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:

Face recognition data is typically produced by assembling a collection of facial images from various sources, such as images from photos or frames from videos. This dataset should contain a wide diversity of persons, expressions, lighting conditions, and backgrounds to ensure reliable model training. Rotation, scaling, and noise addition are examples of data augmentation techniques that can be used to increase dataset diversity and improve model generalization.

Recognizing faces:

Face recognition is an automated technique for identifying and classifying human faces in still images or moving images. In order to categorize people based on characteristics like age, gender, or emotions or to match them with known individuals, this is done by looking at face features such the eyes, nose, and mouth. Privacy and ethical issues are raised by the use of facial recognition technology for security, authentication, and personalization.

Preparing the training data:

Data preparation is the act of gathering the information you need and putting it into a form that is comprehensible to computers. Accurate data collection is the initial step in every AI or machine learning project, and it is frequently more challenging and time-consuming than developing the machine learning algorithms themselves.

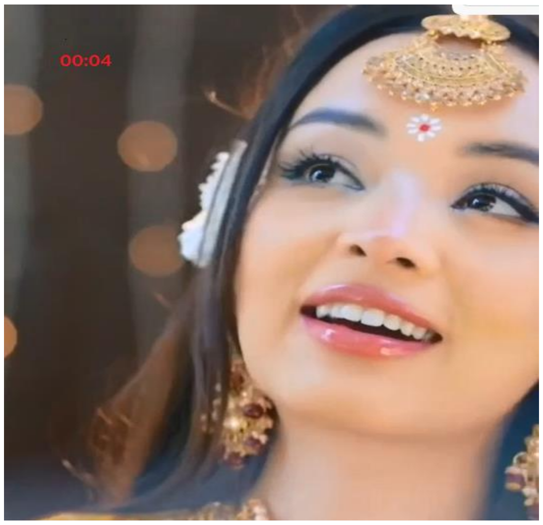
Loading the data and recognizing faces:

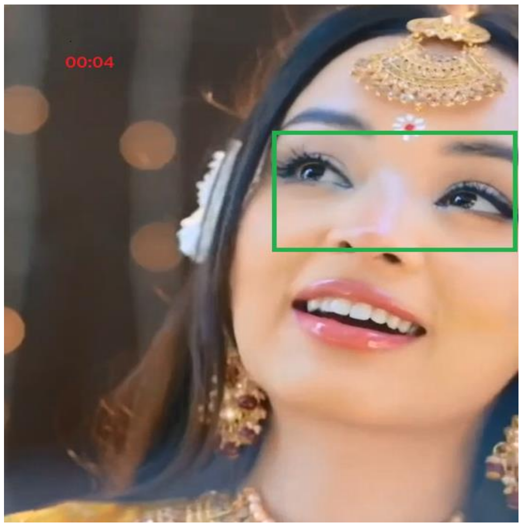
The steps involved in gathering and preprocessing picture or video data, then employing facial detection and recognition algorithms, are known as "loading the data" and "recognizing faces." Accurate face recognition can be achieved using a variety of deep learning techniques. Usual method for loading datasets is.CSV files that can display a broad range of input photos in a dataset.

Code:

```
import cv2
import datetime
from google.colab.patches import cv2_imshow
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
video_path='/content/sample_data/v.mp4'
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))
timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
cv2.putText(frame, timestamp, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7,(0, 0, 255), 2)
for (x, y, w, h) in faces:
cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
cv2_imshow(frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
break
```

Output :





Conclusion :

The simplest technique for object detection is Haar Cascade. The Haar cascade technique is capable of real-time object detection in videos, regardless of the objects' size, location, or frame rate. Every window's features are computed in an attempt to determine whether or not it could be an object.We successfully used the Haar Cascade approach for object detection in video in this experiment.