VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belgavi-590014

Computer Graphics and Image Processing Mini Project Report on

"Recognition of Face Emotion in Real Time"

Submitted to

Visvesvaraya Technological University

In the partial fulfilment of requirements for the award of degree

Bachelor of Engineering

In

COMPUTER SCIENCE AND ENGINEERING

Submitted By

UMME KULSUM	**4RA21CS107**
VARSHITHA M	**4RA21CS112**

Under The Guidance of

Mr. SANJAY M

Assistant Professor, Dept. of CSE

Rajeev Institute of Technology, Hassan

RAJEEV INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

HASSAN-573201

2023-2024

RAJEEV INSTITUTE OF TECHNOLOGY, HASSAN

(Approved by AICTE, New Delhi and Affiliated to VTU, Belagavi.)

Plot #1-D, Growth Centre, Industrial Area, B-M Bypass Road, Hassan – 573201 Ph:(08172)-243180/83/84 Fax:(08172)-243183

Department of Computer Science and Engineering

CERTIFICATE

Certified that the **Computer Graphics and Image Processing Mini Project** entitled **"RECOGNITION OF FACE EMOTION IN REAL TIME"** is carried out by Ms. UMME KULSUM [4RA21CS107], Ms. VARSHITHA M [4RA21CS112] respectively, a bonafide students of **RAJEEV INSTITUTE OF TECHNOLOGY**, Hassan in partial fulfilment for the award of **BACHELOR OF ENGINEERING** in **COMPUTER SCIENCE AND ENGINEERING** of the Visvesvaraya Technological University, Belagavi during the year 2023-2024. The Project Phase-1 report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Mr. SANJAY M	**Dr. SHRISHAIL MATH**	**Dr. MAHESH P K**
Assistant Professor	Head of the Department	Principal
Dept. of Computer Science & Engineering	Computer Science & Engineering	**RIT, HASSAN**
RIT, HASSAN	**RIT, HASSAN**	

Name of the examiners Signature with date

1. .... ……………………

2. ……………..... ……………………

DECLARATION

We **UMME KULSUM, VARSHITHA M** bearing USN **4RA21CS107**,**4RA21CS112,** students of 6th sem B.E in **Computer Science and Engineering**, **Rajeev Institute of Technology, Hassan**, hereby declare that the work being presented in the dissertation entitled "**Recognition of Face Emotion in Real Time**" has been carried out by us under the supervision of guide **Mr. SANJAY M**, Assistant Professor, Computer Science and Engineering, **Rajeev Institute of Technology, Hassan**, as partial fulfilment of requirement for the award of B.E Degree of **Bachelor of Engineering in Computer Science and Engineering at Visvesvaraya Technological University, Belagavi** is and authentic record of my own carried out by us during the academic year 2023-2024.

UMME KULSUM	VARSHITHA M
4RA21CS107	4RA21CS112

PLACE: HASSAN

DATE: ....

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful of any task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

We would like to profoundly thank our **Management of Rajeev Institute of Technology** & our President **Dr.Rachana Rajeev** for providing such a healthy environment.

We would like to express our sincere thanks to our principal **Dr.Mahesh P K**, Rajeev Institute of Technology for his encouragement that motivated us for successful completion of project work.

We wish to express our gratitude to **Dr.Shrishail Math**, Head of the Department of Computer Science & Engineering for providing a good working environment and for his constant support and encouragement.

It gives us great pleasure to express our gratitude to **Mr.Sanjay M** Assistant Professor, Department of Computer Science & Engineering for her expert guidance, initiative and encouragement that led us to complete this project.

We would also like to thank all our staffs of Information Science and Engineering department who have directly or indirectly helped us in the successful completion of this phase-1 project and also, we would like to thank our parents.

UMME KULSUM
4RA21CS107

VARSHITHA M
4RA21CS112

ABSTRACT

The project “RECOGNITION OF FACE EMOTION IN REAL TIME” is used to demonstrate the use of OpenCV functions that is defined in OpenCV. Our objective is to develop a robust and efficient system that accurately detects and interprets human emotions in real-time from live video streams, enabling applications in various domains such as healthcare, education, entertainment, and human-computer interaction.

Real-time recognition of face emotions is a pivotal area in computer vision and artificial intelligence, aiming to interpret and respond to human emotional states from live video streams. This study explores the application of deep learning techniques, specifically convolutional neural networks (CNNs), for detecting facial expressions such as happiness, sadness, anger, surprise, and others. The methodology involves preprocessing frames using OpenCV, detecting faces with Haar cascades, and analyzing emotional features using models like DeepFace. Experimental results demonstrate robust performance in real-world scenarios, showcasing high accuracy in identifying dominant emotional states from diverse facial expressions. The findings highlight the potential of real-time emotion recognition systems in applications ranging from healthcare and education to human-computer interaction, paving the way for more empathetic and responsive technology interfaces.

CONTENTS

CHAPTER 1

INTRODUCTION

Computer graphics are graphics created using computers and more generally, the representation and manipulation of image data by a computer.

The development of computer graphics has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have a profound impact on many types of media and have revolutionized animation, movies and the video game industry.

1.1 Overview of Computer Graphics

The term computer graphics has been used in a broad sense to describe “almost everything on computers that is not text or sound. It is one of the most powerful and interesting facts of computer. There is a lot that you can do apart from drawing figures of various shapes.

Today, computers and computer-generated images touch many aspects of daily life. Computer image is found on television, in newspapers, for example in weather reports, in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports, and other presentation material.

Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D , 3D , 5D, and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with “the visualization of three-dimensional phenomena” (architectural, meteorological, medical, biological, etc.), where emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time component).

1.2 History of Computer Graphics

Computer graphics was first created as a visualization tool for scientists and engineers in government and corporate research centers such as Bell Labs and Boeing in the 1950s. Later the tools would be developed at universities in the 6os and 70s at places such as Ohio State University, MIT, University of Utah Cornshell, North Carolina and the New York Institute of technology. The early breakthroughs that took place in academic centers continued at research centers such as the famous Xerox PARC in the 1970's. These efforts broke first into broadcast video graphics and then major motion pictures in the late 70's and early 1980's. Computer graphic research continues today around the production companies. Companies such as George Luca's Industrial light and magic are constantly redefining the cutting edge of computer graphic technology in order to present the world with a new synthetic digital reality.

1.3 Applications of Computer Graphics

Nowadays Computer Graphics used in almost all the areas ranges from science, engineering, medicine, business, industry government, art, entertainment, education and training.

1.3.1 CG in the field of CAD

Computer Aided design methods are routinely used in the design of buildings, automobiles, aircraft, watercraft, spacecraft computers, textiles and many other applications.

1.3.2 CG in Presentation Graphics

Another major application area presentation graphics used to produce illustrations for reports or generate slides. Presentation graphics is commonly used to summarize financial, statistical, mathematical, scientific data for research reports and other types of reports.2D and 3D bar chart to illustrate some mathematical or statistical report.

1.3.3 CG in computer Art

CG methods are widely used in both fine art and commercial art applications. Artists use a variety of computer methods including special purpose hardware, artist's paintbrush program, other pain packages, desktop packages, mathematics packages, animation packages that provide facility for designing object motion. Ex: cartoons decision is an example of computer art which uses CG.

1.3.4 Image Processing

Concerned with fundamentally different operations. In CG a computer is used to create a picture. Image processing on the other hand applies techniques to modify existing pictures such as photo scans, TV scans.

1.3.5 User Interface

It is a common for software packages to provide a graphical interface . A major component of a graphical interface is a window manager that allows a user to display multiple window area. Interface also displays menus, icons for fast selection and processing.

1.3.6 Education and Training

Computer generated models of physical, financial, economic system often acts as education aids. For some training application special system are designed. Ex: specialized system for simulator for practice sessions or training of ship captain, aircraft pilots and traffic control.

1.4 Statement of the Project

To design and implement the Balloon Blast game using OpenGL. This game is enriched with OpenGL functions which is learnt from the classes of Computer Graphics.

1.5 Objectives

To implement the concepts of Computer Graphics we have learnt.

1.6 Organization of the Report

The next chapter 2 deals with the introduction to OpenGL. The chapter 3 gives the concept of this project. The chapter 4 describes the design and implementation of the project work, followed by presentation of few output snapshots. The chapter 6 provides the conclusion and also mentions the future scope. At the end the references used for the project are listed.

1.7 Hardware Requirements

This project requires a hardware requirement of a processor of speed 1.0GHz, 512MB or above RAM capacity, a graphic card, a Keyboard and a free space minimum of 500KB on hard disk.

- ❖ **Computer**: Modern multi-core CPU, recommended with a GPU (e.g., NVIDIA GTX 1060+).
- ❖ **Webcam**: HD webcam (720p or higher).
- ❖ **RAM**: At least 8GB (16GB recommended).
- ❖ **Storage**: SSD with at least 256GB.
- ❖ **Internet**: Stable connection for downloading software and models.

1.8 Software Requirements

- ❖ **Operating System**: Windows 10/11, macOS, or Linux.
- ❖ **Programming Language**: Python 3.6+.
- ❖ **Libraries**:
  - ✓ OpenCV (pip install openCV-python)
  - ✓ Dlib (pip install dlib)
  - ✓ TensorFlow or PyTorch (pip install tensorflow or pip install torch)
  - ✓ Keras (pip install keras)
  - ✓ NumPy (pip install numpy)

1.9 Technology Used

Here graphical implementation is done using OpenCV graphical package. Python programming language is used for development of this project.

CHAPTER 2

INTRODUCTION TO OPENCV

OpenCV (Open-Source Computer Vision Library) is a powerful open-source library for computer vision and machine learning. Developed by Intel, it provides a comprehensive set of tools for image processing, video analysis, and real-time computer vision applications, supporting multiple programming languages like Python, C++, and Java.

2.1 OpenCV Fundamentals

OpenCV is a versatile library for image processing, video analysis, and object detection. It is widely used for tasks such as face detection and recognition, motion tracking, and augmented reality. Its cross-platform compatibility allows it to run on various operating systems, including Windows, Linux, macOS, Android, and iOS, making it an essential resource for both academic research and industrial applications. OpenCV is essential for developing real-time computer vision applications.

2.1.1 Primitive and Commands

In OpenCV, primitives serve as foundational elements essential for image and video processing tasks. Points are defined by their coordinates in a 2D space, crucial for marking specific locations or key points within images. Lines and shapes encompass basic geometric entities such as circles, rectangles, and polygons, which are used extensively for drawing and delineating regions of interest.

The Mat object stands as the core data structure in OpenCV, organizing image data into matrices of pixels. This structure not only facilitates efficient storage and retrieval of image information but also enables a wide range of operations, including pixel-level manipulation, filtering, and transformations. These primitives collectively empower developers to implement sophisticated computer vision applications, leveraging OpenCV's comprehensive suite of functions for tasks such as object detection, feature extraction, and image enhancement with precision and efficiency.

OpenCV offers a comprehensive suite of commands that facilitate diverse image and video processing tasks. Key functions include **cv2.imread()** for reading images from files and **cv2.imwrite()** for saving processed images. Visualizing results is made simple with **cv2.imshow(),** while drawing primitives like lines (**cv2.line()**), circles (**cv2.circle()**), rectangles (**cv2.rectangle()**), and text (**cv2.putText()**) allows for annotation and highlighting within images.

Image transformations such as resizing (**cv2.resize()**), color space conversion (**cv2.cvtColor()),** and filtering operations like Gaussian blur (**cv2.GaussianBlur())** enable manipulation and enhancement of image quality. For video processing, commands like **cv2.VideoCapture()** capture video frames, and **cv2.VideoWriter()** facilitates writing processed frames to video files. Advanced capabilities include feature detection (**cv2.findContours(), cv2.goodFeaturesToTrack()**) and integration with machine learning models (cv2.ml) and deep learning frameworks (**cv2.dnn**), empowering developers to build sophisticated computer vision applications efficiently.

2.1.2 Procedural versus Descriptive

In OpenCV, the procedural approach involves step-by-step function calls like **cv2.imread()** for reading images and **cv2.imshow()** for displaying them, ensuring precise control over tasks. Meanwhile, the descriptive approach focuses on conceptualizing tasks broadly, defining goals like edge detection or feature extraction without specifying exact function sequences, allowing for flexible application of OpenCV's capabilities based on understanding and intent. Both methods cater to different needs—procedural for detailed task execution and descriptive for strategic planning—depending on the complexity and goals of the computer vision project.

2.1.3 Execution Model

In OpenCV, the execution model follows a sequential paradigm where tasks are carried out in a linear sequence of function calls. Each function, such as cv2.imread() for reading images or cv2.cvtColor() for color space conversion, processes input data—like images or video frames—based on specified parameters and returns processed outputs. This sequential approach ensures that operations are performed in the order they are called, allowing for precise control over the flow of image and video processing tasks. Developers can construct processing pipelines by chaining functions together, enabling complex operations such as image preprocessing, feature extraction, and object detection. This structured execution model in OpenCV supports the implementation of efficient and organized computer vision applications, accommodating various stages of data manipulation and analysis seamlessly.

In addition to its sequential execution model, OpenCV's approach emphasizes modular processing and iterative refinement of tasks. This methodology allows developers to build sophisticated workflows by combining different functions iteratively, refining image processing steps, and adjusting parameters to achieve desired outcomes. Each function call operates independently on data, contributing to a cohesive pipeline where input flows through a series of transformations and analyses, enhancing flexibility and adaptability in addressing diverse computer vision challenges. This iterative and modular approach in OpenCV supports both incremental development and optimization of algorithms, ensuring efficient and effective processing of visual data across various applications and environments.

2.1.4 Basic OpenCV Operation

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. It contains over 2500 optimized algorithms which can be used for various tasks such as image processing, video analysis, and object detection. Here are some basic operations you can perform with OpenCV:

1. Installation

To use OpenCV, you need to install it first. You can install it using pip:

- pip install opencv-python

2. Reading and Displaying Images

```
import cv2

# Read an image from file
image = cv2.imread('image.jpg')

# Display the image in a window
cv2.imshow('Image', image)

# Wait for a key press and close the image window
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. Converting to Grayscale

```
# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
# Display the grayscale image
cv2.imshow('Grayscale Image', gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4. Resizing an Image

```
# Resize the image
resized_image = cv2.resize(image, (width, height))

# Display the resized image
cv2.imshow('Resized Image', resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5. Drawing Shapes

```
# Draw a rectangle
cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), thickness=2)

# Draw a circle
cv2.circle(image, (center_x, center_y), radius, (255, 0, 0), thickness=2)

# Draw a line
cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), thickness=2)

# Display the image with shapes
cv2.imshow('Shapes', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

6. Writing Text

```
# Write text on the image
cv2.putText(image, 'Hello, OpenCV!', (x, y), cv2.FONT_HERSHEY_SIMPLEX, font_scale=1,
color=(255, 255, 255), thickness=2)

# Display the image with text
cv2.imshow('Text', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

7. Edge Detection

```
# Perform edge detection using Canny
edges = cv2.Canny(gray_image, threshold1=100, threshold2=200)

# Display the edges
cv2.imshow('Edges', edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

8. Video Capture

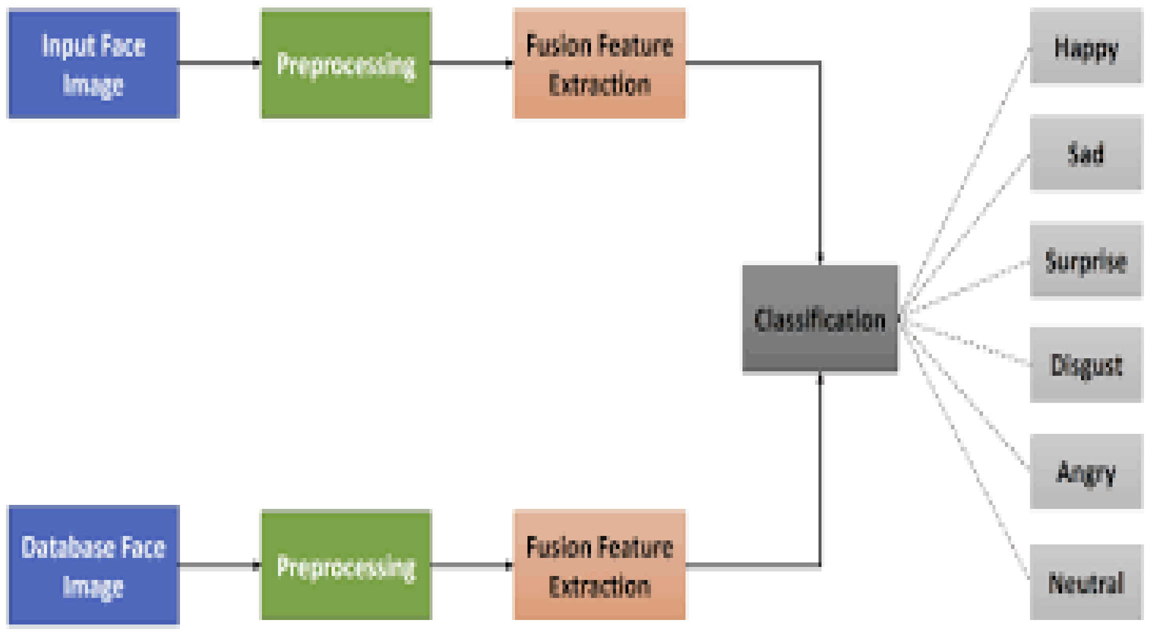
```
# Open a connection to the webcam
cap = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Display the resulting frame
    cv2.imshow('Webcam', frame)
    # Break the loop on 'q' key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture and close windows
cap.release()
cv2.destroyAllWindows()
```

2.1.5 Basic Operation of Recognition of Face Emotion in Real Time

To recognize facial emotions in real-time using OpenCV and a pre-trained model, follow these steps:

1. **Install Libraries**: Install OpenCV, TensorFlow, and Keras using pip.
2. **Import and Load Models**: Import the required libraries, load the pre-trained emotion recognition model, and the Haar Cascade face detector.
3. **Capture Video**: Start video capture from the webcam.
4. **Process Frames**:
   - Convert each frame to grayscale.
   - Detect faces in the frame using the Haar Cascade classifier.
   - For each detected face, extract and resize the face region.
   - Normalize the face region and prepare it for prediction.
   - Use the emotion recognition model to predict the emotion.
5. **Display Results**:
   - Draw a rectangle around the detected face.
   - Display the predicted emotion label above the rectangle.
6. **Run Loop**: Continuously capture, process, and display video frames in real-time.
7. **Exit on Key Press**: Break the loop and close all windows when the 'q' key is pressed

2.1.6 ARCHITECTURE



2.2 OpenCV API

OpenCV's API stands as a versatile toolkit for implementing advanced computer vision solutions effortlessly. It provides developers with a rich array of functions tailored for tasks ranging from basic image processing to sophisticated operations.

This API supports seamless integration across multiple platforms and programming languages, enabling developers to leverage its capabilities for applications in fields like augmented reality, autonomous systems, and medical imaging. With its extensive documentation, active community, and continual updates, OpenCV's API remains pivotal in driving innovation and advancing the boundaries of visual computing technologies.

2.2.1 Display Control Functions

Display control functions in OpenCV are essential for visualizing and interacting with images and videos during the development and deployment of computer vision applications. The primary functions include:

1. **cv2.imshow()**: This function displays an image in a window. It takes the window name and the image matrix (loaded using cv2.imread()) as arguments. This is crucial for inspecting intermediate processing results or final outputs.
2. **cv2.waitKey()**: Used in conjunction with cv2.imshow(), cv2.waitKey() waits for a specific delay for a key event in milliseconds. It is essential for controlling the duration of displayed frames and handling user interactions.
3. **cv2.destroyAllWindows()**: This function closes all OpenCV windows createdby cv2.imshow(). It is used to clean up and release resources after processing or when terminating an application.

These display control functions are fundamental for developers to visualize and debug their computer vision algorithms effectively, providing real-time feedback on image processing and analysis tasks. They facilitate interactive exploration of data and results, essential for refining and optimizing computer vision applications across various domains.

2.2.2 OpenCV Basic Drawing

OpenCV provides robust functionality for drawing basic shapes and text on images, crucial for annotating visual data and highlighting regions of interest. Key drawing functions include:

1. **Lines**: Draw straight lines on images using cv2.line(). This function requires specifying the image to draw on, starting and ending coordinates, color, and line thickness.
2. **Rectangles**: Create rectangles with cv2.rectangle(), which allows defining the image, top-left and bottom-right corner coordinates, color, and thickness. This function is useful for bounding boxes and visual annotations.
3. **Circles**: Draw circles on images using cv2.circle(), specifying the image, centre coordinates, radius, color, and thickness. It's commonly used for marking points of interest or visualizing circular objects.
4. **Text**: Add text to images with cv2.putText(), which requires specifying the image, text string, position, font face, font scale, color, and thickness. This function is essential for labelling images with information or annotations.
5. **Polygons**: Use cv2.polylines() to draw polygons on images by specifying the image, vertices, whether to close the shape, color, and thickness. This function is versatile for outlining irregular regions or complex shapes.
6. **Ellipses**: Draw ellipses with cv2.ellipse(), which allows defining the image, centre coordinates, axes lengths, rotation angle, start and end angles, color, and thickness. This function is useful for highlighting elliptical features or areas of interest.
7. **Customization**: Each drawing function in OpenCV allows customization of color using RGB or BGR values, line thickness, and other parameters to tailor annotations to specific visualization needs. This flexibility empowers developers to create informative and visually appealing representations of image analysis results.

2.2.3 OpenCV Utilities for Complex Objects

OpenCV provides a robust suite of utilities tailored for handling complex objects in computer vision applications. These utilities encompass a range of functionalities essential for tasks such as object detection, tracking, and 3D reconstruction:

❖ **Feature Detection and Description**: OpenCV includes algorithms like SIFT, SURF, and ORB for detecting key points and describing local features in images, crucial for matching objects across different views or frames.

❖ **Object Tracking**: Algorithms such as the Kalman Filter, CAMShift, and MOSSE in OpenCV facilitate robust object tracking across video frames. They maintain continuity in tracking objects' positions, even when they undergo changes in appearance or occlusion.

❖ **3D Reconstruction**: Techniques like Structure from Motion (SfM) and Stereo Vision allow OpenCV to reconstruct three-dimensional scenes from multiple images or stereo pairs. This capability supports applications requiring spatial understanding, such as augmented reality and 3D modeling.

❖ **Image Segmentation**: Algorithms such as GrabCut and Watershed in OpenCV are employed for segmenting complex objects from backgrounds in images. This aids in isolating objects of interest for further analysis or manipulation.

2.2.3 Transformations and Perspective:

In OpenCV, transformations and perspective play crucial roles in manipulating and understanding images and video frames.

❖ **Geometric Transformations**: OpenCV offers functions like cv2.warpAffine() and cv2.warpPerspective() for geometric transformations.

i. **cv2.warpAffine()**: This function applies an affine transformation to an image, which includes translation, rotation, scaling, and shearing.

ii. **cv2.warpPerspective()**: Used for perspective transformations, this function warps an image according to a specified 3x3 transformation matrix, allowing corrections for perspective distortions in images captured from different viewpoints.

- ❖ **Homography**: OpenCV computes a homography matrix using methods such as cv2.findHomography(). This matrix describes the transformation between two planes seen from different viewpoints, essential for tasks like image stitching and augmented reality.
- ❖ **Camera Calibration**: OpenCV supports camera calibration with functions like cv2.calibrateCamera() and cv2.undistort(). These functions estimate intrinsic and extrinsic camera parameters, correcting lens distortions and enabling accurate spatial measurements and object tracking.
- ❖ **Perspective Correction**: For correcting perspective in images, OpenCV provides tools like cv2.getPerspectiveTransform() and cv2.warpPerspective(). These functions are crucial for transforming images to rectify skewed perspectives or aligning images for accurate measurements and analysis.
- ❖ **Augmented Reality (AR)**: By combining geometric transformations with camera calibration techniques, OpenCV facilitates AR applications where virtual objects are seamlessly integrated into real-world scenes. This involves estimating camera poses and rendering virtual objects from the correct perspectives in real-time.

2.2.4 Associated utility libraries

OpenCV is often complemented by several utility libraries that extend its capabilities or provide additional functionalities for specific tasks in computer vision and image processing. Some of the associated utility libraries commonly used alongside OpenCV include:

1. **NumPy**: A fundamental library for numerical computing in Python, NumPy provides support for multidimensional arrays and matrices, essential for handling and manipulating image data efficiently within OpenCV.

2. **SciPy**: Built on NumPy, SciPy offers advanced mathematical functions, optimization, signal processing, and statistical algorithms. It complements OpenCV by providing tools for tasks such as interpolation, signal filtering, and scientific computations.
3. **Matplotlib**: A plotting library in Python, Matplotlib is useful for visualizing data and displaying images or graphs generated from OpenCV operations. It helps in presenting analysis results, debugging, and understanding the output of computer vision algorithms.
4. **Scikit-image**: This library provides a collection of algorithms for image processing tasks not directly covered by OpenCV, such as segmentation, feature extraction, and morphology. It offers additional tools for advanced image analysis and manipulation.
5. **Pillow (PIL)**: Pillow is a fork of the Python Imaging Library (PIL) and provides capabilities for image processing and manipulation, including opening, manipulating, and saving various image file formats. It complements OpenCV by supporting additional image operations and format conversions.
6. **TensorFlow and PyTorch**: Deep learning frameworks like TensorFlow and PyTorch integrate with OpenCV through their respective bindings (cv2.dnn module in OpenCV). They enable leveraging pre-trained neural network models for tasks such as object detection, image classification, and image segmentation, enhancing the capabilities of OpenCV in complex deep learning applications.
7. **CUDA and cuDNN**: NVIDIA's CUDA platform and cuDNN library accelerate deep learning and image processing tasks on NVIDIA GPUs. OpenCV provides CUDA-enabled modules (cv2.cuda) for leveraging GPU acceleration, significantly speeding up operations like matrix calculations and neural network inference.
8. **Dlib**: Dlib is a C++ library with Python bindings that provides tools and algorithms for machine learning, computer vision, and numerical optimization. It includes implementations of facial detection, landmark detection, object tracking, and deep learning-based face recognition, complementing OpenCV in advanced face and object-related tasks.
9. **OpenCV Contrib Modules**: OpenCV Contrib Modules are additional modules not included in the main OpenCV library but offer specialized algorithms and tools. These modules include features like text detection (text module), 3D reconstruction (sfm and rgbd modules), and

computational photography (photo module), expanding the capabilities of OpenCV for specific applications.

10. **SimpleCV**: SimpleCV is an open-source framework based on Python that wraps around OpenCV, providing a higher-level interface for beginners and enthusiasts to work with computer vision. It simplifies common tasks such as image capture, processing, feature detection, and visualization, making it accessible for educational and hobbyist projects.

11. **Mahotas**: Mahotas is a computer vision library for Python that provides implementations of

traditional image processing algorithms, including feature detection, filtering, morphology, and texture analysis. It serves as a complementary library to OpenCV, particularly for applications requiring specific image processing techniques not covered in OpenCV's core modules.

12. **VTK (Visualization Toolkit)**: VTK is an open-source software system for 3D computer graphics, image processing, and visualization. It offers advanced capabilities for rendering, volume rendering, and surface extraction from medical images, which can be integrated with OpenCV for applications in medical imaging, scientific visualization, and virtual reality.

CHAPTER 3

PROJECT DESCRIPTION

Objective

The objective of this project is to develop a real-time system for detecting facial emotions using a webcam. The system will capture video frames, detect faces, analyze facial expressions to determine the dominant emotion, and display the results on the screen.

Overview

This project combines OpenCV for real-time video capture and face detection with DeepFace for emotion analysis. The system will process video frames in real-time, detect faces, analyze emotions, and display the detected emotions on the screen.

Requirements

1. **Hardware**:
   - A webcam for capturing video.
   - A computer with sufficient processing power.
2. **Software**:
   - Operating System: Compatible with the libraries used (Windows, macOS, or Linux).
   - Python environment with the following libraries installed:
     - OpenCV (opencv-python)
     - DeepFace (deepface)
     - Other dependencies such as NumPy, TensorFlow/Keras.

3.1 Graphics Functions

1. **Video Capture**: Captures video frames from the webcam.

```
cap = cv2.VideoCapture(0)
```

2. **Face Detection**: Detects faces in the captured video frames.

```
face_classifier = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

faces = face_classifier.detectMultiScale(frame_gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
```

3. **Emotion Analysis**: Analyzes the detected face regions for emotions.

```
response = DeepFace.analyze(frame, actions=("emotion",), enforce_detection=False)
```

4. **Drawing Graphics**: Draws rectangles around detected faces and displays the predicted emotion labels.

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

    if dominant_emotion:

        cv2.putText(frame, text=dominant_emotion, org= (x, y), fontFace=cv2.FONT_HERSHEY_COMPLEX, fontScale=1, color= (0, 255, 0), thickness=2)

    cv2.rectangle(frame, (x, y), (x + w, y + h), color= (255, 0, 0), thickness=2)
```

5. **Display Frame**: Displays the processed video frames with annotations.

```
cv2.imshow("Facial Emotion Analysis", frame)
```

6. **Exit Mechanism**: Allows the user to exit the program by pressing a specific key.

```
if cv2.waitKey(30) & 0xFF == ord('q'):

    break
```

3.2 USER DEFINED FUNCTIONS

1. **initialize_face_classifier**:
   - **Purpose**: Load and return the face classifier.
   - **Parameters**: None.
   - **Returns**: Face classifier object.

2. **open_video_capture**:
   - **Purpose**: Open and return the video capture object.
   - **Parameters**: None.
   - **Returns**: Video capture object.

3. **capture_frame**:
   - **Purpose**: Capture a frame from the video stream.
   - **Parameters**: cap (video capture object).
   - **Returns**: Tuple containing a boolean (ret) indicating success and the captured frame.

4. **convert_to_grayscale**:
   - **Purpose**: Convert the captured frame to grayscale.
   - **Parameters**: frame (the captured frame).
   - **Returns**: Grayscale frame

5. **detect_faces**:
   - **Purpose**: Detect faces in the grayscale frame.
   - **Parameters**: face_classifier (the face detection model), frame_gray (grayscale frame).
   - **Returns**: List of rectangles, where each rectangle contains the coordinates of a detected face.

6. **analyze_emotion**:

- **Purpose**: Analyze the frame for emotions using DeepFace.
- **Parameters**: frame (the current video frame).
- **Returns**: The dominant emotion detected in the frame or None if no emotion is detected.

7. **draw_annotations**:

- **Purpose**: Draw rectangles around detected faces and annotate with the detected emotion.
- **Parameters**: frame (the current video frame), faces (list of detected faces), dominant_emotion (the detected emotion).
- **Returns**: The annotated frame.

8. **display_frame**:

- **Purpose**: Display the processed frame.
- **Parameters**: frame (the current video frame).
- **Returns**: None.

Main Function

- **main**:
  - **Purpose**: Main loop to run the facial emotion recognition system.
  - **Parameters**: None.
  - **Returns**: None.
  - **Description**: Initializes the system, captures frames, detects faces, analyzes emotions, draws annotations, and displays the frame. It continues this loop until the 'q' key is pressed.

3.3 IMPLEMENTATION

```
import os
import cv2
from deepface import DeepFace
# Ensure CUDA is not used
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

# Load the face classifier
face_classifier = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Open the video capture
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(frame_gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))

    # Analyze the frame for emotion
    response = DeepFace.analyze(frame, actions=("emotion",), enforce_detection=False)
    print(response)

    # Check if "dominant_emotion" key exists in the response
    dominant_emotion = response[0]['dominant_emotion'] if response and isinstance(response, list)
```

```
and 'dominant_emotion' in response[0] else None

    # Draw rectangles around detected faces and annotate with the detected emotion
    for (x, y, w, h) in faces:
        if dominant_emotion:
            cv2.putText(frame, text=dominant_emotion, org=(x, y),
fontFace=cv2.FONT_HERSHEY_COMPLEX, fontScale=1, color=(0, 255, 0), thickness=2)
        cv2.rectangle(frame, (x, y), (x + w, y + h), color=(255, 0, 0), thickness=2)
# Show the frame
    cv2.imshow("Facial Emotion Analysis", frame)

    # Break the loop if the 'q' key is pressed
    if cv2.waitKey(30) & 0xFF == ord('q'):
        break

# Release the video capture and destroy all windows
cap.release()
cv2.destroyAllWindows()
```

CHAPTER 4

SNAPSHOTS

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

Real-time facial emotion recognition uses technologies like OpenCV and deep learning frameworks such as DeepFace to analyze emotions from live video feeds. This innovation is crucial in fields like healthcare for patient monitoring, retail for customer sentiment analysis, and entertainment for personalized experiences. By detecting faces with algorithms like Haar cascades and analyzing expressions to identify emotions (e.g., happiness, sadness, anger), these systems enhance user engagement and enable empathetic human-machine interactions. As the technology advances, its accuracy and efficiency in understanding emotions will continue to improve.

5.2 FUTURE SCOPE

- ❖ **Advancements in AI**: Continued improvements in artificial intelligence (AI), especially deep learning algorithms, will enhance the accuracy and reliability of emotion recognition systems.
- ❖ **Integration with IoT**: Emotion recognition technologies will increasingly integrate with Internet of Things (IoT) devices for real-time emotional analysis in various environments.
- ❖ **Healthcare Applications**: There will be significant applications in healthcare for monitoring patient emotional states, aiding in mental health diagnosis, and supporting personalized treatment plans.
- ❖ **Personalized User Experiences**: Emotion recognition will enable personalized user experiences in entertainment, education, and digital content delivery based on real-time emotional feedback.
- ❖ **Ethical Considerations**: Addressing ethical concerns related to privacy, data security, and bias will be crucial for the responsible development and deployment of these technologies.
- ❖ **Cross-Industry Adoption**: Industries beyond healthcare and entertainment, such as marketing, automotive, and public safety, will adopt emotion recognition for diverse applications like customer insights and behavioral analysis.

REFERENCES

[1]. www.Github.com

[2]. https://www.hackster.io/opencv/projects

[3]. www.stackoverflow.com

[4]. https://opencv.org/

[5]. www.youtube.com