**Name :** Akshat Sharma

**Aim :** Viola Jones Algorithm

**Theory:** The Viola-Jones algorithm is a widely used real-time object detection framework, primarily known for face detection. Proposed by Paul Viola and Michael Jones in 2001, it is one of the first algorithms capable of detecting faces in images or videos in real time. It combines Haar-like features, Integral Image, Adaboost, and a Cascade Classifier to achieve high efficiency and accuracy.

## Key Components of the Viola-Jones Algorithm

**Haar-like Features:**

- Haar features are simple rectangular features used to detect patterns in images.
- These features work by computing the difference between pixel intensities in adjacent rectangular regions.

**Integral Image:**

- A technique that allows rapid computation of Haar-like features.
- It enables fast summation of pixel values in any rectangular region, significantly reducing computation time.

**Adaboost (Adaptive Boosting):**

- A machine learning technique that selects the most important Haar-like features from a large set.
- It assigns different weights to weak classifiers, combining them to form a strong classifier.

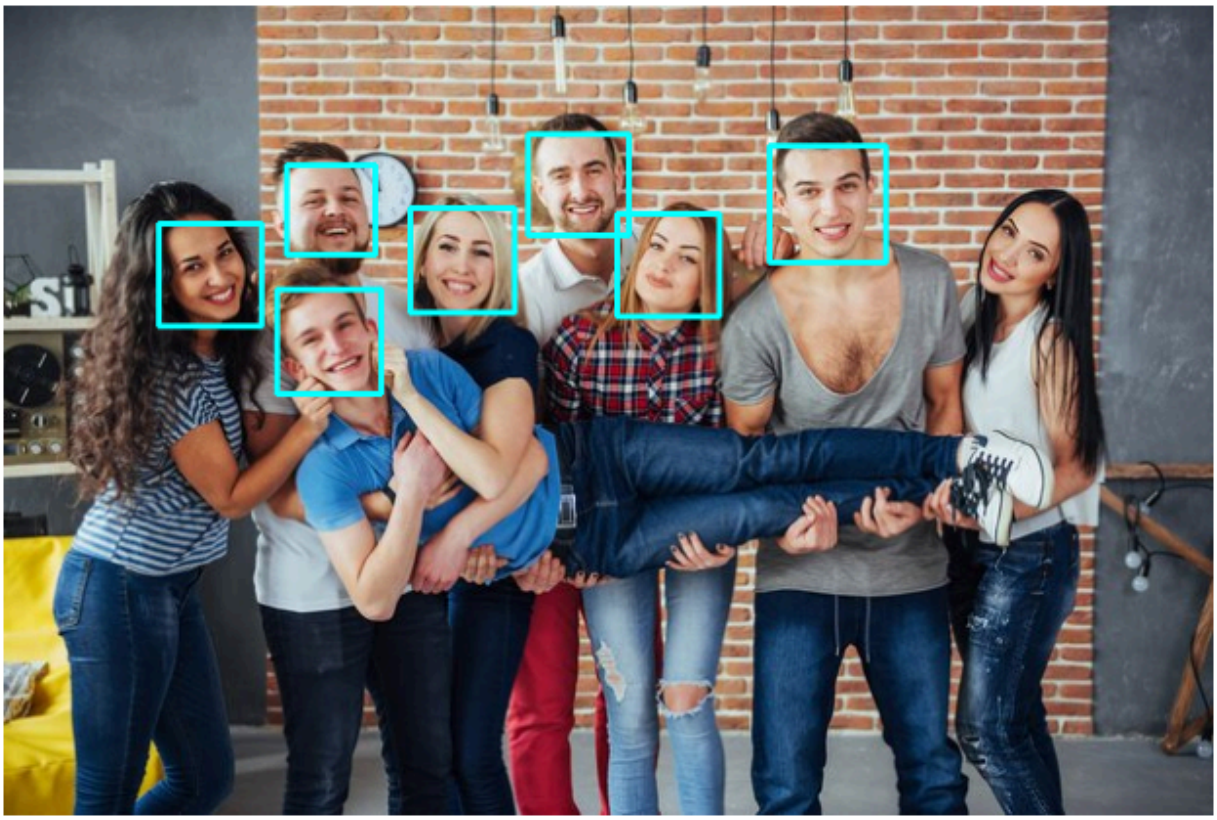
**Cascade Classifier:**

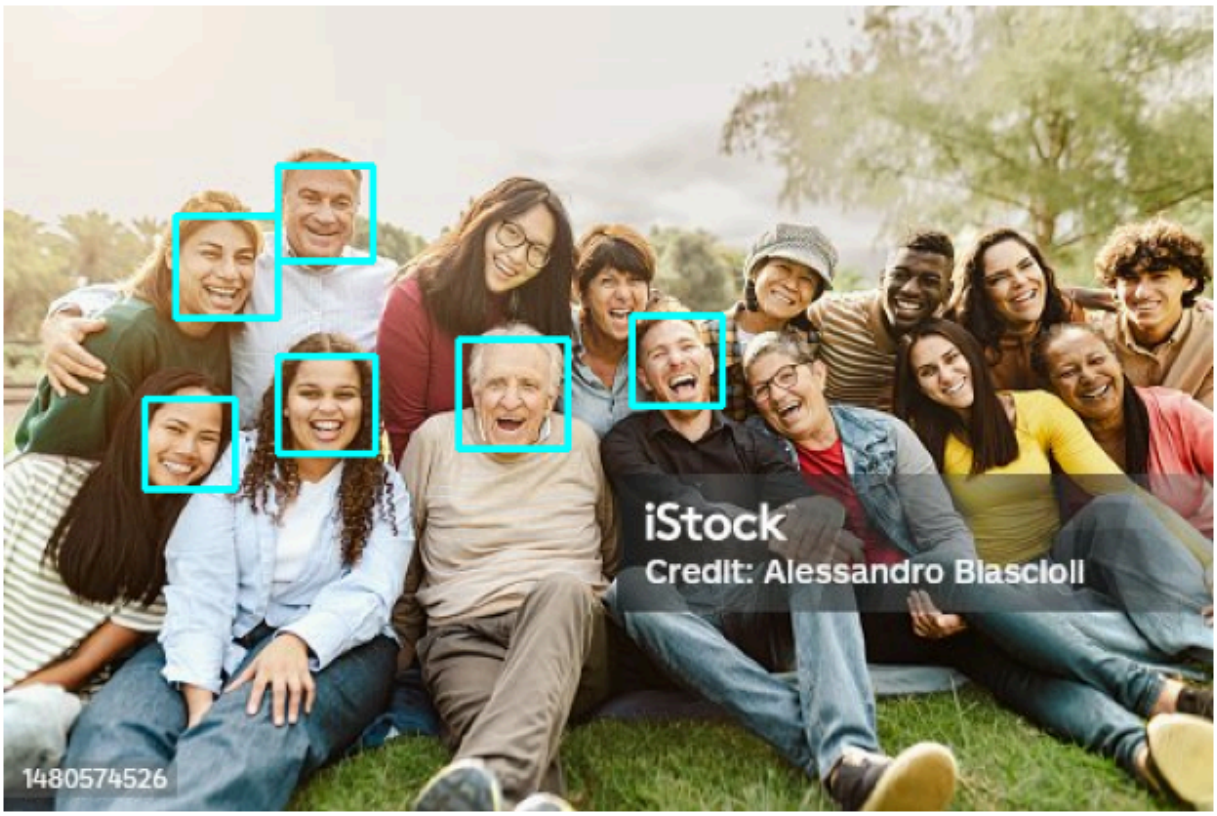
- A multi-stage process where the classifier quickly eliminates non-face regions and focuses computation on potential face regions.
- The cascade structure ensures that detection remains **fast and efficient** by discarding negative samples early**.**

## CODE: FOR IMAGE

```
import cv2
import matplotlib.pyplot as plt
face_cascade =
cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
image_path = "Group.jpg"
image = cv2.imread(cv2.samples.findFile(image_path))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray,
scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
for (x, y, w, h) in faces:
 cv2.rectangle(image, (x, y), (x + w, y + h), (255,
255, 5), 2)
plt.figure(figsize=(8, 6))
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

## OUTPUT:



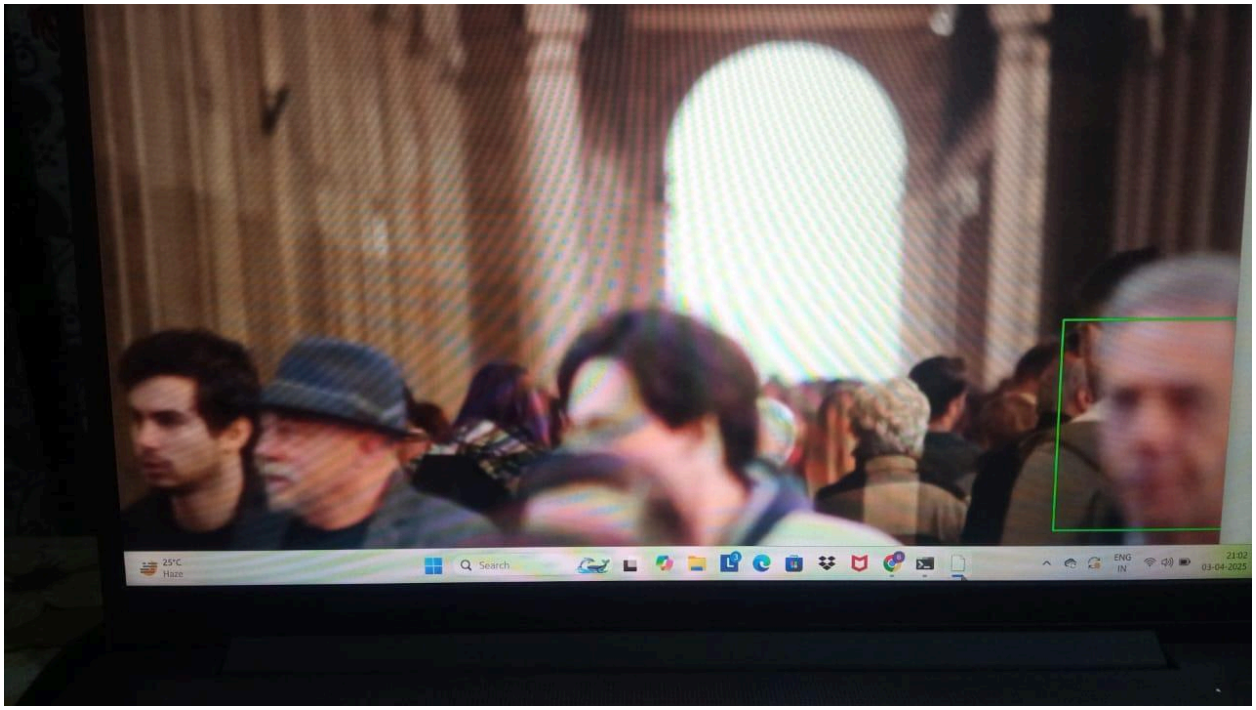

iStock
Credit: Alessandro Biascioli
1480574526

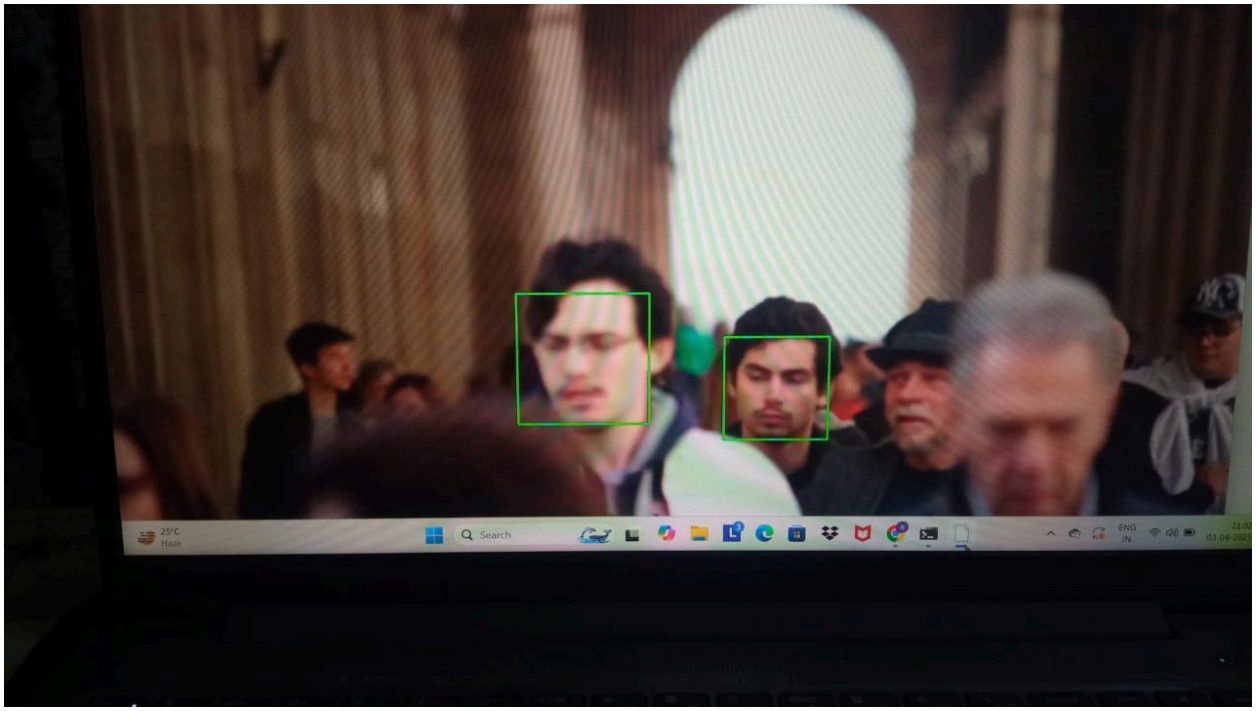
**CODE : FOR VIDEO**

```
import cv2
import os

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

# Open the video file
video_capture = cv2.VideoCapture('Crowd.mp4')

# Get video properties
frame_width = int(video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(video_capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(video_capture.get(cv2.CAP_PROP_FPS))

# Check if video opened successfully
if not video_capture.isOpened():
    print("Error: Could not open video file.")
    exit()

# Define the codec and create VideoWriter object
output_path = os.path.expanduser('~/output_video.mp4')
output_video = cv2.VideoWriter(output_path,
cv2.VideoWriter_fourcc(*'mp4v'), fps, (frame_width, frame_height))

while True:
    # Read a frame from the video
    ret, frame = video_capture.read()
    if not ret:
        break

    # Convert frame to grayscale for better accuracy
    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))

    # Draw rectangles around detected faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Write the processed frame to the output video
    output_video.write(frame)

    # Display the video with detected faces
    cv2.imshow('Face Detection', frame)

    # Press 'q' to exit the video loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
video_capture.release()
output_video.release()
cv2.destroyAllWindows()
print(f"Output video saved at: {output_path}")
```

**OUTPUT :**





## Conclusion:

- **Efficient and Real-Time:** The algorithm is one of the fastest and most efficient object detection methods, making it suitable for real-time applications.

- **Robust Feature Selection:** By using Adaboost, the model selects only the most relevant features, reducing computational complexity.

- **Lightweight and Computationally Feasible:** Unlike deep learning methods that require high computational power, the Viola-Jones algorithm is efficient on lower-end hardware.

- **Limited to Rigid Objects:** It works best for frontal face detection but struggles with variations like **tilted faces, occlusions, and non-frontal views**.

- **Outdated by Modern Methods:** While it was a breakthrough in early 2000s, modern deep learning-based approaches such as **CNNs (e.g., Faster R-CNN, SSD, YOLO)** have surpassed it in accuracy and robustness.

Overall, the **Viola-Jones algorithm** remains a **foundational technique** in object detection, particularly for **real-time face detection**, but modern deep learning models now provide **more accurate and versatile alternatives**.