Mid Term Report On

# Face Recognition Using Open cv

Project-I

**BACHELOR OF TECHNOLOGY**

(Computer Science and Engineering)



**SUBMITTED BY:**

Harshdeep singh (2230777)

Gaurav kumar (2230769)

Dushyant(2230762)

Fazil Fayaz (2230765)

**Under the Guidance of**

MS. Anju Bala

Assisstant Professor

**Department of Computer Science & Engineering**  **Chandigarh Engineering College**

**Jhanjeri Mohali - 140307**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr**  **No** | **Contents** | **Page No** |
| 1. | Introduction | 3 |
| 2. | System Requirements | 5 |
| 3. | Software Requirement Analysis | 7 |
| 4. | Software Design | 11 |
| 5. | Implementation | 14 |
| 6. | References | 19 |

**Chapter - 1**

## Introduction

### Facial recognition is one of the most transformative technologies in the field of artificial intelligence, particularly in security and identification systems. Its ability to identify individuals through their facial features enables a wide range of applications from surveillance and attendance systems to user authentication and smart access control. With the increasing need for secure, contactless, and real-time identity verification, facial recognition has gained prominence in both public and private sectors. OpenCV, an open-source computer vision library, provides a powerful platform for developing such applications efficiently. This mid-term phase of the project focuses on building the foundational components of a facial recognition system using OpenCV and Python.

### The purpose of this stage is to implement a functional face detection system capable of identifying human faces in live video streams captured via webcam. While this version does not yet perform full recognition (i.e., identifying who the person is), it establishes a critical base for advanced modules to be added later, such as training recognition models, optimizing performance, and ensuring privacy and security compliance. The following sections explore the motivation, technical background, and goals of the current implementation.

### Background

Facial recognition technology works by capturing facial features from images or video frames and comparing them with stored data to identify or verify individuals. With OpenCV, developers can access optimized algorithms for real-time image processing, face detection, and basic recognition functions. The rise of deep learning has further enhanced this technology by enabling systems to learn complex patterns from large datasets.

**1.2 Motivation**

The increasing demand for fast and contactless identity verification systems in various fields like education, corporate environments, and public safety has driven interest in facial recognition technology. Traditional methods like ID cards or passwords are easily compromised, whereas facial recognition provides a non-intrusive, secure, and user-friendly alternative.

**1.3 Problem Statement**

While facial recognition systems have become more advanced, challenges still remain. These include varying lighting conditions, pose changes, partial occlusion of faces, and the threat of spoofing attacks. Additionally, concerns around data privacy and computational efficiency must also be addressed. The

mid-term phase of this project is focused on developing a system that can accurately detect faces in real time using OpenCV, with future phases targeting more complex challenges**.**

**1.4 Aim of the Mid-Term Project Phase**

The objective for this phase is to successfully implement a real-time face detection module using OpenCV's Haar Cascade Classifier. This includes integrating webcam-based video input, identifying face locations in frames, and drawing bounding boxes around detected faces. This basic but essential functionality serves as a stepping stone toward full facial recognition capabilities.

**1.5 Technologies Used So Far**

The development so far has relied on Python for coding and OpenCV for image processing. The Haar Cascade Classifier has been used to detect facial regions in grayscale video frames. Initial results confirm the effectiveness of this approach in real-time scenarios, forming the groundwork for implementing recognition and model training.

**1.6 Scope for Further Development**

Following this phase, the project will focus on implementing facial recognition features by integrating deep learning models such as FaceNet or ArcFace. Additionally, efforts will be made to optimize performance for real-time applications, address dataset bias, and include privacy-preserving technologies like differential privacy and federated learning.

**Chapter - 2**

## System Requirement

A facial recognition system based on OpenCV requires a combination of hardware and software components to ensure smooth development, execution, and testing. Since the project deals with real-time video processing and image recognition, it is essential to consider both computational performance and compatibility. The following section outlines the minimum and recommended system requirements categorized into hardware, software, and additional tools essential for successful implementation.

**2.1 Hardware Requirements**

**2.1.1. Processor**

• Minimum: Intel Core i3 (8th Gen or higher) / AMD Ryzen 3

• Recommended: Intel Core i5/i7 (10th Gen or higher) / AMD Ryzen 5 or above

A multi-core processor is preferred for handling real-time video capture and processing without lag.

**2.1.2. RAM**

• Minimum: 4 GB

• Recommended: 8 GB or higher

Adequate RAM ensures smooth handling of video data and multiple libraries without crashing.

**2.1.3 Storage**

• Minimum: 256 GB HDD or SSD

• Recommended: 512 GB SSD

Faster storage devices like SSDs reduce file read/write latency, improving data processing speed.

**2.1.4 Camera**

• A built-in or external HD webcam is required for capturing live video streams.

• Optional: Infrared (IR) or 3D cameras for future enhancements such as liveness detection.

**2.1.5 GPU (Optional but Beneficial)**

• While not mandatory for basic face detection, a GPU (e.g., NVIDIA GTX 1650 or higher) is useful for future deep learning model training and testing.

**2.2 Software Requirements**

**2.2.1 Operating System**

• Minimum: Windows 10 / Linux (Ubuntu 18.04 or later)

• The project is currently developed on Windows 10, but the codebase is compatible with Linux distributions.

**2.2.2 Programming Language**

• Python 3.8 or higher

Python is used for development due to its simplicity and rich support for image processing and AI libraries.

**2.2.3 Libraries and Frameworks**

• OpenCV: For face detection and image processing

• NumPy: For array manipulation

• Matplotlib (optional): For visualization during debugging

• Haar Cascade Classifier: For real-time face detection

• Pip: For package installation and environment setup

**2.2.4. IDE/Code Editor**

• Recommended: Visual Studio Code, PyCharm, or Jupyter Notebook for writing and testing Python code effectively.

**2.3 Additional Tools and Platforms**

**2.3.1 Python Environment Manager**

• Anaconda (optional): For easy management of Python environments and packages.

**2.3.2 Version Control System**

• Git: For version tracking and collaborative development (especially useful in future phases).

**2.3.3 Documentation Tools**

• Microsoft Word or Google Docs: For preparing project reports and documentation.

**2.3.4 Internet and Connectivity**

• Stable Internet Connection: Required for downloading libraries, dependencies, and datasets, and for accessing online documentation or tutorials.

## Chapter – 3

## Software Requirement Analysis

The success of a facial recognition system depends significantly on selecting the right set of software tools that can meet performance, compatibility, and scalability demands. Since this project involves real-time image processing and video feed integration, a detailed analysis of the required software components is essential. This section outlines and analyzes the software environment, development tools, libraries, and frameworks required to develop, test, and deploy the system effectively.

**3.1 Operating System**

**3.1.1 Platform Compatibility**

The project is being developed on a Windows 10 platform due to ease of hardware compatibility and driver support. However, all libraries and code used are cross-platform compatible, making it feasible to shift or scale to Linux-based systems (e.g., Ubuntu) if needed in the future.

**3.1.2 System Stability**

Windows offers a stable GUI for development and testing, while Linux could be considered for deployment due to better resource efficiency and compatibility with edge computing devices like Raspberry Pi or Jetson Nano.

**3.2 Programming Environment**

**3.2.1 Language Selection**

* Python 3.8 or higher has been chosen for this project because of its:
  + Rich support for image processing and machine learning libraries.
  + Simplicity in syntax, which speeds up development.
  + Active community and abundant resources for troubleshooting.

**3.2.2 IDEs and Editors**

* Visual Studio Code and Jupyter Notebook are used for development and testing.
* These environments offer:
  + Built-in debugging tools.
  + Extension support (e.g., Python extensions).
  + Interactive visualization for image processing outputs.

**3.3 Core Libraries and Frameworks**

**3.3.1 OpenCV (Open Source Computer Vision Library)**

* Core component used for:
  + Face detection.
  + Video stream handling.
  + Image preprocessing.
* OpenCV supports real-time applications and has built-in Haar cascade classifiers, ideal for mid-term functionality.

**3.3.2 NumPy**

* Used for:
  + Numerical operations.
  + Efficient matrix and array handling, essential for image processing tasks.

**3.3.3 Haar Cascade Classifier**

* A pre-trained XML-based classifier included in OpenCV.
* Lightweight and efficient, making it suitable for real-time face detection without requiring deep learning models.

**3.3.4 Optional Tools**

* Matplotlib: Used for plotting or visual debugging.
* Pillow: Used for additional image format handling when needed.

**3.4 Dependency and Package Management**

**3.4.1 Pip**

* Python’s default package manager is used to install and manage required libraries.
* Ensures consistent environment setup across machines.

**3.4.2 Virtual Environment (Optional)**

* A virtual environment can be used to isolate dependencies and prevent conflicts between different projects.

**3.5 Development and Version Control Tools**

**3.5.1 Git**

* While not mandatory for mid-term development, Git is recommended for:
  + Version control.
  + Backup and tracking changes.
  + Collaboration in future project phases.

**3.5.2 GitHub/GitLab (Optional)**

* For remote repository management, useful if the project expands to a team-based approach.

**3.6 Deployment Readiness (Future Consideration)**

Though not immediately required in the mid-term phase, the software stack is being designed with future scalability in mind:

* Ready for integration with Flask or FastAPI for creating RESTful APIs.
* Compatible with deep learning frameworks such as TensorFlow or PyTorch in later stages of development.

**Chapter - 4**

## Software Design and Architecture

The system design for this facial recognition project revolves around a modular approach, where each component performs a specific task such as video input, face detection, and output display. Since the current scope is focused on real-time face detection using OpenCV, the architecture is kept lightweight and optimized for quick performance. This section describes the structural layout and functional interaction between the different components of the system.

**4.1 System Overview**

The proposed facial recognition system is designed to detect human faces in a live video stream using OpenCV and Haar Cascade Classifier. The architecture follows a **client-side real-time processing model**, where all computation is performed on the local machine. This design ensures fast execution and minimal latency without relying on external servers or cloud APIs.

**4.2 Architectural Design**

**4.2.1 Layered Architecture**

The system is divided into four primary layers:

* **Input Layer**: Captures live video frames using the device's webcam.
* **Processing Layer**: Performs grayscale conversion and applies Haar cascade-based face detection.
* **Detection Layer**: Identifies the coordinates of facial regions and draws bounding boxes.
* **Output Layer**: Displays the processed video feed with detected faces highlighted in real time.

**4.2.2 Flow Diagram**

Here is a simplified flow of the current system:

pgsql

CopyEdit

[ WebCam Input ]

↓

[ Frame Acquisition ]

↓

[ Grayscale Conversion ]

↓

[ Face Detection (Haar Cascade) ]

↓

[ Draw Bounding Box ]

↓

[ Display Output Frame ]

Each frame is processed independently in a loop to ensure continuous detection during live video streaming.

**4.3 Module Design**

**4.3.1 Video Capture Module**

* Uses cv2.VideoCapture(0) to initialize and read input from the default webcam.
* Continuously captures video frames in real time for further processing.

**4.3.2 Preprocessing Module**

* Converts each frame to **grayscale** to reduce computational load and improve detection accuracy.
* Grayscale images provide better contrast for the Haar Cascade algorithm.

**4.3.3 Face Detection Module**

* Applies the pre-trained Haar Cascade Classifier from OpenCV.
* Detects multiple faces and returns the coordinates of bounding rectangles around detected facial areas.

**4.3.4 Visualization Module**

* Draws rectangles around detected faces using cv2.rectangle() function.
* Displays the updated video frames in a live feed window titled “video\_live”.

**4.4 Technology Stack**

| **Component** | **Technology Used** |
| --- | --- |
| Programming Language | Python 3.x |
| Image Processing | OpenCV |
| Face Detection Model | Haar Cascade Classifier |
|  |  |
| Display and GUI | OpenCV (imshow) |
| Input Source | Built-in/External Webcam |

**Chapter - 5**

## Implementation

The implementation phase of the project focused on creating a functional face detection system using OpenCV. This system captures real-time video input from a webcam and detects human faces within the frames using the Haar Cascade Classifier. The implementation is coded in Python, making use of OpenCV’s built-in libraries and functions for image processing and visualization. This section provides a detailed overview of the steps taken, along with code integration and explanation.

**5.1 Environment Setup**

**5.1.1 Installation of Required Libraries**

The development environment was set up using Python 3.11. The essential libraries were installed using the pip package manager:

* Bash
* CopyEdit
* pip install opencv-python
* pip install numpy

**1.2. Tools Used**

* Python 3.11
* OpenCV (cv2)
* Visual Studio Code as the code editor
* Default system webcam for video input

**5.2 Face Detection Using OpenCV**

**5.2.1 Loading the Haar Cascade Classifier**

A pre-trained Haar Cascade Classifier was used for detecting frontal faces. The classifier is stored in an XML file provided by OpenCV.

python

CopyEdit

face\_cap = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

**5.2.2 Capturing Video from Webcam**

The VideoCapture() method is used to access the system’s webcam.

python

CopyEdit

video\_cap = cv2.VideoCapture(0)

**5.2.3 Processing Frames in Real-Time**

Frames are continuously captured, converted to grayscale, and passed to the classifier for face detection.

python

CopyEdit

while True:

ret, video\_data = video\_cap.read()

gray = cv2.cvtColor(video\_data, cv2.COLOR\_BGR2GRAY)

faces = face\_cap.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

**5.2.4 Drawing Bounding Boxes**

For each detected face, a rectangle is drawn around the facial region.

python

CopyEdit

for (x, y, w, h) in faces:

cv2.rectangle(video\_data, (x, y), (x + w, y + h), (0, 255, 0), 2)

**5.2.5 Displaying the Output**

The processed frames with bounding boxes are displayed using OpenCV's imshow() function.

python

CopyEdit

cv2.imshow("video\_live", video\_data)

if cv2.waitKey(10) == ord('a'):

break

**5.2.6 Releasing Resources**

After the program ends, resources are released and all OpenCV windows are closed.

python

CopyEdit

video\_cap.release()

cv2.destroyAllWindows()

**5.3 Output and Verification**

**5.3.1 Live Output**

The system successfully detects faces from the live webcam feed, draws bounding boxes in real-time, and displays them without noticeable lag.

**5.3.2 Performance**

* Detection is quick and consistent under good lighting.
* Accuracy drops slightly in low-light or with partially visible faces (to be improved in future stages).

**5.4 Challenges Encountered**

* Initial issues with classifier path compatibility across systems.
* Performance fluctuation depending on lighting conditions.
* Difficulty in detecting faces at extreme angles or with occlusions.

**5.5 Future Improvements**

* Integrating face recognition models (e.g., FaceNet or ArcFace).
* Implementing liveness detection to prevent spoofing.

**Chapter – 6**

**REFERENCES**

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
2. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.
3. Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 815–823.
4. Parkhi, O. M., Vedaldi, A., & Zisserman, A. (2015). Deep Face Recognition. *British Machine Vision Conference (BMVC)*.
5. Deng, J., Guo, J., Xue, N., & Zafeiriou, S. (2019). ArcFace: Additive angular margin loss for deep face recognition. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 4690–4699.
6. OpenCV Documentation. (n.d.). Retrieved from <https://docs.opencv.org>
7. TensorFlow Documentation. (n.d.). Retrieved from <https://www.tensorflow.org>
8. PyTorch Documentation. (n.d.). Retrieved from <https://pytorch.org/docs>
9. Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint face detection and alignment using multi-task cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10), 1499–1503.
10. Taigman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014). DeepFace: Closing the gap to human-level performance in face verification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1701–1708.