Aim: To perform Face detection on Video

Objective: Performing face recognition Generating the data for face recognition Recognizing faces preparing the training data Loading the data and recognizing faces.

Theory:

Generating the Data for Face Recognition:
Generating data for face recognition typically involves capturing images or videos of individuals' faces. This can be done through cameras or video recording devices. It's essential to ensure good lighting conditions and a variety of facial expressions and poses to train a robust face recognition model. These images or video frames are used as the dataset for training and testing the face recognition system.

Recognizing Faces:
• Face recognition is the process of identifying and verifying individuals based on their facial features. It involves the following steps:

• Face Detection: Locate and isolate faces within an image or video frame. This can be done using methods like Haar Cascade face detection or deep learning-based techniques.

• Feature Extraction: Extract facial features from the detected faces. Common methods include Eigenfaces, Local Binary Patterns (LBP), and deep learning based feature extraction using Convolutional Neural Networks (CNNs).

• Face Representation: Represent the extracted features as a feature vector or set of descriptors. These representations capture the unique characteristics of each face.

• Training: Train a face recognition model using a labeled dataset. This model learns to map the feature vectors to the corresponding individuals' identities.

• Recognition: In the recognition phase, the model is used to identify individuals in new images or video frames. The input features are compared to the learned representations, and the model outputs the most likely identity or a confidence score.

Preparing the Training Data:
Preparing training data for face recognition involves organizing the captured images or video frames into a labeled dataset. Each image or frame should be associated with the correct identity of the person in the frame. This dataset is used to train the face recognition model. It's crucial to have a diverse dataset that includes various individuals, poses, lighting conditions, and expressions to ensure the model's robustness.

• Loading the Data and Recognizing Faces:

• To recognize faces in videos, you need to follow these steps:

• Data Loading: Load the video frames or images into your program or application.

• Face Detection: Apply a face detection algorithm to locate faces in each frame. This can be a single image or a sequence of frames in a video.

• Feature Extraction: Extract facial features from the detected faces.

• Face Representation: Represent the features as a feature vector for each face.
• Model Loading: Load the pre-trained face recognition model that has been trained on your labeled dataset.

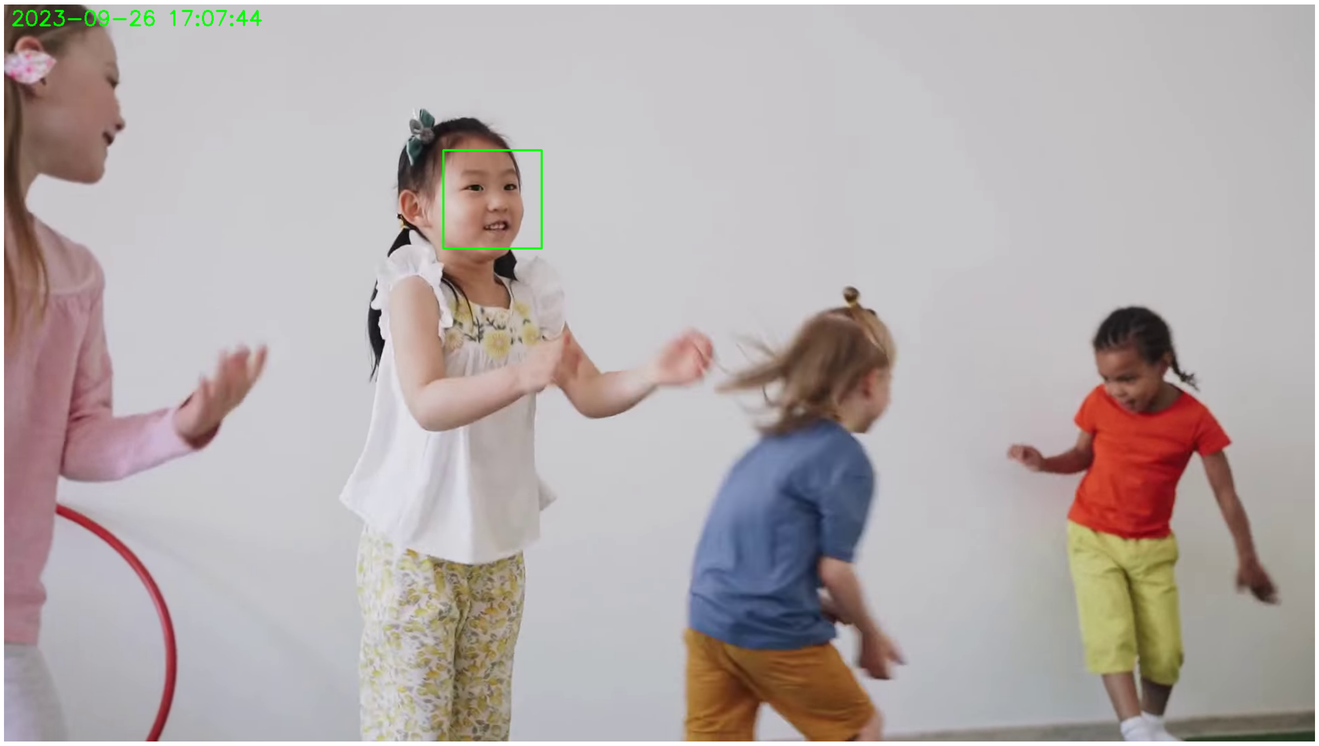
• Recognition: Use the loaded model to recognize faces by comparing the feature vectors of the detected faces to the known representations in your dataset.

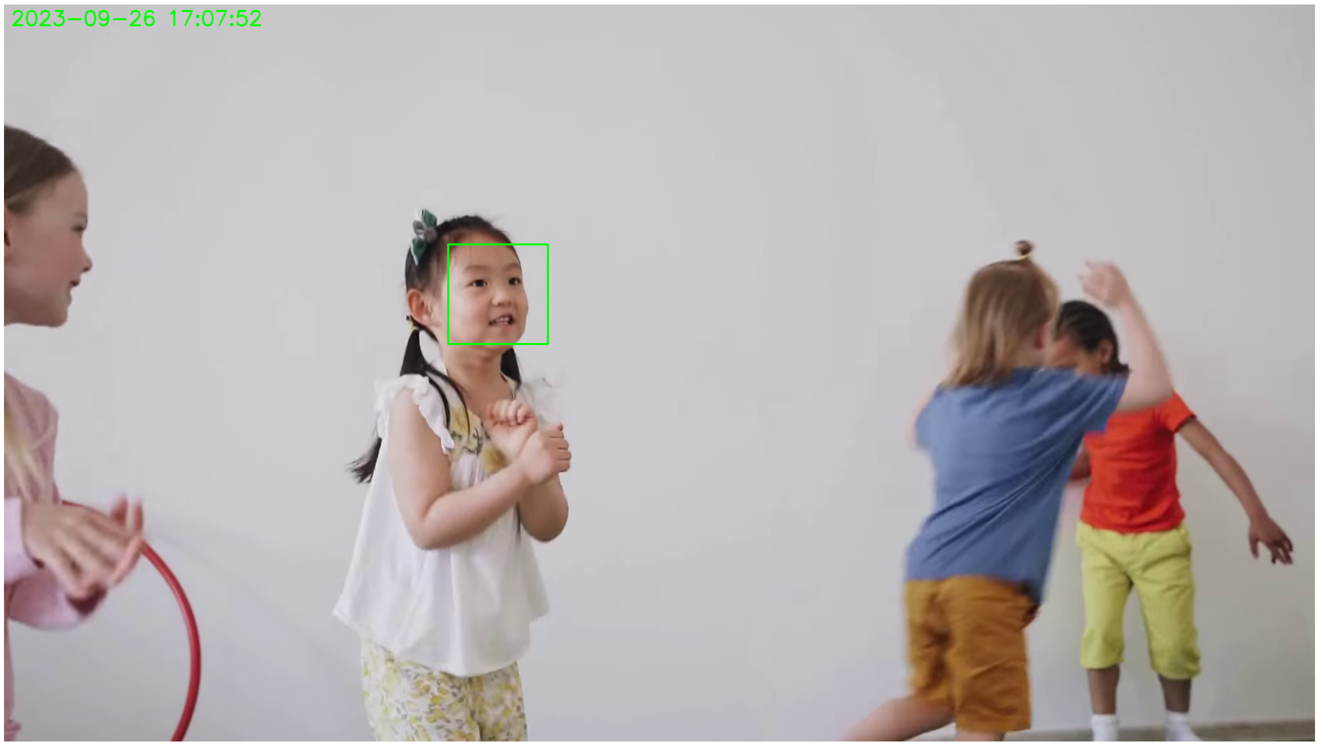
• Output: Display the recognized faces, along with their identities or confidence scores, in the video or images.

Code:

```
import cv2
import datetime
from google.colab.patches import cv2_imshow

# Load the pre-trained face detection cascade
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Open the WebM video file (replace 'video.webm' with your WebM video file)
video_path = '/content/child.webm'
cap = cv2.VideoCapture(video_path)

while True:
    ret, frame = cap.read()

    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5,
minSize=(30, 30))

    # Get the current timestamp
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Add timestamp text to the frame
    cv2.putText(frame, timestamp, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 0), 2)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv2_imshow(frame)  # Display the frame with detected faces

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

cap.release()
cv2.destroyAllWindows()
```

Output:





Conclusion:

We created a Python script that uses OpenCV to process a WebM video, detect faces, and overlay timestamps on the frames. The following are the script's main highlights:

The OpenCV library is used to open and read a WebM video file.

A face detection cascade is used to identify and highlight faces in each frame.

Adding a timestamp to the top-left corner of each frame in the format "YYYY-MM-DD HH:MM:SS".

Displaying frames with detected faces and timestamps and allowing for font size, position, and color customization.

Limiting the output to only show the first and last frames of the video, allowing for greater flexibility in specific use cases.