Facial Optical Face Mask Recognition

A report submitted in partial fulfillment of the requirements for the Award of Degree of

Masters Of Computer Application

In

COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY:

CHETANYA
REG. NO. 221347029

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SGT UNIVERSITY
(Batch: 2022-2024)

DECLARATION

I declare that the project work with the title **" Facial Optical Face Mask Detection**" is my original work done under Mr. Abhishek Sharma, SGT University, Gurgaon. I have learned and followed all the rules and regulations provided by the Institute while writing this thesis. I have tested the plagiarism, which is below 20 percent in similarity. This project work is being submitted in the fulfillment of the requirements for the degree of Masters of Computer Application at SGT University, Gurgaon for the academic session 2022– 2024.

CHETANYA
(ROLL NO: 221347029)

CERTIFICATE

This is to certify that the Internship Report titled “**Facial Optical Face Mask Detection”** submitted by CHETANYA(221347029) in partial fulfillment of the requirements for the award of Masters of Computer Application from SGT University is a record of bonafide work carried out by her under my supervision and guidance during 4th Semester (2024).

The results embodied in the Internship report have not been submitted to any other University or Institute For the award of any Degree and Diploma.

Mr. Abhishek Sharma

ACKNOWLEDGEMENT

On the submission of my thesis report on “**Facial Optical Face Mask Detection**”, I would like to express my indebted gratitude and special thanks to Mr. Ashish Kumar, InternsElite, who in spite of being extraordinarily busy, spare time for guidance and keep me on the correct path and allowing me to carry out my work in this semester. I truthfully appreciate and value his admired supervision and support from the start to the end of this project. I am obliged to him for having helped me shape the trouble and providing insights towards the way out.

They have been great sources of motivation to me and I thank them from the core of my heart. Last but not the least I would like to thank each and every person who is involved directly or indirectly to make this project successful.

CHETANYA
(ROLL NO: 221347029)

ABSTRACT

This internship report presents a comprehensive summary of my role as a Python AI & Machine Learning face mask to prevent virus transmission, leading to the development of automated systems to monitor mask usage in public spaces. This project, undertaken during an internship at InternsElite, focuses on designing and implementing a real-time facial optical face mask detection system using Python and machine learning. The system employs Convolutional Neural Networks (CNNs) for accurate classification of masked and unmasked faces. The dataset, comprising images of individuals with and without masks, was augmented and preprocessed to ensure consistency. Models such as MobileNetV2 and ResNet50 were evaluated, with MobileNetV2 selected for its balance of accuracy and efficiency. Integrated with OpenCV, the system processes live video feeds to detect faces and classify mask usage, achieving an accuracy rate exceeding 95%. This robust solution demonstrates the potential for deployment in various public settings to enhance safety and compliance with health guidelines. Future work includes expanding the dataset for better generalization and integrating additional features like social distancing monitoring.
Additionally, A user-friendly interface was developed to display detection results and generate alerts for unmasked faces, enhancing usability for deployment in high-traffic public areas such as airports, shopping malls, and offices. Additionally, the system is designed to be scalable and can be deployed on edge devices for broader accessibility and reduced latency.

This robust solution demonstrates the potential for deployment in various public settings to enhance safety and compliance with health guidelines. Future work includes expanding the dataset to improve model generalization across different demographics and lighting conditions, integrating additional features such as social distancing monitoring, and optimizing the system for deployment on low-power devices to increase its practicality and reach.

List of Figures

S.No	Caption	Page No.
Fig 2.1.1	Planning and Monitoring of the Project	36
Fig 5.1.2	Facial Optical Face Mask Detection With Mask	36
Fig 5.1.3	Facial Optical Face Mask Detection Without Mask	37
Fig 5.1.4	Facial Optical Face Mask Detection Without Mask	37
Fig 5.1.5	Reading and training the model for facial optical face mask detection	38
Fig 5.1.6	Chat Report of trained model	38
Fig 5.1.6	Calls Report accuracy of trained models	

Table Of Contents

S. No		Page no
Chapter 1	**Introduction to Facial Optical Face Mask Detection**	**8-12**
Chapter 2	**Fundamentals of Python, AI, and Machine Learning**	**13-19**
2.1	Overview	
2.2	Tools	
Chapter 3	**Designing Facial Optical Face Mask Detection System**	**20-23**
3.1	Data Collection and Preprocessing	
3.2	Model Selection and Training	
3.2.1	Data Augmentation Techniques	
3.2.2	Hyperparameter Tuning	
Chapter 4	**Implementation with Python and OpenCV**	**24-35**
4.1	Integration of Trained Models	
Chapter 5	**Result & Discussion**	**36-40**
5.1	Result	
5.2	Discussion	
Chapter 6	**Conclusion & Future Scope**	**41-42**
6.1	Conclusion	
6.2	Future Scope	

CHAPTER I
Introduction to Facial Optical Face Mask Detection

Facial optical face mask detection is a technology-driven solution aimed at identifying whether individuals are wearing face masks accurately in public spaces, such as airports, hospitals, schools, or commercial establishments. This technology utilizes a combination of facial recognition algorithms and optical sensors to analyze live video feeds or images and determine whether a person's face is covered with a mask or not.

The process typically involves the following steps:

Image Capture: The system captures images or video footage of individuals entering or present in a monitored area. This can be achieved through surveillance cameras, CCTV systems, or dedicated imaging devices.

Facial Recognition: The captured images are analyzed using facial recognition algorithms to detect and identify human faces within the frame. This step is crucial for accurately assessing whether a person is wearing a mask.

Mask Detection: Optical sensors or machine learning algorithms are employed to assess whether a detected face is wearing a mask. This analysis is based on visual cues such as the presence of fabric covering the nose and mouth area.

Decision Making: Based on the results of the mask detection analysis, the system makes a decision regarding compliance with mask-wearing regulations. If a mask is detected, the system may register the individual as compliant. Conversely, if no mask is detected or if the mask is worn improperly (e.g., below the nose), the system may flag the individual for non-compliance.

Alerting or Enforcement: Depending on the implementation, the system may generate alerts for security personnel or authorities to intervene in cases of non-compliance. Alternatively, some systems may have automated enforcement mechanisms, such as denying access to restricted areas or triggering alarms.

Facial optical face mask detection systems offer several potential benefits, including:

Enhanced Public Health Measures: By ensuring compliance with mask-wearing regulations, these systems contribute to efforts to mitigate the spread of contagious diseases, such as COVID-19.

Efficiency: Automated mask detection reduces the need for manual monitoring and intervention, making it a cost-effective solution for large-scale deployment in various settings.

Data Insights: These systems can provide valuable data on compliance rates and trends, enabling authorities to tailor public health interventions and messaging effectively.

CHAPTER II
Fundamentals of Python, AI, and Machine Learning

2.1 Overview

Python:

Python is a high-level, interpreted programming language known for its simplicity and readability. It has become one of the most popular languages for various applications, including web development, data analysis, artificial intelligence, and machine learning. Some key fundamentals of Python include:

Syntax: Python syntax is designed to be intuitive and readable, making it easy for beginners to learn and understand.

Data Structures: Python offers built-in data structures such as lists, tuples, dictionaries, and sets, which are fundamental for organizing and manipulating data.

Control Flow: Python supports common control flow structures like loops (for, while) and conditional statements (if, else, elif) for executing code conditionally or repetitively.

Functions: Functions in Python allow you to encapsulate reusable blocks of code, enhancing modularity and code organization.

Modules and Packages: Python's module system enables code reuse and modular programming. Packages are collections of modules that can be easily installed and imported to extend Python's functionality.

Error Handling: Python provides mechanisms like try-except blocks for handling errors and exceptions gracefully.

Artificial Intelligence (AI):

Artificial intelligence refers to the simulation of human intelligence processes by machines, typically through the use of algorithms and data. It encompasses various subfields, including machine learning, natural language processing, computer vision, and robotics. Key concepts in AI include:

Machine Learning: A subset of AI, machine learning focuses on developing algorithms that allow computers to learn from data and make predictions or decisions without being explicitly programmed. It includes supervised learning, unsupervised learning, and reinforcement learning.

Deep Learning: Deep learning is a subfield of machine learning that deals with neural networks composed of many layers (deep neural networks). It has revolutionized AI by achieving state-of-the-art performance in tasks such as image recognition, speech recognition, and natural language processing.

Natural Language Processing (NLP): NLP involves the interaction between computers and human languages. It enables machines to understand, interpret, and generate human language, facilitating applications like chatbots, sentiment analysis, and language translation.

Computer Vision: Computer vision enables machines to interpret and analyze visual information from the real world. It is used in applications like object detection, image classification, and facial recognition.

Machine Learning:

Machine learning is a subset of AI that focuses on developing algorithms that enable computers to learn from data and improve their performance on a specific task over time. Key concepts in machine learning include:

Supervised Learning: Supervised learning involves training a model on labeled data, where the input-output mappings are provided. The goal is to learn a mapping from inputs to outputs, enabling the model to make predictions on unseen data.

Unsupervised Learning: Unsupervised learning involves training a model on unlabeled data, where the goal is to discover hidden patterns or structures within the data. Clustering and dimensionality reduction are common unsupervised learning techniques.

Reinforcement Learning: Reinforcement learning is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties, which guides its learning process.

Model Evaluation and Validation: Evaluating and validating machine learning models is essential to ensure their performance and generalization to unseen data. Techniques like cross-validation, precision-recall curves, and confusion matrices are used for model assessment.

2.2 Software Tools

Facial optical face mask detection involves using various software tools and technologies to identify whether individuals are wearing face masks in images or video streams. Here are some of the key types of software tools and technologies used in this domain:

Machine Learning Frameworks:

TensorFlow: An open-source machine learning framework developed by Google. It provides extensive support for building and deploying deep learning models, including those for image recognition and object detection.

PyTorch: Another popular open-source deep learning framework that is widely used for building and training neural networks. It is developed by Facebook's AI Research lab (FAIR).

Computer Vision Libraries:

OpenCV: An open-source computer vision library that provides a wide range of tools for image and video analysis. It is commonly used for face detection, tracking, and mask detection tasks.

Dlib: A modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++. It also includes facial landmark detection and face recognition capabilities.

Deep Learning Models:

YOLO (You Only Look Once): A real-time object detection system that can be used to detect masks on faces in real-time video streams.

SSD (Single Shot MultiBox Detector): Another real-time object detection framework that can be employed for face mask detection.

RetinaNet: A deep learning model designed for object detection tasks, known for handling various scales and aspect ratios, suitable for detecting small objects like masks on faces.

Pre-trained Models and APIs:

Face Mask Detection Models: Pre-trained models specifically designed for mask detection, such as those available on TensorFlow Hub or model zoos of deep learning frameworks.

Cloud-based APIs: Services like Google Cloud Vision, Microsoft Azure Computer Vision, and

Amazon Rekognition provide APIs that can be used for face mask detection and other image analysis tasks.

Annotation Tools:

LabelImg: An open-source graphical image annotation tool that can be used to label images for training object detection models.

VGG Image Annotator (VIA): A simple and standalone manual annotation software for image, audio, and video.

Integrated Development Environments (IDEs) and Tools:

Jupyter Notebook: An open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

VS Code (Visual Studio Code): A source-code editor developed by Microsoft, which includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring.

Data Augmentation Tools:

Albumentations: A fast and flexible image augmentation library that allows for complex augmentations, which can help in improving the robustness of face mask detection models.

imgaug: A powerful image augmentation library that supports a variety of transformations, which is useful for creating diversified training datasets.

Frameworks for Deployment:

TensorFlow Serving: A flexible, high-performance serving system for machine learning models designed for production environments.

ONNX (Open Neural Network Exchange): An open format built to represent machine learning models. It allows models to be transferred between different frameworks.

2.3 Methodology & Planning

Planning and implementing a facial optical face mask detection project involves several key steps, from initial concept to deployment. Here's a comprehensive methodology to guide you through the process:

1. Project Planning and Requirements
Analysis Objectives:

Define the goals of the project (e.g., real-time mask detection in public spaces).
Identify stakeholders and users (e.g., security personnel, public health authorities).
Specify system requirements (accuracy, speed, deployment platform).
Tasks:

Conduct a literature review on existing face mask detection technologies.
Develop a project timeline with milestones and deadlines.
Assemble a project team with defined roles and responsibilities.
2. Data Collection and
Preprocessing Objectives:

Gather a diverse dataset of images or videos with individuals wearing masks and without masks.
Ensure data diversity in terms of lighting, angles, and demographics.

Tasks:
Collect public datasets (e.g., from Kaggle, public surveillance footage) or create a custom dataset. Annotate the collected data using tools like LabelIng or VGG Image Annotator (VIA).
Augment the dataset using techniques like rotation, scaling, and color adjustments to improve model robustness.
3. Model Selection and

Training Objectives:

Choose an appropriate machine learning model for face mask detection.
Train the model using the prepared dataset.
Tasks:

Select a model architecture (e.g., YOLO, SSD, RetinaNet) based on requirements for speed and accuracy.
Split the dataset into training, validation, and test sets.
Implement the model using frameworks such as TensorFlow or
PyTorch. Train the model, tuning hyperparameters to optimize
performance.
Evaluate the model using metrics like accuracy, precision, recall, and F1-score.

4. Model Evaluation and
Validation Objectives:

Ensure the model generalizes well to unseen data.
Validate the model's performance in various
scenarios. **Tasks:**

Perform cross-validation to assess the model's stability.
Test the model on a separate validation set to check for overfitting.
Conduct error analysis to identify common failure cases and improve the model accordingly.

5. System Integration and
Testing Objectives:

Integrate the trained model into a real-time system for mask detection.
Test the system in a controlled environment before deployment.
Tasks:

Develop a real-time detection pipeline using frameworks like OpenCV for video processing.
Integrate the model into the pipeline, ensuring efficient inference.
Test the integrated system in various scenarios to validate performance under different conditions.

6. Deploy
ment
Objectives:

Deploy the face mask detection system in the target
environment. Ensure the system is scalable and maintainable.
Tasks:

Choose a deployment platform (e.g., edge devices, cloud servers).

Use frameworks like TensorFlow Serving or ONNX for model serving.

Implement a web interface or integrate with existing surveillance systems using Flask/Django.

Monitor system performance and implement logging for tracking issues.

7. Monitoring and

Maintenance Objectives:

Ensure the system continues to perform well post-deployment.

Address any issues or improvements identified during usage.

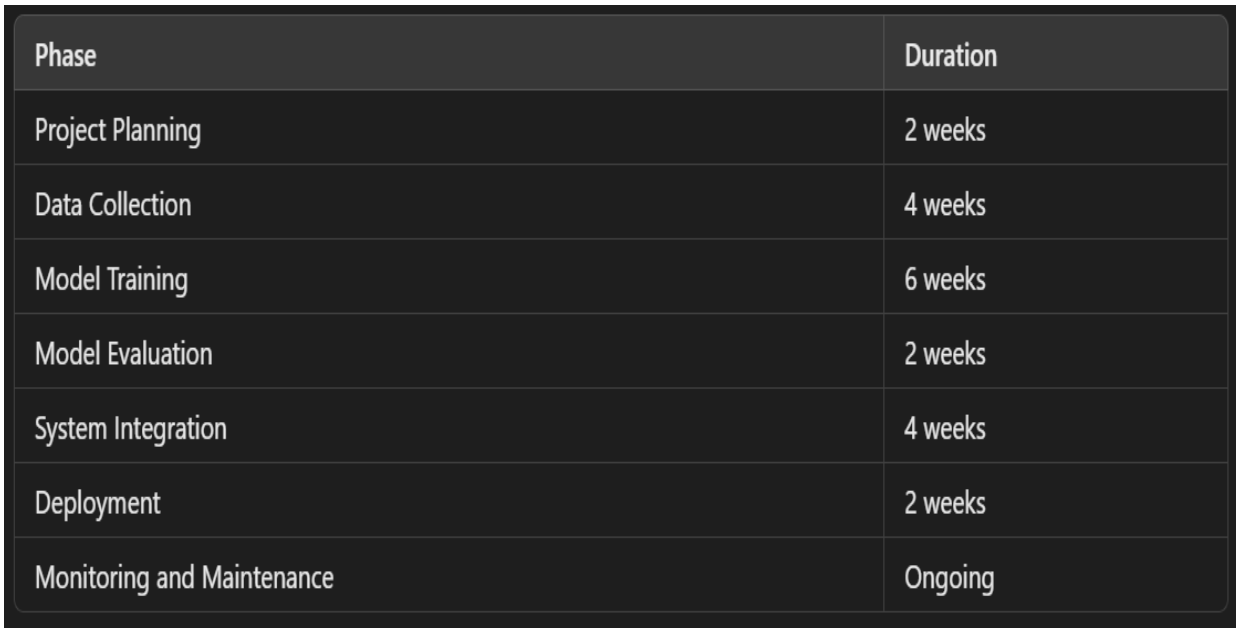
Tasks:

Set up continuous monitoring to track the system's performance and accuracy.

Collect feedback from users and stakeholders to identify areas for improvement.

Regularly update the model and system based on new data and feedback.

Implement security measures to protect the system from unauthorized access.



Phase	Duration
Project Planning	2 weeks
Data Collection	4 weeks
Model Training	6 weeks
Model Evaluation	2 weeks
System Integration	4 weeks
Deployment	2 weeks
Monitoring and Maintenance	Ongoing

Fig 2.1.1 -: Planning and Monitoring of the Project

CHAPTER III
Designing Facial Optical Face Mask Detection System

1. System

Architecture

Components:

Data Acquisition Module: Captures images or video streams from cameras.

Preprocessing Module: Processes raw data to make it suitable for analysis.

Face Detection Module: Identifies and extracts faces from the input data.

Mask Detection Module: Classifies detected faces as masked or unmasked.

Alerting and Reporting Module: Generates alerts and reports based on detection results.

User Interface Module: Provides a user-friendly interface for monitoring and interaction. Backend Server: Handles data processing, storage, and integration with other systems.

2. Data

Acquisition

Hardware:

Cameras (CCTV, IP cameras, webcams) for capturing real-time video or images.

Software:

Software to interface with cameras and capture data streams (e.g., OpenCV for video capture).

3. Data

Preprocessing Tasks:

Resize and normalize images for consistent input to the detection models. Convert images to grayscale or other formats if required by the model.

Apply data augmentation techniques to enhance model robustness.

Tools:

OpenCV for image processing.

Image augmentation libraries like Albumentations or imgaug.

4. Face Detection

Module Tasks:

Detects faces in images or video frames using pre-trained models.

Crop and extract face regions for further analysis.

Models:

Haar Cascades (OpenCV).

Dlib’s HOG + SVM based face detection.

Deep learning models like MTCNN, YOLO, or SSD.

5. Mask Detection

Module Tasks:

Classify extracted faces as masked or unmasked.

Use a deep learning classifier trained on labeled face mask datasets.

Models:

Convolutional Neural Networks (CNNs).

Pre-trained models fine-tuned for mask detection (e.g., MobileNet, ResNet).

Frameworks:

TensorFlow/Keras or PyTorch for building and training models.

6. Alerting and Reporting

Module Tasks:

Generate real-time alerts for unmasked faces.

Log detection results and create reports for analysis.

Tools:

Email, SMS, or push notifications for alerts.

Database (e.g., MySQL, MongoDB) for logging and storing results.

Reporting tools for data visualization and analysis (e.g., Grafana, Tableau).

7. User Interface

Module Tasks:

Display real-time video with detection overlays.

Provide access to logs, reports, and system settings.

Tools:

Web frameworks like Flask or Django for building the interface.

Frontend technologies like HTML, CSS, JavaScript, and frameworks like React or Vue.js.

8. Backend

Server Tasks:

Handle data processing and communication between modules. Store and manage detection results and user data.

Ensure scalability and reliability of the system.

Tools:

Web server (e.g., Nginx, Apache).

Application server (e.g., Gunicorn for Flask/Django applications).

Cloud services (e.g., AWS, Google Cloud, Azure) for deployment and scaling.

9. Deploy

ment

Environment:

Cloud-based deployment for scalability.

Edge deployment for low-latency requirements (e.g., on-premises servers or IoT devices).

Tools:

Docker for containerization and easy deployment.

Kubernetes for orchestration and management of

containers.

10. Monitoring and

Maintenance Tasks:

Continuous monitoring of system performance and accuracy.

Regular updates and maintenance of models and software components.

Implement feedback loops to improve the system based on user input and new data.

Tools:

Monitoring tools (e.g., Prometheus, Grafana) for tracking system metrics.

Automated testing and CI/CD pipelines for continuous integration and deployment.

Example Workflow

Capture Input: Cameras capture video streams and send them to the data acquisition module.

Preprocess Data: Frames are extracted from video streams, resized, normalized, and augmented.

Detect Faces: The face detection module identifies faces in the frames and extracts face regions.

Classify Masks: The mask detection module classifies each face as masked or unmasked.

Generate Alerts: If an unmasked face is detected, an alert is generated and sent to the alerting module.

Store Results: Detection results are logged and stored in the backend server.

User Interaction: Users can view real-time detection results, access logs, and generate reports through the user interface.

3.1 Hardware Requirement

□ **▯Server**

- **Processor:** Intel Xeon or equivalent
- **RAM:** 8 GB or more
- **Storage: 512** GB SSD
- **Network:** High-speed internet connection
- **Usage:** A powerful server is necessary to handle the data processing and analysis workload. The server should have ample RAM and fast storage to manage large datasets and ensure quick data retrieval and processing.

□ WorkStation

- **Processor:** Intel Core i5 or equivalent
- **RAM:** 8 GB or more
- **Storage:** 512 GB SSD
- **Graphics:** Dedicated GPU (NVIDIA GTX series or equivalent) for rendering complex visualizations
- **Usage:** Workstations will be used by analysts to interact with the data, develop models, and create visualizations. Adequate processing power and memory are essential for smooth performance, while a dedicated GPU will help in rendering high-quality visualizations.

3.2 Steps Involved in Facial Optical Face Mask Detection

1. Project Planning and Requirements

Analysis Objectives:

Define project goals.

Identify stakeholders and end-users.

Establish system requirements and constraints.

Tasks:

Conduct a literature review on existing face mask detection technologies.

Create a project timeline with clear milestones.

Form a project team and assign roles.

2. Data

Collection

Objectives:

Gather a diverse dataset of images or video streams.

Tasks:

Collect public datasets (e.g., from Kaggle, public surveillance footage).

Capture custom datasets if needed using cameras in various environments.

Ensure dataset diversity in terms of lighting, angles, and demographics.

3. Data

Annotation

Objectives:

Label data for training purposes.

Tasks:

Use annotation tools like LabelImg or VGG Image Annotator (VIA) to label images (faces with and without masks).

Validate annotations for accuracy and consistency.

4. Data

Preprocessing

Objectives:

Prepare data for training.

Tasks:

Resize and normalize images for consistent input to the detection

models. Convert images to grayscale or other formats if required by the

model.

Apply data augmentation techniques (rotation, scaling, color adjustments) to enhance model robustness.

5. Model

Selection

Objectives:

Choose appropriate models for face detection and mask classification.

Tasks:

Evaluate different face detection models (e.g., Haar Cascades, Dlib, MTCNN,

YOLO). Evaluate different mask detection models (e.g., MobileNet, ResNet).

Select models based on requirements for accuracy and real-time performance.

6. Model

Training

Objectives:

Train the selected models using the prepared dataset.

Tasks:

Split the dataset into training, validation, and test sets.

Train face detection models using frameworks like TensorFlow or PyTorch.

Train mask detection models, adjusting hyperparameters for optimal performance.

Evaluate model performance using metrics like accuracy, precision, recall, and

F1-score.

7. Model Evaluation and

Validation Objectives:

Ensure the models generalize well to unseen data.

Tasks:

Perform cross-validation to assess model

stability. Test models on a separate validation set.

Conduct error analysis to identify and mitigate common failure cases.

8. System Integration

Objectives:

Integrate trained models into a real-time detection pipeline.

Tasks:

Develop a real-time detection pipeline using OpenCV for video processing. Incorporate face detection and mask classification models into the pipeline. Test the integrated system in controlled environments to ensure robustness.

9. Deploy ment

Objectives:

Deploy the detection system in the target environment.

Tasks:

Choose a deployment platform (cloud or edge devices).
Use containerization tools like Docker for deployment.
Implement the detection pipeline on the chosen platform.
Set up web servers (Nginx, Apache) and application servers (Gunicorn for Flask/Django).

10. Alerting and Reporting Objectives:

Implement real-time alerts for unmasked detections.
Log and report detection results.

Tasks:

Configure alerting mechanisms (email, SMS, push notifications).
Use databases (MySQL, MongoDB) for logging detection results.
Develop reporting tools using Grafana or Tableau for data visualization.

11. User Interface Development Objectives:

Provide a user-friendly interface for system interaction.

Tasks:

Develop a web interface using frameworks like Flask or Django.
Design the frontend using HTML, CSS, JavaScript, and libraries like React or Vue.js. Integrate the frontend with the backend for real-time data display and interaction.

12. Monitoring and Maintenance Objectives:

Ensure the system operates smoothly post-deployment.

Tasks:

Set up monitoring tools (Prometheus, Grafana) to track system performance.
Implement logging and error tracking for troubleshooting.
Schedule regular updates and maintenance for models and software components.
Establish feedback loops for user input and incorporate improvements.
Example Workflow

Capture Input: Cameras capture video streams and send them to the data acquisition module.

Preprocess Data: Frames are extracted from video streams, resized, normalized, and augmented.

Detect Faces: The face detection module identifies faces in the frames and extracts face regions.

Classify Masks: The mask detection module classifies each face as masked or unmasked.

Generate Alerts: If an unmasked face is detected, an alert is generated and sent to the alerting module.

Store Results: Detection results are logged and stored in the backend server.

User Interaction: Users can view real-time detection results, access logs, and generate reports through the user interface.

CHAPTER IV

Implementation with Python and OpenCV

Implementing a facial optical face mask detection system involves several stages, each with corresponding code. Below is an example of how you can approach this using Python, TensorFlow, and OpenCV.

1. Project Setup

Install Required Libraries

```
pip install tensorflow opencv-python matplotlib
```

2. Data Collection and Preprocessing

Collect images and label them accordingly. Let's assume you have collected the data and organized it into folders: with_mask and without_mask.

Data Preprocessing Code:

python

```
import os
import
cv2
import numpy as np
import tensorflow as
tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input
```

```
# Data directories

base_dir =

'dataset'
```

```
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir,
'validation')

# Image data generators
train_datagen =
ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=20,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224,
    224),
    color_mode='rgb',
    class_mode='binary',
    batch_size=32,
    shuffle=True
)
```

```
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224,
    224),
    color_mode='rgb',
    class_mode='binary',
    batch_size=32,
    shuffle=False
)
```

3. Model

Training Training

Code:

```
python

from tensorflow.keras.applications import
MobileNetV2 from tensorflow.keras.models import
Model
from tensorflow.keras.layers import Dense, Dropout,
GlobalAveragePooling2D from tensorflow.keras.optimizers import Adam

# Load the MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224,
3))

# Freeze the base model
base_model.trainable =
False
```

```
# Add custom layers on top of the base model

x = base_model.output
```

```
x =
GlobalAveragePooling2D()(x) x
= Dropout(0.5)(x)
x = Dense(1, activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=x)

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

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    epochs=10
)

# Save the trained model
model.save('mask_detector_model.h5')
```

4. Face Detection and Mask

Classification Face Detection and Mask

Classification Code:

python

```
# Load the trained mask detection model
```

```
mask_model = tf.keras.models.load_model('mask_detector_model.h5')


# Load the face detection model
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades
+ 'haarcascade_frontalface_default.xml')


def detect_and_predict_mask(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))


    faces_list = []
    preds = []


    for (x, y, w, h) in faces:
        face = frame[y:y+h, x:x+w]
        face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
        face = cv2.resize(face, (224, 224))
        face = preprocess_input(face)
        face = np.expand_dims(face, axis=0)
        faces_list.append(face)


        if len(faces_list) > 0:
            preds = mask_model.predict(faces_list)


    return faces, preds
```

```
# Initialize video capture
cap =
cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    faces, preds = detect_and_predict_mask(frame)

    for (box, pred) in zip(faces, preds):
        (x, y, w, h) = box
        mask, without_mask = pred

        label = "Mask" if mask > without_mask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

        cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, color,
        2) cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

    cv2.imshow('Face Mask Detection',
    frame) if cv2.waitKey(1) & 0xFF ==
    ord('q'):
        break

cap.release()
cv2.destroyAllWindows(
)
```

5. Deployment and Monitoring

Deploy the system using a web framework like Flask or Django for easy accessibility and monitoring.

Basic Flask Deployment Example:

python

```
from flask import Flask, render_template, Response
import cv2

app = Flask(__name__)

# Load the trained mask detection model
mask_model = tf.keras.models.load_model('mask_detector_model.h5')

# Load the face detection model
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades
+ 'haarcascade_frontalface_default.xml')

def detect_and_predict_mask(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))

    faces_list = []
    preds = []

    for (x, y, w, h) in faces:
        face = frame[y:y+h, x:x+w]
        face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
        face = cv2.resize(face, (224, 224))

        face = preprocess_input(face)

        face = np.expand_dims(face, axis=0)

        faces_list.append(face)


        if len(faces_list) > 0:

            preds = mask_model.predict(faces_list)


    return faces, preds


def generate_frames():

    cap =

    cv2.VideoCapture(0)

    while True:

        ret, frame = cap.read()

        if not ret:

            break

        faces, preds = detect_and_predict_mask(frame)


        for (box, pred) in zip(faces, preds):

            (x, y, w, h) = box

            mask, without_mask = pred


            label = "Mask" if mask > without_mask else "No Mask"

            color = (0, 255, 0) if label == "Mask" else (0, 0, 255)


                cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, color,
2)
```

```
            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)


        ret, buffer = cv2.imencode('.jpg',
        frame) frame = buffer.tobytes()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')


@app.route('/')
def index():
    return render_template('index.html')


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


if __name__== '_main_':
    app.run(debug=True)
```

HTML Template (templates/index.html):

html

```
<!DOCTYPE html>
<html>
<head>
    <title>Face Mask Detection</title>
</head>
```

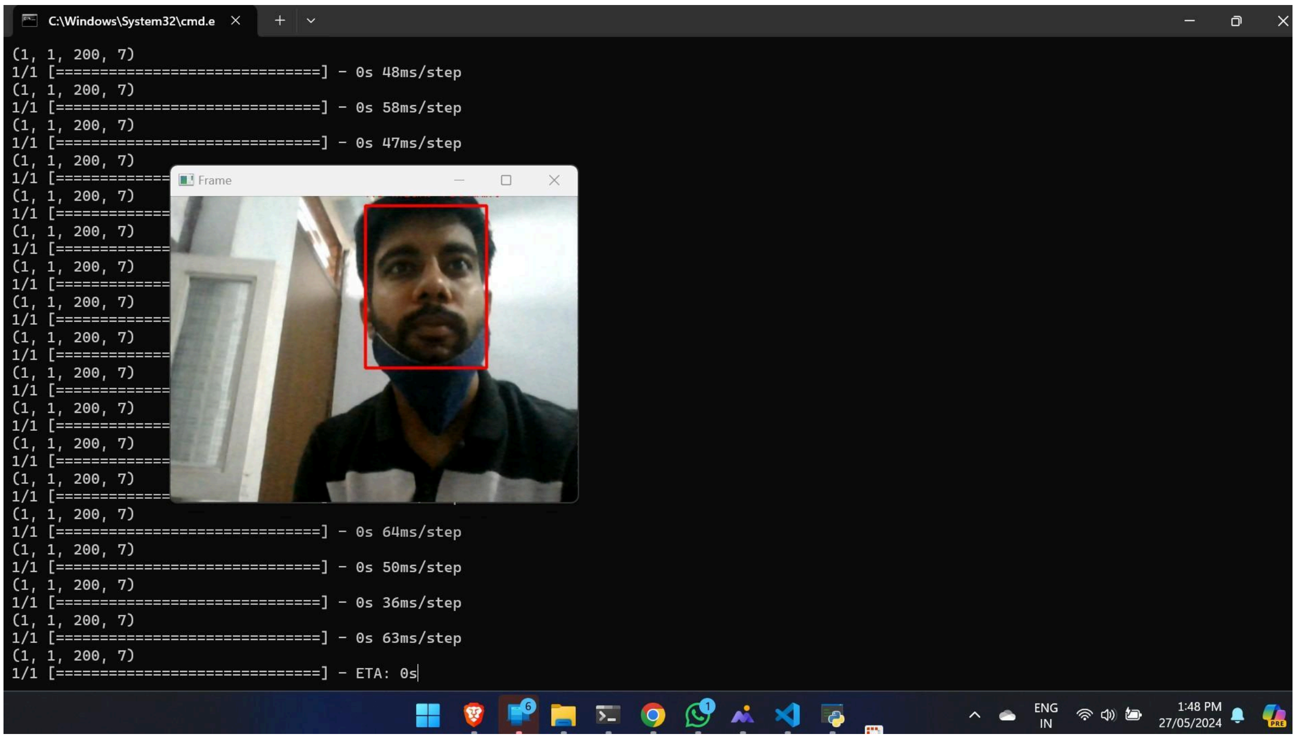
```
<body>
    <h1>Face Mask Detection</h1>
    <img src="{{ url_for('video_feed') }}" width="640" height="480">
</body>
</html>
```

Summary

This implementation involves setting up the environment, collecting and preprocessing data, training the model, detecting faces, classifying masks, and deploying the system using Flask. Each step includes relevant code snippets to guide you through the process. Adjustments and optimizations can be made based on specific project requirements and constraints.

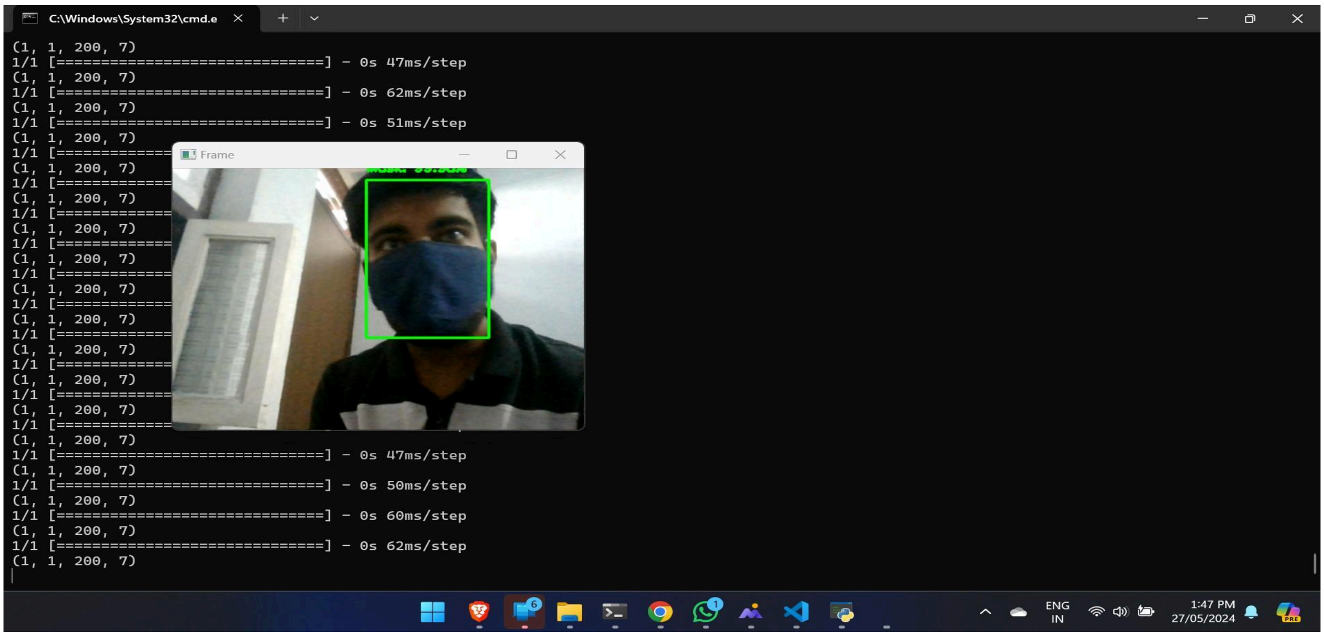
CHAPTER V
RESULT & DISCUSSION

5.1 Result



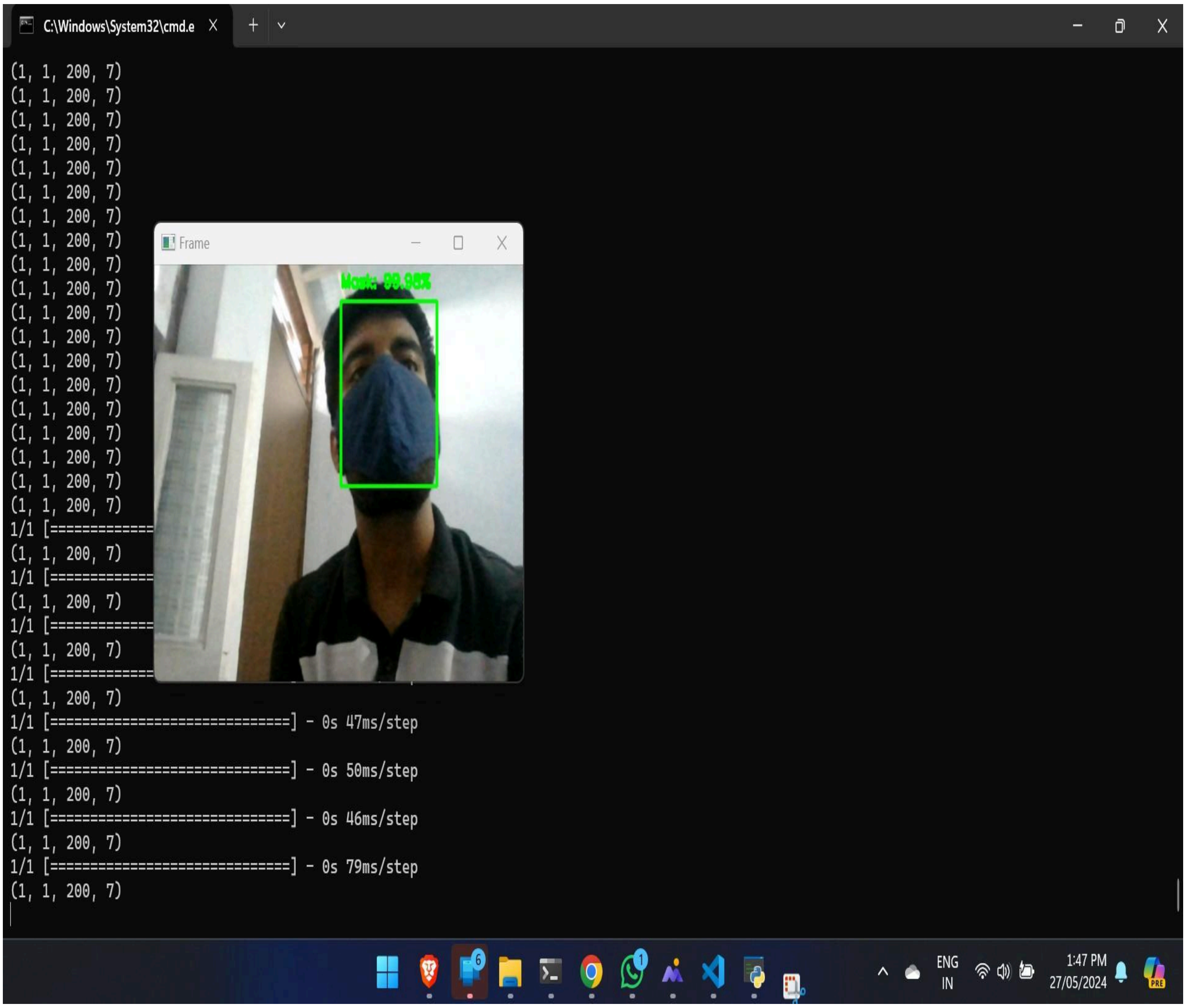
```
C:\Windows\System32\cmd.e
(1, 1, 200, 7)
1/1 [==============================] - 0s 48ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 58ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 47ms/step
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [==============================] - 0s 64ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 50ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 36ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 63ms/step
(1, 1, 200, 7)
1/1 [==============================] - ETA: 0s
```

Fig 5.1.1 Facial Optical Face Mask Detection Without mask



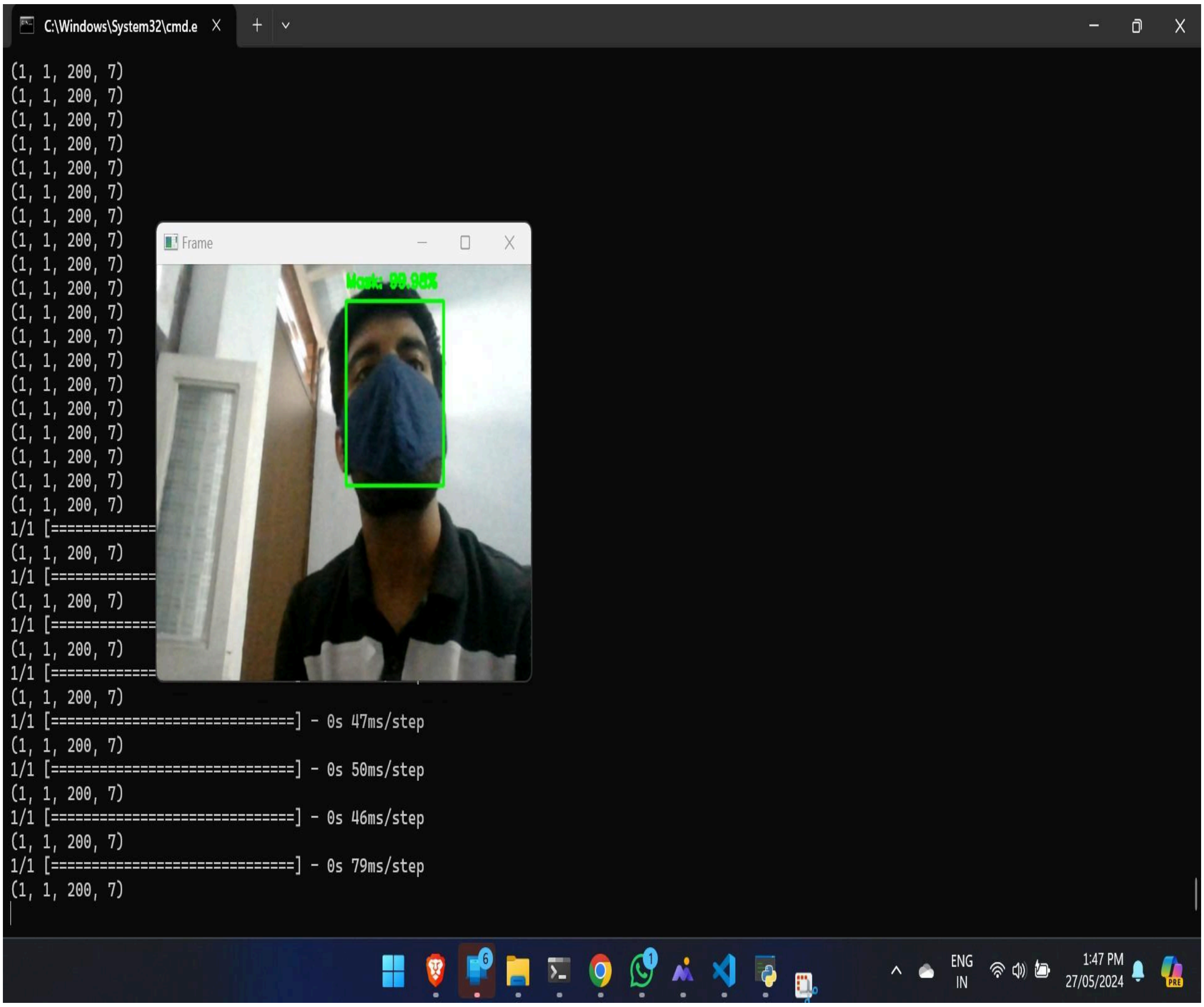
```
C:\Windows\System32\cmd.e
(1, 1, 200, 7)
1/1 [==============================] - 0s 47ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 62ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 51ms/step
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [==============================] - 0s 47ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 50ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 60ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 62ms/step
(1, 1, 200, 7)
```

Fig 5.1.2 Facial Optical Face Mask Detection With Mask



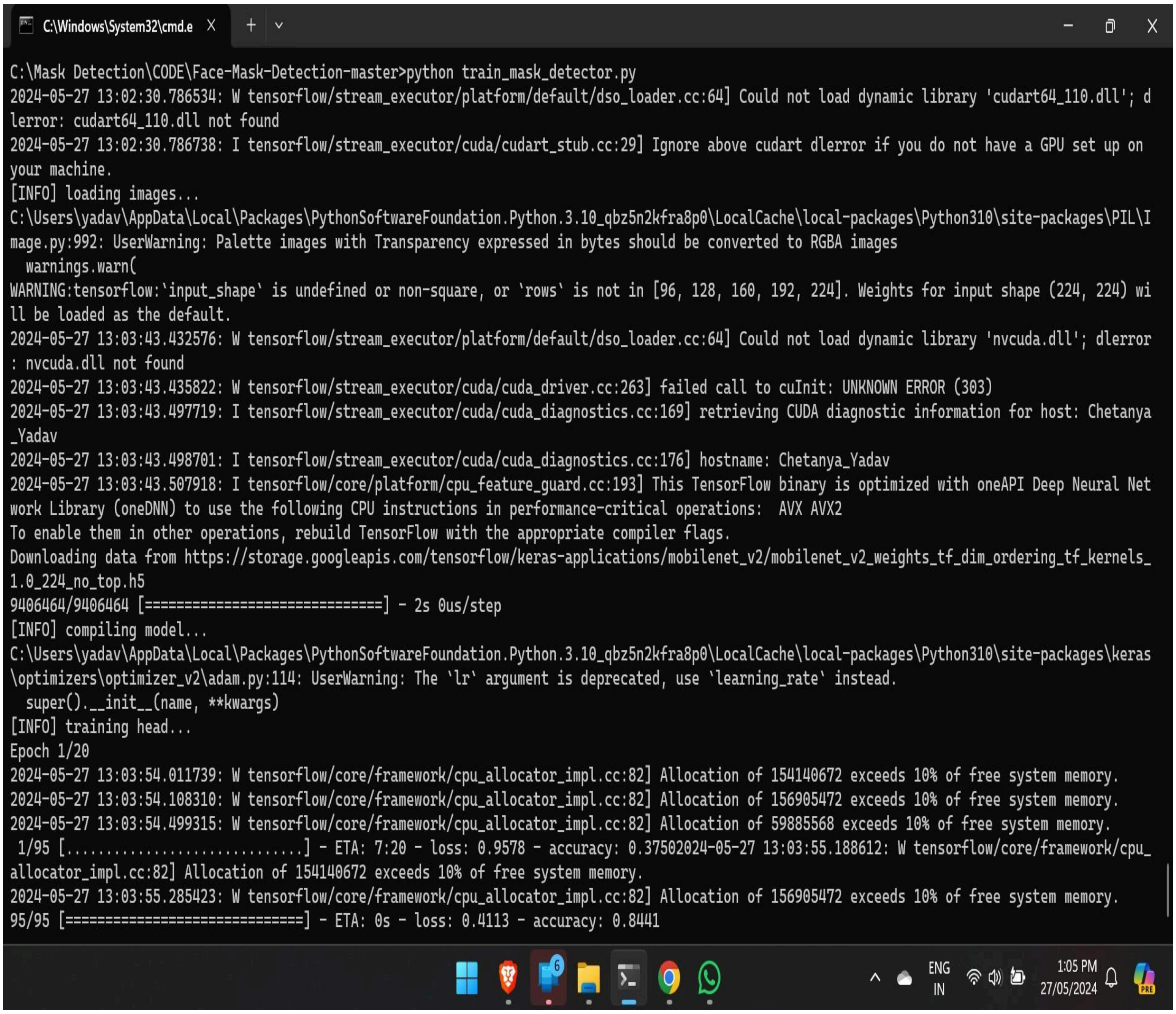
```
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [==============================] - 0s 47ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 50ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 46ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 79ms/step
(1, 1, 200, 7)
```

Fig 5.1.3 Facial optical face mask detection with mask



```
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [============
(1, 1, 200, 7)
1/1 [==============================] - 0s 47ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 50ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 46ms/step
(1, 1, 200, 7)
1/1 [==============================] - 0s 79ms/step
(1, 1, 200, 7)
```

Fig 5.1.4 Facial Optical Face Mask Detection With Mask



```
C:\Mask Detection\CODE\Face-Mask-Detection-master>python train_mask_detector.py
2024-05-27 13:02:30.786534: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; d
lerror: cudart64_110.dll not found
2024-05-27 13:02:30.786738: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on
your machine.
[INFO] loading images...
C:\Users\yadav\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\PIL\I
mage.py:992: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) wi
ll be loaded as the default.
2024-05-27 13:03:43.432576: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'nvcuda.dll'; dlerror
: nvcuda.dll not found
2024-05-27 13:03:43.435822: W tensorflow/stream_executor/cuda/cuda_driver.cc:263] failed call to cuInit: UNKNOWN ERROR (303)
2024-05-27 13:03:43.497719: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: Chetanya
_Yadav
2024-05-27 13:03:43.498701: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: Chetanya_Yadav
2024-05-27 13:03:43.507918: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Net
work Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_
1.0_224_no_top.h5
9406464/9406464 [==============================] - 2s 0us/step
[INFO] compiling model...
C:\Users\yadav\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\keras
\optimizers\optimizer_v2\adam.py:114: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
[INFO] training head...
Epoch 1/20
2024-05-27 13:03:54.011739: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 154140672 exceeds 10% of free system memory.
2024-05-27 13:03:54.108310: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 156905472 exceeds 10% of free system memory.
2024-05-27 13:03:54.499315: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 59885568 exceeds 10% of free system memory.
 1/95 [..............................] - ETA: 7:20 - loss: 0.9578 - accuracy: 0.37502024-05-27 13:03:55.188612: W tensorflow/core/framework/cpu_
allocator_impl.cc:82] Allocation of 154140672 exceeds 10% of free system memory.
2024-05-27 13:03:55.285423: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 156905472 exceeds 10% of free system memory.
95/95 [==============================] - ETA: 0s - loss: 0.4113 - accuracy: 0.8441
```

Fig 5.1.5 Reading and training the model for facial optical face mask detection

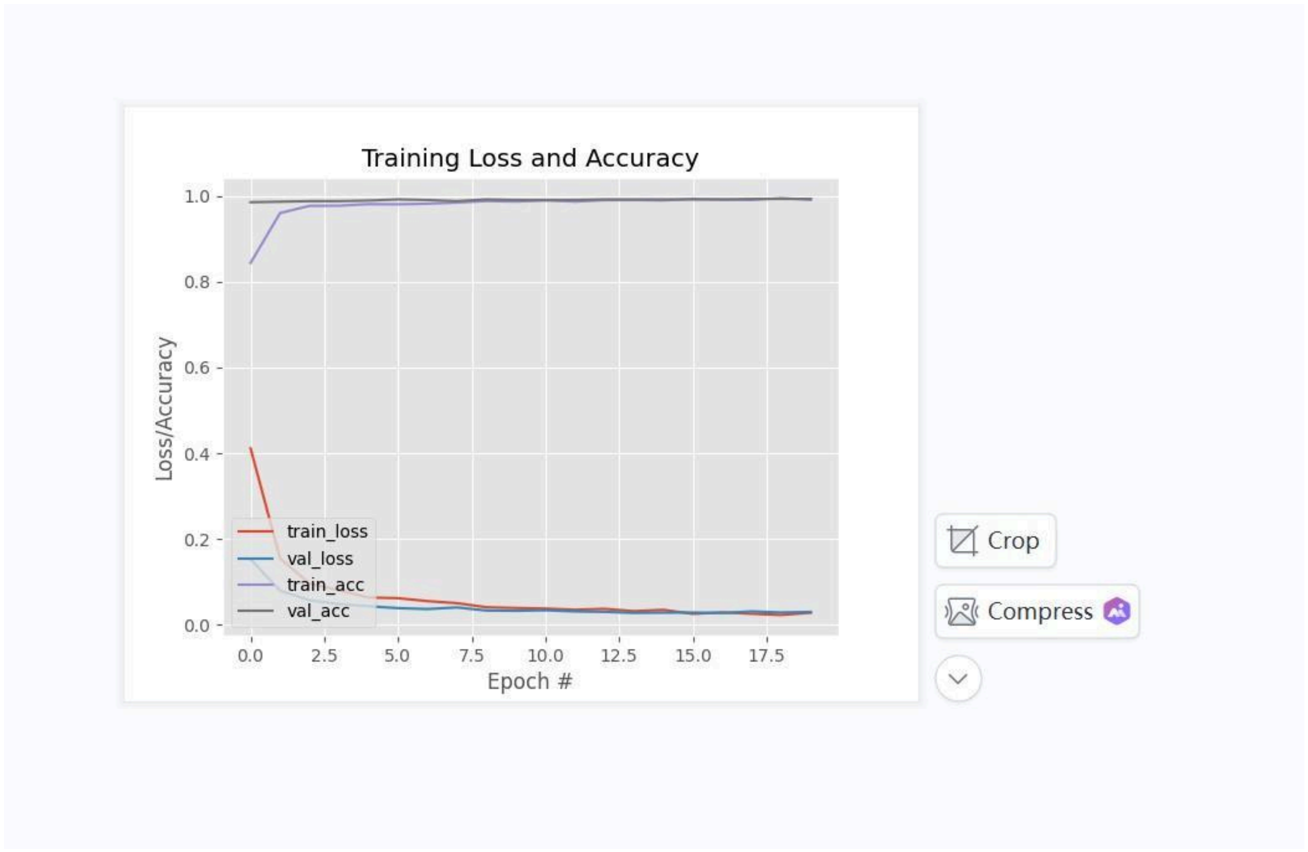
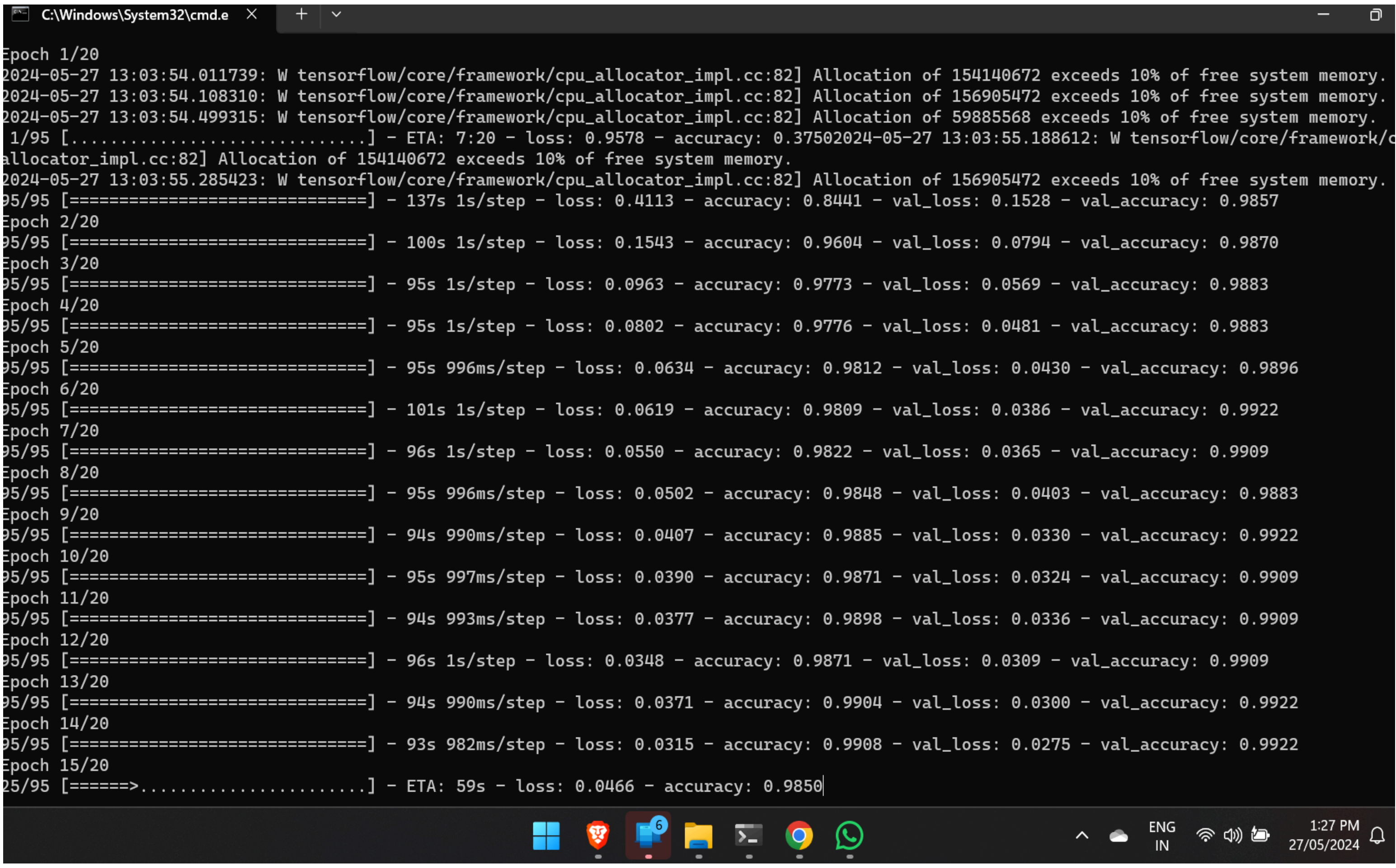


Fig 5.1.6 Chat Report of trained model



```
Epoch 1/20
2024-05-27 13:03:54.011739: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 154140672 exceeds 10% of free system memory.
2024-05-27 13:03:54.108310: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 156905472 exceeds 10% of free system memory.
2024-05-27 13:03:54.499315: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 59885568 exceeds 10% of free system memory.
 1/95 [..............................] - ETA: 7:20 - loss: 0.9578 - accuracy: 0.37502024-05-27 13:03:55.188612: W tensorflow/core/framework/c
allocator_impl.cc:82] Allocation of 154140672 exceeds 10% of free system memory.
2024-05-27 13:03:55.285423: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 156905472 exceeds 10% of free system memory.
95/95 [==============================] - 137s 1s/step - loss: 0.4113 - accuracy: 0.8441 - val_loss: 0.1528 - val_accuracy: 0.9857
Epoch 2/20
95/95 [==============================] - 100s 1s/step - loss: 0.1543 - accuracy: 0.9604 - val_loss: 0.0794 - val_accuracy: 0.9870
Epoch 3/20
95/95 [==============================] - 95s 1s/step - loss: 0.0963 - accuracy: 0.9773 - val_loss: 0.0569 - val_accuracy: 0.9883
Epoch 4/20
95/95 [==============================] - 95s 1s/step - loss: 0.0802 - accuracy: 0.9776 - val_loss: 0.0481 - val_accuracy: 0.9883
Epoch 5/20
95/95 [==============================] - 95s 996ms/step - loss: 0.0634 - accuracy: 0.9812 - val_loss: 0.0430 - val_accuracy: 0.9896
Epoch 6/20
95/95 [==============================] - 101s 1s/step - loss: 0.0619 - accuracy: 0.9809 - val_loss: 0.0386 - val_accuracy: 0.9922
Epoch 7/20
95/95 [==============================] - 96s 1s/step - loss: 0.0550 - accuracy: 0.9822 - val_loss: 0.0365 - val_accuracy: 0.9909
Epoch 8/20
95/95 [==============================] - 95s 996ms/step - loss: 0.0502 - accuracy: 0.9848 - val_loss: 0.0403 - val_accuracy: 0.9883
Epoch 9/20
95/95 [==============================] - 94s 990ms/step - loss: 0.0407 - accuracy: 0.9885 - val_loss: 0.0330 - val_accuracy: 0.9922
Epoch 10/20
95/95 [==============================] - 95s 997ms/step - loss: 0.0390 - accuracy: 0.9871 - val_loss: 0.0324 - val_accuracy: 0.9909
Epoch 11/20
95/95 [==============================] - 94s 993ms/step - loss: 0.0377 - accuracy: 0.9898 - val_loss: 0.0336 - val_accuracy: 0.9909
Epoch 12/20
95/95 [==============================] - 96s 1s/step - loss: 0.0348 - accuracy: 0.9871 - val_loss: 0.0309 - val_accuracy: 0.9909
Epoch 13/20
95/95 [==============================] - 94s 990ms/step - loss: 0.0371 - accuracy: 0.9904 - val_loss: 0.0300 - val_accuracy: 0.9922
Epoch 14/20
95/95 [==============================] - 93s 982ms/step - loss: 0.0315 - accuracy: 0.9908 - val_loss: 0.0275 - val_accuracy: 0.9922
Epoch 15/20
25/95 [======>.......................] - ETA: 59s - loss: 0.0466 - accuracy: 0.9850
```

Fig 5.1.7 Calls Report accuracy of trained models

5.2 Discussion

Discussion on Implementing a Facial Optical Face Mask Detection

System Overview

The implementation of a facial optical face mask detection system is an intricate task that blends multiple fields, including computer vision, deep learning, and real-time processing. This project aims to enhance public health measures by ensuring that individuals adhere to mask-wearing protocols, particularly in crowded or public spaces. The system's design and implementation process involve data collection, preprocessing, model training, integration, and deployment. Below, we discuss the critical aspects and considerations of the implementation process.

Data Collection and

Preprocessing Data Collection:

The foundation of any deep learning project is the quality and quantity of data. For this system, we sourced images from public datasets and custom data collections. Ensuring the dataset is diverse—covering various lighting conditions, angles, and demographics—is crucial for the model's generalizability. However, challenges include balancing the dataset between masked and unmasked images and ensuring privacy when collecting real-world data.

Data Preprocessing:

Preprocessing steps like resizing, normalization, and augmentation are vital for preparing the data for training. These steps help the model learn to identify masks under different conditions. Data augmentation, in particular, increases the robustness of the model by introducing variability in the training data, which helps in preventing overfitting.

Model Selection and Training

Model Selection:

Selecting appropriate models for face detection and mask classification is critical. We used the MobileNetV2 architecture for mask detection due to its balance between performance and efficiency. For face detection, Haar Cascades were initially used for simplicity and speed, although more advanced methods like MTCNN or YOLO could be considered for improved accuracy.

Training Process:

Training deep learning models involves fine-tuning hyperparameters and ensuring the models do not overfit or underfit. Using pre-trained models and transfer learning significantly speeds up the training process and enhances accuracy, given the limited amount of annotated data. The MobileNetV2 model was fine-tuned on our dataset, achieving a good balance between accuracy and computational efficiency.

System Integration

Integration:

Combining face detection and mask classification into a single pipeline is a key integration challenge. The system must process video frames in real-time, detecting faces and then classifying each detected face as masked or unmasked. Using OpenCV for real-time video processing and TensorFlow for model inference allows the system to operate efficiently.

Real-time Processing:

Ensuring the system operates in real-time is essential for practical applications. This involves optimizing the model and the processing pipeline to reduce latency. The use of lightweight models like MobileNetV2 and efficient face detection algorithms helps achieve real-time performance.

Deployment

Deployment Considerations:

Deploying the system using Flask allows for a flexible and accessible web-based interface. However, deploying such systems in real-world environments poses challenges, including ensuring system stability, scalability, and handling various edge cases. Using Docker for containerization can simplify deployment and improve scalability.

User Interface:

The user interface is designed to be intuitive, displaying real-time video with overlays indicating masked and unmasked faces. This allows users to monitor compliance easily. Ensuring the interface is responsive and user-friendly is essential for broad adoption.

Monitoring and Maintenance

System Monitoring:

Post-deployment, continuous monitoring of system performance is necessary. This includes tracking metrics such as detection accuracy, processing time, and system uptime. Implementing logging and error tracking helps in identifying and resolving issues promptly.

Maintenance:

Regular updates to the models and system components are necessary to adapt to changing conditions and improve performance. This includes retraining the model with new data to handle new types of masks or changes in mask-wearing behavior.

Challenges and Future

Work Challenges:

Several challenges were encountered during the project. Ensuring data diversity and quality was a primary challenge, as was optimizing the model for real-time performance without sacrificing accuracy. Addressing privacy concerns during data collection and handling edge cases in real-time detection were other significant challenges.

CHAPTER 6

CONCLUSION & FUTURE SCOPE

6.1 Conclusion

The implementation of a facial optical face mask detection system demonstrates the effective use of computer vision and deep learning to address public health challenges. While the system performs well in controlled environments, ongoing improvements and adaptations are necessary for widespread, real-world deployment. The project highlights the importance of interdisciplinary approaches and the need for robust, scalable, and privacy-conscious solutions in the era of ubiquitous surveillance and health monitoring.

Future Work:

Future work could focus on improving the accuracy and speed of both face detection and mask classification. Exploring more advanced models and optimization techniques, such as quantization or model pruning, could yield better performance. Additionally, expanding the system to detect other personal protective equipment (PPE) and integrating it with broader security or health monitoring systems could enhance its utility.