Aim: To study Detecting and Recognizing Faces

Objective: To Conceptualizing Haar Cascades Getting Haar cascade data Using Open CV to Perform face detections performing face detection on still images

Theory:

Conceptualizing Haar Cascades :

The haar calculation is done by finding out the difference of the average of the pixel values at the darker region and the average of the pixel values at the lighter region. If the difference is close to 1, then there is an edge detected by the haar feature.

Getting Haar Cascade Data:

The objective here is to find out the sum of all the image pixels lying in the darker area of the haar feature and the sum of all the image pixels lying in the lighter area of the haar feature. And then find out their differences. Now if the image has an edge separating dark pixels on the right and light pixels on the left, then the haar value will be closer to 1. That means, we say that there is an edge detected if the haar value is closer to 1. In the example above, there is no edge as the haar value is far from 1.

Using Open CV to perform Face Detection:

- Install OpenCV:
- Import OpenCV:
- Load the Haar Cascade Classifier:
- Load an Image or Capture from Webcam: ● Perform Face Detection:
- Draw Rectangles Around Detected Faces:
- Display the Result:

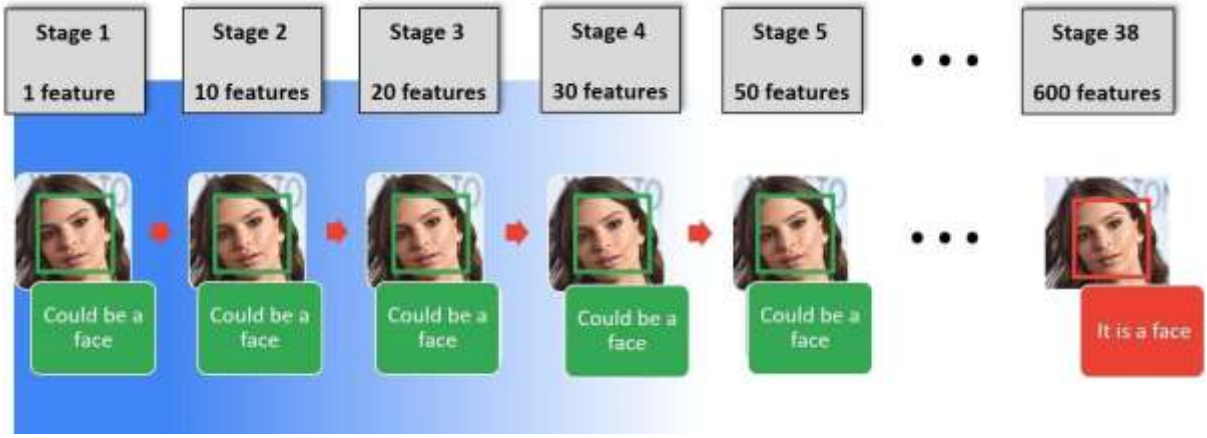
Performing Face detection on a still image:

- Import the OpenCV Package
- Read the Image

- Convert the Image to Grayscale
- Load the Classifier
- Perform the Face Detection
- Drawing a Bounding Box
- Displaying the Image

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.



Why Use Haar Cascade Algorithm for Object Detection?

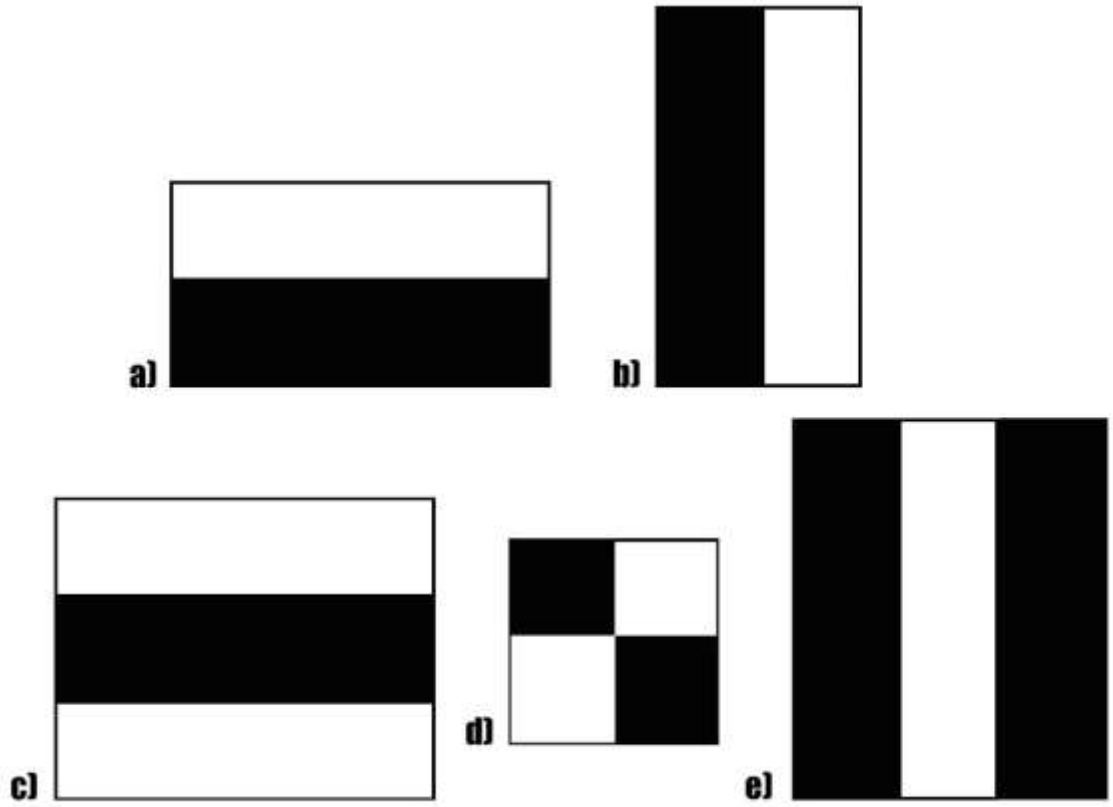
Identifying a custom object in an image is known as object detection. This task can be done using several techniques, but we will use the haar cascade, the simplest method to perform object detection in this article.

What is Haar Cascade Algorithm?

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:

```
import cv2
import numpy as np
from google.colab import files
from io import BytesIO
from IPython.display import display, Image
import PIL

# Function to perform face detection and display the result
def detect_faces(image_path):
    # Load the pre-trained face detection model from OpenCV
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    # Load the image from the provided path
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Perform face detection
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5,
minSize=(30, 30))

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

    # Display the result
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    pil_img = PIL.Image.fromarray(img_rgb)
    display(pil_img)

# Upload an image for face detection
```

```
uploaded = files.upload()

# Process the uploaded image
if len(uploaded) > 0:
    image_path = list(uploaded.keys())[0]
    detect_faces(image_path)
else:
    print("No image uploaded.")
```

Input & Output:

Conclusion:

In short, the Haar Cascade face recognition method is a popular and effective way to find faces in pictures and videos. To recognise facial traits, it uses a cascade of straightforward classifiers and pre-trained classifiers. Although it is quick and efficient for real-time applications, it might not be able to handle changes in pose, lighting, and occlusions. However, Haar Cascade face detection continues to be a useful tool for a variety of real-world applications, particularly when real-time efficiency is important. For the purpose of identifying and detecting faces, OpenCV is a flexible and effective toolkit. For academics and developers working in this field, its versatility, performance, and active community support make it a significant asset.