**NAME :Dua zahra**

**Roll number:036**

**SUBJECT:PAI-Lab**

**Image Processing Using OpenCV**

## Introduction

Image processing is a crucial field in computer vision that allows us to manipulate, analyze, and enhance images using various techniques. This report covers fundamental image processing operations performed using OpenCV, a powerful Python library for computer vision.

## Setting Up the Environment

To begin, we created a virtual environment and installed the necessary libraries. This step ensures a clean workspace for our project.

python -m venv venv

venv/Scripts/activate

pip install opencv-python numpy matplotlib pandas pillow

## Image Handling Operations

### 1. Reading and Displaying an Image

We used OpenCV's cv2.imread() function to read an image and cv2.imshow() to display it. The window remains open until a key is pressed.

import cv2

img = cv2.imread("image.jpeg", cv2.IMREAD\_COLOR)

cv2.imshow("Image", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

### 2. Saving an Image

We saved the image using cv2.imwrite().

cv2.imwrite("output.jpeg", img)

## Image Operations

### 1. Adding and Subtracting Images

We performed addition and subtraction on two images using cv2.add() and cv2.subtract().

img1 = cv2.imread("image1.jpeg")

img2 = cv2.imread("image2.jpeg")

added = cv2.add(img1, img2)

subtracted = cv2.subtract(img1, img2)

cv2.imshow("Added Image", added)

cv2.imshow("Subtracted Image", subtracted)

cv2.waitKey(0)

### 2. Bitwise Operations

We applied bitwise\_and, bitwise\_or, bitwise\_xor, and bitwise\_not operations to images.

bitwise\_and = cv2.bitwise\_and(img1, img2)

bitwise\_or = cv2.bitwise\_or(img1, img2)

bitwise\_xor = cv2.bitwise\_xor(img1, img2)

bitwise\_not = cv2.bitwise\_not(img1)

## Image Transformations

### 1. Resizing

We resized an image using different scaling factors.

resized = cv2.resize(img, (800, 600))

### 2. Blurring

We applied different blurring techniques such as Gaussian, Median, and Bilateral filtering.

gaussian\_blur = cv2.GaussianBlur(img, (7, 7), 0)

median\_blur = cv2.medianBlur(img, 5)

bilateral\_blur = cv2.bilateralFilter(img, 9, 75, 75)

## Edge and Feature Detection

### 1. Detecting Corners

Using the Harris Corner Detection method, we highlighted corners in an image.

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

corners = cv2.cornerHarris(gray, 2, 3, 0.04)

### 2. Detecting Circles

We used the Hough Circle Transform method to detect circles in an image.

detected\_circles = cv2.HoughCircles(gray, cv2.HOUGH\_GRADIENT, 1, 20, param1=50, param2=30, minRadius=1, maxRadius=40)

## Histogram Analysis

We analyzed the image histogram to understand the pixel intensity distribution.

histogram = cv2.calcHist([img], [0], None, [256], [0, 256])

## Video Processing

### 1. Playing a Video

We loaded and displayed a video frame by frame using OpenCV.

cap = cv2.VideoCapture("video.mp4")

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

cv2.imshow("Video", frame)

if cv2.waitKey(25) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

### 2. Face and Smile Detection

We implemented face and smile detection using Haar cascades.

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

smile\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_smile.xml')

## Conclusion

This project explored various image processing techniques using OpenCV, covering image manipulation, transformations, feature detection, and video processing. These techniques are fundamental in computer vision applications like object detection and facial recognition.