

INDIVIDUAL REPORT 1

AI & VISION SYSTEM LABORATORY



January 20, 2024

NAGALAXMAN G

21R221

**https://drive.google.com/drive/folders/1enQi0Y66fRYw0nP0BEw4rJx6ud4cO1FA**

**Question No. 13**

Implement masking concept (masking persons face) in the video frames.

**Software Packages Used**

1. Pycharm IDE

2. Libraries used:

* opencv-python
* numpy

**IMAGE MASKING :**

* OpenCV Image Masking is a powerful for manipulating images. It allows you to apply effects to a single image and create an entirely new look. With OpenCV Image Masking, you can selectively modify colors, contrast, lighten or darken, add or remove noise, and even erase parts or objects from an image. For example, let’s say that we were building a computer vision system to recognize faces. The only part of the image we are interested in finding and describing is the parts of the image that contain faces — we simply don’t care about the rest of the image’s content. Provided that we could find the faces in the image, we may construct a mask to show only the faces in the image. Another image masking application you’ll encounter is alpha blending and transparency (e.g., in this guide on Creating GIFs with OpenCV). When applying transparency to images with OpenCV, we need to tell OpenCV what parts of the image transparency should be applied to versus not — masks allow us to make that distinction.

**Program**

import cv2

# Load the pre-trained face cascade classifier

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

# Load the mask image

mask = cv2.imread('img.png', cv2.IMREAD\_UNCHANGED)

# Create a VideoCapture object

cap = cv2.VideoCapture(0) # Use 0 for the default camera

while True:

# Read a frame from the video capture

ret, frame = cap.read()

# Convert the frame to grayscale for face detection

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

# Detect faces in the frame

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5, minSize=(30, 30))

# Iterate over detected faces and apply the mask

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

# Resize the mask to fit the detected face

resized\_mask = cv2.resize(mask, (w, h), interpolation=cv2.INTER\_AREA)

# Extract the alpha channel from the mask (transparency)

alpha\_mask = resized\_mask[:, :, 3] / 255.0

# Resize the face region to match the mask dimensions

face\_region = frame[y:y + h, x:x + w]

# Blend the face region with the masked region

for c in range(0, 3):

face\_region[:, :, c] = face\_region[:, :, c] \* (1 - alpha\_mask) + resized\_mask[:, :, c] \* alpha\_mask

# Display the frame with the face mask

cv2.imshow('Face Masking', frame)

# Break the loop when 'q' key is pressed

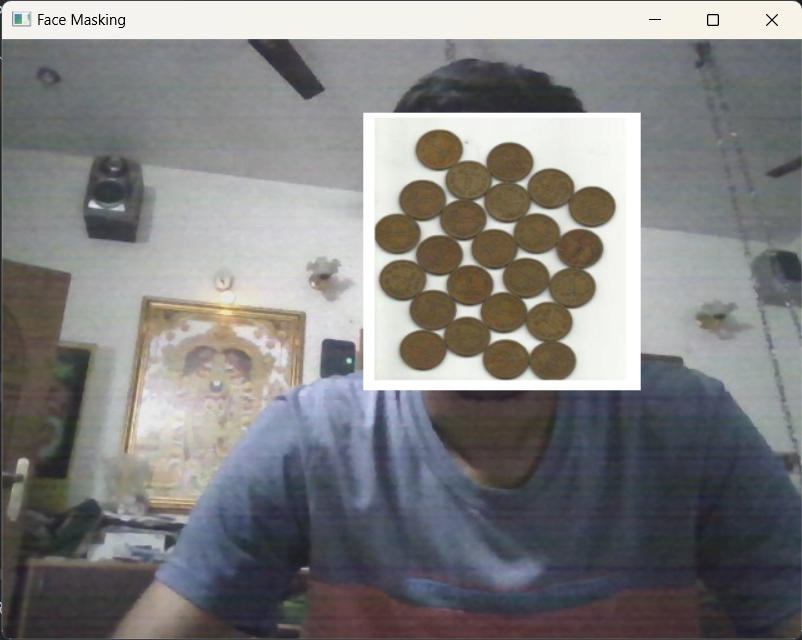
if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the VideoCapture and close all windows

cap.release()

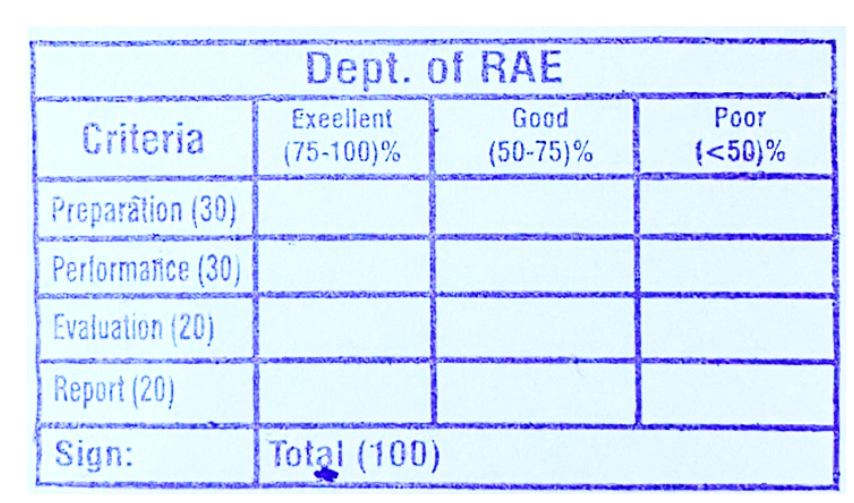
cv2.destroyAllWindows()

**Output :**

**Explanation**

Certainly! Here's an explanation of image masking vedio in points without code:

1. **Haarcascades Classifier:**
   * The code uses the Haarcascades classifier for face detection, a pre-trained model specifically designed for detecting faces in images.
2. **Mask Image:**
   * A mask image is loaded, and it should be in PNG format with an alpha channel for transparency. This image will be applied to the detected faces.
3. **Video Capture:**
   * The script captures video frames from the default camera using the VideoCapture object from OpenCV.
4. **Frame Processing:**
   * Each frame is converted to grayscale to enhance face detection performance.
5. **Face Detection:**
   * The Haarcascades classifier is applied to the grayscale frame to detect faces. Detected faces are represented as rectangles.
6. **Face Masking:**
   * For each detected face, the script resizes the loaded mask image to match the dimensions of the face.
   * It extracts the alpha channel (transparency) from the mask image.
   * The original face region is blended with the resized mask region using the alpha channel for transparency.
7. **Real-time Display:**
   * The script continuously displays the video frames with the applied face masks in real-time.
8. **User Interaction:**
   * The loop continues until the user presses the 'q' key, at which point the program exits.
9. **Resource Release:**
   * Upon exiting the loop, the script releases the resources associated with the video capture and closes all OpenCV windows.



**RESULT**

Thus the implementation of image masking vedio recognition was done using OpenCV in python.