**FACE DETECTION**

**FACE DETECTION:**

Face detection is a computer vision technique used to automatically identify and locate human faces within images, videos, or live camera feeds. The primary goal of face detection is to determine the presence and position of faces within a given visual data source. It plays a fundamental role in various applications, including facial recognition, face tracking, emotion analysis, and human-computer interaction.

[**Face Detection with OpenCV**](https://github.com/Kirubasettu/FACE-DETECTOR#face-detection-with-opencv)**:**

image face detection with OpenCV is a fundamental computer vision task that involves identifying and locating human faces within a static image. OpenCV, an open-source computer vision library, provides a robust and versatile toolkit for performing this task efficiently.

[**Key Features and Capabilities:**](https://github.com/Kirubasettu/FACE-DETECTOR#key-features-and-capabilities)

***Haar Cascade Classifier***: OpenCV offers a Haar Cascade Classifier-based face detection method, which is a machine learning-based approach that excels at locating faces in static images. These pre-trained models have been meticulously trained on large datasets to recognize facial features.

***Accurate Face Detection***: OpenCV's face detection in image can handle various challenges, including variations in lighting, head orientations, and partial occlusions, to accurately locate faces.

***Customization***: OpenCV provides flexibility to fine-tune parameters and adapt the face detection model to specific image analysis requirements. This customization is especially valuable for addressing unique use cases and image characteristics.

***Python Integration***: OpenCV is commonly used with Python, providing an accessible and widely-adopted programming interface for developers to implement face detection on images.

***Open Source***: OpenCV is open-source software, making it freely available to the community for use and enhancement. Developers can contribute to its improvement and customizati

[**Applications:**](https://github.com/Kirubasettu/FACE-DETECTOR#applications)

* ***Photography and Image Editing***: OpenCV's image face detection is utilized in photography and image editing software to identify and enhance facial features, apply filters, or perform retouching.
* ***Auto-tagging and Organizing Photos***: Face detection assists in automatically tagging people in photos or organizing image collections based on individuals present in the images.
* ***Meme Creation***: Memes often involve adding humorous text or graphics to images. Face detection can be used to identify faces in the source image to determine where to add captions or elements.
* ***Image Annotation***: In research and data labeling tasks, face detection can be used to locate and annotate faces in images for further analysis or categorization.
* ***Forensic Analysis***: Face detection in single images is valuable for forensic investigations to identify and analyze individuals in photographs as part of criminal investigations.
* ***Visual Content Analysis***: Businesses and organizations use single image face detection for demographic analysis, customer behavior studies, and other visual content analysis purposes. Face detection with OpenCV serves as a foundational step for various image processing and analysis tasks. Its versatility, accuracy, and adaptability make it a valuable asset for developers and researchers working on projects that require the detection and localization of human faces within static images.

**Import Libraries**

import cv2

import matplotlib.pyplot as plt

**Read the Image and display shape**

image = cv2.imread("C:\\Users\\kirub\\OneDrive\\Pictures\\people1.jpg")

display (image.shape)

**Display the image with Matplotlib**

plt.imshow(image)

plt.show()

**Print Image Values – Pixel Values**

print(image)

**Display the image with CV2**

The image will be opened in a new window

The image will open and close quickly, it is necessary to add wait key

Once the wait is added the image will be opened till you press a key

waitKey() waits for a key press to close the window and 0 specifies indefinite loop . 1 means 1 milli second, 1000 milli second is 1 second.

cv2.imshow('', image)

cv2.waitKey(10000)

# cv2.destroyAllWindows() simply destroys all the windows we created.

cv2.destroyAllWindows()

**Resize the image**

**The image pixel size is 1280 X 1920. Since this is a**

**big image use resizes to reshape the image 800X600**

image = cv2.resize(image, (800, 600))

image.shape

**Display the image – small size image**

cv2.imshow('',image)

cv2.waitKey(10000)

cv2.destroyAllWindows()

**Gray scale the image – Convert to Back and White**

image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

cv2.imshow('',image\_gray)

cv2.waitKey(10000)

cv2.destroyAllWindows()

**Display the Size of the Grayscale image (Back and White)**

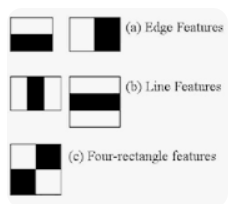
**Once dimension has reduced – No colour dimension**

display (image\_gray.shape)

Create the Model CascadeClassifier

**cascade classifier** is a machine learning-based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Cascading classifiers are **trained with several hundred "positive" sample views of a particular object and arbitrary "negative" images of the same size**. After the classifier is trained it can be applied to a region of an image and detect the object in question.



**haarcascade\_frontalface\_default. xml** : **Detects faces**.

haarcascade\_eye. xml Detects the left and right eyes on the face.

**Load Face Detector .xml file**

face\_detector = cv2.CascadeClassifier("C://Users/kirub//.conda//pkgs//opencv-4.6.0-py310ha36de5b\_5//Library/etc//haarcascades//haarcascade\_frontalface\_default.xml")

display (face\_detector)

**Display the Face detection co-ordinates**

Each row for each face detected -The values in each row have the co-ordinates such as x,y, width, height.

In this example, the algorithm detected 6 faces, that is 6 rows in the result.

detections = face\_detector.detectMultiScale(image\_gray)

display (detections)

**Display the length- Number of faces detected**

display (len(detections))

**Display each detected face with rectangle**

x=390 # X - Co ordinates

y=323 # Y- Co ordinates

w=56 # Face Width

h=56 # Face Height

# Draw the rectangle with co-ordinate specified above

# (0,255,255) - This is RBG colour for rectangle

# 1 is the thickness of rectangle

cv2.rectangle(image\_gray, (x, y), (x + w, y + h), (0,255,255), 1)

cv2.imshow('', image\_gray)

cv2.waitKey(10000)

cv2.destroyAllWindows()

**Display second detected face with rectangle**

x=115 # X - Co ordinates

y=124 # Y- Co ordinates

w=52 # Face Width

h=52 # Face Height

# Thickness is 5, thicker rectangle

cv2.rectangle(image\_gray, (x, y), (x + w, y + h), (0,255,255), 5)

cv2.imshow('', image\_gray)

cv2.waitKey(10000)

cv2.destroyAllWindows()

**Display third detected face with rectangle**

x=475 # X - Co ordinates

y=123 # Y- Co ordinates

w=59 # Face Width

h=59 # Face Height

cv2.rectangle(image\_gray, (x, y), (x + w, y + h), (0,255,255), 5)

cv2.imshow('', image\_gray)

cv2.waitKey(10000)

cv2.destroyAllWindows()

**Display all detected face with rectangle**

for (x, y, w, h) in detections:

#print (x, y, w, h)

cv2.rectangle(image\_gray, (x, y), (x + w, y + h), (0,255,255), 5)

cv2.imshow('', image\_gray)

cv2.waitKey(10000)

plt.show()

cv2.destroyAllWindows()

**Face Detection by Colour image**

image = cv2.imread("C:\\Users\\kirub\\OneDrive\\Pictures\\people1.jpg")

display (image.shape)

image = cv2.resize(image, (800, 600)) # Resize image

display (image.shape)

detections = face\_detector.detectMultiScale(image)

display (detections)

**Display Image**

for (x, y, w, h) in detections:

#print (x, y, w, h)

cv2.rectangle(image, (x, y), (x + w, y + h), (0,255,255), 5)

cv2.imshow('',image)

cv2.waitKey(10000)

plt.show()

cv2.destroyAllWindows()

**Parameter -Scale Factor**

The argument scaleFactor **determines the factor by which the detection window of the classifier is scaled down per detection pass**. A factor of 1.1 corresponds to an increase of 10%. Hence, increasing the scale factor increases performance, as the number of detection passes is reduced

image = cv2.imread("C:\\Users\\kirub\\OneDrive\\Pictures\\people1.jpg")

image = cv2.resize(image, (800, 600)) # Resize image

image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

detections = face\_detector.detectMultiScale(image\_gray, scaleFactor = 1.09)

display (detections)

for (x, y, w, h) in detections:

cv2.rectangle(image, (x, y), (x + w, y + h), (0,255,0), 5)

cv2.imshow('',image)

cv2.waitKey(10000)

plt.show()

cv2.destroyAllWindows()

**Parameter –** **minNeighbors**

Load the image and display it

image = cv2.imread("C:\\Users\\kirub\\OneDrive\\Pictures\\people1.jpg")

cv2.imshow('',image)

cv2.waitKey(10000)

plt.show()

cv2.destroyAllWindows()

Face Detections in the normal way – It has many False Positive

image = cv2.imread("C:\\Users\\kirub\\OneDrive\\Pictures\\people1.jpg")

image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

detections = face\_detector.detectMultiScale(image\_gray, scaleFactor = 1.09)

display (detections)

for (x, y, w, h) in detections:

print(w, h)

cv2.rectangle(image, (x, y), (x + w, y + h), (0,255,0), 2)

cv2.imshow('',image)

cv2.waitKey(10000)

plt.show()

cv2.destroyAllWindows()

**Use Parameter –** **minNeighbors – It reduces False Positive**

minNeighbors – **Parameter specifying how many neighbors each candidate rectangle should have to retain it**. In other words, this parameter will affect the quality of the detected faces. Higher value results in less detections but with higher quality

image = cv2.imread("C:\\Users\\kirub\\OneDrive\\Pictures\\people1.jpg")

image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

detections = face\_detector.detectMultiScale(image\_gray, scaleFactor = 1.2,minNeighbors=7)

display (detections)

for (x, y, w, h) in detections:

print(w, h)

cv2.rectangle(image, (x, y), (x + w, y + h), (0,255,0), 2)

cv2.imshow('',image)

cv2.waitKey(10000)

plt.show()

cv2.destroyAllWindows()

**Use Parameter –** **minSize and  maxSize**

1. **minSize** : Minimum possible object size. Objects smaller than that are ignored. That is the rectangle size.
2. **maxSize** : Maximum possible object size. Objects larger than that are ignored.

image = cv2.imread("C:\\Users\\kirub\\OneDrive\\Pictures\\people1.jpg")

image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

detections = face\_detector.detectMultiScale(image\_gray, scaleFactor = 1.2,

minNeighbors=7,minSize=(20,20), maxSize=(100,100))

display (detections)

for (x, y, w, h) in detections:

print(w, h)

cv2.rectangle(image, (x, y), (x + w, y + h), (0,255,0), 2)

cv2.imshow('',image)

cv2.waitKey(10000)

plt.show()

cv2.destroyAllWindows()

**Eye Detector**

**Load eye detector .xml file**

eye\_detector= cv2.CascadeClassifier("C:\\Users\\kirub\\OneDrive\\Pictures\\people1.jpg")

display (eye\_detector)

**Use both eye detection and Face detection**

image = cv2.imread("C:\\Users\\kirub\\OneDrive\\Pictures\\people1.jpg")

display (image.shape)

image = cv2.resize(image, (1600,1000)) # Resize image

print(image.shape)

face\_detections = face\_detector.detectMultiScale(image, scaleFactor = 1.3, minSize = (30,30))

for (x, y, w, h) in face\_detections:

cv2.rectangle(image, (x, y), (x + w, y + h), (0,255,0), 2)

eye\_detections = eye\_detector.detectMultiScale(image, scaleFactor = 1.1, minNeighbors=10, maxSize=(60,60))

for (x, y, w, h) in eye\_detections:

print(w, h)

cv2.rectangle(image, (x, y), (x + w, y + h), (0,0,255), 2)

cv2.imshow('',image)

cv2.waitKey(10000)

plt.show()

cv2.destroyAllWindows()