12 - Aditi Sawant

Aim: To study Detecting and Recognizing Faces.

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

Theory:

Conceptualizing Haar Cascades

Photographic images, even from a webcam, may contain a lot of detail for our (human) viewing pleasure. However, image detail tends to be unstable with respect to variations in lighting, viewing angle, viewing distance, camera shake, and digital noise. Moreover, even real differences in physical detail might not interest us for the purpose of classification. I was taught in school that no two snowflakes look alike under a microscope. Fortunately, as a Canadian child, I had already learned how to recognize snowflakes without a microscope, as the similarities are more obvious in bulk. Thus, some means of abstracting image detail is useful in producing stable classification and tracking results. The abstractions are called features, which are said to be extracted from the image data. There should be far fewer features than pixels, though any pixel might influence multiple features. The level of similarity between two images can be evaluated based on Euclidean distances between the images' corresponding features.

Getting Haar Cascade Data

Once you have a copy of the source code of OpenCV 3, you will find a folder, data/haarcascades. This folder contains all the XML files used by the OpenCV face detection engine to detect faces in still images, videos, and camera feeds. Once you find haarcascades, create a directory for your project, in this folder, create a subfolder called cascades, and copy the following files from haarcascades into cascades: haarcascade profileface.xml haarcascade righteye 2splits.xml haarcascade russian plate number.xml haarcascade smile.xml haarcascade_upperbody.xml As their names suggest, these cascades are for tracking faces, eyes, noses, and mouths. They require a frontal, upright view of the subject. We will use them later when building a face detector. If you are curious about how these data sets are generated, refer to Appendix B. Generating Haar Cascades for Custom Targets. OpenCV Computer Vision with Python. With a lot of patience and a powerful computer, you can make your own cascades and train them for various types of objects.

Using Open CV to perform Face Detection

Unlike what you may think from the outset, performing face detection on a still image or a video feed is an extremely similar operation. The latter is just the sequential version of the former. Face detection on videos is simply face detection applied to each frame read into the program from the camera. Naturally, a whole host of concepts are applied to video face detection such as tracking, which does not apply to still images, but it's always good to know that the underlying theory is the same

Performing Face detection on a still image

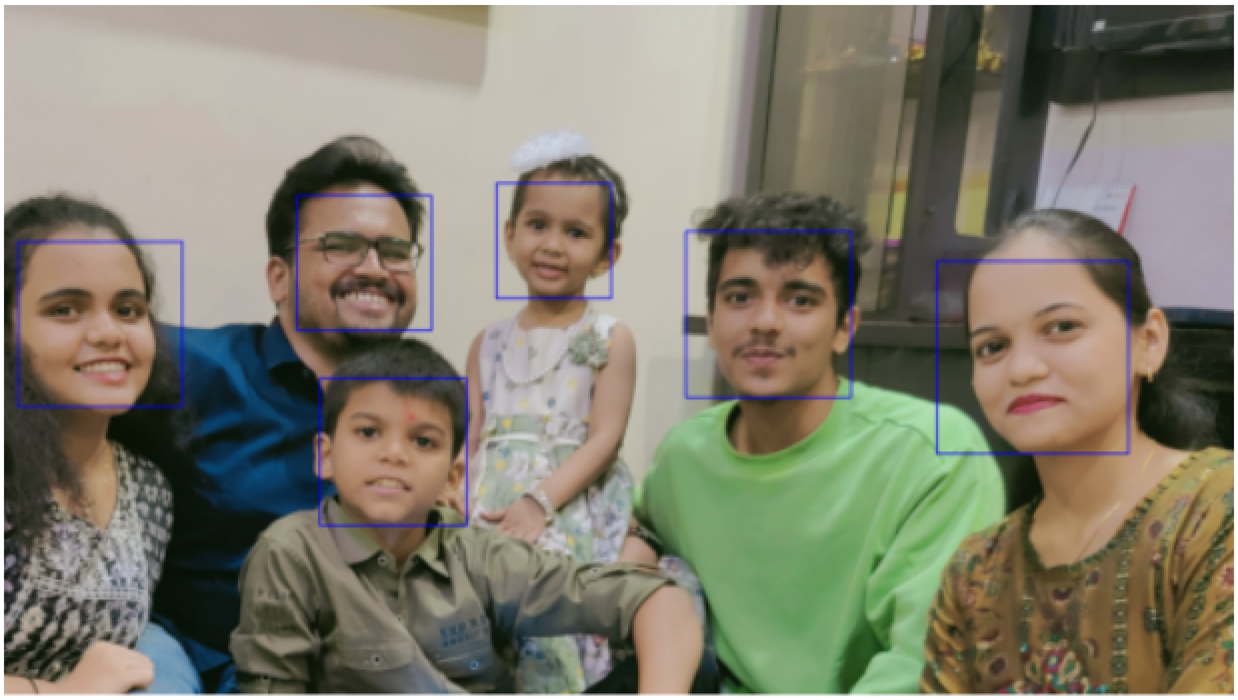
The first and most basic way to perform face detection is to load an image and detect faces init. To make the result visually meaningful, we will draw rectangles around faces on the original image. Now that you have haar cascades included in your project, let's go ahead and create a basic script to perform face detection.

Code:

```
import cv2
import matplotlib.pyplot as plt
# Load the pre-trained Haar cascades for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
# Load the image
img = cv2.imread('/content/IMG_20230926_222050.jpg')
# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30,
30))
```

```
# Draw rectangles around the faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
# Convert BGR image to RGB for displaying with matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Display the image with rectangles around faces
plt.imshow(img_rgb)
plt.axis('off')
plt.show()
```

Output:



Conclusion:

Haar cascades serve as effective tools to abstract image features, aiding in stable object recognition. Pre-trained cascade classifiers, like the one for facial detection, are included within OpenCV. By utilizing these cascades, one can detect faces in images with relative ease. Haar cascades are optimal for detecting frontal and upright faces, making them suitable for a range of applications. However, their effectiveness diminishes when faced with non-frontal or rotated facial orientations.