

# CONDUCCIÓN AUTÓNOMA DE VEHÍCULOS (SELF-DRIVING CAR)

## Enrique Camacho



**SDC - 7 -**  
**Clasificador Haar Cascade**



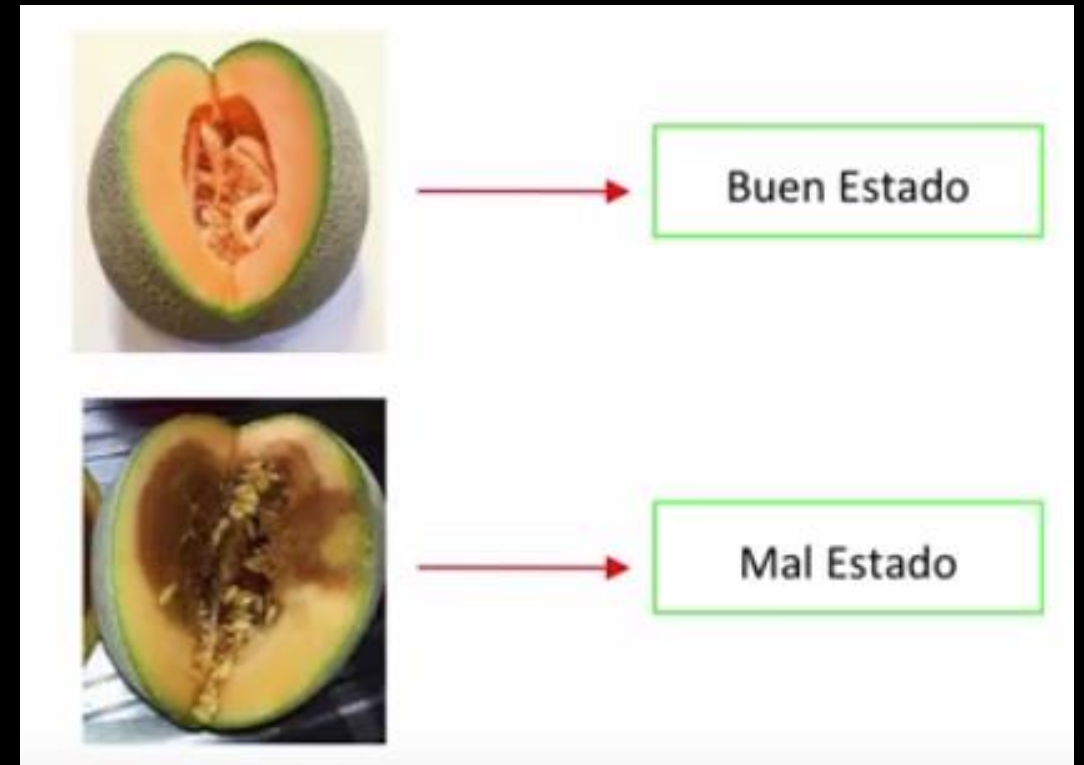
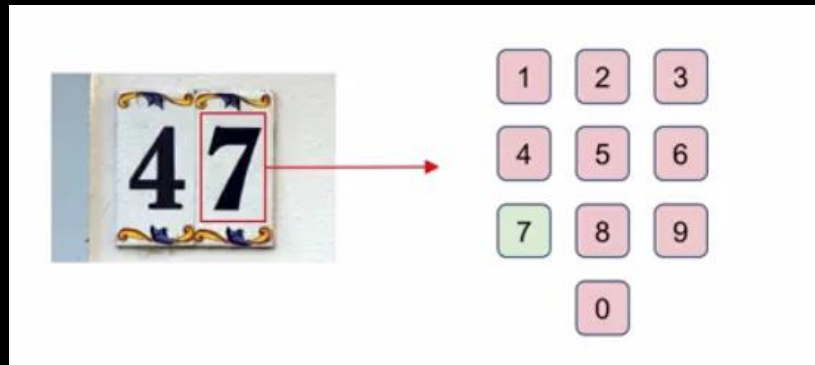
**UADY**  
UNIVERSIDAD  
AUTÓNOMA  
DE YUCATÁN

FACULTAD DE INGENIERÍA

**LAB**  
**datos**

# Problema de clasificación

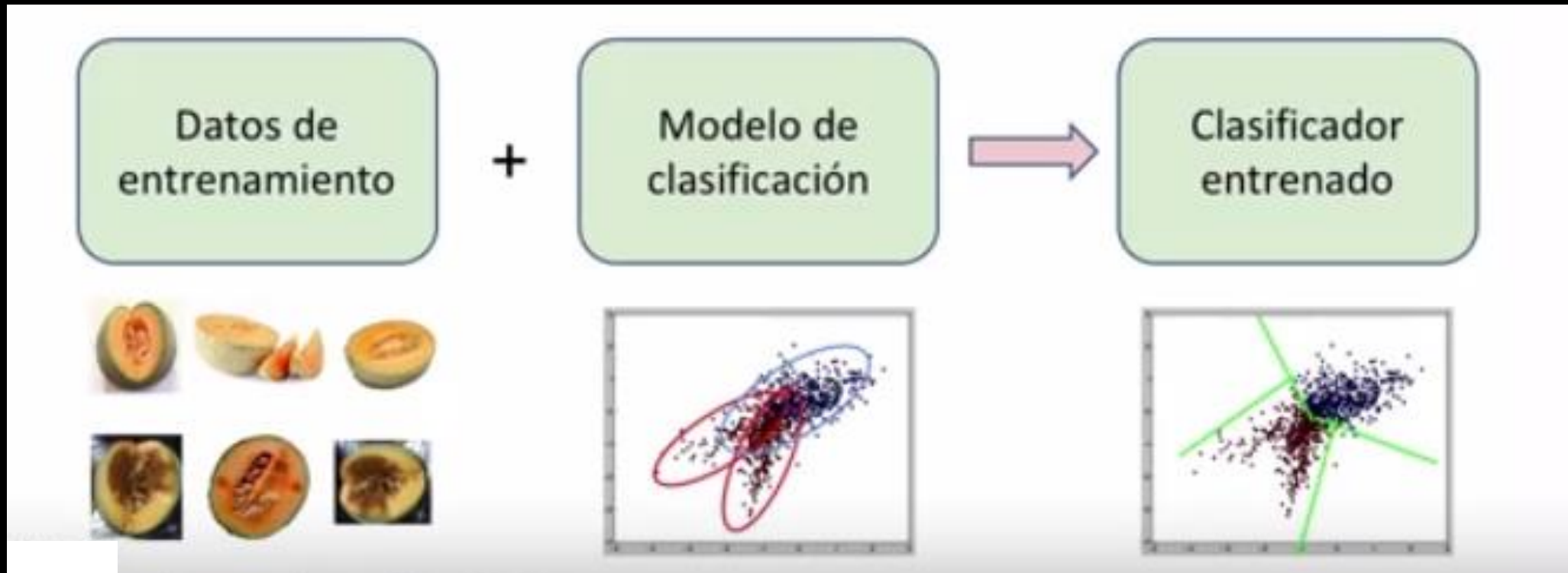
- El objetivo de un problema de clasificación es encontrar un sistema capaz de identificar automáticamente para cada objeto la clase a la cual pertenece

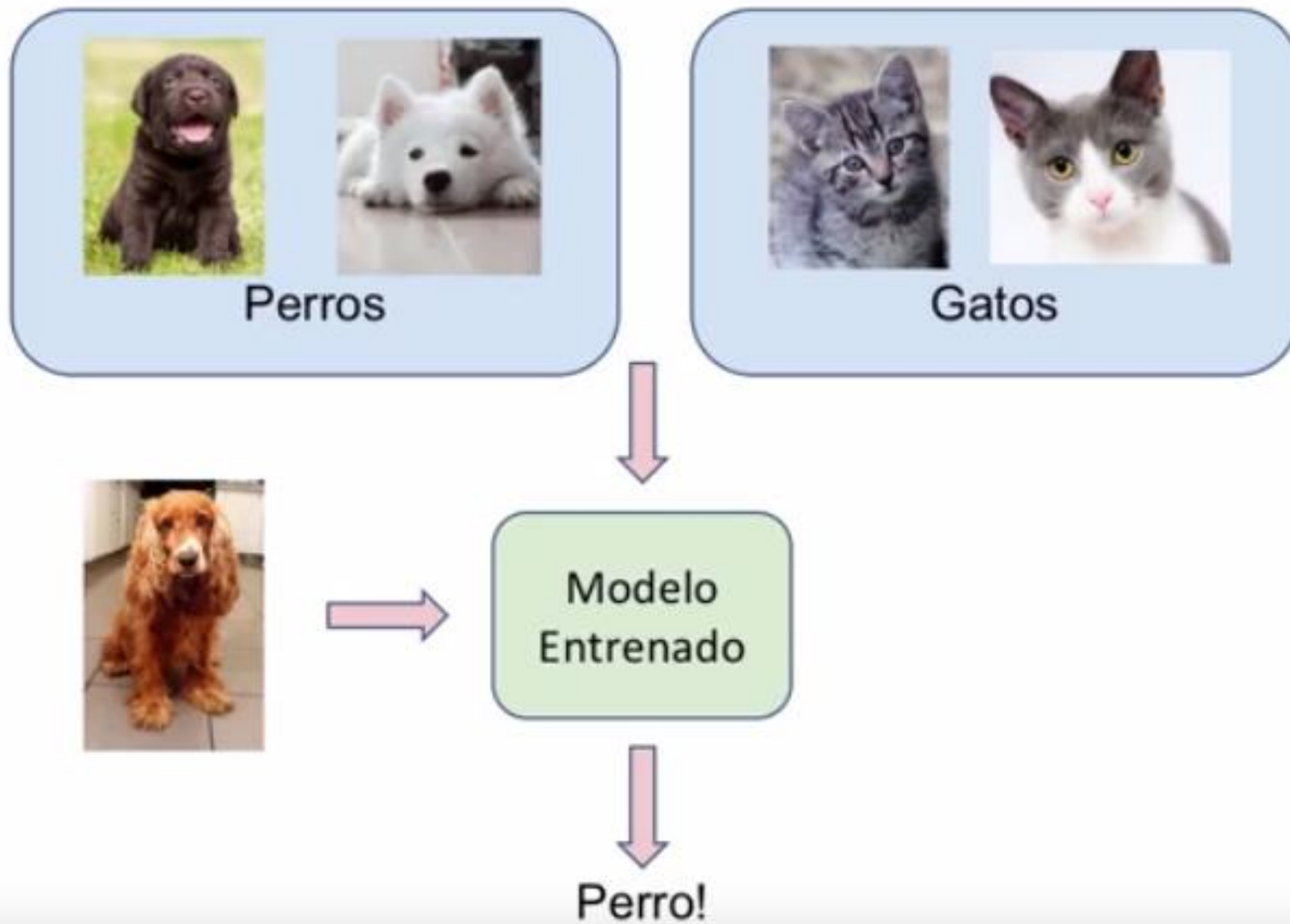




# ¿Que necesitamos para un clasificador?

- El modelo de clasificación se ajusta usando datos de entrenamiento





# ¿Cómo representar objetos?



# ¿Cómo representar objetos?

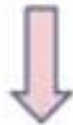
- Usando características propias del objeto



id	Total Compra	Profesión	Ubicación tienda	Género	...
1	105	Arquitecto	Santiago	Femenino	...

# ¿Cómo representar objetos?

- Usando características propias del objeto



id	Total Compra	Profesión	Ubicación tienda	Género	...	Clase
1	105	Arquitecto	Santiago	Femenino	...	Preferencial

# ¿Cómo representar objetos?

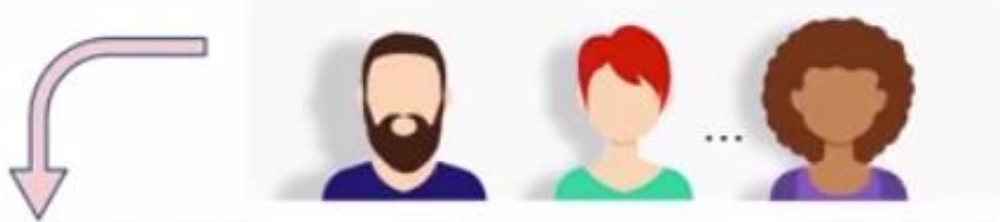
id	Total Compra	Profesión	Ubicación tienda	Género	...	Clase
1	105	Arquitecto	Santiago	Femenino	...	Preferencial





# ¿Cómo representar objetos?

- Los objetos corresponden a las filas en la base de datos



id	Total Compra	Profesión	Ubicación tienda	Género	...	Clase
1	105	Arquitecto	Santiago	Masculino	...	Preferencial
2	98	Músico	La Florida	Femenino	...	Frecuente
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1200	82	Periodista	Mirador	Femenino	...	Preferencial

# ¿Cómo representar objetos?

- En el caso de una imagen por medio de los pixeles



Img. de 300x300

id	Pixel1	Pixel2	...	Pixel 300	Animal
1	120	8	...	230	Reno

# ¿Cómo representar objetos?



id	Pixel1	Pixel2	...	Pixel 300	Numero
1	120	8	...	230	47

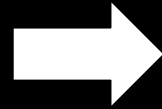
# Clasificador Haar Cascade

- Propuesto por Paul Viola and Michael Jones en su artículo, "Rapid Object Detection using a Boosted Cascade of Simple Features" en 2001.
- Es un enfoque basado en aprendizaje automático (machine learning) en el que una función se entrena a partir de muchas imágenes positivas y negativas. Luego se utiliza para detectar objetos en otras imágenes.
- Hay disponibles para OpenCV varios clasificadores entrenados para detectar el rostro, ojos, personas, señales de tráfico entre otros.

<https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>



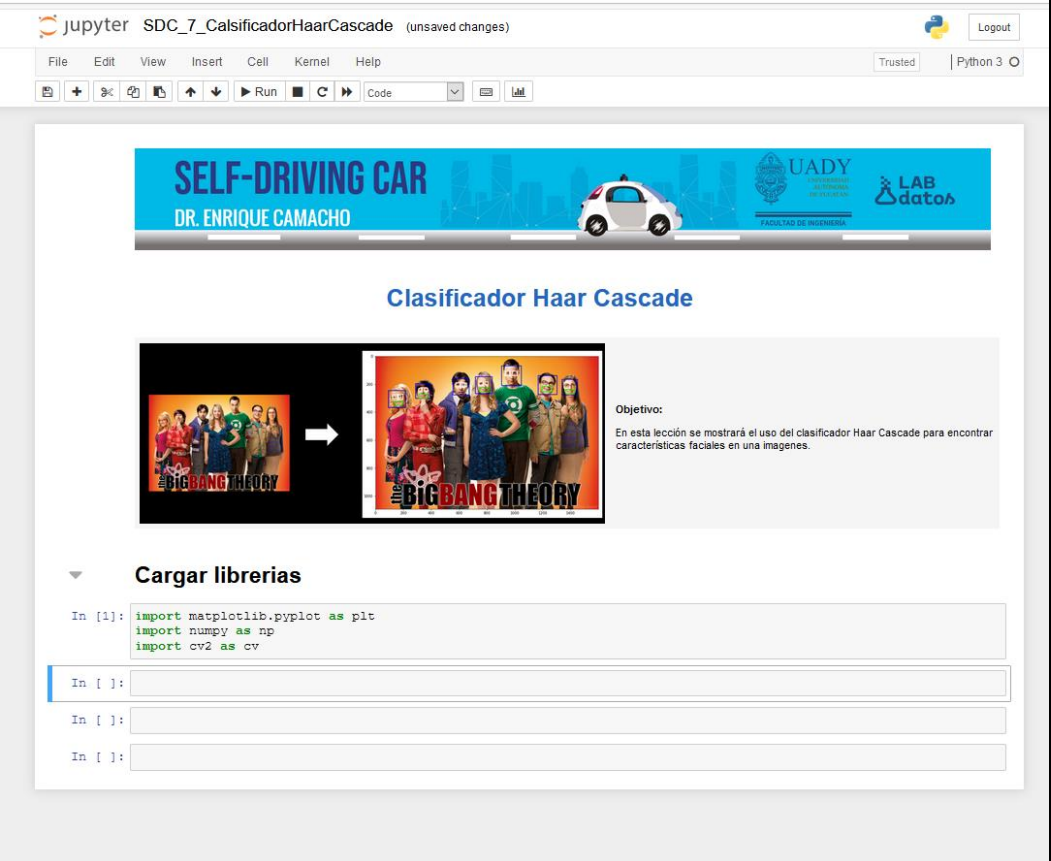




# Parte A: Encontrando características faciales en una imagen

# Actividad

- Realizar este ejercicio con la notebook **SDC\_7\_CalsificadorHaarCascade**
- Seguir el tutorial de la parte A con la imagen que se adjunta
- Después repita el código utilizando 5 imágenes de películas o series (como el del ejemplo).



The screenshot shows a Jupyter Notebook interface with the title "SDC\_7\_CalsificadorHaarCascade" and a status of "(unsaved changes)". The notebook contains a header banner for "SELF-DRIVING CAR" by "DR. ENRIQUE CAMACHO" from "UADY" and "LAB datos". Below the banner is the title "Clasificador Haar Cascade". A diagram shows two images of the Big Bang Theory cast, with an arrow pointing from the original image to a processed version where faces are highlighted with bounding boxes. To the right of the diagram is the "Objetivo:" section, which states: "En esta lección se mostrará el uso del clasificador Haar Cascade para encontrar características faciales en una imagen." Below this is a section titled "Cargar librerías" with a code cell containing the following code:

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

Below the code cell are three empty input fields for subsequent code cells.

- Cargar librerías generales

```
import matplotlib.pyplot as plt  
import numpy as np  
import cv2 as cv
```

## Cargar librerías

```
import matplotlib.pyplot as plt  
import numpy as np  
import cv2 as cv
```



- Mostramos la imagen original

```
#Mostramos la imagen original  
img = cv.imread('figuras/bbt.jpeg')  
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)  
plt.imshow(img)  
plt.show()
```

```
#Mostramos la imagen original  
img = cv.imread('figuras/bbt.jpeg')  
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)  
plt.imshow(img)  
plt.show()
```



# Buscando rostros en una imagen

```
#Cargando la imagen  
img = cv.imread('figuras/bbt.jpeg')
```

```
#Cargando el clasificador  
face_cascade = cv.CascadeClassifier('datos/haarcascade_frontalface_default.xml')
```

## Buscando rostros en una imagen

```
#Cargando la imagen  
img = cv.imread('figuras/bbt.jpeg')  
  
#Cargando el clasificador  
face_cascade = cv.CascadeClassifier('datos/haarcascade_frontalface_default.xml')
```

- Seguimos el mismo procedimiento, pero es posible realizarlo en una sola celda
- El archivo que contiene la información del clasificador:  
**haarcascade\_frontalface\_default.xml**

# Buscando rostros en una imagen

```
#Convertimos a escala de grises la imagen original
img_grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
plt.figure(figsize=(13,8))
plt.imshow(img_grey,cmap='gray')
```




- El clasificador sólo funciona en imágenes en escala de grises.

# Buscando rostros en una imagen

```
#Realizamos la búsqueda de rostros en la imagen
# 1.1 y 5 son valores típicos para una buena búsqueda
faces = face_cascade.detectMultiScale(grey, 1.12, 5)
print(faces)
```

```
#Realizamos la búsqueda de rostros en la imagen
faces = face_cascade.detectMultiScale(img_grey, 1.12, 5)
print(faces)
```

```
[[ 905   68  138  138]
 [1168  142  126  126]
 [ 552  157  121  121]
 [1339  165  113  113]
 [ 716  146  121  121]
 [ 286  196  107  107]
 [ 103  255  101  101]]
```



- El resultado son sublistas, donde cada una de ellas representa la región en donde se encuentra un rostro.
- La región es representada por un rectángulo o cuadrado



```
#Realizamos la búsqueda de rostros en la imagen
faces = face_cascade.detectMultiScale(img_grey, 1.12, 5)
print(faces)
```

```
[ [ 905 68 138 138]
  [1168 142 126 126]
  [ 552 157 121 121]
  [1339 165 113 113]
  [ 716 146 121 121]
  [ 286 196 107 107]
  [ 103 255 101 101]]
```

- 905 es el punto en X
- 68 es la coordenada en Y
- 138 es el ancho del rectángulo
- 138 es el alto del rectángulo



# Buscando rostros en una imagen

```
#Dibujamos rectángulos sobre la imagen original y mostramos nuevamente la imagen
for (x,y,w,h) in faces:
    cv.rectangle(img, (x,y), (x+w,y+h), (255,0,0),2)
plt.figure(figsize=(13,8))
img=cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```



- El ciclo *for* por cada iteración toma los 4 elementos  $(x,y,w,h)$  que hay en cada sublista
- $(x,y)$  : coordenada de inicial
- $(x+w,y+h)$ : coordenada final
- $(255,0,0)$  : Color del rectángulo
- 2: Grosor de la línea

# Buscando los ojos en una imagen

```
#Buscando los ojos en una imagen
img = cv.imread('figuras/bbt.jpeg')
img_grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
eye_cascade = cv.CascadeClassifier('datos/haarcascade_eye.xml')
eyes = eye_cascade.detectMultiScale(img_grey, 1.1 , 5)
for (x,y,w,h) in eyes:
    cv.rectangle(img, (x,y), (x+w,y+h), (255,255,255),2)
plt.figure(figsize=(13,8))
img =cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```

```
#Buscando los ojos en una imagen
img = cv.imread('figuras/bbt.jpeg')
img_grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
eye_cascade = cv.CascadeClassifier('datos/haarcascade_eye.xml')
eyes = eye_cascade.detectMultiScale(img_grey, 1.1 , 5)
for (x,y,w,h) in eyes:
    cv.rectangle(img, (x,y), (x+w,y+h), (255,255,255),2)
plt.figure(figsize=(13,8))
img =cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```

<matplotlib.image.AxesImage at 0x12dfc2f8>

- Seguimos el mismo procedimiento, pero es posible realizarlo en una sola celda
- Se utiliza el archivo **haarcascade\_eye.xml**





# Buscando sonrisas en una imagen

```
#Buscando los sonrisas en una imagen
img = cv.imread('figuras/bbt.jpeg')
img_grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
smile_cascade = cv.CascadeClassifier('datos/haarcascade_smile.xml')
smiles = smile_cascade.detectMultiScale(img_grey, 1.3, 20)
for (x,y,w,h) in smiles:
    cv.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
plt.figure(figsize=(13,8))
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```

```
#Buscando los sonrisas en una imagen
img = cv.imread('figuras/bbt.jpeg')
img_grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
smile_cascade = cv.CascadeClassifier('datos/haarcascade_smile.xml')
smiles = smile_cascade.detectMultiScale(img_grey, 1.3, 20)
for (x,y,w,h) in smiles:
    cv.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
plt.figure(figsize=(13,8))
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```

<matplotlib.image.AxesImage at 0x12e2ba60>

- Seguimos el mismo procedimiento, pero es posible realizarlo en una sola celda
- Se utiliza el archivo **haarcascade\_smile.xml**





# Combinando la detección del rostro, ojos y sonrisa

```
img = cv.imread('figuras/bbt.jpeg')
for (x,y,w,h) in faces:
    cv.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 3)
    for (x_s,y_s,w_s,h_s) in eyes:
        if( (x <= x_s) and (y <= y_s) and ( x+w >= x_s+w_s) and ( y+h >= y_s+h_s)):
            cv.rectangle(img, (x_s,y_s), (x_s+w_s,y_s+h_s), (255,255,255), 3)
    for (x_s,y_s,w_s,h_s) in smiles:
        if( (x <= x_s) and (y <= y_s) and ( x+w >= x_s+w_s) and ( y+h >= y_s+h_s)):
            cv.rectangle(img, (x_s,y_s), (x_s+w_s,y_s+h_s), (0,255,0), 3)
plt.figure(figsize=(13,8))
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```

```
img = cv.imread('figuras/bbt.jpeg')
for (x,y,w,h) in faces:
    cv.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 3)
    for (x_s,y_s,w_s,h_s) in eyes:
        if( (x <= x_s) and (y <= y_s) and ( x+w >= x_s+w_s) and ( y+h >= y_s+h_s)):
            cv.rectangle(img, (x_s,y_s), (x_s+w_s,y_s+h_s), (255,255,255), 3)
    for (x_s,y_s,w_s,h_s) in smiles:
        if( (x <= x_s) and (y <= y_s) and ( x+w >= x_s+w_s) and ( y+h >= y_s+h_s)):
            cv.rectangle(img, (x_s,y_s), (x_s+w_s,y_s+h_s), (0,255,0), 3)
plt.figure(figsize=(13,8))
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```

<matplotlib.image.AxesImage at 0x1bc1f130>

- En este caso restringimos que solo se muestren los ojos y sonrisas que están dentro del recuadro de un rostro.

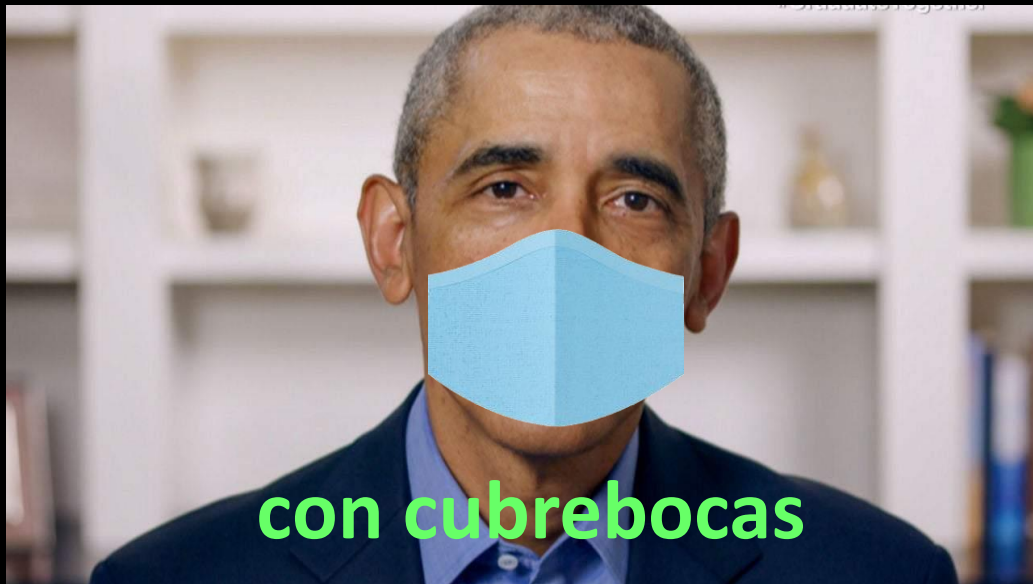


# EJEMPLO: Implementación en una webcam

- Se adjunta el archivo smile\_webcam.py que se puede correr en tiempo real usando una webcam.
- Utilizar el idle de Python para ejecutarlo
- Presionar la letra “q” para cerrar el video
- Tarda unos segundos en comenzar

```
1 import numpy as np
2 import cv2 as cv
3
4 cap = cv.VideoCapture(0)
5 eye_cascade = cv.CascadeClassifier('datos/haarcascade_eye.xml')
6 face_cascade = cv.CascadeClassifier('datos/haarcascade_frontalface_default.xml')
7 smile_cascade = cv.CascadeClassifier('datos/haarcascade_smile.xml')
8
9 def gris(image):
10     return cv.cvtColor(img, cv.COLOR_BGR2GRAY)
11
12 def rostros(image):
13     return face_cascade.detectMultiScale(gris(image), 1.12, 5)
14
15 def ojos(image):
16     return eye_cascade.detectMultiScale(gris(image), 1.1, 5)
17
18 def sonrisas(image):
19     return smile_cascade.detectMultiScale(gris(image), 1.3, 20)
20
21
22
23 while(True):
24     # Capturamos frame por frame
25     ret, frame = cap.read()
26
27     # Desplegamos el frame
28     img = frame
29     faces = rostros(img)
30     eyes = ojos(img)
31     smiles = sonrisas(img)
32     for (x,y,w,h) in faces:
33         cv.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 3)
34         for (x_s,y_s,w_s,h_s) in eyes:
35             if( (x <= x_s) and (y <= y_s) and (x+w >= x_s+w_s) and (y+h >= y_s+h_s)):
36                 cv.rectangle(img, (x_s,y_s), (x_s+w_s,y_s+h_s), (255,255,255), 3)
37         for (x_s,y_s,w_s,h_s) in smiles:
38             if( (x <= x_s) and (y <= y_s) and (x+w >= x_s+w_s) and (y+h >= y_s+h_s)):
39                 cv.rectangle(img, (x_s,y_s), (x_s+w_s,y_s+h_s), (0,255,0), 3)
40     cv.imshow('frame',img)
41     # Presionar q para terminar
42     if cv.waitKey(1) & 0xFF == ord('q'):
43         break
44
45 #Se cierra la ventana
46 cap.release()
47 cv.destroyAllWindows()
```





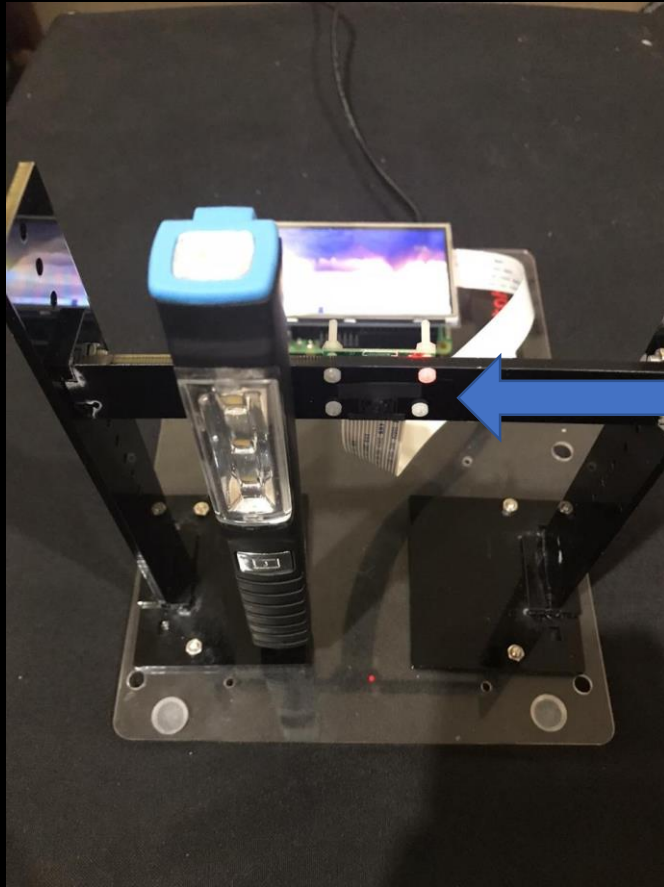
**Parte B**  
**Detector de cubrebocas**

# Actividad

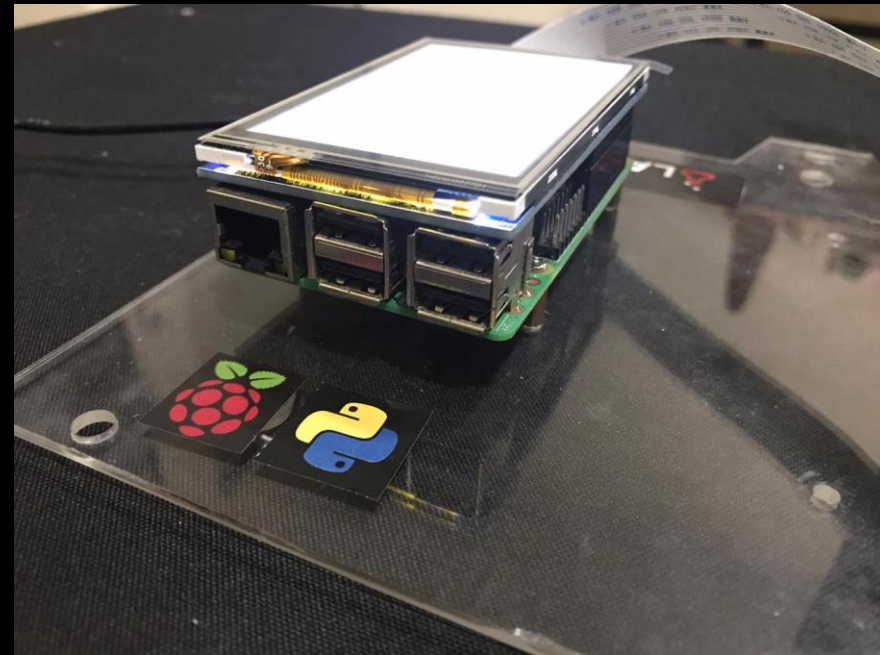
- Realizar este ejercicio en un nuevo notebook.
- **Escribir el código necesario para un detector de cubrebocas en imágenes.**
- Pruébalo con 5 imágenes con cubrebocas y 5 imágenes sin cubrebocas. Es necesario agregar la leyenda en verde (con cubrebocas) y rojo (sin cubrebocas).
- Utilizar los siguientes clasificadores
  - ❖ haarcascade\_eye.xml
  - ❖ haarcascade\_frontalface\_alt.xml
  - ❖ Haarcascade\_mouth.xml
- *OPCIONAL: Implementarlo con una webcam, esto tendrá que realizarse en el IDLE de Python con el nombre detector\_cubrebocas\_NOMBRE\_APELLIDO.py.*

# Ejemplo de aplicación en la RaspberryPi

- Este algoritmo se puede implementar en una RaspberryPi para la detección de cubrebocas en tiempo real



Cámara



RaspberryPi con pantalla





# Instrucciones

- Guarde los notebooks con los nombres  
**SDC\_7A\_Nombre\_Apellido.ipynb**  
**SDC\_7B\_Nombre\_Apellido.ipynb**
- Comprima toda su carpeta en un archivo .ZIP con el nombre **SDC\_7\_Nombre\_Apellido.ZIP** (debe incluir los dos notebooks y sus imágenes).
- Incluir también el archivo *detector\_cubrebocas\_NOMBRE\_APELLIDO.py* en caso de haberlo realizado, no es obligatorio este archivo.
- En caso de que sea muy grande el archivo ZIP, subirlo a su Google drive o alguna plataforma en la nube y compartirme el link.