*Chapter 5*

# Face Detection

This chapter deals with face detection using Python 3.8.13. I hope you will like it.

## Contents:



## 1. Face Detection - A Brief Theory

Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images. Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene.

Face detection can be regarded as a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Examples include upper torsos, pedestrians, and cars. Face detection simply answers two question, 1. are there any human faces in the collected images or video? 2. where is the located?

Face-detection algorithms focus on the detection of frontal human faces. It is analogous to image detection in which the image of a person is matched bit by bit. Image matches with the image stores in database. Any facial feature changes in the database will invalidate the matching process.

A reliable face-detection approach based on the genetic algorithm and the eigen-face technique:

Firstly, the possible human eye regions are detected by testing all the valley regions in the gray-level image. Then the genetic algorithm is used to generate all the possible face regions which include the eyebrows, the iris, the nostril and the mouth corners.

Each possible face candidate is normalized to reduce both the lighting effect, which is caused by uneven illumination; and the shirring effect, which is due to head movement. The fitness value of each candidate is measured based on its projection on the eigen-faces. After a number of iterations, all the face candidates with a high fitness value are selected for further verification. At this stage, the face symmetry is measured and the existence of the different facial features is verified for each face candidate.

### *Applications:*

1. Facial Motion Capture
2. Facial Recognition

3. Face detection is used in biometrics, often as a part of (or together with) a facial recognition system. It is also used in video surveillance, human computer interface and image database management.
4. Photography: Some recent digital cameras use face detection for autofocus. Face detection is also useful for selecting regions of interest in photo slideshows that use a pan-and-scale Ken Burns effect.
5. Modern appliances also use smile detection to take a photograph at an appropriate time.
6. Marketing: Face detection is gaining the interest of marketers. A webcam can be integrated into a television and detect any face that walks by. The system then calculates the race, gender, and age range of the face. Once the information is collected, a series of advertisements can be played that is specific toward the detected race/gender/age. An example of such a system is OptimEyes and is integrated into the Amscreen digital signage system.
7. Emotional Inference: Face detection can be used as part of a software implementation of emotional inference. Emotional inference can be used to help people with autism under stand the feelings of people around them.
8. Lip Reading: Face detection is essential for the process of language inference from visual cues. Automated lip reading has applications to help computers determine who is speaking which is needed when security is important.

### *Commonly Used Methods for Face Detection:*

1. HAAR Cascade Algorithm
2. Local Binary Pattern(LBP) Classifier

Here we are going to discuss about face detection using Haar Cascade Algorithm.

## 2. Face Detection using Haar Cascade Algorithm

### *Haar Feature-Based Cascade Classifier:*

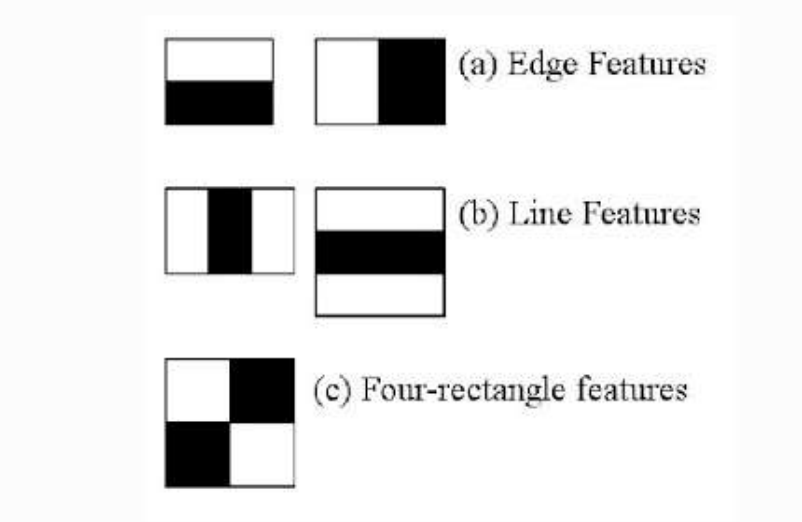
Haar-like features are digital image features used in object recognition. They owe their name to their intuitive similarity with Haar wavelets and were used in the first real-time face detector. Paul Viola and Michael Jones in their paper titled "Rapid Object Detection using a Boosted Cascade of Simple Features" used the idea of Haar-feature classifier based on the Haar wavelets. This classifier is widely used for tasks like face detection in computer vision industry.

Haar cascade classifier employs a machine learning approach for visual object detection which is capable of processing images extremely rapidly and achieving high detection rates. This can be attributed to three main reasons:

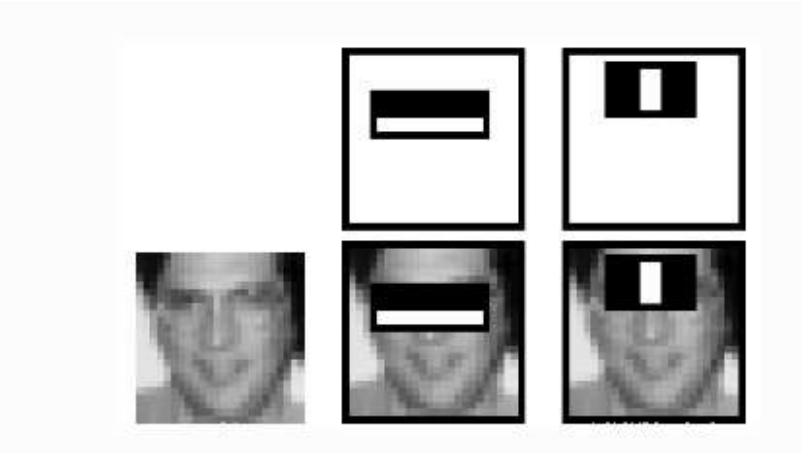
Haar classifier employs 'Integral Image' concept which allows the features used by the detector to be computed very quickly. The learning algorithm is based on AdaBoost. It selects a small number of important features from a large set and gives highly efficient classifiers. More complex classifiers are combined to form a 'cascade' which discard any non-face regions in an image, thereby spending more computation on promising object-like regions. Let us now try and understand how the algorithm works on images in steps:

**1. 'Haar features' Extraction:**

After the tremendous amount of training data (in the form of images) is fed into the system, the classifier begins by extracting Haar features from each image. Haar Features are kind of convolution kernels which primarily detect whether a suitable feature is present on an image or not. Some examples of Haar features are mentioned below:




These Haar Features are like windows and are placed upon images to compute a single feature. The feature is essentially a single value obtained by subtracting the sum of the pixels under the white region and that under the black. The process can be easily visualized in the example below.

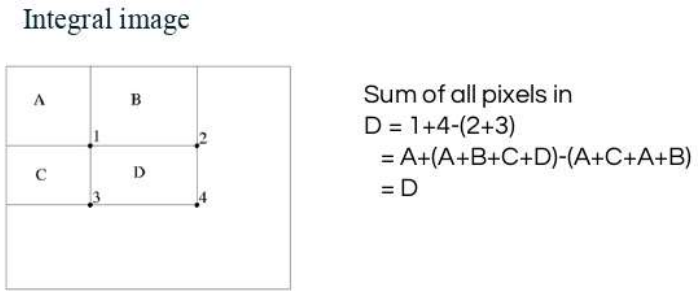


For demonstration purpose, let's say we are only extracting two features, hence we have only two windows here. The first feature relies on the point that the eye region is darker than the adjacent cheeks and nose region. The second feature focuses on the fact that eyes are kind of darker as compared to the bridge of the nose. Thus, when the feature window moves over the eyes, it will calculate a single value. This value will then be compared to some threshold and if it passes that it will conclude that there is an edge here or some positive feature.

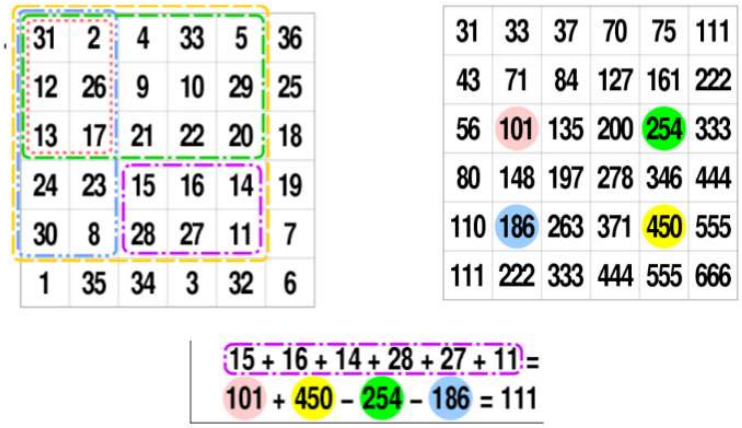
**2. 'Integral Images' Concept**

The algorithm proposed by Viola Jones uses a 24X24 base window size, and that would result in more than 180,000 features being calculated in this window. Imagine calculating the pixel difference for all the features? The solution devised for this computationally intensive process is to

go for the Integral Image concept. The integral image means that to find the sum of all pixels under any rectangle, we simply need the four corner values.




This means, to calculate the sum of pixels in any feature window, we do not need to sum them up individually. All we need is to calculate the integral image using the 4 corner values. The example below will make the process transparent.

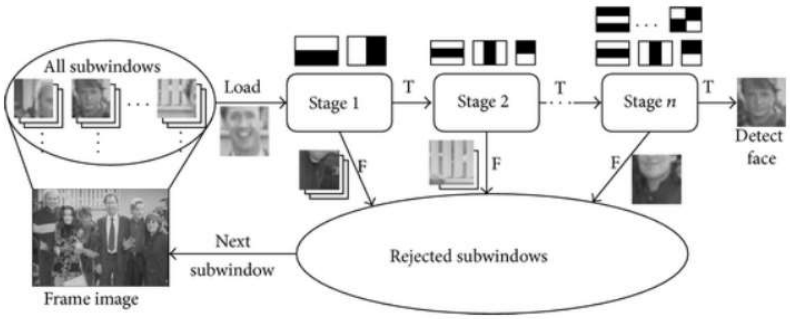



### 3. 'Adaboost': To Improve Classifier Accuracy

As pointed out above, more than 180,000 features values result within a 24X24 window. However, not all features are useful for identifying a face. To only select the best feature out of the entire chunk, a machine learning algorithm called Adaboost is used. What it essentially does is that it selects only those features that help to improve the classifier accuracy. It does so by constructing a strong classifier which is a linear combination of a number of weak classifiers. This reduces the amount of features drastically to around 6000 from around 180,000.

### 4. Using 'Cascade of Classifiers'

Another way by which Viola Jones ensured that the algorithm performs fast is by employing a cascade of classifiers. The cascade classifier essentially consists of stages where each stage consists of a strong classifier. This is beneficial since it eliminates the need to apply all features at once on a window. Rather, it groups the features into separate sub-windows and the classifier at each stage determines whether or not the sub-window is a face. In case it is not, the sub-window is discarded along with the features in that window. If the sub-window moves past the classifier, it continues to the next stage where the second stage of features is applied. The process can be understood with the help of the diagram below.




The Paul-Viola algorithm can be visualized as follows:


## *Demonstration:*

For More Information Please Visit The Given Links:

https://www.datacamp.com/tutorial/face-detection-python-opencv
(https://www.datacamp.com/tutorial/face-detection-python-opencv)

https://www.geeksforgeeks.org/opencv-python-program-face-detection/
(https://www.geeksforgeeks.org/opencv-python-program-face-detection/)

https://www.geeksforgeeks.org/face-detection-using-python-and-opencv-with-webcam/
(https://www.geeksforgeeks.org/face-detection-using-python-and-opencv-with-webcam/)

https://towardsdatascience.com/face-detection-in-2-minutes-using-opencv-python-90f89d7c0f81
(https://towardsdatascience.com/face-detection-in-2-minutes-using-opencv-python-90f89d7c0f81)

https://www.javatpoint.com/face-recognition-and-face-detection-using-opencv#:~:text=Face%20recognition%20and%20Face%20detection%20using%20the%20OpenCV,c
(https://www.javatpoint.com/face-recognition-and-face-detection-using-

opencv#:~:text=Face%20recognition%20and%20Face%20detection%20using%20the%20OpenCV,c

https://developers.google.com/ml-kit/vision/face-detection (https://developers.google.com/ml-kit/vision/face-detection)

https://firebase.google.com/docs/ml-kit/detect-faces (https://firebase.google.com/docs/ml-kit/detect-faces)

https://www.techtarget.com/searchenterpriseai/definition/face-detection (https://www.techtarget.com/searchenterpriseai/definition/face-detection)

https://towardsdatascience.com/face-detection-for-beginners-e58e8f21aad9 (https://towardsdatascience.com/face-detection-for-beginners-e58e8f21aad9)

https://en.wikipedia.org/wiki/Face_detection (https://en.wikipedia.org/wiki/Face_detection)

https://docs.aws.amazon.com/rekognition/latest/dg/faces.html (https://docs.aws.amazon.com/rekognition/latest/dg/faces.html)

https://en.wikipedia.org/wiki/Face_detection (https://en.wikipedia.org/wiki/Face_detection)

https://cloud.google.com/vision/docs/detecting-faces (https://cloud.google.com/vision/docs/detecting-faces)

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
img = cv2.imread('kc1.jpg') ## by default the image is read as BGR
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.title('Original Image')
plt.imshow(img)
plt.axis('off')
plt.show()
```

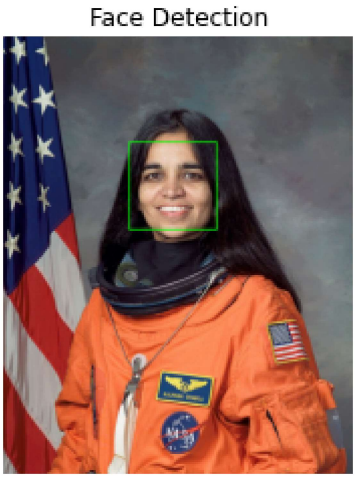
In [5]:
```
gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)#grey
plt.title('Original Grayscale Image')
plt.imshow(gray, cmap='gray')
plt.axis('off')
plt.show()
```


In [10]:
```
# pre-defined classifier to identify faces
# Haar cascade classifier employs a machine learning approach for visual object d
haar_cascade_face = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
# detectMultiscale module of the classifier. This function will return a rectangl
# around the detected face.
# scalefactor In a group photo, there may be some faces which are near the camera
# Naturally, such faces would appear more prominent than the ones behind. This fa
# minNeighbors This parameter specifies the number of neighbors a rectangle shoul
# these are the default and standard values
faces_rects = haar_cascade_face.detectMultiScale(gray, scaleFactor = 1.2, minNeig
# Let us print the no. of faces found
print('Faces found: ', len(faces_rects))
```

```
Faces found:  1
```

In [11]:
```
for (x,y,w,h) in faces_rects:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

In [13]:
```
#convert image to RGB and show image
plt.title('Face Detection')
plt.imshow(img)
plt.axis('off')
plt.show()
```




## 2. Face Detection Using DNN

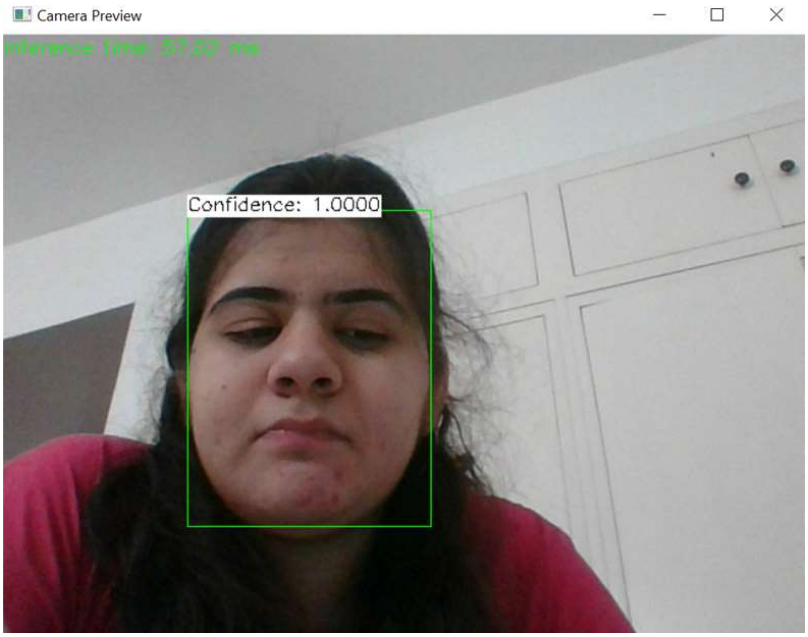
Write the following code for performing face detection using DNN:

***Code:***

Face Detection using DNN.py

```python
import cv2

source = cv2.VideoCapture(0)

win_name = 'Camera Preview'
cv2.namedWindow(win_name, cv2.WINDOW_NORMAL)

net = cv2.dnn.readNetFromCaffe("deploy.prototxt",
                               "res10_300x300_ssd_iter_140000_fp16.caffemodel")

# Model parameters
in_width = 300
in_height = 300
mean = [104, 117, 123]
conf_threshold = 0.7

while (cv2.waitKey(1)!=27):
    has_frame, frame = source.read()
    if not has_frame:
        break
    frame = cv2.flip(frame,1)
    frame_height = frame.shape[0]
    frame_width = frame.shape[1]

    # Create a 4D blob from a frame.
    blob = cv2.dnn.blobFromImage(frame,1.0,(in_width, in_height), mean, swapRB = False, crop = False)
    # Run a model
    net.setInput(blob)
    detections = net.forward()

    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if (confidence>conf_threshold):
            x_left_bottom = int(detections[0, 0, i, 3] * frame_width)
            y_left_bottom = int(detections[0, 0, i, 4] * frame_height)
            x_right_top = int(detections[0, 0, i, 5] * frame_width)
            y_right_top = int(detections[0, 0, i, 6] * frame_height)

            cv2.rectangle(frame, (x_left_bottom, y_left_bottom), (x_right_top, y_right_top), (0, 255, 0))
            label = "Confidence: %.4f" % confidence
            label_size, base_line = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)

            cv2.rectangle(frame, (x_left_bottom, y_left_bottom - label_size[1]),
                                 (x_left_bottom + label_size[0], y_left_bottom + base_line),
                                 (255, 255, 255), cv2.FILLED)
            cv2.putText(frame, label, (x_left_bottom, y_left_bottom),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))

            t,_ = net.getPerfProfile()
            label = 'Inference time: %.2f ms' % (t * 1000.0 / cv2.getTickFrequency())
            cv2.putText(frame, label, (0, 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0))
            cv2.imshow(win_name, frame)

source.release()
cv2.destroyWindow(win_name)
```

## *Output:*




# 3. Face Detection Using Mediapipe

Here, we are going to use mediapipe api for detecting faces in images and videos

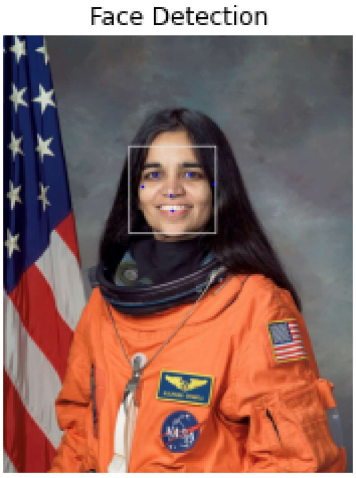
## 3.1 On Images - Demonstration

In [14]:
```
import mediapipe as mp
```

In [16]:
```
mp_face_detection = mp.solutions.face_detection
mp_drawing = mp.solutions.drawing_utils
```

In [20]:
```
with mp_face_detection.FaceDetection(model_selection=1,  min_detection_confidence
    image = cv2.imread('kc1.jpg')
    image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    # Convert the BGR image to RGB and process it with MediaPipe Face Detection.
    results = face_detection.process(image)
    # Draw face detections of each face.
    if (not results.detections):
        print('Face Cannot Be Detected')
    annotated_image = image.copy()
    for detection in results.detections:
        print('Nose tip:')
        print(mp_face_detection.get_key_point(detection, mp_face_detection.FaceKe
        mp_drawing.draw_detection(annotated_image, detection)
    plt.title('Face Detection')
    plt.imshow(annotated_image)
    plt.axis('off')
    plt.show()
```

```
Nose tip:
x: 0.48034185
y: 0.36611181
```




## 3.2 On Videos - Demonstration

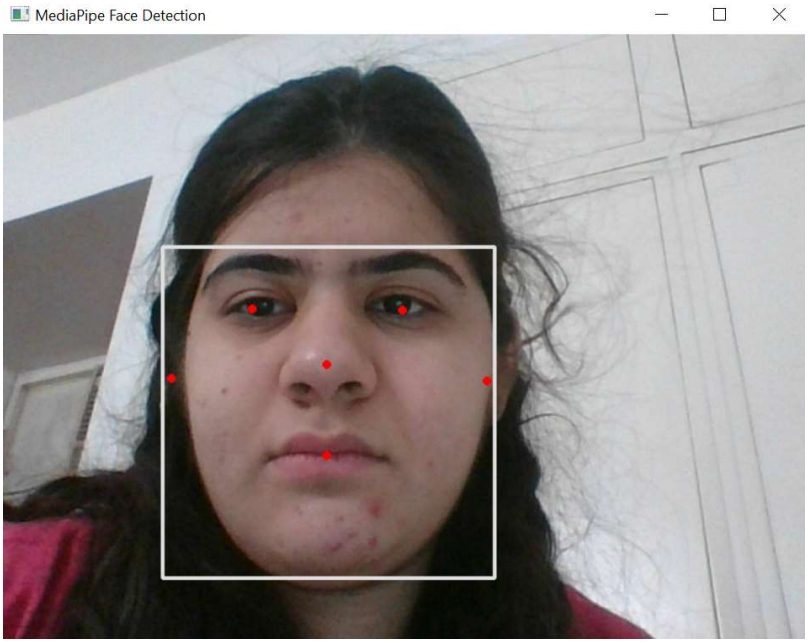
Write the following code for performing face detection on videos using mediapipe:

### *Code:*

Face Detection using Mediapipe.py

```python
import cv2
import mediapipe as mp
import numpy as np

mp_face_detection = mp.solutions.face_detection
mp_drawing = mp.solutions.drawing_utils

cap = cv2.VideoCapture(0)

with mp_face_detection.FaceDetection(
    model_selection=0, min_detection_confidence=0.5) as face_detection:
  while cap.isOpened():
    success, image = cap.read()
    if not success:
      print("Ignoring empty camera frame.")
      # If loading a video, use 'break' instead of 'continue'.
      continue

    # To improve performance, optionally mark the image as not writeable to
    # pass by reference.
    image.flags.writeable = False
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = face_detection.process(image)

    # Draw the face detection annotations on the image.
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    if results.detections:
      for detection in results.detections:
        mp_drawing.draw_detection(image, detection)
    # Flip the image horizontally for a selfie-view display.
    cv2.imshow('MediaPipe Face Detection', cv2.flip(image, 1))
    if cv2.waitKey(5) & 0xFF == 27:
      break
cap.release()
```

***Output:***




# The End of Chapter 5 !!!