Vidyavardhini's College of Engineering & Technology
Department of Computer Engineering

Name – Divya Patil

Roll no - 08

Experiment No: 06

Aim: To study Detecting and Recognizing Faces

Objective: To Conceptualize Haar Cascades Getting Haar cascade data. Using Opencv to Perform face detections performing face detection on still images.

Theory:

1. Conceptualizing Haar Cascades:

Conceptualizing Haar Cascades refers to a method of object detection that uses a cascade of simple Haar-like features to efficiently identify objects within images or video streams. These cascades are trained on positive and negative samples to distinguish between the object of interest and background, making them suitable for real-time applications such as face detection in cameras and security systems. Haar Cascades are known for their speed and accuracy, making them a popular choice in the field of computer vision.

2. Getting Haar Cascade Data:

Getting Haar Cascade data involves obtaining pre-trained Haar Cascade classifiers, which are used for object detection. These classifiers are based on Haar-like features and can recognize specific objects or patterns. To obtain Haar Cascade data, you typically download or create these classifiers and then use them with a computer vision library like OpenCV for tasks such as face detection or object recognition.

3. Using OpenCV to perform Face Detection:

Using OpenCV for face detection involves utilizing the OpenCV library's pre-trained deep learning models, such as Haar cascades or deep neural networks like Single Shot MultiBox Detector (SSD) or Faster R-CNN, to identify and locate faces within images or video streams. OpenCV provides a convenient and efficient framework to apply these models, making it a popular choice for various face detection applications, including facial recognition, emotion analysis, and more.

4. Performing Face detection on a still image:

Performing face detection on a still image is a computer vision technique that involves identifying and locating human faces within the image. This process typically employs algorithms and models to detect facial features like eyes, nose, and mouth, and it is a fundamental step in applications such as facial recognition, emotion analysis, and image indexing. The output of face detection is usually a bounding box or a list of coordinates that delineate the detected faces within the image, enabling further analysis or manipulation.

Introduction

Discover object detection with the Haar Cascade algorithm using OpenCV. Learn how to employ this classic method for detecting objects in images and videos. Explore the underlying principles, step-by-step implementation, and real-world applications. From facial recognition to vehicle detection, grasp the essence of Haar Cascade and OpenCV's role in revolutionizing computer vision. Whether you're a novice or an expert, this article will equip you with the skills to harness the potential of object detection in your projects.

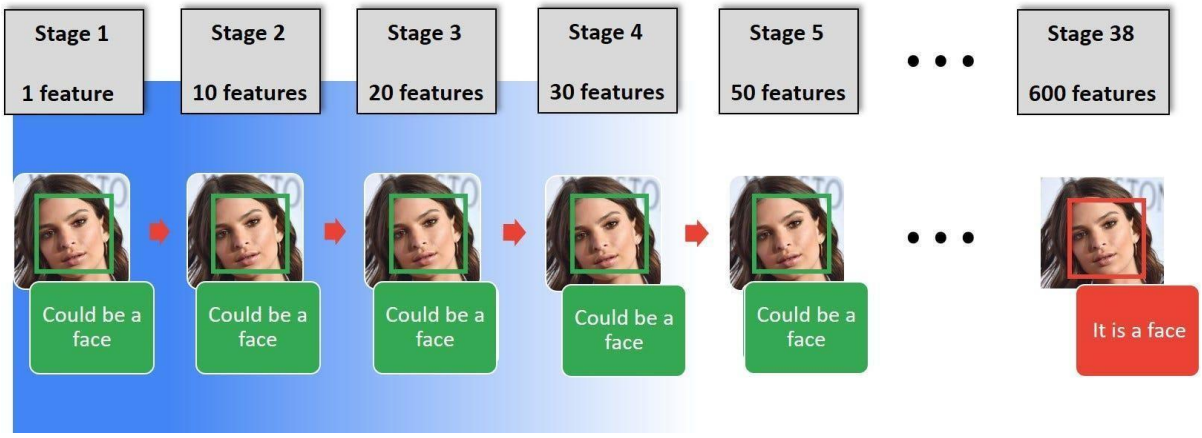


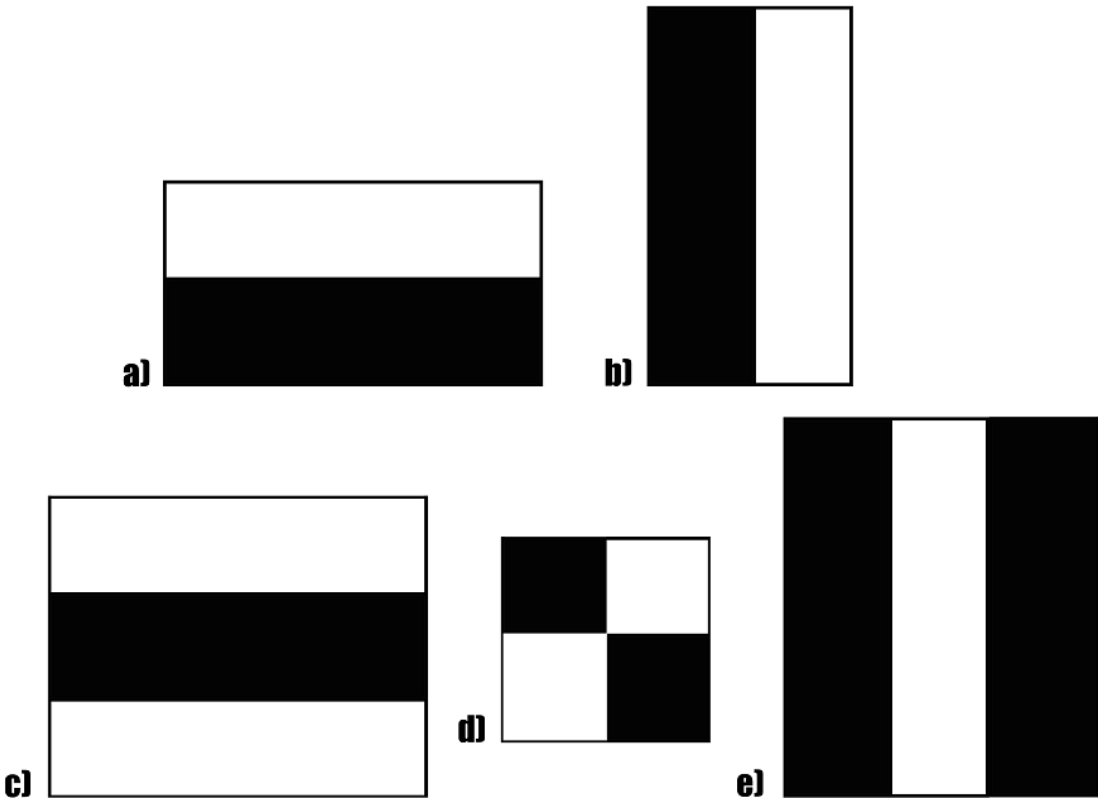
Table of contents:

Identifying a custom object in an image is known as object detection. This task can be done using several techniques, but we will use the haar cascade, the simplest method to perform object detection in this article.

Haar cascade is an algorithm that can detect objects in images, irrespective of their scale in image and location.

This algorithm is not so complex and can run in real-time. We can train a haar-cascade detector to detect various objects like cars, bikes, buildings, fruits, etc.

Haar cascade uses the cascading window, and it tries to compute features in every window and classify whether it could be an object.



Haar cascade works as a classifier. It classifies positive data points → that are part of our detected object and negative data points → that don't contain our object.

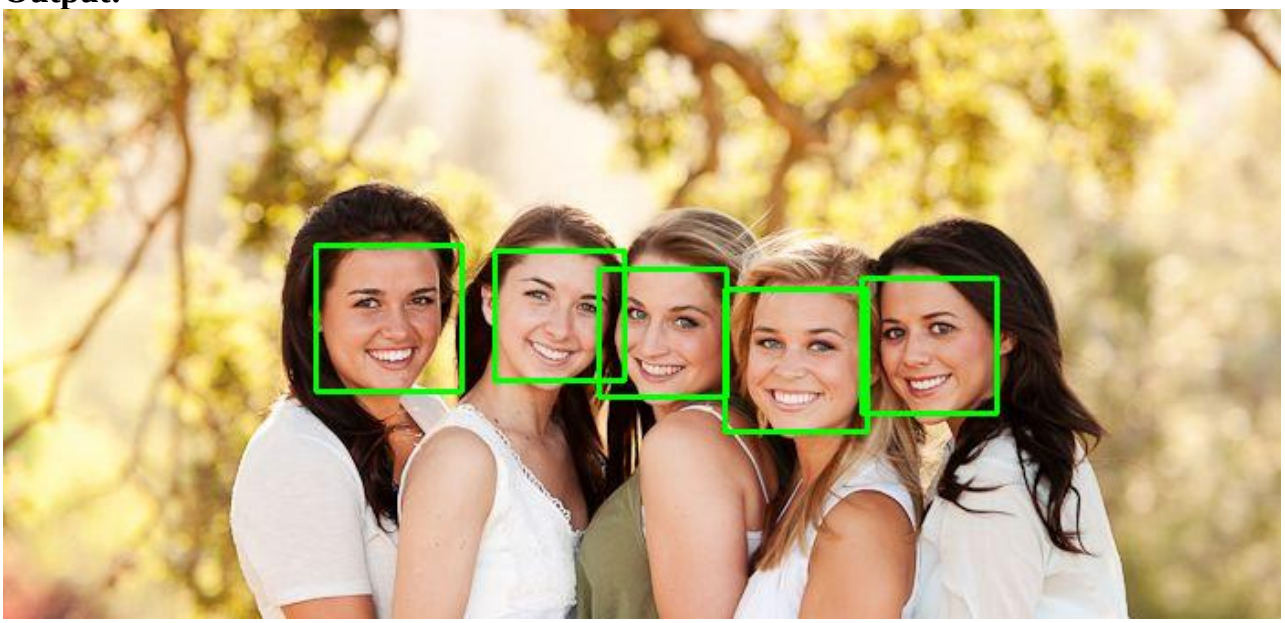
- Haar cascades are fast and can work well in real-time.
- Haar cascade is not as accurate as modern object detection techniques are.
- Haar cascade has a downside. It predicts many false positives.
- Simple to implement, less computing power required.

Code and Output:

```
pip install opencv-python
import cv2
imagePath = 'face.jpg'
img = cv2.imread(imagePath)
img.shape
->(275, 183, 3)

gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray_image.shape
->(275, 183)

face_classifier = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
)
face = face_classifier.detectMultiScale(
    gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(40, 40)
)
for (x, y, w, h) in face:
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 4)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)\

import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
plt.imshow(img_rgb)
plt.axis('off')
```

Input:

Output:



Conclusion :

Performing face detection on static images encompasses the utilization of Haar cascades, specialized data structures tailored for object recognition, primarily oriented towards identifying faces. Acquiring Haar cascade data entails the assembly of datasets containing positive and negative samples, followed by the training and customization of the cascade classifier for specific detection objectives. OpenCV, a widely adopted computer vision library, offers comprehensive resources and tools for the implementation of Haar cascade-based face detection, streamlining the development process