Project Documentation

Project Title:- Face Recognition System

Abstract:-

This project aims to develop a robust face recognition system using Python, leveraging libraries such as OpenCV for detecting multiple faces in both images and video streams. The system is designed to implement recognition of multiple faces concurrently in real-time video feeds. Through the integration of computer vision techniques, the software provides functionalities for accurately detecting and recognizing faces in various scenarios. By harnessing the capabilities of OpenCV and other supporting libraries, the system offers a flexible and efficient solution for face recognition tasks, facilitating applications in surveillance, security, and human-computer interaction domains.

MOTIVATION:-

In today's digital age, face recognition technology plays a crucial role in various domains, ranging from security and surveillance to personalized user experiences. Developing a face recognition system in Python using libraries such as OpenCV offers several compelling motivations:

1. Enhanced Security: Face recognition systems provide a robust means of enhancing security measures by accurately identifying individuals in images and videos.

2. Surveillance Applications: Face recognition systems enable efficient monitoring and surveillance in public spaces, airports, and other high-traffic areas. By detecting multiple faces in real-time video streams, the system can aid law enforcement agencies and security personnel in identifying and tracking individuals of interest.

3. Improved User Experience: Integrating face recognition capabilities into applications can enhance user experiences by offering personalized services and features. From unlocking devices to customizing content recommendations, face recognition technology enables seamless interaction with digital platforms.

4. Automation and Efficiency: Automating face detection and recognition processes using Python libraries such as OpenCV can streamline workflows and improve operational efficiency. Whether in retail environments for customer analytics or in industrial settings for employee tracking, automated face recognition systems offer time-saving benefits.

5. Research and Innovation: Developing a face recognition system presents an opportunity for research and innovation in the field of computer vision. By leveraging cutting-edge algorithms and techniques, developers can explore novel approaches to improve the accuracy and performance of face recognition systems.

SOFTWARE REQUIRED:-

1. Python: Ensure Python is installed on your system. The code is written in Python and requires Python interpreter to execute.

2. OpenCV: Install OpenCV library, a popular computer vision library, to perform face detection and image processing tasks. You can install OpenCV using pip:

pip install opencv-python

3. RetinaFace: Install RetinaFace library, which is used for face detection. You can install RetinaFace using pip:

pip install retinaface

4. PIL (Python Imaging Library): Install PIL library for image processing tasks. You can install PIL using pip:

pip install pillow

5. Transformers: Install Transformers library, which may be used for additional processing tasks. You can install Transformers using pip:

pip install transformers

6. tqdm: Install tqdm library for displaying progress bars during processing tasks. You can install tqdm using pip:

pip install tqdm

Once all the above software requirements are met, you can run the provided code for developing a face recognition system in Python using OpenCV to detect multiple faces in images and videos, and implement recognition of multiple faces in video streams.

IMPLEMENTATION (Coding Part):-

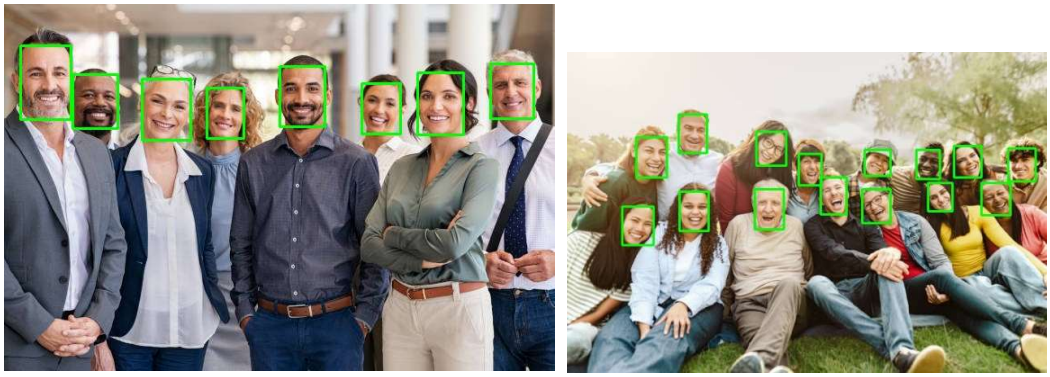
1. **Process on folder:**

```
import os
import cv2
from PIL import Image
from retinaface import RetinaFace
from transformers import pipeline
from tqdm import tqdm
import time
def detect_image(input_folder, output_folder):
    # Create the output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
```

```
# Initialize tqdm to show progress bar
image_files = [f for f in os.listdir(input_folder) if f.endswith(('.png', '.jpg', '.jpeg'))]
progress_bar = tqdm(total=len(image_files), desc="Processing")

# Iterate over the images in the input folder
for image_file in image_files:
    start_time = time.time()
    # Path to input image
    image_path = os.path.join(input_folder, image_file)

    # Detect faces using RetinaFace
    resp = RetinaFace.detect_faces(image_path)

    # Load the image
    img = cv2.imread(image_path)

    # Iterate over detected faces
    for face_id, face_data in resp.items():
        # Extract facial area coordinates
        x1, y1, x2, y2 = face_data['facial_area']

        # Extract the face region
        face_img = img[y1:y2, x1:x2]

        # Convert the face image to PIL image format
        pil_image = Image.fromarray(cv2.cvtColor(face_img, cv2.COLOR_BGR2RGB))

        # Draw the bounding box on the image
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
```

```
        # Save the processed image to the output folder
        output_path = os.path.join(output_folder, f"processed_{image_file}")
        cv2.imwrite(output_path, img)

        # Update progress bar
        progress_bar.update(1)

        end_time = time.time()
        process_time = end_time - start_time
        print(f"Processing time for {image_file}: {process_time:.2f} seconds")

    # Close the progress bar
    progress_bar.close()

# Specify the input and output folders
input_folder = "input_folder"
output_folder = "output_frames"

# Detect image and save processed images
detect_image(input_folder, output_folder)
```

Output:



Process On Video:

```
import cv2
import os
from PIL import Image
from retinaface import RetinaFace
from transformers import pipeline
from tqdm import tqdm

def detect_video(input_video_path, output_video_path):
    # Load the emotion detection pipeline
    emotion_pipe = pipeline('image-classification',
model='Saravanan290702/facial_emotions_image_detection')

    # Create a VideoCapture object
    cap = cv2.VideoCapture(input_video_path)
    if not cap.isOpened():
        print("Error: Could not open the video file.")
        return

    # Get the video properties
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    # Create a VideoWriter object
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width, frame_height))

    # Initialize tqdm to show progress bar
    progress_bar = tqdm(total=frame_count, desc="Processing")
```

```
# Process each frame of the video
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Detect faces using RetinaFace
    resp = RetinaFace.detect_faces(frame)

    # Iterate over detected faces
    for face_id, face_data in resp.items():
        # Extract facial area coordinates
        x1, y1, x2, y2 = face_data['facial_area']

        # Extract the face region
        face_img = frame[y1:y2, x1:x2]

        # Convert the face image to PIL image format
        pil_image = Image.fromarray(cv2.cvtColor(face_img, cv2.COLOR_BGR2RGB))

        # Draw the bounding box on the frame
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

    # Write the frame to the output video
    out.write(frame)

    # Update progress bar
    progress_bar.update(1)

# Release VideoCapture and VideoWriter objects
cap.release()
```

```
    out.release()

    # Close the progress bar
    progress_bar.close()

    print("Processing complete. Output video saved successfully.")

# Specify the input and output video paths
input_video_path = "1.mp4"
output_video_path = "output_video.avi"

# Detect emotions and draw bounding boxes on the input video frames, and output the processed video
detect_video(input_video_path, output_video_path)
```

Run On webcam:

```
import cv2
from PIL import Image
from retinaface import RetinaFace
from transformers import pipeline

def detect_webcam():

    # Create a VideoCapture object for webcam
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Could not open the webcam.")
        return

    # Get the webcam properties
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
```

```
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = 15  # Assuming a standard webcam captures at 30 frames per second

# Create a VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output_video.avi', fourcc, fps, (frame_width, frame_height))

# Process each frame from the webcam
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Detect faces using RetinaFace
    resp = RetinaFace.detect_faces(frame)

    # Iterate over detected faces
    for face_id, face_data in resp.items():
        # Extract facial area coordinates
        x1, y1, x2, y2 = face_data['facial_area']

        # Extract the face region
        face_img = frame[y1:y2, x1:x2]

        # Convert the face image to PIL image format
        pil_image = Image.fromarray(cv2.cvtColor(face_img, cv2.COLOR_BGR2RGB))

        # Draw the bounding box on the frame
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

    # Write the frame to the output video
```

```
        out.write(frame)

        # Display the frame
        cv2.imshow('Webcam Detection', frame)

        # Break the loop if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release VideoCapture and VideoWriter objects
    cap.release()
    out.release()
    cv2.destroyAllWindows()

    print("Processing complete. Output video saved successfully.")

# Call the function to start webcam detection
detect_webcam()
```

UTILITIES:-

"Unlock the Power of Faces:

Detect multiple faces in images and videos effortlessly.

Seamlessly recognize and track multiple faces in real-time video streams.

Enhance security, streamline surveillance, and personalize user experiences with cutting-edge face recognition technology.

Empower your applications with Python, OpenCV, and advanced computer vision capabilities."