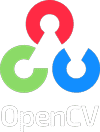
**Proyecto 1 – Detección de Caras**

**I - Introducción**

La importancia de hacer aplicaciones es fundamental a la hora de programar, nos obliga a retarnos, aprender nuevas tecnologías y también a la resolución de problemas, cosas que no encontraremos en un libro. El día a día laboral es de esta manera. Realizaremos un detector de caras con una famosa librería llamada OpenCV.

**II - ¿Qué es OpenCV?**



OpenCV es una librería de código abierto que principalmente se enfoca a visión por computador. El acrónimo descompuesto sería “Open Source Computer Vision” library. Liberada en los principios del año 2000 fue en sus orígenes solo para C++, pero ha sido actualizada y cada vez son más los lenguajes de programación a los cuales son migrados.

Podemos ver más de OpenCV en esta dirección: <https://opencv.org/>

En este caso utilizaremos OpenCV Python. Regularmente hacen actualizaciones. Ahora realizaremos un proyecto con detección de caras.

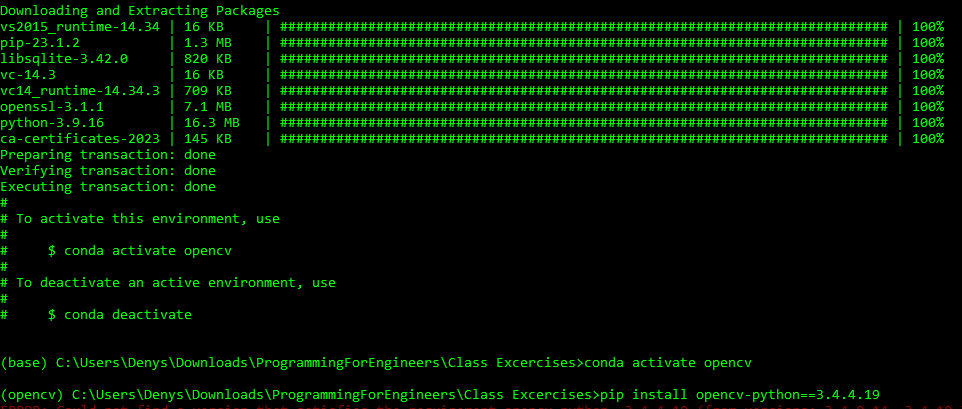
Para este ejemplo realizaremos el uso de la librería OpenCV Python y validaremos la librería en su version.

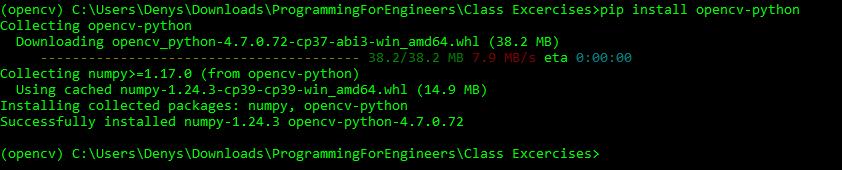
Primero crearemos un ambiente virtual llamado opencv

**conda create –n opencv python=3.9**

**conda activate opencv**

**pip install opencv-python==4.7.0**





Ahora descargaremos el modelo de Haar Cascades para clasificación de imágenes. Haar cascades es básicamente un clasificador que se utiliza para particularmente detectar objetos de la misma fuente, en especial este que utilizaremos se realiza para detección frontal de caras. Nosotros hemos descargado el archivo haarcascade\_frontalface\_default.xml que servirá para este propósito pero podemos encontrar más modelos en: <https://github.com/opencv/opencv/tree/master/data/haarcascades>

**III – Creación del Programa de Detección de Caras**

Lo primero que realizaremos es importar la librería de opencv

**import cv2**

Seguido llamaremos al detector de caras

**face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')**

ahora que cargamos el algoritmo necesitamos una imagen a detectar. Note que tanto el xml y la imagen de Elon Musk están en el mismo folder, por ende estas están disponibles directamente y las podens cargar sin una ruta en específico.

**cap = cv2.VideoCapture("elon.jpg")**

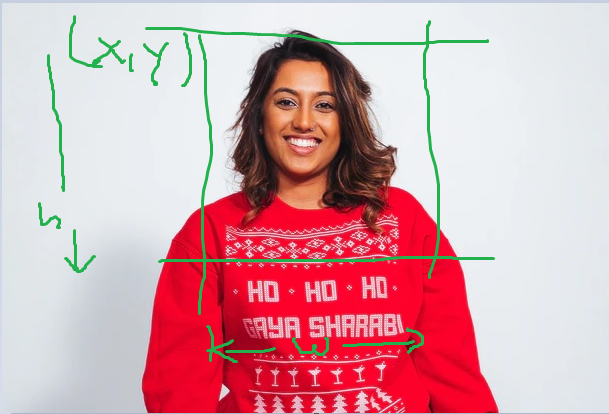
Ahora leeremos con el método read() la imagen, este nos da dos salidas, la imagen y el resultado de retorno que es un validador si la operación se realizó correctamente.

**ret, img = cap.read()**

Lo siguiente es realizar la conversión de a escala de grises. Como el filtro Haar es entrenado en este tipo de imágenes necesitamos realizar las conversiones de cada una de las imágenes a esto

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

Lo siguiente es intentar detectar la cara de la persona, para esto necesitamos entender como opera el detector haar en su salida



Digamos que tenemos la siguiente imagen anterior y detectamos una cara. Normalmente OpenCV empieza por 0,0 como en superior izquierdo. Si detectamos una cara en (x,y), la salida del detector será la posición inicial del cuadro y (w,h) o el ancho y el alto de la cara.

Ahora invoquemos la función de detección a multi escalas.

**faces = face\_cascade.detectMultiScale(gray, 1.3, 5)**

Vamos a realizar la inspección de la imagen leída con imshow(), por ahora, para pruebas el programa nos queda de la siguiente manera…

**import cv2**

**face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')**

**cap = cv2.VideoCapture("elon.jpg")**

**ret, img = cap.read()**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = face\_cascade.detectMultiScale(gray, 1.3, 5)**

**cv2.imshow('img', img)**

**cv2.waitKey(0)**

WaiKey(0) hará que esperemos por siempre hasta la entrada de alguna tecla,esto nos da la posibilidad de ver la imagen.

Lo último es liberar memoria para cuando luego de waitkey() demos ESC por ejemplo.

**cap.release()**

**cv2.destroyAllWindows()**

El resultado debe verse como sigue, cuando este conforme con el resultado presione ESC para cerrar la pantalla.



**IV – Marcar el rectángulo de la cara detectada**

Para esto luego de la línea …

faces = …

insertaremos el siguiente código

**for (x,y,w,h) in faces:**

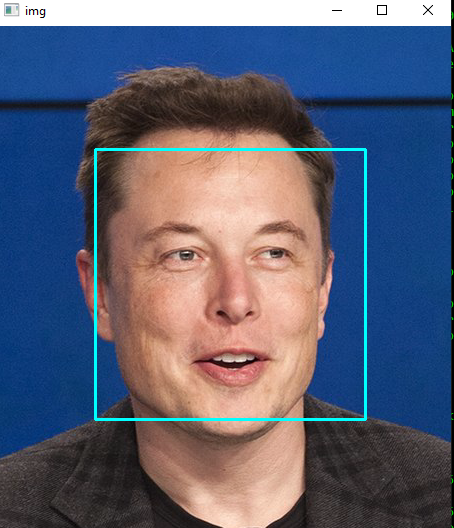
**cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)**

Lo anterior se traduce en lo siguiente…

for (x,y,w,h) in faces = Para cada cara detectada, que corresponde a un tuple de cuatro variables cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2) = dibuja un rectángulo en la imagen original con posición inicial (x,y) y posición final (x+w, y+h), el tono de color del cuadro es (intensidad\_de azul, intensidad\_de\_verde, intensidad\_de\_rojo) , y el grosor de las líneas es 2.

Ejecutemos el programa para ver los resultados…

**python project.py**



Perfecto, hemos detectado una cara.

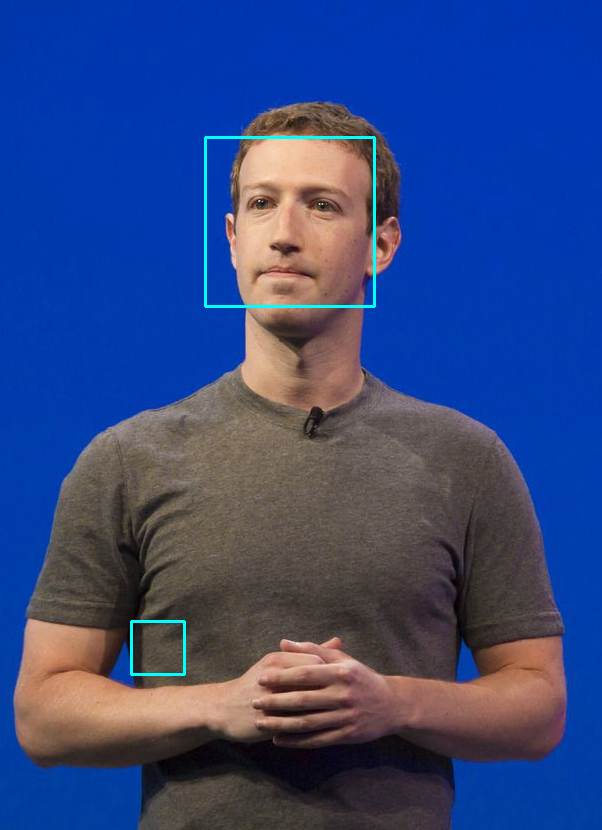
Ahora, cambiemos en la línea donde detectamos a Elon Musk y probemos con Mark Zuckerberg.

**cap = cv2.VideoCapture("mark.jpg")**

Ejecutamos nuevamente para ver resultados

**python project.py**

Interesante, no solo detectamos la cara de Mark sino que tenemos algo llamado un “Falso Positivo”



Esto se puede corregir un poco con el ajuste de parámetros del detector Haar, pero seguiremos adelante.

**V – Creación de Detección de Múltiples Caras**

Ahora vamos a modificar un poco el programa para detectar múltiples caras. Para esto haremos una sencilla modificación al programa original, pero primero guárdelo con otro nombre para tener un código base.

Eliminaremos las líneas

**cap = cv2.VideoCapture("mark.jpg")**

**ret, img = cap.read()**

Y sobre la línea

**face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')**

importaremos glob y leeremos una carpeta que tiene las imágenes que queremos detectar

**import glob**

**all\_images = glob.glob(“multi-image-detector/\*.jpg”)**

También cambiaremos un poco nuestro programa insertando el cambio de imagen de color en el ciclo for de la siguiente manera:

**for image in all\_images:**

**img = cv2.imread(image)**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = face\_cascade.detectMultiScale(gray, 1.3, 2)**

**for (x,y,w,h) in faces:**

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

De esta manera nos aseguraremos que cada imagen en la carpeta es detectada

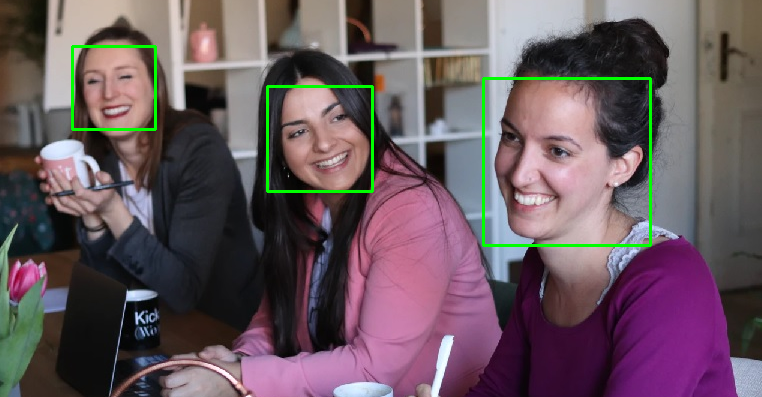
Finalmente realizaremos la detección de imágenes cada dos segundos

**cv2.waitKey(2000)**

Finalmente ejecutamos y realizamos una inspección ejecutando el código.

**python face-detector.py**

Vemos como va ciclando por pantalla y detectando caras



**VI – Modificación del Primer Programa para detección de Caras en tiempo real.**

Todo esto esta perfecto, pero el mundo real no es estático, es dinámico. Ahora haremos una copia del primer programa y lo modificaremos para realizar detección de caras en tiempo real.

Para hacer esto, insertaremos luego de

face\_cascade = …

lo siguiente:

**cap = cv2.VideoCapture(0)**

**# Detects faces of different sizes in the input image**

**while True:**

**ret, img = cap.read()**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = face\_cascade.detectMultiScale(gray, 1.3, 2)**

**for (x,y,w,h) in faces:**

**# To draw a rectangle in a face**

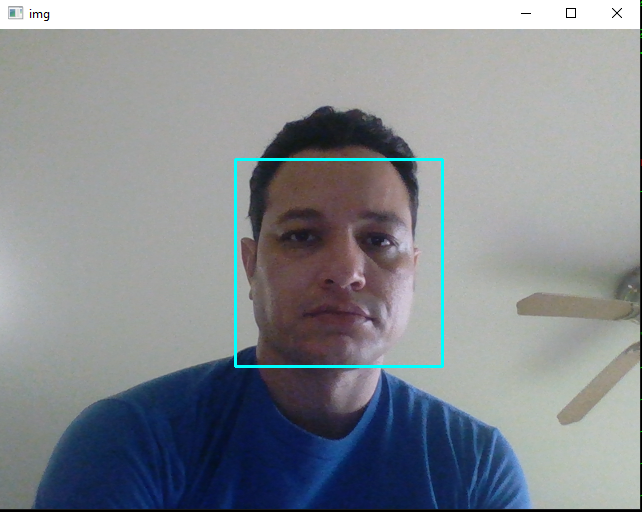
**cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)**

**cv2.imshow('img',img)**

**k = cv2.waitKey(25)**

**if k == 27:**

**break**



**VII – Conclusión**

Finalmente hemos concluidos este proyecto. Imagínese la posibilidad de ejecutar luego de detectar.

* Aumentar la imagen original poniendo lentes
* Cambiando caras
* Hacer que un drone le siga
* Detectar un intruso

Si bien es cierto existen tecnologías más de vanguardia para realizar detección de caras y con mejores resultados, esto le da una idea del mundo de visión por computador y la inteligencia artificial además de sus aplicaciones.

Existe un problema con esta aplicación, y es su portabilidad y uso, note que tiene que instalar diferentes aplicaciones para realizar esto, lo otro es su uso. Solamente las personas que utilizan o tienen instalado Python podrán lanzar la aplicación.

Intentaremos para la siguiente semana realizar una aplicación con simple HTML, JS y CSS hacerla vistosa y que sea posible que muchos usuarios la accesen. Por el momento esto es todo.

**VIII – Apéndice**

Lista de Código de Detección de Una cara

**import cv2**

**face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')**

**cap = cv2.VideoCapture("mark.jpg")**

**ret, img = cap.read()**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**# Detects faces of different sizes in the input image**

**faces = face\_cascade.detectMultiScale(gray, 1.3, 2)**

**for (x,y,w,h) in faces:**

**# To draw a rectangle in a face**

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

**cv2.imshow('img',img)**

**k = cv2.waitKey(0)**

**# Close the window**

**cap.release()**

**# De-allocate any associated memory usage**

**cv2.destroyAllWindows()**

Lista de Código de Detección de Múltiples caras

**import cv2, glob**

**all\_images = glob.glob("multi-image-detector/\*.jpg")**

**detect = cv2.CascadeClassifier("haarcascade\_frontalface\_default.xml")**

**for image in all\_images:**

**img = cv2.imread(image)**

**gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = detect.detectMultiScale(gray\_img, 1.1, 5)**

**for (x, y, w, h) in faces:**

**final\_img = cv2.rectangle(img, (x,y), (x+w,y+h), (0, 255, 0), 2)**

**cv2.imshow("Face Detection", final\_img)**

**cv2.waitKey(2000)**

**cv2.destroyAllWindows**

Lista de Código de Detección de Caras en Tiempo Real

**import cv2**

**face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')**

**cap = cv2.VideoCapture(0)**

**# Detects faces of different sizes in the input image**

**while True:**

**ret, img = cap.read()**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = face\_cascade.detectMultiScale(gray, 1.3, 2)**

**for (x,y,w,h) in faces:**

**# To draw a rectangle in a face**

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

**cv2.imshow('img',img)**

**k = cv2.waitKey(25)**

**if k == 27:**

**break**

**# Close the window**

**cap.release()**

**# De-allocate any associated memory usage**

**cv2.destroyAllWindows()**