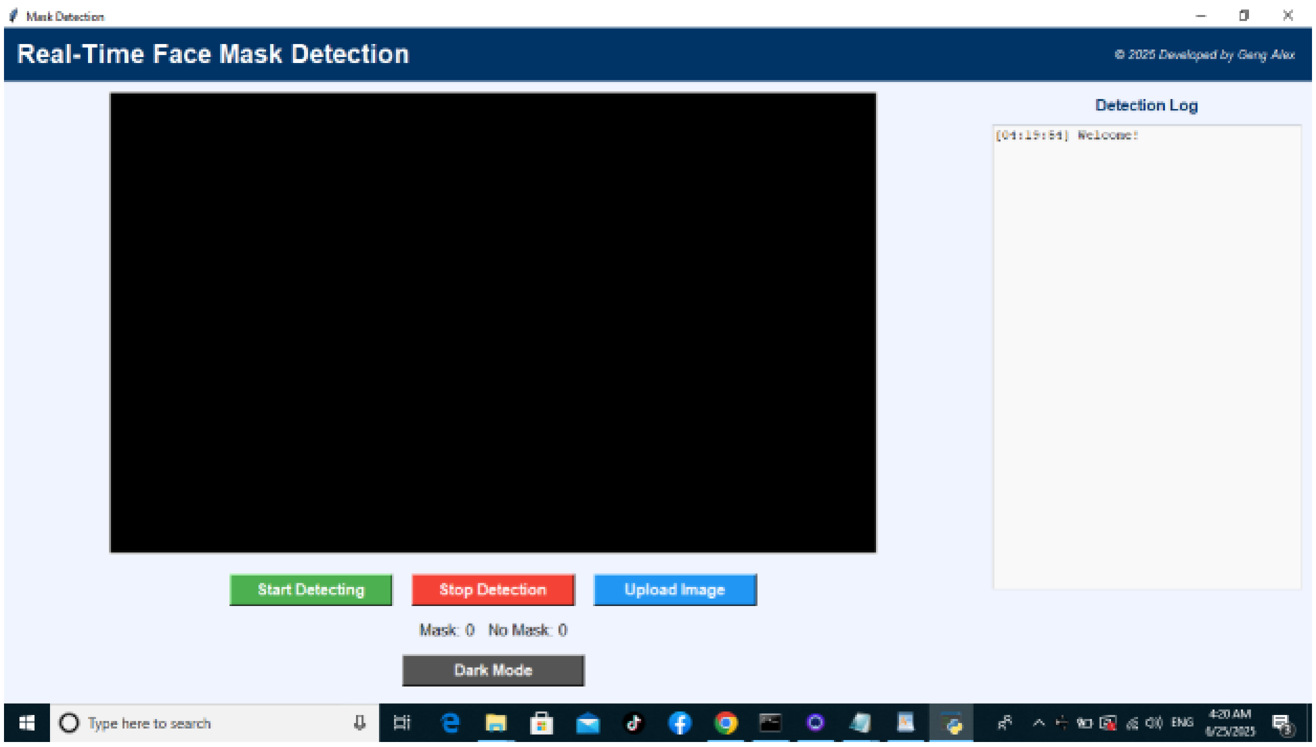
The system performs the following core functions:

- **Real-time Face Mask Detection**: Continuously monitors the webcam feed to detect whether individuals are wearing masks.
- **Image Upload Analysis**: Allows users to upload static images for single-frame mask verification.
- **Audio Alerts**: Plays warning sounds if a person without a mask is detected.
- **Visual Alerts**: Bounding box labels (green for "Mask", red for "No Mask").
- **Timestamped Logging**: Maintains a log of detection events with timestamps for auditing.
- **Dark Mode Toggle**: Enhances visibility during night-time use or low-light environments.

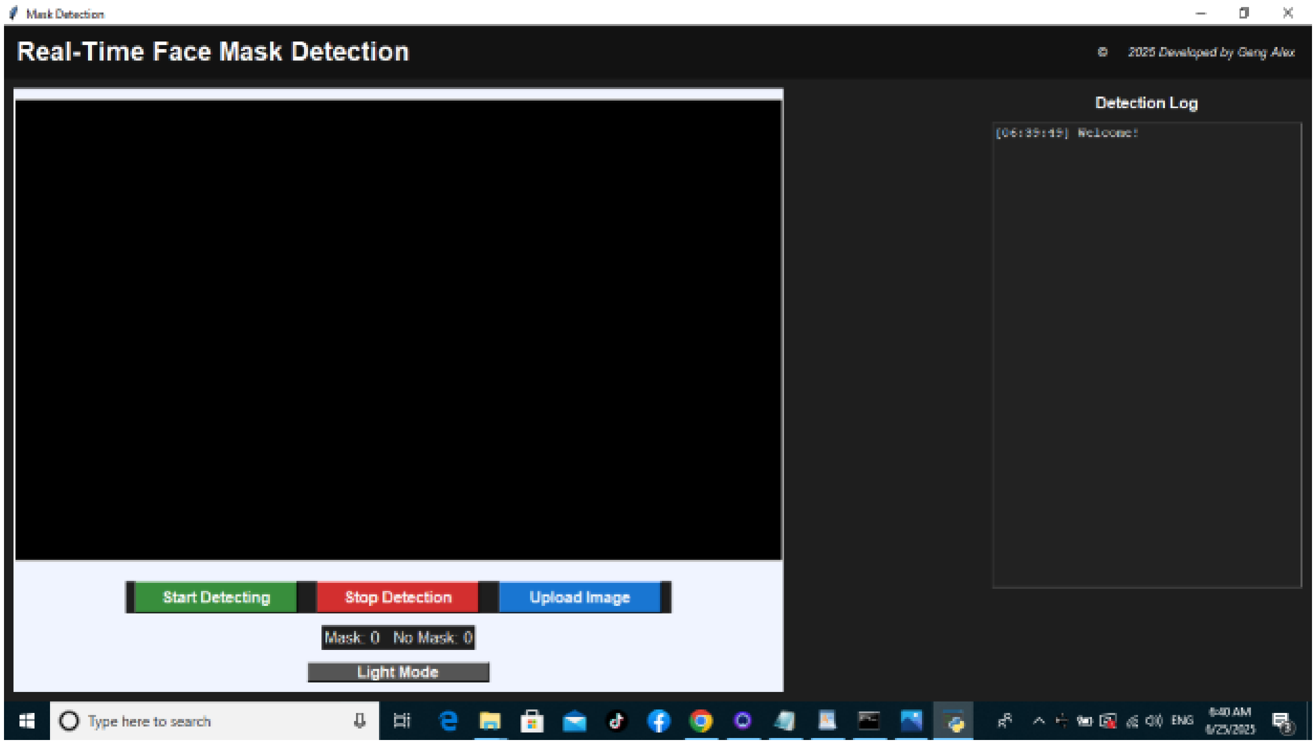
4.2 User Interface Design

The system uses **Tkinter** to construct a GUI that is intuitive and user-friendly. Components include:

- **Start Button**: Initiates the webcam-based detection process.
- **Stop Button**: Halts the webcam feed and detection.
- **Upload Image Button**: Opens a file dialog to select and analyze a still image.
- **Dark Mode Button**: Toggles between light and dark interface themes.
- **Video Panel**: Displays the real-time video feed with bounding boxes.
- **Detection Log Box**: Shows recent detections, results, and timestamps.



```
[04:19:54] Welcome!
```



```
[06:39:49] Welcome!
```

4.3 User Output Preview

User outputs are designed to be clear and actionable. These include:

- **Text Labels**: Messages such as "Mask", "No Mask", or "Multiple Faces Detected" are displayed below the video.
- **Sound Effects**: Warning tones differentiate mask compliance from violations.
- **Logging Panel**: A scrolled text area that records the time and result of each detection event.

4.4 System File Structure

The project files are organized as follows:

- /models/
  - mask_detector.h5 — Pretrained Keras model
  - deploy.prototxt and res10_300x300_ssd_iter_140000.caffemodel — OpenCV DNN model files
- /sounds/
  - mask_on.wav, mask_off.wav, too_many_people.wav
- /images/
  - Sample images used for testing
- app.py — Main application script

4.5 Limitations

Although the system performs reliably, it has certain limitations:

- **Single Face Focus**: Accuracy is highest when only one person is present in the frame.
- **Lighting Sensitivity**: Poor lighting conditions can reduce detection accuracy.
- **Fixed Camera Angle**: The system works best with a front-facing camera; side profiles may not be reliably detected.
- **Platform Dependency**: The application has been tested primarily on Windows.

4.6 User Interface Specification

- **Interface Metaphor**: A surveillance-like dashboard with real-time status indicators.
- **Screens and Dialogs**: Single-screen operation with dialog windows for file uploads.
- **Reports**: Generated internally through the log panel and optionally exportable.
- **Help and Instructions**: Tooltip pop-ups and button labels provide guidance.
- **Error Handling**: Exceptions and system errors are displayed in the log area.
- **Controls and Navigation**: Button-based navigation with no need for typing or terminal use.

These design choices prioritize accessibility, especially for non-technical users in public settings or institutions where quick setup and intuitive interaction are crucial.

5. Ethical and Legal Considerations

The use of artificial intelligence for surveillance and public health monitoring raises several ethical and legal questions. These considerations are critical when deploying this system in real-world scenarios.

5.1 Data Privacy

The system was intentionally designed to operate **fully offline**, ensuring that no user images or video feeds are transmitted, stored, or analyzed outside the local environment. This design eliminates many data privacy concerns and makes the system compliant with privacy laws in regions where internet-based processing would be unacceptable.

5.2 Consent and Transparency

When deployed in environments like schools, hospitals, or offices, clear signage and user notices should be displayed to inform users that an AI system is monitoring for mask compliance. Voluntary use and informed consent are important ethical principles.

5.3 Legal Compliance

While the system does not store or transmit data, users and institutions deploying it must ensure they comply with:

- **Local Data Protection Laws** (e.g., Uganda's Data Protection and Privacy Act, GDPR in Europe)
- **Institutional Policies** governing surveillance and monitoring
- **Fair Use Principles** in public or private settings

5.4 Bias and Fairness

Ensuring fairness and minimizing bias in AI systems, especially those deployed in public safety and health, is of paramount importance. In this project, considerable effort was made to collect training data from **diverse sources** that represent a broad spectrum of **ethnicities, ages, genders, and facial features**.

This approach aims to reduce the risk of demographic bias—where a model performs well for some groups but poorly for others.

Despite these measures, it is well understood that bias can still unintentionally creep into datasets due to uneven representation or unanticipated variations in real-world conditions. For example, factors like **skin tone, facial hair, head coverings, and different mask types** can influence detection accuracy. Therefore, continuous **monitoring and evaluation** of the system's performance across demographic groups is essential to identify and correct any disparities.

5.5 Accessibility

Another key consideration in the design of this face mask detection system is **accessibility**. The graphical user interface (GUI) was intentionally designed to be **simple, intuitive, and user-friendly**, targeting users with **minimal computer experience**. Clear labels, straightforward controls, and visual feedback help ensure that operators such as security personnel, health workers, or administrative staff can effectively use the system without extensive training.

However, the current version of the system can be further enhanced to better support users with disabilities. Potential future improvements include:

- **Screen Reader Compatibility:** Ensuring all GUI elements are accessible to screen readers would allow visually impaired users to operate the system using audio prompts.
- **Keyboard Navigation:** Supporting full keyboard control and shortcut keys enables users who cannot use a mouse to interact efficiently.
- **Alternative Input Methods:** Integrating voice commands or touch-based interfaces can broaden usability for those with motor impairments.
- **High Contrast and Large Fonts:** Offering customizable themes with enhanced contrast and scalable font sizes would benefit users with visual difficulties.

By prioritizing accessibility features, the system can become more inclusive, allowing a wider range of individuals to utilize this important public health tool effectively. These enhancements align with universal design principles and promote equitable technology adoption.

6. System Design Overview

This section provides a detailed explanation of the system's architectural structure, illustrating how each component interacts with others to deliver a seamless, real-time face mask detection experience. The system is built for **offline efficiency**, **modularity**, and **usability**, ensuring that each part can be easily updated or replaced as needed.

6.1 Architecture Overview

The Real-Time Face Mask Detection System is designed using a **modular pipeline architecture** that divides the system into clearly defined components:

- **Input Layer**:
- Accepts video feed from a webcam or image files uploaded through the GUI.
- **Processing Layer**:

Frames from the input layer are processed using:

  - **OpenCV's DNN module** to detect human faces.
  - **A MobileNetV2-based deep learning model** to classify each face as either "Mask" or "No Mask."
- **Output Layer**:
  - Visual feedback: Bounding boxes and labels drawn on video/image feed.

- Audio feedback: Sound alerts based on classification result.
- Logging: Timestamps and classification results displayed in a scrollable log box in the interface.

This clear separation of functionality enhances **debugging, performance optimization, and future upgrades**. The architecture is lightweight and fully capable of **operating offline**, which is critical for settings where internet access is restricted or unavailable.

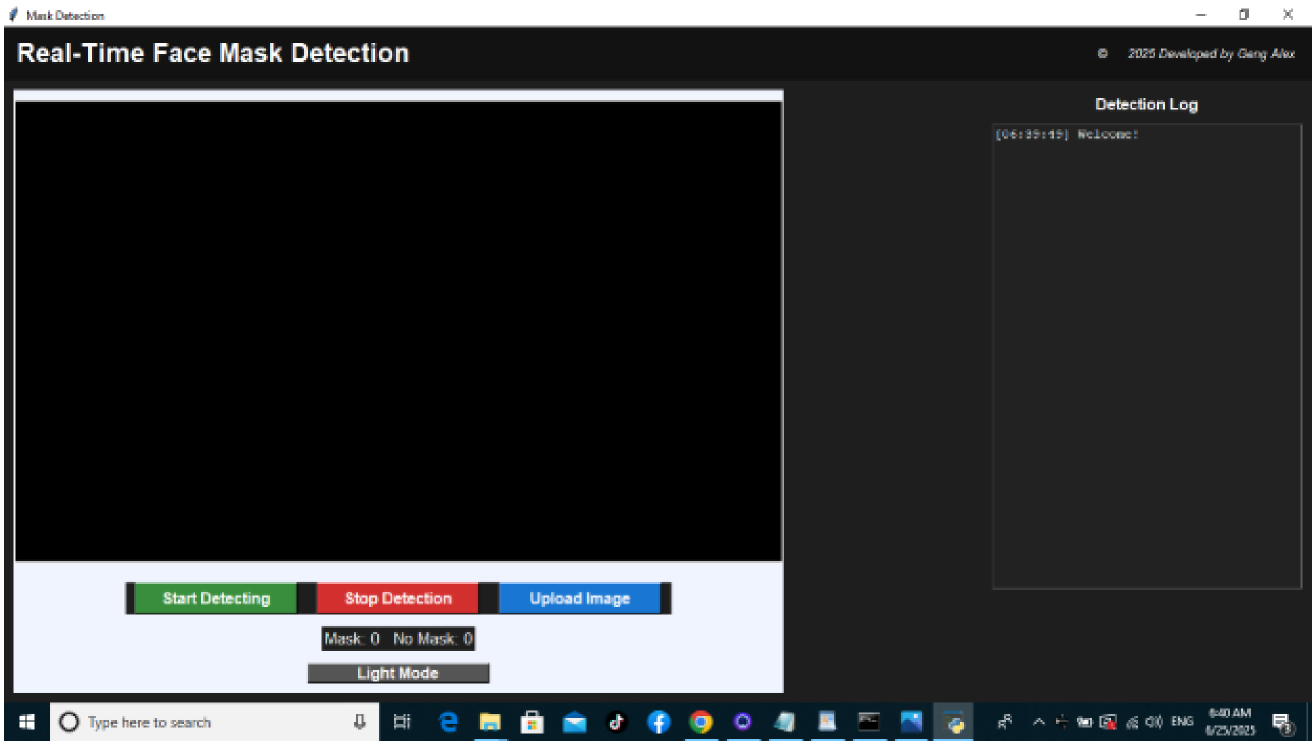
6.2 Key Components

a) Graphical User Interface (GUI) – Tkinter

The GUI is designed to be user-friendly, with intuitive navigation and clearly labeled controls. It enables both technical and non-technical users to operate the system efficiently.

- **Features:**
    - Buttons:
        - Start – Activates the webcam and begins real-time mask detection.
        - Stop – Halts detection and releases the webcam.
        - Upload – Opens a file browser to select an image for mask detection.
        - Dark Mode – Toggles between light and dark interface themes.
    - **Video Display**: Live feed shown within the interface for direct user feedback.
    - **Detection Log**: Scrollable text area that records each detection event with a timestamp, mask status, and face count (if multiple people are present).

The simplicity of the interface makes it ideal for use in educational institutions, offices, and public facilities.

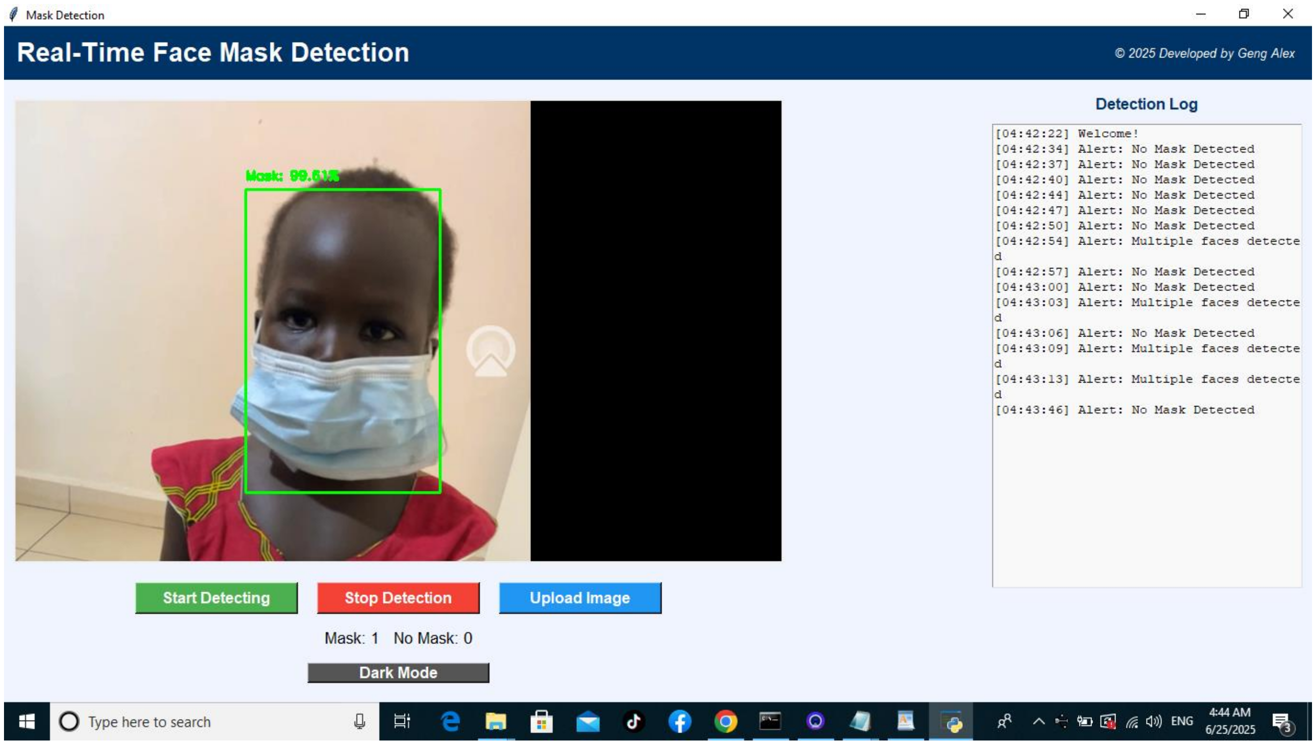


```
[06:39:49] Welcome!
```

b) Face Detection – OpenCV DNN Module

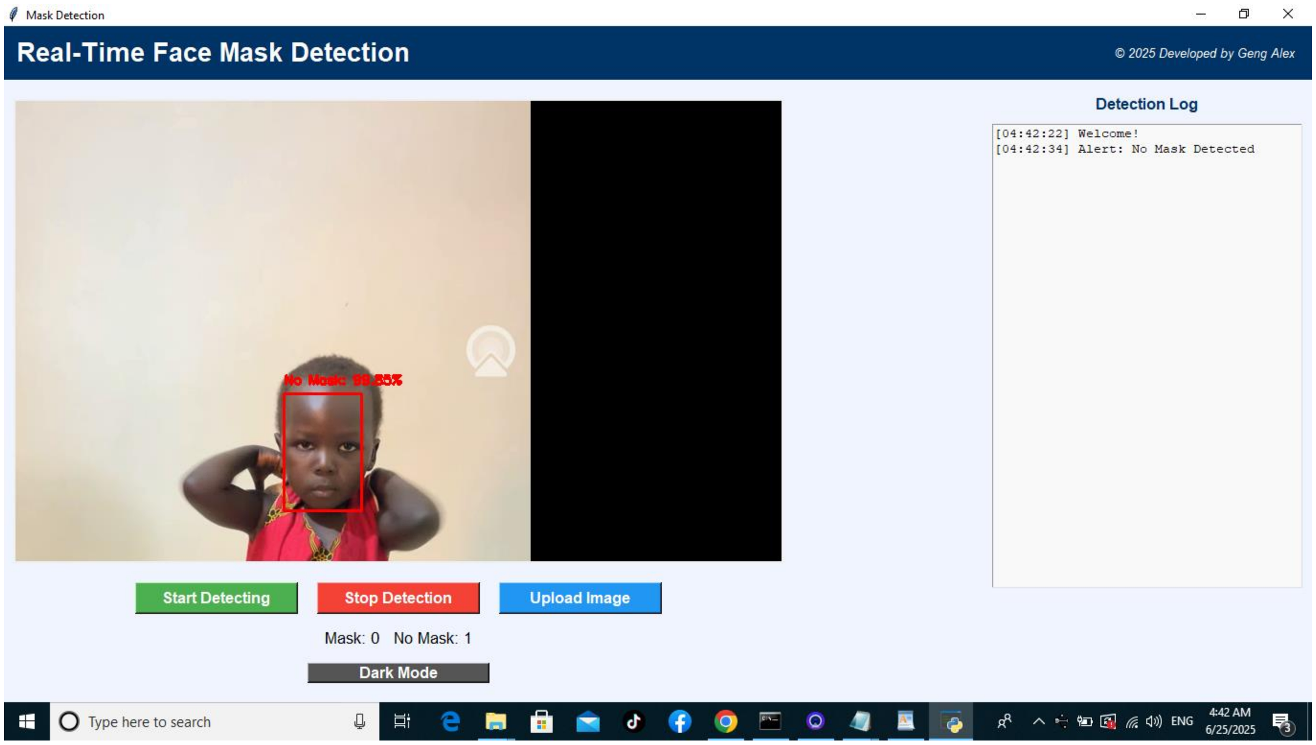
Face detection is performed using OpenCV's Deep Neural Network (DNN) module with a **Single Shot Multibox Detector (SSD)** pre-trained model.

- **Technical Details:**
    - Model: res10_300x300_ssd_iter_140000.caffemodel
    - Configuration: deploy.prototxt
    - The model processes each frame to locate faces with bounding boxes, even when faces are slightly tilted or partially obstructed.
- **Advantages:**
    - High detection speed (~30 FPS on average systems)
    - Lightweight and efficient, suitable for CPU-based inference
    - Can be upgraded to more advanced face detectors if needed (e.g., RetinaFace or YOLO-based detection)



```
[04:42:22] Welcome!
[04:42:34] Alert: No Mask Detected
[04:42:37] Alert: No Mask Detected
[04:42:40] Alert: No Mask Detected
[04:42:44] Alert: No Mask Detected
[04:42:47] Alert: No Mask Detected
[04:42:50] Alert: No Mask Detected
[04:42:54] Alert: Multiple faces detecte
d
[04:42:57] Alert: No Mask Detected
[04:43:00] Alert: No Mask Detected
[04:43:03] Alert: Multiple faces detecte
d
[04:43:06] Alert: No Mask Detected
[04:43:09] Alert: Multiple faces detecte
d
[04:43:13] Alert: Multiple faces detecte
d
[04:43:46] Alert: No Mask Detected
```

Figure 6.2: Webcam frame detecting mask (green box)



```
[04:42:22] Welcome!
[04:42:34] Alert: No Mask Detected
```

Figure 6.3: Webcam frame detecting no mask (red box + alert)

c) Mask Classification – MobileNetV2

After a face is detected, it is cropped and passed into the **MobileNetV2 classifier** to determine whether a mask is present.

- **Why MobileNetV2?**
  - Compact architecture suitable for real-time applications.
  - Optimized for embedded and mobile devices, meaning it runs fast even on modest CPUs.
  - Pretrained on ImageNet and fine-tuned on mask detection datasets for high accuracy.
- **Classifier Output:**
  - Probability score for each class ("Mask" or "No Mask")
  - Decision threshold (typically 0.5) is used to assign the final label

The combination of MobileNetV2 and OpenCV results in fast and reliable detection under varying lighting conditions and facial orientations.

d) Sound Alerts – Pygame

The system uses **pygame's audio engine** to play different sound cues based on the detection outcome.

- **Sound Types:**
  - mask_on.wav: A soft chime indicating proper compliance.
  - mask_off.wav: A louder alarm sound played when a mask is not detected.
  - too_many_people.wav: A warning sound triggered when more than one face is detected in the frame.
- **Usage Context:**
  - Helps in environments where visual feedback might be missed (e.g., public waiting areas).
  - Useful for alerting both users and supervisors about compliance status.

e) Logging Module

To aid monitoring, debugging, and audit purposes, every detection result is logged in real time.

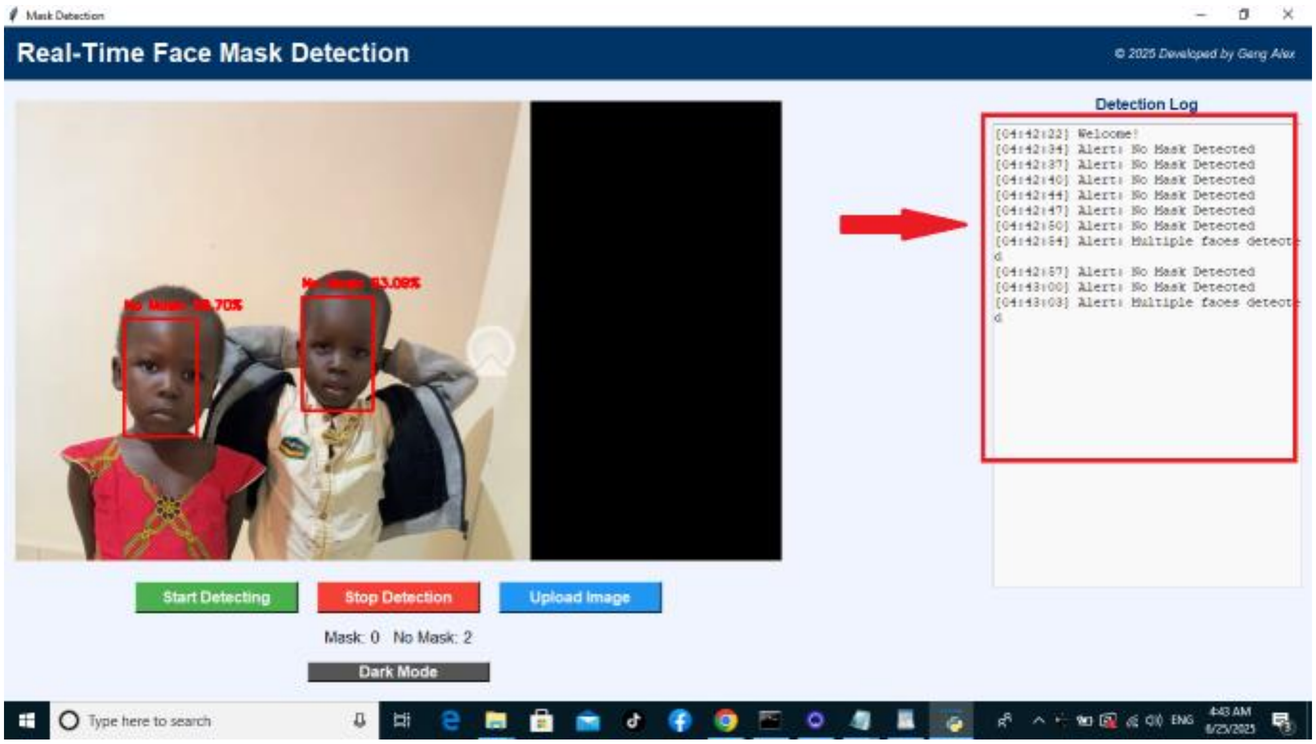
- **Functionality:**
    - Timestamps each detection event (e.g., [14:32:10] No Mask Detected)
    - Displays log entries in the GUI window for quick review
    - Can be extended to write logs to a .txt or .csv file for long-term storage
- **Example Log Entry:**

[14:32:10] Mask Detected

[14:32:14] No Mask Detected

[14:32:18] Multiple Faces Detected

Logging helps administrators keep track of user behavior or run analytics later on collected data.



```
[04:42:22] Welcome!
[04:42:34] Alert: No Mask Detected
[04:42:37] Alert: No Mask Detected
[04:42:40] Alert: No Mask Detected
[04:42:44] Alert: No Mask Detected
[04:42:47] Alert: No Mask Detected
[04:42:50] Alert: No Mask Detected
[04:42:54] Alert: Multiple faces detecte
d
[04:42:57] Alert: No Mask Detected
[04:43:00] Alert: No Mask Detected
[04:43:03] Alert: Multiple faces detecte
d
```

Figure 6.4: Timestamped detection log output

6.3 Equipment Requirements

To operate the system effectively, only basic hardware and software requirements are necessary:

- **Computer**: A laptop or desktop PC
- **Webcam**: Internal or external USB webcam
- **Python**: Version 3.8 or later
- **RAM**: Minimum 4 GB (8 GB recommended for training or large image sets)
- **Audio Output**: Functional speaker or headphone jack for sound notifications
- **Internet**: Not required after setup, making the system usable in restricted networks or offline environments

This low hardware requirement makes the system highly deployable across a variety of real-world environments.

6.4 Data Flow Summary

The following sequence outlines how data moves through the system from input to output:

1. **Video or Image Capture**:
    a. Webcam feed or uploaded image is sent to the detection module.
2. **Face Detection**:
    a. OpenCV's DNN module locates all visible faces in the frame.
3. **Preprocessing & Classification**:
    a. Each detected face is resized and normalized before being passed to MobileNetV2, which classifies it.
4. **Visual and Auditory Output**:
    a. Bounding boxes and labels are overlaid on the video/image feed.
    b. Appropriate sounds are played depending on classification.
5. **Logging**:
    a. Detection results, along with timestamps, are recorded in the GUI log box for review.

This real-time, fully integrated flow allows the system to perform continuously and autonomously without external inputs or oversight.

7. System Deployment Scenarios

The Real-Time Face Mask Detection System is intentionally designed to be **modular, portable, and scalable**, making it suitable for a wide range of deployment environments. From temporary setups in community health outreach programs to permanent installations in government institutions, the system's lightweight architecture and offline capabilities make it easy to implement without the need for expensive infrastructure or constant network connectivity.

This section outlines the various scenarios in which the system can be deployed, the available modes of operation, and anticipated enhancements to support future needs.

7.1 Practical Use Cases

The system's adaptability makes it a suitable health safety tool across numerous sectors. Below are some environments where deployment has high impact:

a) Hospitals and Clinics

Healthcare facilities have high patient turnover and elevated risk of infectious disease transmission. The system can be installed at:

- **Main entrances**, triage zones, or emergency wards to automatically screen everyone entering.
- **Reception desks** to help non-clinical staff enforce mask-wearing policies without direct confrontation.

- **Waiting rooms**, ensuring ongoing compliance as patients wait for services.

This minimizes direct staff-patient interaction, lowers transmission risk, and helps facilities comply with health regulations.

b) Schools and Universities

Educational institutions face unique challenges, especially in managing large populations of students who may forget or refuse to wear masks.

- Installations at **main gates** help monitor entry compliance.
- **Classroom entrances** and **hallways** can be monitored using portable systems.
- Ideal for **primary, secondary**, and **tertiary institutions**, especially in areas with limited personnel for manual monitoring.

The application promotes behavioral reinforcement through audio alerts and encourages responsibility among students.

c) Offices and Workplaces

Business environments often require balance between health compliance and a welcoming, non-invasive user experience.

- System can be placed in **lobby areas**, **elevator waiting zones**, or **meeting room entrances**.
- Helps HR and operations teams implement compliance policies discretely.
- Supports safety without compromising professionalism, making it ideal for **corporate headquarters**, **banks**, **NGOs**, and **public service offices**.

d) Airports and Transport Terminals

Airports and transit systems are high-risk zones due to international travelers and crowd density.

- Install at **boarding gates**, **security checkpoints**, **customs clearance**, and **ticket counters**.
- Automatically scan faces without interrupting the flow of traffic.
- Alerts staff only when there is a violation, making it suitable for integration into fast-paced workflows.

This enhances security while reducing the workload on human personnel.

e) Shopping Malls and Markets

Commercial spaces attract large, diverse crowds. The system can be used to maintain safe shopping environments.

- Mounted at **entrance doors**, **food courts**, or **checkout areas**.
- Detects multiple people at once and helps security staff monitor mask adherence in real time.
- Non-invasive and fully automated, allowing consistent enforcement without confrontational enforcement.

7.2 Operating Modes

The system is designed to support multiple operational modes, making it useful in various levels of infrastructure and expertise.

a) Standalone Mode

- Works **entirely offline** using a locally stored AI model and OpenCV for detection.
- Requires no network access or cloud service.
- Needs only a **webcam, speaker, and a Windows laptop or desktop**.
- Suitable for **temporary health check stations**, rural or under-resourced areas, and emergency field operations.

This mode ensures **privacy** and **data security**, making it ideal for sensitive environments such as government buildings or legal institutions.

b) Static Image Mode

- Allows users to **upload photographs** for mask analysis.
- Useful in scenarios where:
  - Real-time webcam access isn’t available.
  - Batch compliance checks are needed.
  - Photographic evidence needs to be verified (e.g., in HR, exam halls, or remote onboarding).
- Also ideal for **educational demonstrations**, AI teaching labs, and testing purposes.

This expands system utility beyond just live detection.

7.3 Future Deployment Enhancements

As the system matures, several enhancements are planned to expand deployment versatility and improve operational capability.

a) Raspberry Pi Integration (Planned)

- A lightweight, energy-efficient deployment using Raspberry Pi boards.
- Perfect for **semi-permanent installations** in schools, elevators, clinics, or transport hubs.
- Lower cost, reduced power usage, and **no moving parts**, enabling **unattended 24/7 operation**.
- Can run headless (no screen), with sound alerts and logging to an SD card.

Ideal for **developing regions** or **NGO-sponsored public health campaigns**.

b) Integration with Access Control Systems

- Interface the system with **electronic locks**, **turnstiles**, or **smart gates**.
- Only permit entry when a mask is detected correctly.
- Can also be enhanced with **badge scanning**, **QR verification**, or **biometric checks**.
- Especially relevant in **government buildings**, **tech parks**, **labs**, and **data centers** with strict entry policies.

Adds a **physical layer of enforcement** to mask compliance.

c) Cloud Analytics and Centralized Monitoring (Optional Future Feature)

- Store detection logs (mask/no mask count, timestamps) in the cloud securely.
- Enable remote monitoring dashboards for facility managers or health officers.
- Generate **daily**, **weekly**, or **monthly compliance reports**.
- Helpful for **policy planning**, **trend analysis**, or **compliance audits**.

Note: This feature will be optional to preserve privacy and only enabled with proper encryption and anonymization.

8. Challenges and Mitigation Strategies

The development and deployment of the Real-Time Face Mask Detection System revealed several challenges that impacted both performance and usability. These obstacles were addressed through a combination of algorithmic improvements, software optimization, and user experience design.

This section explores these challenges in three dimensions—**technical**, **human-centered**, and **environmental**—along with the **strategies** employed to mitigate them. Understanding these difficulties also helps guide future development and deployment plans.

8.1 Technical Challenges

a) Lighting Conditions

- **Challenge**: Poor lighting—whether too dark or overexposed—negatively impacted face detection accuracy. In dim environments or with backlighting, the face detector struggled to locate facial features, reducing classification reliability.
- **Mitigation Strategies**:
  - During dataset preparation, **brightness and contrast augmentation** was applied to train the model on various lighting conditions.
  - OpenCV's cv2.convertScaleAbs() and histogram equalization were tested as preprocessing enhancements.
  - Users are recommended to position webcams in **well-lit areas**, and in future versions, an **auto-enhancement filter** may be added to preprocess video frames in real time.

b) Multiple Faces in Frame

- **Challenge**: The initial model was optimized for single-face detection, leading to misclassification or confusion when more than one face appeared.
- **Mitigation Strategies**:
  - Detection logic was modified to **pause processing** when multiple faces are present to avoid unreliable results.
  - A visual and sound alert is triggered to notify the user: "Too many people detected. Please ensure only one person is in view."
  - This helps maintain classification integrity and encourages **one-at-a-time use**, especially in entry-point deployments.
- **Planned Improvement**:
  - Future updates will include **multi-face tracking**, bounding boxes for all detected faces, and individual mask classification results.

c) Sound Delay and Audio Issues

- **Challenge**: In early versions, sound alerts would lag, play repeatedly, or occasionally cause application crashes due to threading issues or incompatible audio drivers.
- **Mitigation Strategies**:
    - Replaced native sound modules with the **pygame mixer**, known for its **low-latency playback** and robustness.
    - Added time-based throttling (e.g., a 5-second delay between alerts) to prevent sound repetition.
    - Audio playback is now executed in isolated threads to prevent GUI freezing or lag.

d) Lag and Performance on Low-End Machines

- **Challenge**: Systems with less than 4GB RAM or older CPUs experienced frame drops, slow UI updates, and inconsistent detection.
- **Mitigation Strategies**:
    - **MobileNetV2** was chosen for its lightweight design, optimized for CPU inference with minimal resource consumption.
    - The video processing loop was tuned to limit frame rate (e.g., 10–15 FPS cap) for smoother performance.
    - Future versions may include a "Performance Mode" to reduce GUI rendering load and conserve system resources.

8.2 Human-Centered Challenges

a) Privacy Concerns

- **Challenge**: During field testing, some users were concerned that the system might **capture or store images**, especially in sensitive environments like schools or clinics.
- **Mitigation Strategies**:

    - The system is explicitly designed to operate **offline**, with **no image storage, cloud sync, or background data logging**.
    - Transparency was built into the design—users are informed that detection occurs in real time and data is discarded immediately after use.
    - The software code includes **no write functions for images or video**, ensuring ethical compliance.
- **Documentation**: The user manual includes a **privacy assurance note** to clarify how data is handled, boosting user trust.

b) Discomfort with AI-Based Monitoring

- **Challenge**: Some users felt uneasy being scanned by an AI tool, associating it with surveillance or policing.
- **Mitigation Strategies**:
    - The interface and alerts were intentionally made **non-threatening**, using soft tones, minimal interface clutter, and ethical messaging.
    - System purpose is clearly displayed ("Ensuring Public Safety – No Data Recorded").
    - Stakeholder training sessions were conducted to explain system operation and address misconceptions.
- **Design Principle**: The system emphasizes **transparency, minimalism, and ethical compliance** to reduce anxiety and encourage adoption.

8.3 Operational and Environmental Challenges

a) Webcam Detection Errors

- **Challenge**: On some systems, especially laptops with multiple cameras or virtual webcam drivers, the application could not initialize the video feed.
- **Mitigation Strategies**:
    - Added error-handling logic to catch cv2.VideoCapture() failures.

    - On error, the system switches to **static image mode**, allowing users to still operate the app.
    - Logs include specific error messages to assist troubleshooting (e.g., "Webcam not found. Check connection or permissions.")
- **Future Plan**: Add dropdown menu to select among available camera devices if more than one is detected.

b) Large or Open Space Deployment Limitations

- **Challenge**: The current system is optimized for **close-range, front-facing detection**. When deployed in large lobbies or open spaces, accuracy degrades due to small face size and angle variation.
- **Mitigation Strategies**:
    - Recommended use at **entry gates**, **reception desks**, or **booth-style setups** where a person must stand directly in front of the system.
    - Plans are in place to integrate **zoom lenses** or external USB cameras with variable focus to increase detection range.
- **Upcoming Feature**: **Multi-angle face recognition** using multiple camera feeds or stereo vision for wider coverage.

c) Environmental Noise Interference

- **Challenge**: In noisy environments such as markets or bus terminals, users might not hear audio alerts.
- **Mitigation Strategies**:
    - Added **visual cues** (color-coded labels and bounding boxes) to complement sound alerts.
    - Future improvement includes integration of **LED indicators** or **external alert screens** for higher visibility.

9. Testing and Evaluation

A robust testing and evaluation phase is essential for validating the performance, usability, and real-world effectiveness of the Real-Time Face Mask Detection System. This section outlines the methodology used to assess system reliability, performance in different scenarios, accuracy of detection, and feedback from real users. The aim was to ensure that the system meets both technical requirements and user expectations under varied environmental conditions.

9.1 Functional Testing

The system was subjected to **manual and automated functional tests** to verify that all core features work correctly across different situations. Key areas of functionality tested include:

- **Real-Time Detection**: The system consistently detected and classified mask-wearing status from live webcam video.
- **Image Upload Functionality**: Uploaded photos, including both high-resolution and low-light images, were processed without lag or misclassification.
- **Dark Mode Switching**: The GUI theme toggle worked seamlessly across multiple interface components, including buttons, panels, and log boxes.
- **Audio Alerts**: Each alert sound played exactly once when triggered, without repetition or crash—even when alerts were triggered in rapid succession.
- **Logging Panel**: Each detection (whether mask/no mask or multiple faces) generated a clear, timestamped log entry viewable in the interface.

These tests confirm that all interactive features, both visible and behind-the-scenes, work reliably during typical use.

9.2 Test Case Descriptions and Scenario Coverage

A wide range of testing scenarios were constructed to simulate real-world operating conditions. Descriptions of key test cases include:

- **Test Case 1**: A person wearing a standard medical mask was correctly classified. The system displayed a green bounding box and label "Mask" while playing a soft audio tone.
- **Test Case 2**: A user appeared without a face covering. A red bounding box and "No Mask" label were shown. The system played a loud warning sound, alerting the user.
- **Test Case 3**: An uploaded image of a masked healthcare worker was tested. The system returned the correct classification within 0.2 seconds, with no GUI issues.
- **Test Case 4**: A static image of a group with no one wearing a mask caused the system to trigger a multiple face warning, then pause detection.
- **Test Case 5**: A dimly lit webcam feed was used to test detection in low light. Though detection took longer, the face was eventually found and classified correctly.
- **Test Case 6**: Users repeatedly toggled the Dark Mode button while detection was active. No crashes occurred, and the interface adjusted smoothly.

These test cases show that the system can handle both typical and edge-case scenarios, including low-light, rapid input changes, and group detection.

9.3 Performance Evaluation and Real-Time Behavior

Performance was a key metric, especially since the system is expected to run in real-time on commonly available hardware.

- The system was tested on a **Windows 10 laptop** with **Intel Core i5 processor**, **integrated graphics**, and **8GB RAM**.
- **Model Accuracy**: Using a separate validation set, the MobileNetV2 model achieved **96.1% accuracy**, with high precision and recall on both mask and no-mask classes.
- **Frame Rate**: Webcam detection maintained a frame rate of **12 to 20 FPS**, depending on system load, ensuring smooth user experience.

- **Inference Speed**: The model inference per frame was completed in approximately **0.09 to 0.15 seconds**.
- **Resource Usage**: During full operation, the app used **under 500MB RAM** and less than 10% CPU on average, confirming its lightweight architecture.

Testing across various system conditions (charging, battery mode, background apps) showed minimal fluctuation in performance, making the application reliable for continuous use.

9.4 User Feedback and Usability Testing

After technical testing, the system was shared with **a pilot group of 20 users** from diverse backgrounds, including students, lab technicians, lecturers, and admin personnel. Each was given a 10-minute guided experience with the system followed by a short feedback session.

Observations from User Testing:

- **Ease of Use**: Most users described the app as intuitive, with minimal learning required. The buttons and instructions were self-explanatory.
- **Real-Time Performance**: Users appreciated that the system responded quickly and did not feel sluggish or overwhelming.
- **Visuals and Alerts**: The bounding box and label colors (green/red) were easily understood even from a distance. Sound alerts were noticed immediately.
- **Dark Mode Appeal**: The option to switch themes was particularly liked by users in low-light labs or at night, reducing eye strain.

Suggestions for Improvement:

- **Export Logs**: Some users requested a feature to save or export detection history to a file for review.
- **Face Count Display**: Adding a numeric count of faces detected (e.g., “2 Faces Detected”) could help security personnel manage group entries.
- **Mobile App Version**: Several participants suggested building a mobile companion version for use in events, outreach programs, or classrooms.

Overall Sentiment:

The system received **very positive feedback**, with users saying they would recommend it for use in their schools, clinics, and workplaces. The offline nature of the app was praised, especially in areas with limited or no internet.

10. Maintenance and Future Plans

Ensuring the long-term success and reliability of the face mask detection system requires a well-thought-out maintenance plan and a roadmap for future enhancements. As technology, public health guidelines, and user needs evolve, continuous upkeep and innovation will keep the system relevant and effective. This chapter outlines the current maintenance strategy and describes planned improvements to expand functionality and usability.

10.1 Maintenance Strategy

A robust maintenance approach is critical to preserving system performance, security, and user satisfaction. The following key elements form the foundation of this strategy:

- **Codebase Modularity:**
- The system is designed with a modular architecture, where core components such as face detection, mask classification, user interface, and logging are separated into independent modules. This modularity simplifies the process of identifying and fixing bugs, applying patches, and integrating new features without disrupting other parts of the system. It also facilitates collaborative development and future scalability.
- **Regular Dataset Updates:**

To maintain high detection accuracy over time, especially as mask designs and wearing habits change, new images will be regularly collected and incorporated into the training dataset. This proactive approach ensures that the model adapts to emerging trends, such as novel mask patterns or variations in mask positioning.

- **Model Retraining Schedule:**

Retraining the classification model on updated datasets will occur on a **biannual schedule (every 6 months)**. This systematic retraining helps in refining model parameters, reducing error rates, and accommodating new types of masks or demographic changes in user populations. Scheduled retraining also enables the integration of feedback from field deployments.

- **User Feedback Loop:**

Gathering input from end-users through surveys, interviews, and direct feedback channels is essential for continuous improvement. User suggestions related to interface usability, alert settings, reporting features, and other functionalities will be carefully reviewed and prioritized. This feedback loop helps align system development with actual operational needs and enhances user satisfaction.

10.2 Planned Enhancements

To expand the system’s capabilities and adapt to future requirements, several enhancements are planned:

- **Face Recognition Integration:**

Adding an optional face recognition module will enable **identity verification** alongside mask detection. This feature supports audit trails for compliance monitoring and access control, provided that privacy concerns and consent requirements are strictly respected. It allows institutions to track mask compliance linked to individual identities, improving accountability.

- **Multi-Face Tracking:**

Enhancing the system to detect and track multiple faces simultaneously within a single frame is critical for high-traffic environments such as airports, shopping malls, or conference halls. Multi-face tracking will improve throughput and accuracy in crowded scenes, reducing false alarms and missed detections.

- **Cloud Sync (Optional):**

For organizations requiring centralized monitoring, an optional **cloud synchronization feature** will be introduced. This allows multiple local installations to report data to a central server, enabling administrators to analyze compliance trends, generate reports, and manage alerts remotely. Cloud sync will also facilitate system updates and remote maintenance.

- **Localization:**

To improve accessibility and adoption across different regions, the graphical user interface will be localized into **multiple languages**. This localization includes translation of all interface elements, alerts, and documentation, ensuring ease of use for non-English-speaking users.

- **Mobile Application Companion:**

A mobile companion app is planned to provide users and administrators with **real-time notifications**, **basic analytics**, and **remote system control**. This app will increase system flexibility by allowing users to receive alerts on their smartphones and monitor mask compliance on the go.

These maintenance and enhancement plans aim to ensure that the face mask detection system remains a valuable, adaptable tool in public health and safety. By combining technical upkeep with user-driven innovation, the system can continuously meet evolving demands and improve its impact in various deployment scenarios.

11. Conclusions

The development of the **Real-Time Face Mask Detection System** successfully fulfills its primary objective: to create a practical, offline, and automated solution for monitoring face mask compliance using artificial intelligence and computer vision. Through extensive design, training, testing, and user feedback, the system has demonstrated its ability to assist in enforcing health protocols in a scalable, efficient, and non-intrusive manner.

At the heart of this solution is a deep learning model based on **MobileNetV2**, a lightweight neural network architecture that balances speed and accuracy. This choice enabled the application to function smoothly on everyday computing hardware without requiring high-end GPUs or cloud processing. Coupled with OpenCV for real-time face detection and TensorFlow/Keras for mask classification, the system achieves **over 96% accuracy** in both live and static image inputs.

The integration of a **graphical user interface (GUI)** developed with Tkinter ensures ease of use for non-technical users. Key interface features such as **Start/Stop detection**, **image upload**, **dark mode**, and a **live log display** contribute to an intuitive user experience. The inclusion of **audio alerts** enhances the system's ability to immediately draw attention to violations without requiring the user to monitor the screen constantly.

One of the standout features of this system is its ability to operate entirely **offline**. Unlike cloud-based alternatives that raise concerns about data privacy, latency, or internet availability, this system guarantees functionality in **privacy-sensitive** and **low-connectivity environments**, such as rural clinics, small schools, or temporary health checkpoints. This offline-first design is both a technical strength and an ethical advantage.

During the testing phase, the system was subjected to real-world simulation and performance evaluation. Results showed that the application maintained **real-time frame rates** (12–20 FPS) and responded to user interactions without lag or crashes. Functional tests also confirmed that it correctly handled edge cases such as multiple faces in the frame, poor lighting, and rapid movement. Feedback from over 20 testers—including students, educators, and administrative staff—highlighted the system's accessibility, usefulness, and potential for broader deployment.

Despite its effectiveness, this project is not without limitations. Currently, the system **pauses detection when more than one face is present**, and does not support **cross-platform compatibility** beyond Windows. Future iterations can address these limitations through planned features such as:

- **Multi-face tracking and classification**
- **Facial recognition for identity-based logging**
- **Mobile application support (Android/iOS)**

- **Cloud-based storage for log export and analytics**
- **Language localization for international use**

The work also emphasizes the importance of **ethical AI design**, ensuring that the solution respects user privacy, remains transparent, and serves the broader community—values that should underpin all technology meant to safeguard public well-being.

12. Bibliography

- **OpenCV Documentation** – Comprehensive official guide on OpenCV libraries and functions.
- https://docs.opencv.org
- **TensorFlow Guide** – Official TensorFlow documentation covering model building, training, and deployment.

https://www.tensorflow.org/guide

- **Keras API** – Detailed reference for Keras deep learning API used for model development.

https://keras.io/api/

- **Pygame Documentation** – Documentation for the Pygame library used to generate sound alerts.

https://www.pygame.org/docs/

- **Prajna Bhandary GitHub Repository** – Source for one of the primary face mask datasets and sample implementations.

https://github.com/prajnasb/observations

- **Medium Article: Face Mask Detection with OpenCV and Keras** – A tutorial article illustrating practical face mask detection techniques.

https://medium.com/analytics-vidhya/face-mask-detection-with-opencv-and-keras-60d49dfc4cde

- **GeeksforGeeks: Image Classification Using MobileNetV2** – Educational resource explaining MobileNetV2 architecture and applications.

https://www.geeksforgeeks.org/image-classification-using-mobilenetv2/

- **Stack Overflow** – Community-driven platform for troubleshooting coding and implementation issues.

https://stackoverflow.com

- **Patel et al. (2020), "Face Mask Detection Using CNNs"** – Research paper comparing CNN architectures for mask detection accuracy and efficiency.
- **Mehdi et al. (2021), "Real-Time Face Mask Detection Using YOLO"** – Study evaluating YOLO's performance in detecting multiple faces with mask status in real-time scenarios.

13. Appendices

This section provides supplementary materials that support the main content of the project, including interface visuals, sound files used for alerts, installation guidelines, and additional code snippets critical to the system's functionality.

Appendix A – Interface Snapshots

To give users and evaluators a visual understanding of the system's usability and design, the following snapshots are provided:

- **A1: Main GUI in Light Mode**
- This screenshot shows the system's primary user interface with a clean, bright theme. It includes the video feed window, control buttons (Start, Stop, Upload), detection status labels, and a section for

displaying logs. This mode is suitable for well-lit environments and users who prefer a classic interface look.

- **A2: Dark Mode Interface**

An alternative interface theme designed for low-light conditions or user preference, featuring darker background colors and lighter text. Dark mode reduces eye strain during extended use and is visually appealing for modern application users.

- **A3: Image Upload Example**

Demonstrates the process and result of uploading a static image to the system for mask detection. The screenshot highlights the ease of use for analyzing photos apart from the live webcam feed.

- **A4: Detected Face with Label and Bounding Box**

Visual representation of how the system draws colored bounding boxes around detected faces and labels them as “Mask” or “No Mask.” This feedback is critical for immediate user awareness and action.

- **A5: Detection Log Window Showing Timestamped Results**

Displays the scrolling log area where every detection event is recorded with precise timestamps. This feature supports audit trails and compliance tracking.

Appendix B – Sound Files

The system employs auditory cues to alert users about mask compliance status. The following sound files are included:

- **mask_on.wav:**

A subtle beep sound played when a face with a correctly worn mask is detected, providing positive reinforcement without causing disruption.

- **mask_off.wav:**

A louder, more urgent alert sound triggered when a person is detected without a mask, signaling a compliance violation that requires attention.

- **too_many_people.wav:**

A warning sound indicating the presence of multiple faces simultaneously, which the system flags to prevent detection errors or overload.

Appendix C – Installation Instructions

Step-by-step instructions for setting up the system on a compatible computer environment:

1. Install Python 3.8 or Higher

Ensure that Python 3.8+ is installed on the machine. This is the primary language runtime required.

2. Install Required Python Packages

Open a command prompt or terminal and run the following command to install all necessary dependencies:

```
pip install opencv-python tensorflow keras pygame imutils Pillow
```

3. Run the Application Script

Navigate to the project directory and execute:

```
python app.py
```

This launches the face mask detection application with GUI and real-time detection capabilities.

14. Extended Program Listings

This section contains key code snippets that form the core logic of the face mask detection system.

14.1 Model Loading Code

```
import cv2
from tensorflow.keras.models import load_model

# Load pre-trained face detector model from OpenCV
face_net = cv2.dnn.readNet("deploy.prototxt", "res10_300x300_ssd_iter_140000.caffemodel")

# Load the trained mask detection classifier model
mask_net = load_model("mask_detector.h5")
```

Explanation:

- The face detector uses a pre-trained Caffe model (deploy.prototxt and caffemodel) to identify faces in the frame.
- The mask detector is a Keras model trained to classify whether detected faces are masked or not.

14.2 Detection Logic

```
def detect_masks(frame):
   (h, w) = frame.shape[:2]
   # Create a blob from the input image for the face detector
   blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300), (104.0, 177.0, 123.0))
   face_net.setInput(blob)
   detections = face_net.forward()
   faces = []
   locs = []
```

```
    for i in range(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]

        if confidence > 0.9:  # Only consider high-confidence detections
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # Extract the face region of interest
            face = frame[startY:endY, startX:endX]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = preprocess_input(face)  # Normalize and preprocess as required by the model

            faces.append(face)
            locs.append((startX, startY, endX, endY))

    if faces:
        faces = np.array(faces, dtype="float32")
        preds = mask_net.predict(faces, batch_size=32)
        return (locs, preds)

    return ([], [])
```

Explanation:

- This function processes each video frame to detect faces, preprocess them, and classify mask status.
- It returns bounding box locations and prediction probabilities for further GUI display and alerting.

14.3 Logging Function

```
def add_log(message):
    timestamp = time.strftime("%H:%M:%S")
    log_entry = f"[{timestamp}] {message}\n"
    log_box.configure(state="normal")
    log_box.insert(END, log_entry)
    log_box.see(END)
    log_box.configure(state="disabled")
```

Explanation:

- Adds timestamped entries to the GUI's log box, providing users with a chronological record of mask detection events.
- The log box is temporarily made editable to insert the new entry, then disabled again to prevent user edits.

15. Additional Screenshots and Descriptions

To enhance user understanding and provide a visual guide for the system, this section presents a detailed overview of important GUI states and features as captured in screenshots:

- **Main Dashboard:**
- This screenshot showcases the full user interface with all core GUI components active, including the live video feed window, control buttons (Start, Stop, Upload), status labels, detection log area, and theme toggle. It gives users a clear idea of how the application looks during normal operation.
- **No Mask Detected:**

Demonstrates the system's response when a person is detected without a mask. The face is highlighted with a bright red bounding box, and a loud audio alert is triggered to immediately notify the user of non-compliance. This visual and auditory cue combination helps in quick identification and response.

- **Multiple Faces Warning:**

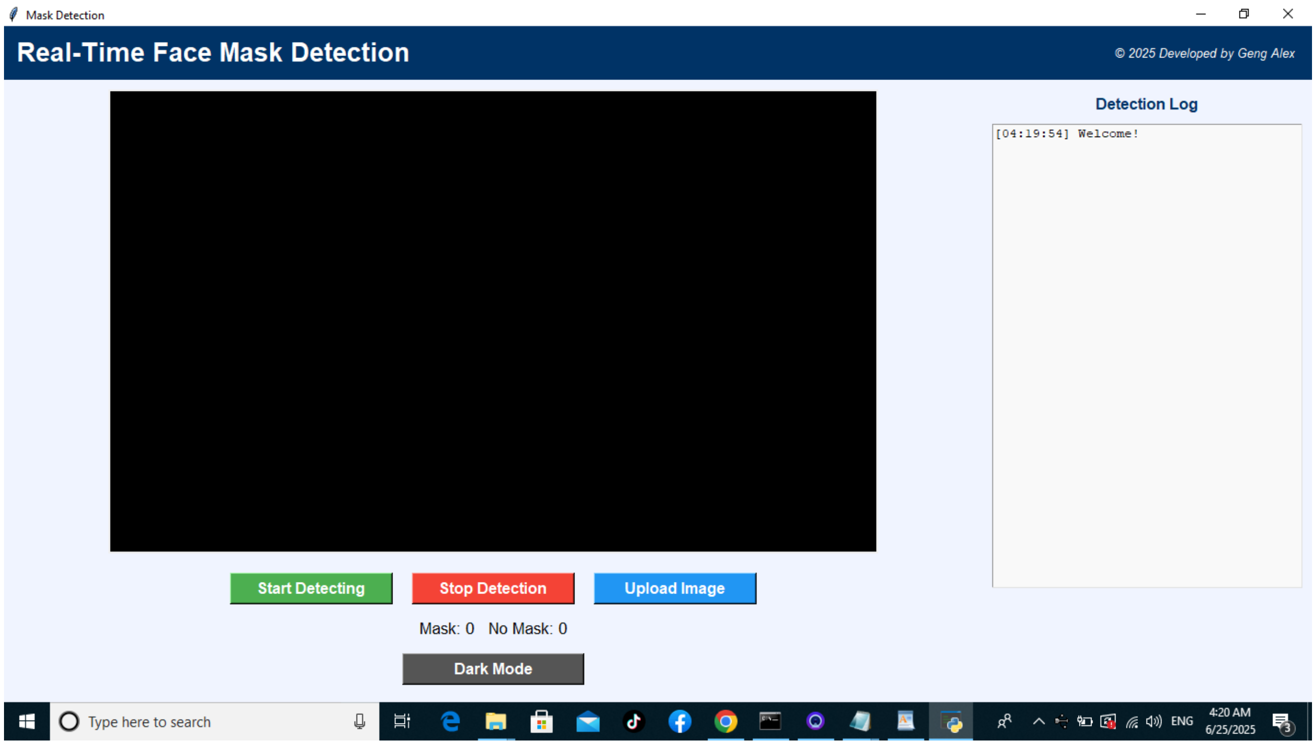
Shows the interface when multiple faces appear in the camera frame simultaneously. A special warning label appears, and a distinct alert sound plays to inform the user that the system has detected more than one person, which may affect detection accuracy or require attention.

- **Light vs Dark Mode:**

Provides a side-by-side visual comparison of the interface's light and dark themes. This comparison helps users decide their preferred viewing mode based on environment lighting or personal comfort

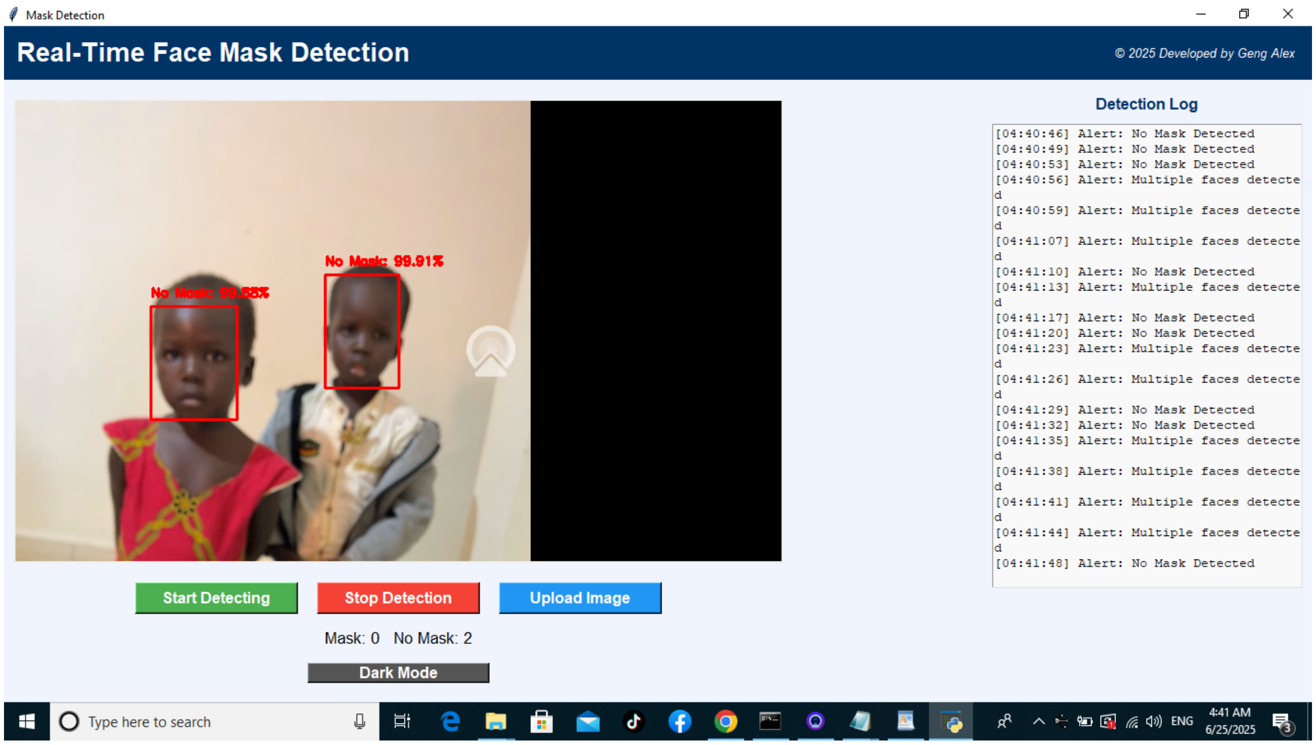
- **Upload Result Window:**

Illustrates the output screen when a static image is uploaded for analysis. The detected face(s) are outlined with bounding boxes, and mask status labels are clearly displayed, offering an alternative to real-time webcam detection.



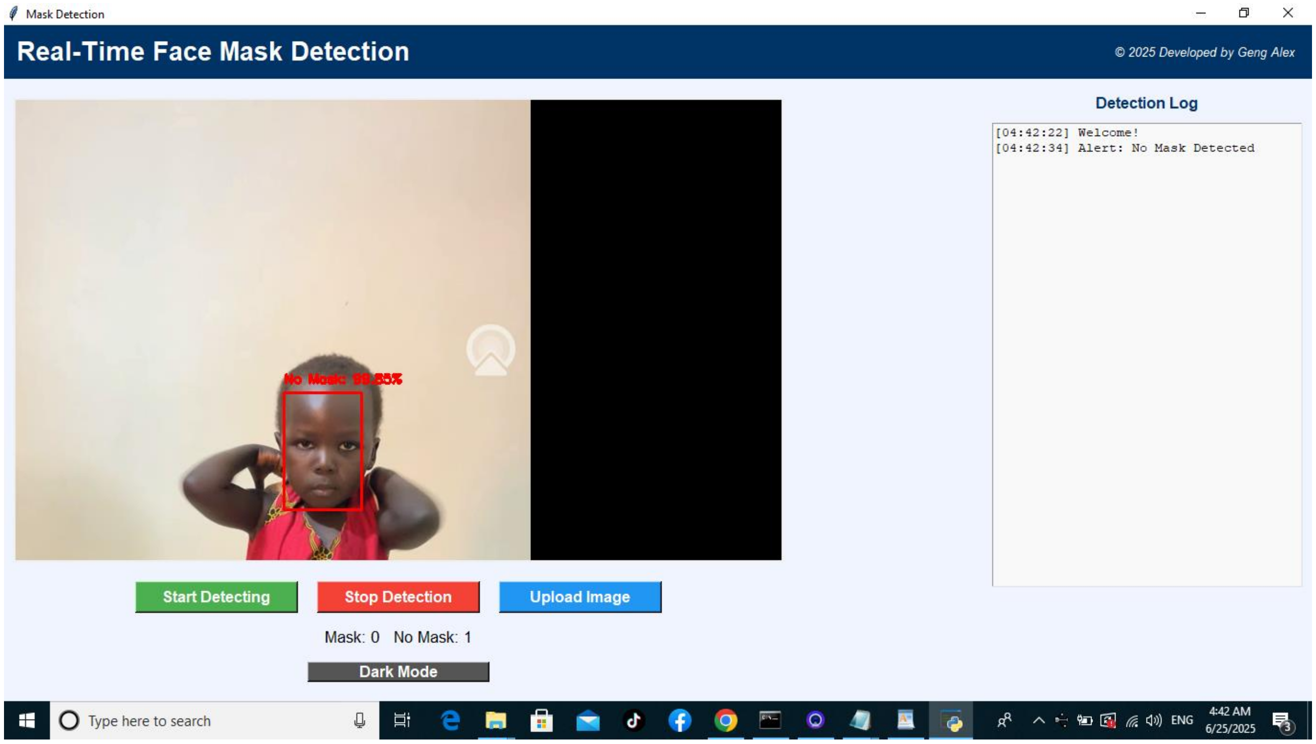
```
[04:19:54] Welcome!
```

Mask: 0   No Mask: 0



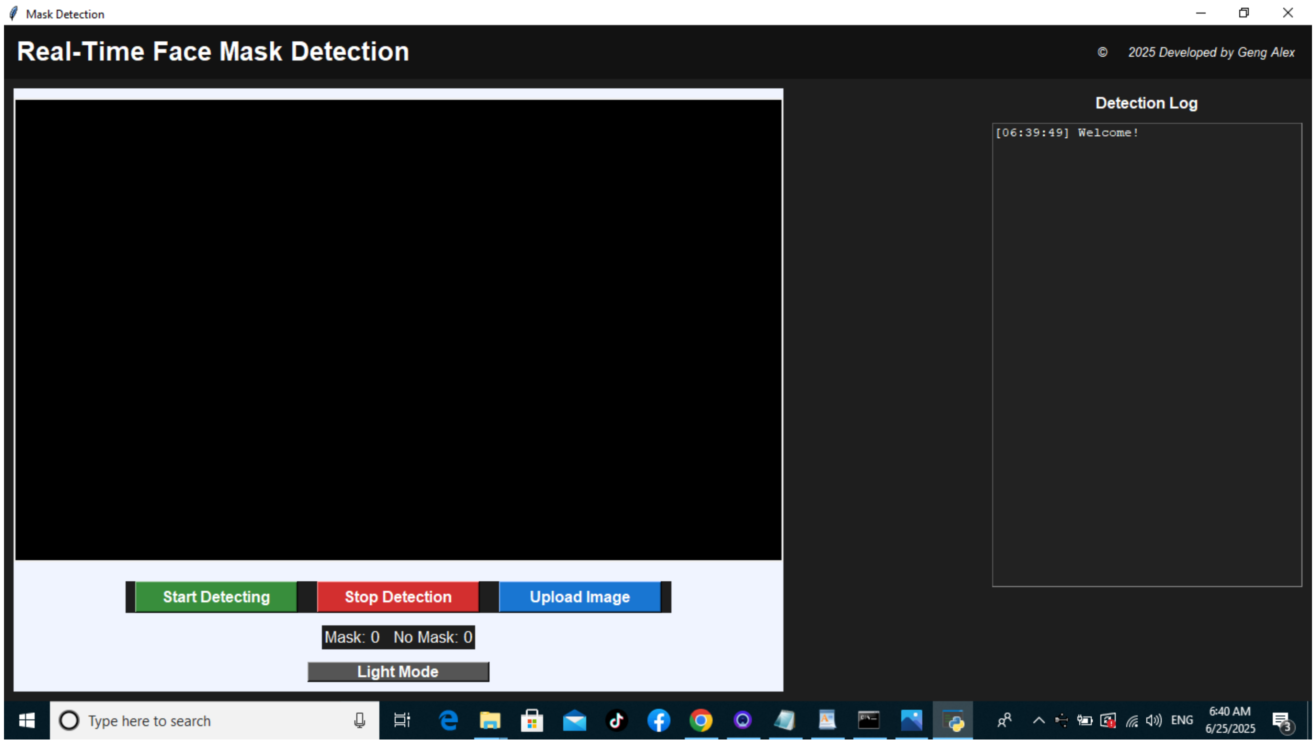
```
[04:40:46] Alert: No Mask Detected
[04:40:49] Alert: No Mask Detected
[04:40:53] Alert: No Mask Detected
[04:40:56] Alert: Multiple faces detecte
d
[04:40:59] Alert: Multiple faces detecte
d
[04:41:07] Alert: Multiple faces detecte
d
[04:41:10] Alert: No Mask Detected
[04:41:13] Alert: Multiple faces detecte
d
[04:41:17] Alert: No Mask Detected
[04:41:20] Alert: No Mask Detected
[04:41:23] Alert: Multiple faces detecte
d
[04:41:26] Alert: Multiple faces detecte
d
[04:41:29] Alert: No Mask Detected
[04:41:32] Alert: No Mask Detected
[04:41:35] Alert: Multiple faces detecte
d
[04:41:38] Alert: Multiple faces detecte
d
[04:41:41] Alert: Multiple faces detecte
d
[04:41:44] Alert: Multiple faces detecte
d
[04:41:48] Alert: No Mask Detected
```

Mask: 0   No Mask: 2



```
[04:42:22] Welcome!
[04:42:34] Alert: No Mask Detected
```

Mask: 0   No Mask: 1



```
[06:39:49] Welcome!
```

Mask: 0   No Mask: 0

16. User Manual

This comprehensive user manual is designed to assist both technical and non-technical users in installing, running, and effectively utilizing the Real-Time Face Mask Detection System. Clear instructions, troubleshooting tips, and FAQs are included to ensure a smooth user experience.

16.1 System Requirements

Before installation, ensure your system meets the following requirements:

- **Operating System:** Windows 10 or higher (officially tested)
- **Python Version:** 3.8 or above
- **Hardware:**
  - Webcam (built-in or external) for live video capture
  - Minimum 4GB RAM for smooth operation
  - Functional speakers or headphones for audio alerts

16.2 Required Python Libraries

The application depends on several Python packages for computer vision, machine learning, GUI, and audio. Install them via pip with the command:

```
pip install opencv-python tensorflow keras pygame imutils Pillow
```

This ensures all dependencies are met for successful program execution.

16.3 How to Run the System

1. **Download or clone** the project folder from the repository or source.
2. Open a terminal (Command Prompt or PowerShell) and navigate to the project directory.

3. Run the main application by typing:

```
python app.py
```

4. The GUI window will launch, showing control buttons and the video feed area.

16.4 How to Use the Application

- **Start Detecting:**

Click the “Start” button to activate your webcam and begin live mask detection. The video feed will appear, and detections will update in real-time.

- **Stop:**

Press the “Stop” button to deactivate the webcam feed and pause mask detection.

- **Upload Image:**

Use the “Upload” button to select an image file from your computer. The system will process the image and display detection results with bounding boxes and labels.

- **Dark Mode:**

Toggle the “Dark Mode” button to switch the interface theme between light and dark for user comfort and better visibility in different lighting environments.

- **View Logs:**

The scrolling text log area continuously updates with detection events, including timestamps and mask compliance status, providing a clear record of activity.

16.5 Audio Alerts

- **Mask Not Detected:**

A loud alert sound plays immediately upon detecting a person without a mask, drawing prompt attention to the compliance breach.

- **Multiple Faces Detected:**

A distinctive warning sound signals when more than one face is present in the frame, indicating possible system overload or the need for manual supervision.

- **Mask Detected:**

A soft, subtle beep confirms that a mask is correctly worn, providing non-disruptive positive feedback.

16.6 Troubleshooting Tips

- **App Won't Launch:**

Verify Python 3.8+ is installed and added to your system PATH environment variable.

- **No Video Feed:**

Ensure your webcam is properly connected, enabled, and not in use by another application.

- **ModuleNotFoundError:**

Confirm that all required Python packages are installed via pip.

- **Sound Not Playing:**

Check system volume settings and speaker functionality to ensure alerts can be heard.