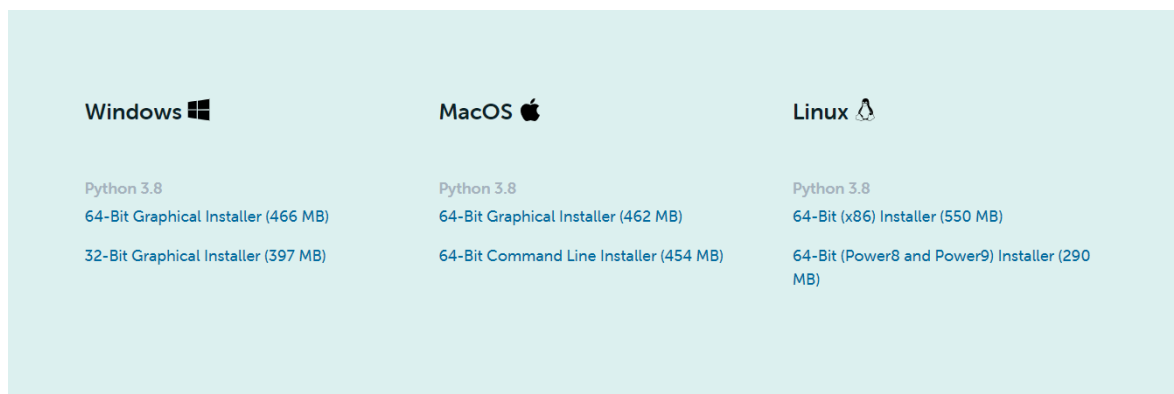


## Documento técnico

Para preparar nuestro ambiente de trabajo seguiremos los siguientes pasos

### 1. Instalación de Anaconda

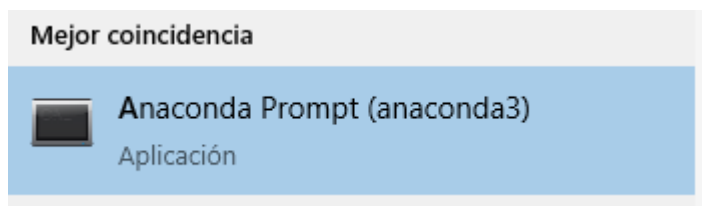
Para descargarla, basta con ir a su sitio web y seleccionar la opción “individual” <https://www.anaconda.com/products/individual> ya que también ofrece opciones de pago para empresas o trabajos específicos.




En su página web nos desplazamos a la parte de abajo donde encontraremos las opciones de instalación para los diferentes sistemas operativos soportados. Como se estará utilizando la línea de comandos propia de Anaconda, las instrucciones de instalación para las paqueterías serán las mismas para Linux y Windows.

Instalar Anaconda es muy conveniente para este tipo de proyectos, pues nos permite tener varios ambientes de trabajo en donde tengamos instaladas paqueterías específicas para ese trabajo. Cosa que es muy útil pues por temas de compatibilidad algunas veces tendremos que instalar versiones específicas de ciertas paqueterías para que un programa funcione correctamente y puede que eso afecte a otros proyectos, con anaconda nos libramos de ese inconveniente además de que en su instalación ya se incorpora Python, su propia línea de comandos, como ya se había mencionado, entre otras aplicaciones interesantes como Spyder o jupyter (El cual también utilizaremos).

Una vez instalada anaconda y considerando toda la paquetería específica que vamos a utilizar, lo más conveniente es crear un ambiente de trabajo específico para ello procederemos a abrir el prom de Anaconda



Al iniciar la aplicación, vamos a ver que el ambiente de trabajo es el que estamos utilizando



```
(base) C:\Users\windows>_
```

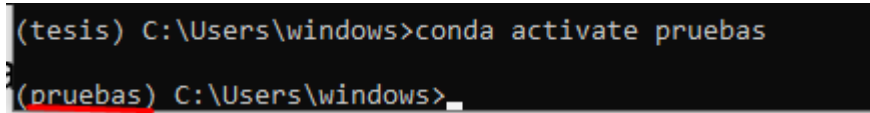
Para crear un nuevo ambiente de trabajo basta con escribir

```
$conda create -n pruebas python=3.7
```

Es muy importante colocar en especifico la versión de Python 3.7

Lo siguiente es entrar a nuestro ambiente recién creado, esto mediante el comando:

```
$ conda activate pruebas
```



```
(tesis) C:\Users\windows>conda activate pruebas  
(pruebas) C:\Users\windows>_
```

## 2. Instalar Jupyter notebook

Para realizar nuestras pruebas de manera más controlada, usaremos Jupyter notebook, para esto usaremos el comando:

```
$python -m pip install jupyter
```

Luego, para abrir el programa ejecutamos

```
$jupyter nootbook
```

## 3. Instalar Numpy 1.16.5

Lo siguiente es instalar numpy 1.16.5, usando el comando

```
$pip install numpy==1.16.5
```

Es muy importante instalar esta versión de numpy **antes** de instalar keras.

## 4 Instalar Keras 2.3.1

La instalación de Keras es sencilla, pues basta con escribir en la línea de comandos de anaconda y escribir

```
$pip install keras==2.3.1
```

Página web: <https://keras.io/api/>

## 5 Instalar TensorFlow 1.14

La instalación de TensorFlow es semejante a la de keras, basta con colocar en la línea de comandos

```
$pip install TensorFlow==1.14
```

Página web: <https://www.tensorflow.org/tutorials?hl=es-419>

Si hemos realizado estas instalaciones de manera correcta, obtendremos el siguiente resultado al ejecutar

```
$python
```

```
>>import keras
```

```
Python 3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import keras
Using TensorFlow backend.
```

En caso de que se tuviera algún problema podría ser relacionado con la versión de “pip” por lo que se sugiere el siguiente comando

```
$pip install --upgrade pip
```

## 6 instalar Opencv

Esta vez basta con instalar la versión por defecto con el siguiente comando:

```
$pip install openCv-python
```

Para probar que todo está en orden, escribimos en la línea de comandos

```
$python
```

```
>>>import cv2
```

Página web: <https://opencv.org/>

Al dar enter, no debería aparecer ningún mensaje de error.

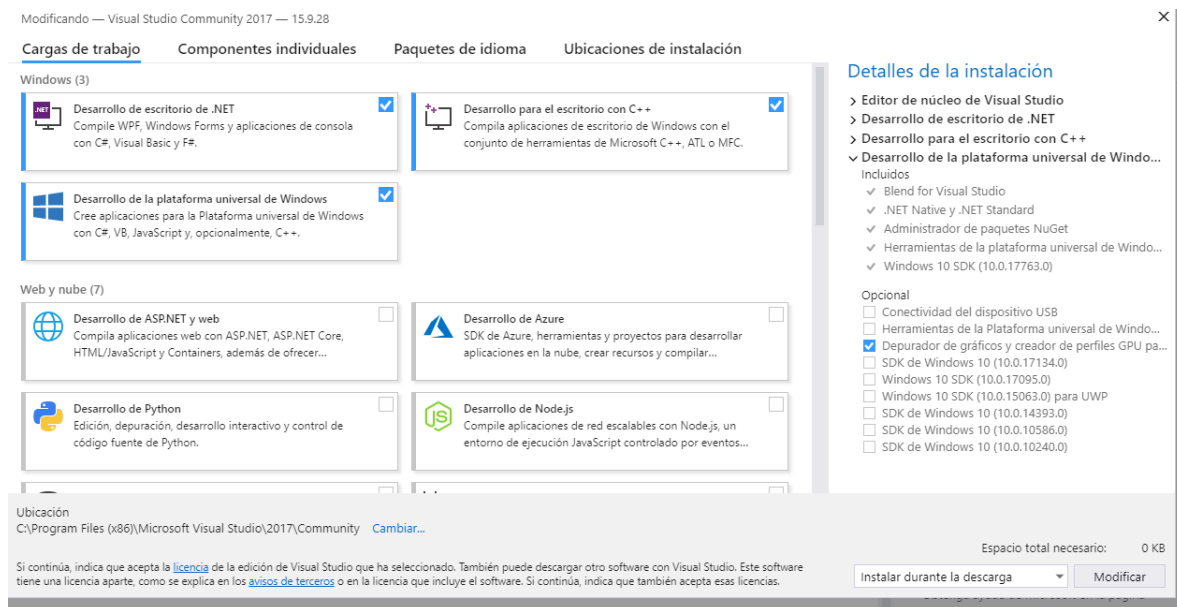
## 7 instalar cmake y dlib

Para usar el algoritmo de HOG para detección de rostros, usaremos la librería *dlib* al menos para la detección de rostros, pues para la detección de siluetas podremos utilizar, para ello la instalaremos como cualquier otra librería de Python, con la excepción de que antes tendremos que instalar la librería *cmake*. Por lo tanto, en la terminar de Anaconda:

```
$ pip install cmake
```

Ahora para que *dlib* se instale correctamente, es necesario que se instalen compiladores de *c++*, esto lo podremos hacer desde el visual studio installer, lo podremos descargar desde la página <https://visualstudio.microsoft.com/es/downloads/>

Luego abrimos el visual studio installer



Y seleccionamos esas opciones. Es importante aclarar que estas instalaciones son pesadas, por lo que podrían omitirse o bien considerar el espacio en el disco duro pues el peso que tienen ronda los 8 GB.

Luego de esto podremos realizar la instalación

```
$ pip install dlib
```

y nuevamente si queremos corroborar la instalación, ejecutamos el comando

```
$python  
>> import dlib
```

Y no tendrá que arrojar ningún error.

## 8 instalación de MTCNN

La instalación de MTCNN, es bastante sencilla, es idéntico a instalar un paquete en Python.

Entramos a su página web (<https://pypi.org/project/mtcnn/>) y se nos señalan las instrucciones necesarias para su implementación, donde una de ellas, es la instalación de tensor Flow, pero al ya tenerlo instalado, podemos omitir este paso por lo que solo nos queda la instalación de MTCNN en sí, por lo que bastaría con utilizar el comando pip para su instalación

```
$pip install mtcnn
```

Si hemos instalado keras y tensor Flow de manera adecuada al realizar los siguientes comandos

```
$python
```






```
>>import mtcnn
```

```
(pruebas) C:\Users\windows>python
Python 3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import mtcnn
Using TensorFlow backend.
```

No debería aparecer algún error.

## 9 Estructura de carpetas














La estructura de carpetas final sería la siguiente

- ▼  Ambiente de trabajo
  - >  Códigos
  -  Imágenes
  - >  Videos
  - >  xmls

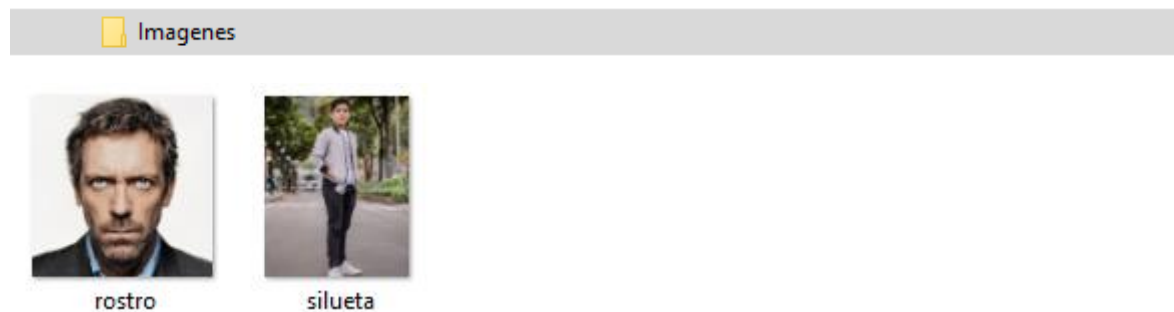
Luego dentro de Códigos encontraremos:

- ▼  Códigos
  - >  jupyter
  -  python

Es claro que la carpeta jupyter contiene los archivos .ipynb propios del Jupyter notebook. Que son los que usaremos para realizar las pruebas.

 .ipynb_checkpoints	08/11/2020 06:16 p. m.	Carpeta de archivos	
 Algoritmo HAAR Cascade-MTCNN.ipynb	06/05/2020 12:41 p. m.	Archivo IPYNB	8 KB
 Cambio de FPS.ipynb	08/11/2020 05:04 p. m.	Archivo IPYNB	4 KB
 Faster R-CNN Completo.ipynb	08/11/2020 06:30 p. m.	Archivo IPYNB	12 KB
 Faster R-CNN Simple.ipynb	04/11/2020 03:53 p. m.	Archivo IPYNB	5 KB
 Haar Cascade Completo.ipynb	08/11/2020 06:05 p. m.	Archivo IPYNB	8 KB
 Haar Cascade simple.ipynb	04/11/2020 02:11 p. m.	Archivo IPYNB	2 KB
 HOG Rostros Completo.ipynb	08/11/2020 06:12 p. m.	Archivo IPYNB	8 KB
 HOG rostros Simple.ipynb	04/11/2020 04:10 p. m.	Archivo IPYNB	3 KB
 HOG Silueta Completo.ipynb	08/11/2020 06:18 p. m.	Archivo IPYNB	8 KB
 Hog Simple Silueta.ipynb	04/11/2020 03:13 p. m.	Archivo IPYNB	3 KB
 MTCNN Completo.ipynb	08/11/2020 06:22 p. m.	Archivo IPYNB	8 KB
 MTCNN simple.ipynb	04/11/2020 02:58 p. m.	Archivo IPYNB	3 KB

Dentro de la carpeta imágenes únicamente encontraremos 2 imágenes que nos serán de utilidad para comprobar que los algoritmos de detección de silueta y rostros funcionan



Dentro de la carpeta de vídeos encontraremos dos carpetas:



Dentro de Datasets irán los conjuntos de datos que estaremos utilizando



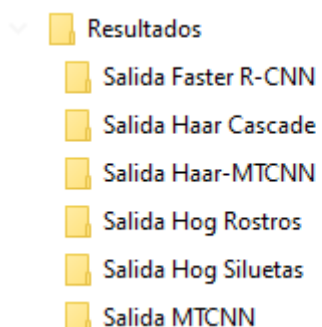
Y a su vez dentro de cada Dataset encontraremos una estructura parecida a esta:



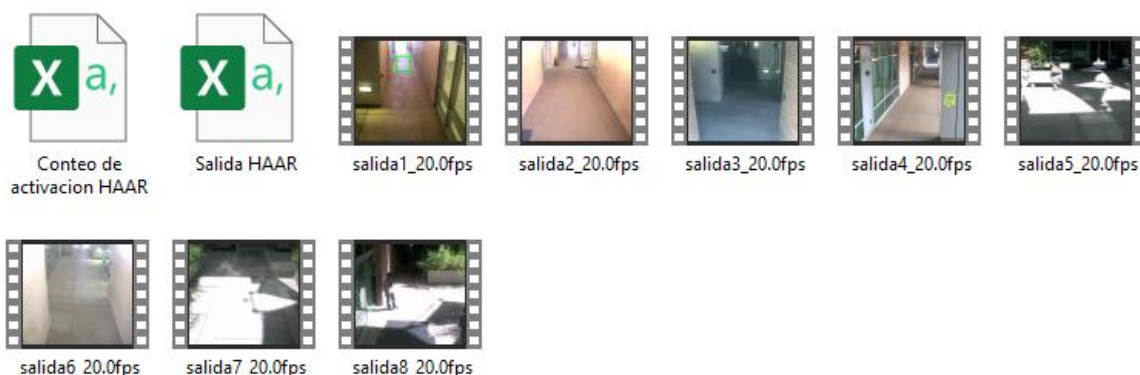
Esta es la imagen perteneciente a el conjunto de datos BEHAVE 352 x 288.

Las subcarpetas con el nombre FPS 3, FPS 4 y FPS 5 son evidencia de como funciona el algoritmo a esos cuadros por segundo, mientras que los videos nombrados con "video\_1", "video\_2", etc. Son los videos del conjunto de datos **originales**, únicamente renombrados y re escalados a una resolución de 352 x 288 pixeles para una mayor facilidad de procesamiento.

Siguiendo en la carpeta videos, su otra subcarpeta es Resultados, la cual contendrá las carpetas creadas por los programas y tendrá una vista similar a esta:

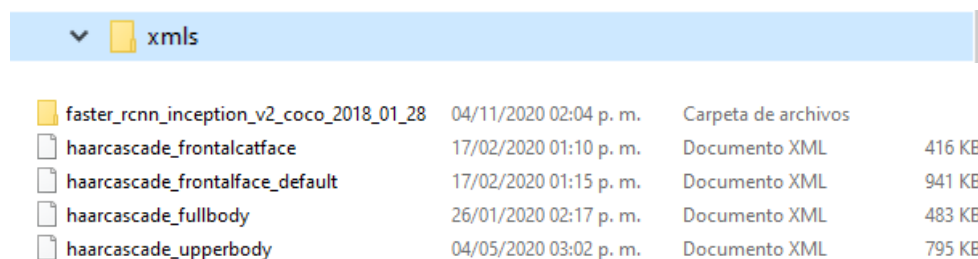


Dentro de cada “Salida” podremos encontrar los siguientes resultados, tomando como ejemplo la carpeta “salida HAAR”






















La cual guardará los archivos de video resultantes del procesamiento por un algoritmo en específico, así como 2 archivos .csv, los cuales contendrán información perteneciente al procesamiento del conjunto de datos.

Por último, tenemos la carpeta xmls, la cual contiene xmls y elementos necesarios para el funcionamiento de la aplicación que se explicaran más a detalle a continuación



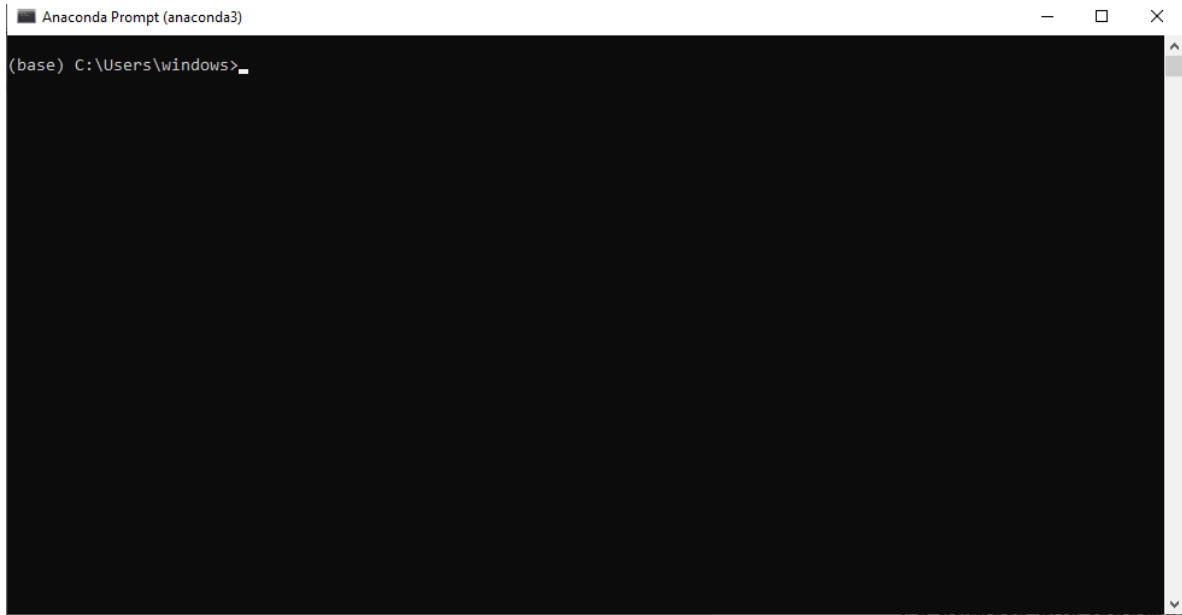


Un esquema general sería el siguiente:

- ▼  Ambiente de trabajo
  - ▼  Códigos
    - >  jupyter
    -  python
    -  Imágenes
  - ▼  Videos
  - ▼  Datasets
    - >  BEHAVE 352 x 288
    - >  CamNet 352 x 288
    - >  CAVIAR 352 x 288
  - ▼  Resultados
    -  Salida Faster R-CNN
    -  Salida Haar Cascade
    -  Salida Haar-MTCNN
    -  Salida Hog Rostros
    -  Salida Hog Siluetas
    -  Salida MTCNN
  - ▼  xmls
    - >  faster\_rcnn\_inception\_v2\_coco\_2018\_01\_28

Ahora vamos a correr uno de los programas, lo primero es tener instalada toda la paquetería necesaria explicada anteriormente y los datasets que necesitemos, estos últimos vendrán incluidos en la carpeta llama “Ambiente de trabajo”.

Lo primero que tenemos que hacer, es abrir el prompt de anaconda



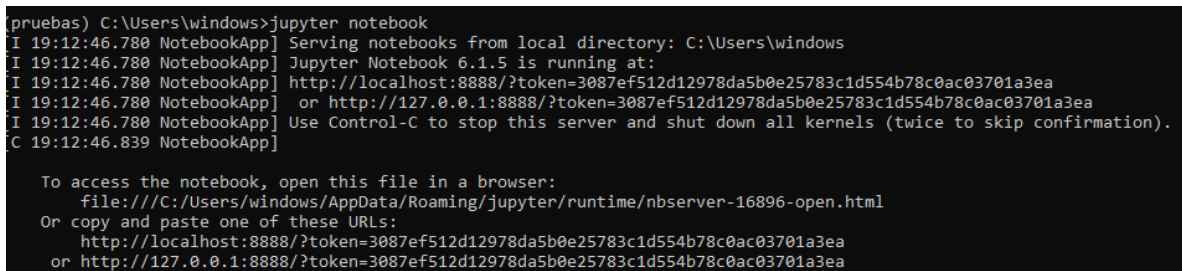
```
Anaconda Prompt (anaconda3)
(base) C:\Users\windows>
```

Posteriormente tendremos que entrar a nuestro ambiente de desarrollo creado anteriormente



```
Anaconda Prompt (anaconda3)
(base) C:\Users\windows>conda activate pruebas
(pruebas) C:\Users\windows>
```

Luego ejecutamos jupyter



```
Anaconda Prompt (anaconda3)
(pruebas) C:\Users\windows>jupyter notebook
[I 19:12:46.780 NotebookApp] Serving notebooks from local directory: C:\Users\windows
[I 19:12:46.780 NotebookApp] Jupyter Notebook 6.1.5 is running at:
[I 19:12:46.780 NotebookApp] http://localhost:8888/?token=3087ef512d12978da5b0e25783c1d554b78c0ac03701a3ea
[I 19:12:46.780 NotebookApp] or http://127.0.0.1:8888/?token=3087ef512d12978da5b0e25783c1d554b78c0ac03701a3ea
[I 19:12:46.780 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
C 19:12:46.839 NotebookApp]

To access the notebook, open this file in a browser:
    file:///C:/Users/windows/AppData/Roaming/jupyter/runtime/nbserver-16896-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=3087ef512d12978da5b0e25783c1d554b78c0ac03701a3ea
    or http://127.0.0.1:8888/?token=3087ef512d12978da5b0e25783c1d554b78c0ac03701a3ea
```

Lo que abrirá en su navegador un explorador de archivos situado desde la raíz, aquí tenemos que navegar hasta la carpeta “Ambiente de trabajo”

<input type="checkbox"/> 0	Documents	Name	Last Modified	File size
		..	hace unos segundos	
<input type="checkbox"/>	Ambiente de trabajo		hace 4 días	
<input type="checkbox"/>	BEDU		hace 4 días	
<input type="checkbox"/>	Documentos personales		hace 2 meses	
<input type="checkbox"/>	ecommerce		hace 2 meses	
<input type="checkbox"/>	GitHub		hace 18 días	

Luego pasamos a Codigos>jupyter

<input type="checkbox"/> 0	Documents / Ambiente de trabajo / Códigos / jupyter	Name	Last Modified	File size
		..	hace unos segundos	
<input type="checkbox"/>	Algoritmo HAAR Cascade-MTCNN.ipynb		hace 17 minutos	6.37 kB
<input type="checkbox"/>	Cambio de FPS.ipynb		hace 2 horas	3.29 kB
<input type="checkbox"/>	Faster R-CNN Completo.ipynb		hace una hora	11.5 kB
<input type="checkbox"/>	Faster R-CNN Simple.ipynb		hace 4 días	4.8 kB
<input type="checkbox"/>	Haar Cascade Completo.ipynb		hace 17 minutos	7.31 kB
<input type="checkbox"/>	Haar Cascade simple.ipynb		hace 4 días	1.88 kB
<input type="checkbox"/>	HOG Rostros Completo.ipynb		hace una hora	8.1 kB
<input type="checkbox"/>	HOG rostros Simple.ipynb		hace 4 días	2.77 kB
<input type="checkbox"/>	HOG Silueta Completo.ipynb		hace una hora	7.89 kB
<input type="checkbox"/>	Hog Simple Silueta.ipynb		hace 4 días	2.26 kB
<input type="checkbox"/>	MTCNN Completo.ipynb		hace una hora	7.73 kB
<input type="checkbox"/>	MTCNN simple.ipynb		hace 4 días	2.3 kB

Ahora solo resta abrir cualquiera de todos los ejemplos

```
jupyter MTCNN simple Last Checkpoint: el miércoles pasado a las 14:14 (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [1]: from mtcnn.mtcnn import MTCNN
import cv2
import os

Using TensorFlow backend.

In [2]: #Seleccionamos el detector
detector = MTCNN()
#Seleccionamos
threshold = 0.7
#Seleccionamos la imagen
frame = cv2.imread('../Imagenes/rostro.jpg')

#Activamos el detector
face_locations = detector.detect_faces(frame)
#Si el detector consigue detectar algo
if len(face_locations):
    #Se recorren el número que fueron detectadas
    for face in zip(face_locations):
        #Se obtienen las coordenadas de las caras encontradas
        (x, y, w, h) = face[0]['box']
        #Se dibuja el rectángulo sobre el frame
        cv2.rectangle(frame, (x,y), (x+w,y+h), (255,0,0), 2)

#Se muestra la imagen
cv2.imshow('MTCNN DL', frame)
cv2.waitKey(0);
```

## 10. Instalación e implementación de Haar Cascade

Una vez instalado OpenCv podemos hacer uso de Haar Cascade, para esto OpenCv nos ofrece una gran variedad de clasificadores pre entrenados (Estos vendrán en archivos XML), donde encontramos no solo de rostros y siluetas, también de partes más específicas del cuerpo, como ojos, boca, cejas, etc. Estos los podemos encontrar en el repositorio de OpenCV en github: <https://github.com/opencv/opencv/tree/master/data/haarcascades>

Para la detección de siluetas usaremos el *haarcascade\_fullbody.xml*

haarcascade_eye.xml	some attempts to tune the performance	7 years ago
haarcascade_eye_tree_eyeglasses.xml	some attempts to tune the performance	7 years ago
haarcascade_frontalcatface.xml	fix files permissions	6 months ago
haarcascade_frontalcatface_extended.xml	fix files permissions	6 months ago
haarcascade_frontalface_alt.xml	some attempts to tune the performance	7 years ago
haarcascade_frontalface_alt2.xml	some attempts to tune the performance	7 years ago
haarcascade_frontalface_alt_tree.xml	some attempts to tune the performance	7 years ago
haarcascade_frontalface_default.xml	some attempts to tune the performance	7 years ago
haarcascade_fullbody.xml	Some mist. typo fixes	3 years ago
haarcascade_lefteye_2splits.xml	some attempts to tune the performance	7 years ago
haarcascade_licence_plate_rus_16stages.xml	Added Haar cascade for russian cars licence plate detection, 16 stage...	7 years ago
haarcascade_lowerbody.xml	Some mist. typo fixes	3 years ago
haarcascade_profileface.xml	some attempts to tune the performance	7 years ago
haarcascade_righteye_2splits.xml	some attempts to tune the performance	7 years ago
haarcascade_russian_plate_number.xml	Create haarcascade_russian_plate_number.xml	7 years ago
haarcascade_smile.xml	fixing models to resolve XML violation issue	3 years ago
haarcascade_upperbody.xml	Some mist. typo fixes	3 years ago

Mientras que para la detección de rostros usaremos el:  
*haarcascade\_frontalface\_default.xml*

haarcascade_eye.xml	some attempts to tune the performance	7 years ago
haarcascade_eye_tree_eyeglasses.xml	some attempts to tune the performance	7 years ago
haarcascade_frontalcatface.xml	fix files permissions	6 months ago
haarcascade_frontalcatface_extended.xml	fix files permissions	6 months ago
haarcascade_frontalface_alt.xml	some attempts to tune the performance	7 years ago
haarcascade_frontalface_alt2.xml	some attempts to tune the performance	7 years ago
haarcascade_frontalface_alt_tree.xml	some attempts to tune the performance	7 years ago
haarcascade_frontalface_default.xml	some attempts to tune the performance	7 years ago
haarcascade_fullbody.xml	Some mist. typo fixes	3 years ago
haarcascade_lefteye_2splits.xml	some attempts to tune the performance	7 years ago
haarcascade_licence_plate_rus_16stages.xml	Added Haar cascade for russian cars licence plate detection, 16 stage...	7 years ago
haarcascade_lowerbody.xml	Some mist. typo fixes	3 years ago
haarcascade_profileface.xml	some attempts to tune the performance	7 years ago
haarcascade_righteye_2splits.xml	some attempts to tune the performance	7 years ago
haarcascade_russian_plate_number.xml	Create haarcascade_russian_plate_number.xml	7 years ago
haarcascade_smile.xml	fixing models to resolve XML violation issue	3 years ago
haarcascade_upperbody.xml	Some mist. typo fixes	3 years ago

Ahora un pequeño ejemplo de la aplicación del modelo

```
#Importamos la paquetería necesaria
import cv2
import os

#Cargamos el modelo, en este caso será haarcascade_frontalface_default.xml o
haarcascade_fullbody.xml
person_cascade = cv2.CascadeClassifier( '../..xmls/haarcascade_frontalface_
default.xml')
print(person_cascade)
#Seleccionamos la imagen
frame = cv2.imread('../..Imagenes/rostro.jpg')

#El Haar-cascade classifier necesita escala de grises
gray_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

rects = person_cascade.detectMultiScale(gray_frame)

for (x, y, w, h) in rects:
    #Dibujar la caja
    cv2.rectangle(frame, (x,y), (x+w,y+h),(0,255,0),2)
#Mostramos la cara dibujada
cv2.imshow("Haar Cascade", frame)
cv2.waitKey(0);
```

## 11. Instalación e implementación de HOG

De esta manera, podremos usar para la detección de rostros

```
hogFaceDetector = dlib.get_frontal_face_detector()
```

y para la detección de siluetas

```
hog = cv2.HOGDescriptor()  
hog.setSVMDetector( cv2.HOGDescriptor_getDefaultPeopleDetector() )
```

La razón de utilizar dos detectores diferentes, es porque la librería cv2 no cuenta con un método “per se” para la detección de rostros, aunque si que puede ser construido ajustando ciertos parámetros, tal como se muestra en este enlace <https://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained/>.

Por ello recurrimos al modelo pre entrenado de detección de rostros que dlib nos permite implementar.

Ahora un ejemplo de la detección de rostros haciendo uso de HOG

```
#Importamos las librerías necesarias  
import cv2  
import os  
import dlib  
  
#Definimos el método que detectará los rostros  
def detectFaceDlibHog(detector, frame, inHeight=300, inWidth=0):  
  
    frameDlibHog = frame.copy()  
    frameHeight = frameDlibHog.shape[0]  
    frameWidth = frameDlibHog.shape[1]  
    if not inWidth:  
        inWidth = int((frameWidth / frameHeight)*inHeight)  
  
    scaleHeight = frameHeight / inHeight  
    scaleWidth = frameWidth / inWidth  
  
    frameDlibHogSmall = cv2.resize(frameDlibHog, (inWidth, inHeight))  
  
    frameDlibHogSmall = cv2.cvtColor(frameDlibHogSmall, cv2.COLOR_BGR2RGB)  
    faceRects = detector(frameDlibHogSmall, 0)  
  
    bboxes = []  
    for faceRect in faceRects:  
  
        cvRect = [int(faceRect.left()*scaleWidth), int(faceRect.top()*scaleHeight),  
int(faceRect.right()*scaleWidth), int(faceRect.bottom()*scaleHeight)],
```

```

            int(faceRect.right()*scaleWidth), int(faceRect.bottom()*scaleHeight) ]
        bboxes.append(cvRect)
        cv2.rectangle(frameDlibHog, (cvRect[0], cvRect[1]), (cvRect[2], cvRect[3]), (0, 255, 0), int(round(frameHeight/150)), 4)
    return frameDlibHog, bboxes

if __name__ == '__main__':
    #Seleccionamos nuestro detector
    hogFaceDetector = dlib.get_frontal_face_detector()

    #Seleccionamos la imagen
    frame = cv2.imread("Path de la imagen")

    #Definimos tamaño para redimensionar
    frame_width = 352
    frame_height = 288

    #Redimensionamos la imagen
    frame = cv2.resize(frame, (frame_width, frame_height))
    #Pasamos el detector al método junto con la imagen
    found, bboxes = detectFaceDlibHog(hogFaceDetector, frame)
    #Mostramos el resultado
    cv2.imshow('HOG Rostros', found)

```

Ahora pasemos a la detección de siluetas

```

#Importamos las librerías
import numpy as np
import cv2
import time
import os

def inside(r, q):
    rx, ry, rw, rh = r
    qx, qy, qw, qh = q
    return rx > qx and ry > qy and rx + rw < qx + qw and ry + rh < qy + qh

#Definimos el método que dibujará los cuadrados sobre el área detectada
def draw_detections(img, rects, thickness = 1):
    for x, y, w, h in rects:

```

```

        #El detector HOG regresa rectangulos un poco más grandes que los objetos
        a los que identifica
        # Por lo que se procede a recortarlos un poco
        pad_w, pad_h = int(0.15*w), int(0.05*h)
        cv2.rectangle(img, (x+pad_w, y+pad_h), (x+w-pad_w, y+h-
pad_h), (0, 255, 0), thickness)

if __name__ == '__main__':
    #Escogemos nuestro descriptor
    hog = cv2.HOGDescriptor()

    #Inicializamos el descriptor de HOG
    hog.setSVMDetector( cv2.HOGDescriptor_getDefaultPeopleDetector() )

    #Definimos el tamaño para redimensionar
    frame_width = 352
    frame_height = 288

    #Redimensionamos la imagen
    frame = cv2.resize(frame, (frame_width, frame_height))

    #Pasamos el detector
    found,w=hog.detectMultiScale(frame, winStride=(8,8), padding=(32,32), scale=
1.05)

    #Dibujamos los cuadros obtenidos desde el detector sobre la imagen original
    draw_detections(frame,found)

    #Mostramos la imagen
    cv2.imshow('HOG siluetas',frame)
    cv2.waitKey(0);

```



## 12. Implementación MTCNN

Para utilizarlo simplemente lo tratamos como una librería en python

Un ejemplo de su implementación es el siguiente:

```
#Se Importan las librerías necesarias
from mtcnn.mtcnn import MTCNN
import cv2
import os

if __name__ == '__main__':

    #Seleccionamos el detector
    detector = MTCNN()
    #Seleccionamos
    threshold = 0.7
    #Seleccionamos la imagen
    frame = cv2.imread("Path de la imagen")

    #Activamos el detector
    face_locations = detector.detect_faces(frame)
    #Si el detector consigue detectar algo
    if len(face_locations):
        #Se recorren el número que fueron detectadas
        for face in zip(face_locations):
            #Se obtienen las coordenadas de las caras encontradas
            (x, y, w, h) = face[0]['box']
            #Se dibuja el rectángulo sobre el frame
            cv2.rectangle(frame, (x,y), (x+w,y+h), (255,0,0), 2)

    #Se muestra la imagen
    cv2.imshow('MTCNN DL', frame)
    cv2.waitKey(0);
```

## 13. Instalación e implementación de Faster-R CNN

Para la siguiente tarea, utilizaremos un modelo pre-entrenado de red neuronal llamado COCO (Common Objects in Context <https://arxiv.org/pdf/1405.0312.pdf>), la cual es una red neuronal desarrollada por Microsoft para la multi clasificación de objetos (más de 80) y en la que los humanos son uno de esos objetos a clasificar. Para el uso de esta es necesario TensorFlow instalado en Python y de algunas otras librerías como lo son numpy.

Además de que necesitamos el modelo previamente entrenado de la red neuronal. Para hacerlo, necesitamos acceder a la siguiente liga:

[https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/detection\\_model\\_zoo.md#coco-trained-models-coco-models](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md#coco-trained-models-coco-models)

Donde nosotros descargaremos el siguiente modelo: faster\_rcnn\_inception\_v2\_coco.tar.gz

<a href="#">ssd_resnet_50_fpn_coco</a> ☆	76	35	Boxes
<a href="#">ssd_mobilenet_v2_coco</a>	31	22	Boxes
<a href="#">ssd_mobilenet_v2_quantized_coco</a>	29	22	Boxes
<a href="#">ssdlite_mobilenet_v2_coco</a>	27	22	Boxes
<a href="#">ssd_inception_v2_coco</a>	42	24	Boxes
<b>faster_rcnn_inception_v2_coco</b>	58	28	Boxes
<a href="#">faster_rcnn_resnet50_coco</a>	89	30	Boxes
<a href="#">faster_rcnn_resnet50_lowproposals_coco</a>	64		Boxes
<a href="#">rfcn_resnet101_coco</a>	92	30	Boxes
<a href="#">faster_rcnn_resnet101_coco</a>	106	32	Boxes
<a href="#">faster_rcnn_resnet101_lowproposals_coco</a>	82		Boxes
<a href="#">faster_rcnn_inception_resnet_v2_atrous_coco</a>	620	37	Boxes
<a href="#">faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco</a>	241		Boxes
<a href="#">faster_rcnn_nas</a>	1833	43	Boxes
<a href="#">faster_rcnn_nas_lowproposals_coco</a>	540		Boxes
<a href="#">mask_rcnn_inception_resnet_v2_atrous_coco</a>	771	36	Masks

Ahora veremos cómo se utiliza el modelo anterior haciendo uso del siguiente código

```
import numpy as np
import tensorflow as tf
import cv2
import time

class DetectorAPI:
    def __init__(self, path_to_ckpt):
        self.path_to_ckpt = path_to_ckpt

        self.detection_graph = tf.Graph()
        with self.detection_graph.as_default():
            od_graph_def = tf.GraphDef()
            with tf.gfile.GFile(self.path_to_ckpt, 'rb') as fid:
                serialized_graph = fid.read()
                od_graph_def.ParseFromString(serialized_graph)
                tf.import_graph_def(od_graph_def, name='')

        self.default_graph = self.detection_graph.as_default()
        self.sess = tf.Session(graph=self.default_graph)

        # Definir los tensores de entrada y salida para detection_graph
```

```

        self.image_tensor =
self.detection_graph.get_tensor_by_name('image_tensor:0')
        # Cada caja representa una parte de la imagen donde un objeto en
particular fue detectado
        self.detection_boxes =
self.detection_graph.get_tensor_by_name('detection_boxes:0')
        # Cada puntaje representa un nivel de precisión

        self.detection_scores =
self.detection_graph.get_tensor_by_name('detection_scores:0')
        self.detection_classes =
self.detection_graph.get_tensor_by_name('detection_classes:0')
        self.num_detections =
self.detection_graph.get_tensor_by_name('num_detections:0')

    def processFrame(self, image):
        # El modelo esperado espera imágenes con el siguiente formato [1, None,
None, 3]
        image_np_expanded = np.expand_dims(image, axis=0)
        # Comienza la detección
        start_time = time.time()
        (boxes, scores, classes, num) = self.sess.run(
            [self.detection_boxes, self.detection_scores,
self.detection_classes, self.num_detections],
            feed_dict={self.image_tensor: image_np_expanded})
        end_time = time.time()

        print("Tiempo transcurrido:", end_time-start_time)

        im_height, im_width, _ = image.shape
        boxes_list = [None for i in range(boxes.shape[1])]
        #Comienza a dibujar las cajas en el lugar donde se encontró algo
        for i in range(boxes.shape[1]):
            boxes_list[i] = (int(boxes[0,i,0] * im_height),
                            int(boxes[0,i,1]*im_width),
                            int(boxes[0,i,2] * im_height),
                            int(boxes[0,i,3]*im_width))
        #Regresa las cajas que encontró
        return boxes_list, scores[0].tolist(), [int(x) for x in
classes[0].tolist()], int(num[0])

    def close(self):
        self.sess.close()
        self.default_graph.close()

#Cargamos el path del modelo
model_path = 'C:\\Users\\Juan\\Documents\\VIEP
2019\\faster_rcnn_inception_v2_coco_2018_01_28\\frozen_inference_graph.pb'
odapi = DetectorAPI(path_to_ckpt=model_path)
threshold = 0.7
img = cv2.imread("C:\\Users\\Juan\\Documents\\VIEP 2019\\amigos.jpg")
img = cv2.resize(img, (600, 600))
boxes, scores, classes, num = odapi.processFrame(img)

#Cuerpo Completo

```

```
for i in range(len(boxes)):
    # La clase 1, representa a los humanos
    if classes[i] == 1 and scores[i] > threshold:
        box = boxes[i]
        # Dibujamos los rectángulos en las regiones obtenidas
        cv2.rectangle(img, (box[1], box[0]), (box[3], box[2]), (255, 0, 0), 2)

cv2.imshow("preview", img)
key = cv2.waitKey(0)
```

### 13.Implementación completa de los algoritmos

Los ejemplos anteriores tratan con una sola imagen, a continuación, se mostrará la implementación completa que se había pensado para este proyecto, el cual contempla que se está trabajando con un vídeo, además de crear dos archivos csv donde se almacenarán los siguientes datos:

1. Salida Haar.csv
  - a. Nombre del vídeo
  - b. Tiempo de activación
  - c. Fps
2. Conteo de activaciones.csv
  - a. Nombre del vídeo
  - b. Cuadros totales
  - c. Cuadros activados

Aunado a eso, también se guardará el video de salida y obviamente mostrar el vídeo corriendo con el algoritmo. El siguiente ejemplo será el de

```
import cv2
import time
import os

#Cargamos el modelo, en este caso será haarcascade_frontalface_default.xml o
#haarcascade_fullbody.xml
person_cascade = cv2.CascadeClassifier('../..//xmls/haarcascade_frontalface_d
efault.xml')

rutaDatasets = '../..//Videos/Datasets/'
nombreDataset = 'CamNet 352 x 288/'
#Asigamos un directorio donde se guardarán nuestros archivos
directorioSalida = '../..//Videos/Resultados/Salida Haar Cascade/'

try:
    # Creamos el directorio de salida
    os.mkdir(directorioSalida)
    print( directorioSalida , " Creado ")
except FileExistsError:
    # Si ya existía, se catcha la excepción
    print( directorioSalida , " Ya existia")

try:
    #Creamos los de salida
    f = open(directorioSalida+"Salida HAAR.csv", "x")
    f2 = open(directorioSalida+"Conteo de activacion HAAR.csv", "x")
```

```

except:
    #Si ya existían, se procede a abrirlas
    f = open(directorioSalida+"Salida HAAR.csv", "w")
    f2 = open(directorioSalida+"Conteo de activacion HAAR.csv", "w")

contador = 1
#Se colocan las cabeceras de los archivos
f.write("nombre, tiempo, fps \n")
f2.write("nombre, cuadros Totales, cuadrosActivados \n")

#En este ciclo 'for' se toman en cuenta todos los videos que se quieran procesar
for i in range(8):
    #Esta bandera es para indicar que aún nos encontramos trabajando con un video en específico
    ban = True
    #El contador de cuadros se inicializa en 0 para cada video
    contadorCuadros = 0
    #Se da el nombre de entrada del video, en este caso los videos
    #son nombrados con "video_n" para automatizar el proceso
    nombreEntrada = "video_"+str(i+1)
    print(nombreEntrada)
    #Se carga el video que queremos utilizar
    cap=cv2.VideoCapture(rutaDatasets+nombreDataset+nombreEntrada+'.mp4')
    #Obtenemos los FPS
    fps = cap.get(cv2.CAP_PROP_FPS)

    #Definimos tamaño a redimensionar
    frame_width = 352#
    frame_height = 288#

    #Le damos un nombre al video que tendremos como salida
    nombreSalida = "salida"+str(contador)+"_"+str(fps)+"fps"

    # Definimos el codec y guardamos el video
    out = cv2.VideoWriter(directorioSalida+nombreSalida+'.avi',cv2.VideoWriter_fourcc('M','J','P','G'), int(fps), (frame_width,frame_height))
    #Asignamos el contador de activación en 0
    contadorActivacion = 0
    #Asignamos un tiempo de inicio
    inicio = time.time()
    #Este ciclo 'While' nos dice si los cuadros de un video fueron ya procesados
    while ban:
        #Se lee un cuadro del video

```

```

r, frame = cap.read()
#Se aumenta el contador de cuadros
contadorCuadros = contadorCuadros + 1
#se verifica si el vídeo aún tiene cuadros por procesar
if r == True:
    #Este proceso es el mismo que se vio anteriormente
    frame = cv2.resize(frame, (frame_width, frame_height))
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    rects = person_cascade.detectMultiScale(gray_frame)

    try:
        if ( all(rects)):
            x = 1

    except:
        #En caso de que haya un fallo en rects, se entiende que se a
        ctivó el algoritmo
        #y aumentamos el contador en uno
        contadorActivacion = contadorActivacion + 1

    #Dibujamos el cuadro de detección en el frame
    for (x, y, w, h) in rects:
        cv2.rectangle(frame, (x,y), (x+w,y+h),(0,255,0),2)
    #Mostramos el frame
    cv2.imshow("preview", frame)
    #Guardamos ese frame
    out.write(frame)

    key = cv2.waitKey(1)
    if key & 0xFF == ord('q'):
        ban = False
else:
    #En caso de que no tenga cuadros por procesar se sale del ciclo
    ban = False

contador = contador + 1
fin = time.time()
segundos = fin - inicio
fpsNuevos = contadorCuadros/segundos
#Se guardan los resultados en los archivos
f.write(str(nombreEntrada) + ',' + str(segundos) + ',' + str(fpsNuevos)+
'\n')
f2.write(str(nombreEntrada) + ',' + str(contadorCuadros) + ',' + str(con
tadorActivacion)+'\n')
#Se imprimen por pantalla los resultados

```

```
print(nombreEntrada + " FPS: " +str(fpsNuevos))
print(nombreEntrada + " Tiempo total: " + str(segundos))
#Se cierran los capturadores
cap.release()
out.release()

#Se cierran los archivos
f.close()
f2.close()
cv2.destroyAllWindows()
```

Esta es básicamente la forma que siguen el resto de los algoritmos para ser probados, la parte que se ve alterada obviamente, es al instanciar el detector y cuando se le pasa el frame que básicamente es lo que está dentro de la instancia

```
if r == True:
```



## HOG (Detección de rostros)

```
import cv2
import time
import os
import dlib
import numpy as np

#Definimos el método que detectará los rostros
def detectFaceDlibHog(detector, frame, inHeight=300, inWidth=0):

    frameDlibHog = frame.copy()
    frameHeight = frameDlibHog.shape[0]
    frameWidth = frameDlibHog.shape[1]
    if not inWidth:
        inWidth = int((frameWidth / frameHeight)*inHeight)

    scaleHeight = frameHeight / inHeight
    scaleWidth = frameWidth / inWidth

    frameDlibHogSmall = cv2.resize(frameDlibHog, (inWidth, inHeight))

    frameDlibHogSmall = cv2.cvtColor(frameDlibHogSmall, cv2.COLOR_BGR2RGB)
    faceRects = detector(frameDlibHogSmall, 0)

    bboxes = []
    for faceRect in faceRects:

        cvRect = [int(faceRect.left()*scaleWidth), int(faceRect.top()*scaleHeight),
                  int(faceRect.right()*scaleWidth), int(faceRect.bottom()*scaleHeight) ]
        bboxes.append(cvRect)
        cv2.rectangle(frameDlibHog, (cvRect[0], cvRect[1]), (cvRect[2], cvRect[3]), (0, 255, 0), int(round(frameHeight/150)), 4)
    return frameDlibHog, bboxes

if __name__ == '__main__':
    #Cargamos nuestro clasificador
    hogFaceDetector = dlib.get_frontal_face_detector()

    rutaDatasets = '../Videos/Datasets/'
    nombreDataset = 'CamNet 352 x 288/'
    #Asigamos un directorio donde se guardarán nuestros archivos
    directorioSalida = '../Videos/Resultados/Salida Hog Rostros/'
```

```

try:
    # Creamos el directorio de salida
    os.mkdir(directorioSalida)
    print( directorioSalida , " Creado ")
except FileExistsError:
    # Si ya existía, se catcha la excepción
    print( directorioSalida , " Ya existia")

try:
    #Creamos los de salida
    f = open(directorioSalida+"Salida HAAR.csv", "x")
    f2 = open(directorioSalida+"Conteo de activacion HOG Rostros.csv", "
x")
except:
    #Si ya existían, se procede a abrirlas
    f = open(directorioSalida+"Salida HAAR.csv", "w")
    f2 = open(directorioSalida+"Conteo de HOG Rostros.csv", "w")

contador = 1
#Se colocan las cabeceras de los archivos
f.write("nombre, tiempo, fps \n")
f2.write("nombre, cuadros Totales, cuadrosActivados \n")

#En este ciclo 'for' se toman en cuenta todos los videos que se quieran
procesar
for i in range(8):
    #Esta bandera es para indicar que aún nos encontramos trabajando con
un video en específico
    ban = True
    #El contador de cuadros se inicializa en 0 para cada video
    contadorCuadros = 0
    #Se da el nombre de entrada del video, en este caso los videos
    #son nombrados con "video_n" para automatizar el proceso
    nombreEntrada = "video_"+str(i+1)

    #Se carga el video que queremos utilizar
    cap=cv2.VideoCapture(rutaDatasets+nombreDataset+nombreEntrada+'.mp4'
)

    #Obtenemos los FPS
    fps = cap.get(cv2.CAP_PROP_FPS)

    #Definimos tamaño a redimensionar
    frame_width = 352#
    frame_height = 288#

```

```

#Le damos un nombre al vídeo que tendremos como salida
nombreSalida = "salida"+str(contador)+"_"+str(fps)+"fps"

# Definimos el codec y guardamos el video
out = cv2.VideoWriter(directorioSalida+nombreSalida+'.avi',cv2.Video
Writer_fourcc('M','J','P','G'), int(fps), (frame_width,frame_height))
#Asignamos el contador de activación en 0
contadorActivacion = 0
#Asignamos un tiempo de inicio
inicio = time.time()
#Este ciclo 'While' nos dice si los cuadros de un video fueron ya p
rocesados
while ban:
    #Se lee un cuadro del video
    r, frame = cap.read()
    #Se aumenta el contador de cuadros
    contadorCuadros = contadorCuadros + 1
    #se verifica si el vídeo aún tiene cuadros por procesar
    if r == True:
        #Este proceso es el mismo que se vio anteriormente
        frame = cv2.resize(frame, (frame_width, frame_height))

        found, bboxes = detectFaceDlibHog(hogFaceDetector,frame)

        try:
            if ( all(found)):
                x = 1

        except:
            #En caso de que haya un fallo en found, se entiende que
se activó el algoritmo
            #y aumentamos el contador en uno
            contadorActivacion = contadorActivacion + 1

        #Mostramos el found
        cv2.imshow('HOG Rostros',found)
        #Guardamos ese frame
        out.write(frame)

        key = cv2.waitKey(1)
        if key & 0xFF == ord('q'):
            ban = False
    else:
        #En caso de que no tenga cuadros por procesar se sale del ci
clo

```

```

        ban = False

        contador = contador + 1
        fin = time.time()
        segundos = fin - inicio
        fpsNuevos = contadorCuadros/segundos
        #Se guardan los resultados en los archivos
        f.write(str(nombreEntrada) + ',' + str(segundos) + ',' + str(fpsNuev
os)+ '\n')
        f2.write(str(nombreEntrada) + ',' + str(contadorCuadros) + ',' + str
(contadorActivacion)+'\n')
        #Se imprimen por pantalla los resulatodos
        print(nombreEntrada + " FPS: " +str(fpsNuevos))
        print(nombreEntrada + " Tiempo total: " + str(segundos))
        #Se cierran los capturadores
        cap.release()
        out.release()

    #Se cierran los archivos
    f.close()
    f2.close()
    cv2.destroyAllWindows()

```

## HOG (Detección de siluetas)

```
#Se importan las librerías necesarias
import cv2
import time
import os
import numpy as np

def inside(r, q):
    rx, ry, rw, rh = r
    qx, qy, qw, qh = q
    return rx > qx and ry > qy and rx + rw < qx + qw and ry + rh < qy + qh

#Definimos el método que dibujará los cuadrados sobre el área detectada
def draw_detections(img, rects, thickness = 1):
    for x, y, w, h in rects:
        #El detector HOG regresa rectangulos un poco más grandes que los objetos a los que identifica
        # Por lo que se procede a recortarlos un poco
        pad_w, pad_h = int(0.15*w), int(0.05*h)
        cv2.rectangle(img, (x+pad_w, y+pad_h), (x+w-pad_w, y+h-pad_h), (0, 255, 0), thickness)

if __name__ == '__main__':
    #Cargamos nuestro clasificador
    hog = cv2.HOGDescriptor()
    #Inicializamos el descriptor de HOG
    hog.setSVMDetector( cv2.HOGDescriptor_getDefaultPeopleDetector())

    #Asignamos un directorio donde se guardarán nuestros archivos
    directorioSalida = 'Path_De_salida'

    try:
        # Creamos el directorio de salida
        os.mkdir(directorioSalida)
        print( directorioSalida , " Creado ")
    except FileExistsError:
        # Si ya existía, se catcha la excepción
        print( directorioSalida , " Ya existia")

    try:
        #Creamos los de salida
        f = open(directorioSalida+"Salida HAAR.csv", "x")
        f2 = open(directorioSalida+"Conteo de activacion HAAR.csv", "x")
    except:
```

```

        #Si ya existían, se procede a abrirlas
        f = open(directorioSalida+"Salida HAAR.csv", "w")
        f2 = open(directorioSalida+"Conteo de activacion HAAR.csv", "w")

    contador = 1
    #Se colocan las cabeceras de los archivos
    f.write("nombre, tiempo, fps \n")
    f2.write("nombre, cuadros Totales, cuadrosActivados \n")

    #En este ciclo 'for' se toman en cuenta todos los videos que se quieran
    procesar
    for i in range(8):
        #Esta bandera es para indicar que aún nos encontramos trabajando con
        un video en especifico
        ban = True
        #El contador de cuadros se inicializa en 0 para cada video
        contadorCuadros = 0
        #Se da el nombre de entrada del video, en este caso los videos
        #son nombrados con "video_n" para automatizar el proceso
        nombreEntrada = "video_"+str(i+1)

        #Se carga el video que queremos utilizar
        cap=cv2.VideoCapture(rutaDatasets+'Nombre_Del_DataSet/'+nombreEntrada
+ '.avi')
        #Obtenemos los FPS
        fps = cap.get(cv2.CAP_PROP_FPS)

        #Definimos tamaño a redimensionar
        frame_width = 352#
        frame_height = 288#

        #Le damos un nombre al video que tendremos como salida
        nombreSalida = "salida"+str(contador)+"_"+str(fps)+"fps"

        # Definimos el codec y guardamos el video
        out = cv2.VideoWriter(directorioSalida+nombreSalida+'.avi',cv2.Video
Writer_fourcc('M','J','P','G'), int(fps), (frame_width,frame_height))
        #Asignamos el contador de activación en 0
        contadorActivacion = 0
        #Asignamos un tiempo de inicio
        inicio = time.time()
        #Este ciclo 'While' nos dice si los cuadros de un video fueron ya p
        rocesados
        while ban:
            #Se lee un cuadro del video

```

```

        r, frame = cap.read()
        #Se aumenta el contador de cuadros
        contadorCuadros = contadorCuadros + 1
        #se verifica si el vídeo aún tiene cuadros por procesar
        if r == True:
            #Este proceso es el mismo que se vio anteriormente
            frame = cv2.resize(frame, (frame_width, frame_height))

            found,w=hog.detectMultiScale(frame, winStride=(8,8), padding
=(32,32), scale=1.05)

            try:
                if ( all(found)):
                    x = 1

            except:
                #En caso de que haya un fallo en found, se entiende que
se activó el algoritmo
                #y aumentamos el contador en uno
                contadorActivacion = contadorActivacion + 1
                #Pasamos las áreas detectadas junto con el frame para que se
an dibujadas

                draw_detections(frame,found)
                #Mostramos el found
                cv2.imshow('HOG Siluetas',frame)
                #Guardamos ese frame
                out.write(frame)

                key = cv2.waitKey(1)
                if key & 0xFF == ord('q'):
                    ban = False
            else:
                #En caso de que no tenga cuadros por procesar se sale del ci
clo

                ban = False

        contador = contador + 1
        fin = time.time()
        segundos = fin - inicio
        fpsNuevos = contadorCuadros/segundos
        #Se guardan los resultados en los archivos
        f.write(str(nombreEntrada) + ',' + str(segundos) + ',' + str(fpsNuev
os)+ '\n')
        f2.write(str(nombreEntrada) + ',' + str(contadorCuadros) + ',' + str
(contadorActivacion)+'\n')

```

```
#Se imprimen por pantalla los resultados
print(nombreEntrada + " FPS: " + str(fpsNuevos))
print(nombreEntrada + " Tiempo total: " + str(segundos))
#Se cierran los capturadores
cap.release()
out.release()

#Se cierran los archivos
f.close()
f2.close()
cv2.destroyAllWindows()
```



## MTCNN

```
#Se importan las librerías necesarias
from mtcnn.mtcnn import MTCNN
import cv2
import time
import os

if __name__ == '__main__':
    #Cargamos nuestro clasificador
    hog = cv2.HOGDescriptor()
    #Inicializamos el descriptor de HOG
    hog.setSVMDetector( cv2.HOGDescriptor_getDefaultPeopleDetector())

    rutaDatasets = '../Videos/Datasets/'
    nombreDataset = 'CamNet 352 x 288/'
    #Asigamos un directorio donde se guardarán nuestros archivos
    directorioSalida = '../Videos/Resultados/Salida Hog Siluetas/'

    try:
        # Creamos el directorio de salida
        os.mkdir(directorioSalida)
        print( directorioSalida , " Creado ")
    except FileExistsError:
        # Si ya existía, se catcha la excepción
        print( directorioSalida , " Ya existia")

    try:
        #Creamos los de salida
        f = open(directorioSalida+"Salida HAAR.csv", "x")
        f2 = open(directorioSalida+"Conteo de activacion Hog siluetas.csv",
"x")
    except:
        #Si ya existían, se procede a abrirlas
        f = open(directorioSalida+"Salida HAAR.csv", "w")
        f2 = open(directorioSalida+"Conteo de activacion Hog siluetas.csv",
"w")

    contador = 1
    #Se colocan las cabeceras de los archivos
    f.write("nombre, tiempo, fps \n")
    f2.write("nombre, cuadros Totales, cuadrosActivados \n")

    #En este ciclo 'for' se toman en cuenta todos los videos que se quieran
procesar
```

```

    for i in range(8):
        #Esta bandera es para indicar que aún nos encontramos trábando con
un vídeo en específico
        ban = True
        #El contador de cuadros se inicializa en 0 para cada vídeo
        contadorCuadros = 0
        #Se da el nombre de entrada del video, en este caso los videos
        #son nombrados con "video_n" para automatizar el proceso
        nombreEntrada = "video_"+str(i+1)

        #Se carga el vídeo que queremos utilizar
        cap=cv2.VideoCapture(rutaDatasets+nombreDataset+nombreEntrada+'.mp4'
)

        #Obtenemos los FPS
        fps = cap.get(cv2.CAP_PROP_FPS)

        #Definimos tamaño a redimensionar
        frame_width = 352#
        frame_height = 288#

        #Le damos un nombre al vídeo que tendremos como salida
        nombreSalida = "salida"+str(contador)+"_"+str(fps)+"fps"

        # Definimos el codec y guardamos el video
        out = cv2.VideoWriter(directorioSalida+nombreSalida+'.avi',cv2.Video
Writer_fourcc('M','J','P','G'), int(fps), (frame_width,frame_height))
        #Asignamos el contador de activación en 0
        contadorActivacion = 0
        #Asignamos un tiempo de inicio
        inicio = time.time()
        #Este ciclo 'While' nos dice si los cuadros de un video fueron ya p
rocesados
        while ban:
            #Se lee un cuadro del video
            r, frame = cap.read()
            #Se aumenta el contador de cuadros
            contadorCuadros = contadorCuadros + 1
            #se verifica si el vídeo aún tiene cuadros por procesar
            if r == True:
                #Este proceso es el mismo que se vio anteriormente
                frame = cv2.resize(frame, (frame_width, frame_height))

                found,w=hog.detectMultiScale(frame, winStride=(8,8), padding
=(32,32), scale=1.05)

```

```

        try:
            if ( all(found)):
                x = 1

        except:
            #En caso de que haya un fallo en found, se entiende que
            se activó el algoritmo
            #y aumentamos el contador en uno
            contadorActivacion = contadorActivacion + 1
            #Pasamos las áreas detectadas junto con el frame para que se
            an dibujadas

            draw_detections(frame,found)
            #Mostramos el found
            cv2.imshow('HOG Siluetas',frame)
            #Guardamos ese frame
            out.write(frame)

            key = cv2.waitKey(1)
            if key & 0xFF == ord('q'):
                ban = False
            else:
                #En caso de que no tenga cuadros por procesar se sale del ci
                clo

                ban = False

        contador = contador + 1
        fin = time.time()
        segundos = fin - inicio
        fpsNuevos = contadorCuadros/segundos
        #Se guardan los resultados en los archivos
        f.write(str(nombreEntrada) + ',' + str(segundos) + ',' + str(fpsNuev
os)+ '\n')
        f2.write(str(nombreEntrada) + ',' + str(contadorCuadros) + ',' + str
(contadorActivacion)+'\n')
        #Se imprimen por pantalla los resultados
        print(nombreEntrada + " FPS: " +str(fpsNuevos))
        print(nombreEntrada + " Tiempo total: " + str(segundos))
        #Se cierran los capturadores
        cap.release()
        out.release()

    #Se cierran los archivos
    f.close()
    f2.close()
    cv2.destroyAllWindows()

```

## Faster R CNN

```
#Se importan las librerías necesarias
import numpy as np
import cv2
import time
import os
import tensorflow as tf

class DetectorAPI:
    def __init__(self, path_to_ckpt):
        self.path_to_ckpt = path_to_ckpt

        self.detection_graph = tf.Graph()
        with self.detection_graph.as_default():
            od_graph_def = tf.GraphDef()
            with tf.gfile.GFile(self.path_to_ckpt, 'rb') as fid:
                serialized_graph = fid.read()
                od_graph_def.ParseFromString(serialized_graph)
                tf.import_graph_def(od_graph_def, name='')

        self.default_graph = self.detection_graph.as_default()
        self.sess = tf.Session(graph=self.detection_graph)

        # Definir los tensores de entrada y salida para detection_graph
        self.image_tensor = self.detection_graph.get_tensor_by_name('image_tensor:0')
        # Cada caja representa una parte de la imagen donde un objeto en particular fue detectado
        self.detection_boxes = self.detection_graph.get_tensor_by_name('detection_boxes:0')
        # Cada puntaje representa un nivel de precisión

        self.detection_scores = self.detection_graph.get_tensor_by_name('detection_scores:0')
        self.detection_classes = self.detection_graph.get_tensor_by_name('detection_classes:0')
        self.num_detections = self.detection_graph.get_tensor_by_name('num_detections:0')

    def processFrame(self, image):
        # El modelo esperado espera imágenes con el siguiente formato [1, None, None, 3]
        image_np_expanded = np.expand_dims(image, axis=0)
        # Comienza la detección
        start_time = time.time()
```

```

        (boxes, scores, classes, num) = self.sess.run(
            [self.detection_boxes, self.detection_scores, self.detection_classes, self.num_detections],
            feed_dict={self.image_tensor: image_np_expanded})
        end_time = time.time()

        im_height, im_width, _ = image.shape
        boxes_list = [None for i in range(boxes.shape[1])]
        #Comienza a dibujar las cajas en el lugar donde se encontró algo
        for i in range(boxes.shape[1]):
            boxes_list[i] = (int(boxes[0,i,0] * im_height),
                            int(boxes[0,i,1]*im_width),
                            int(boxes[0,i,2] * im_height),
                            int(boxes[0,i,3]*im_width))
        #Regresa las cajas que encontró
        return boxes_list, scores[0].tolist(), [int(x) for x in classes[0].tolist()], int(num[0])

    def close(self):
        self.sess.close()
        self.default_graph.close()

if __name__ == '__main__':
    #Cargamos nuestro clasificador
    model_path = '../..\\xmls\\faster_rcnn_inception_v2_coco_2018_01_28\\frozen_inference_graph.pb'
    odapi = DetectorAPI(path_to_ckpt=model_path)
    threshold = 0.7

    rutaDatasets = '../..\\Videos\\Datasets\\'
    nombreDataset = 'CamNet 352 x 288/'
    #Asigamos un directorio donde se guardarán nuestros archivos
    directorioSalida = '../..\\Videos\\Resultados\\Salida Faster R-CNN/'

    try:
        # Creamos el directorio de salida
        os.mkdir(directorioSalida)
        print( directorioSalida , " Creado ")
    except FileExistsError:
        # Si ya existía, se catcha la excepción
        print( directorioSalida , " Ya existia")

    try:

```

```

        #Creamos los de salida
        f = open(directorioSalida+"Salida HAAR.csv", "x")
        f2 = open(directorioSalida+"Conteo de activacion Faster R-
CNN.csv", "x")
    except:
        #Si ya existían, se procede a abrirlas
        f = open(directorioSalida+"Salida HAAR.csv", "w")
        f2 = open(directorioSalida+"Conteo de activacion Faster R-
CNN.csv", "w")

    contador = 1
    #Se colocan las cabeceras de los archivos
    f.write("nombre, tiempo, fps \n")
    f2.write("nombre, cuadros Totales, cuadrosActivados \n")

    #En este ciclo 'for' se toman en cuenta todos los videos que se quieran
    procesar
    for i in range(8):
        #Esta bandera es para indicar que aún nos encontramos trabajando con
        un video en específico
        ban = True
        #El contador de cuadros se inicializa en 0 para cada video
        contadorCuadros = 0
        #Se da el nombre de entrada del video, en este caso los videos
        #son nombrados con "video_n" para automatizar el proceso
        nombreEntrada = "video_"+str(i+1)

        #Se carga el video que queremos utilizar
        cap=cv2.VideoCapture(rutaDatasets+nombreDataset+nombreEntrada+'.mp4'
)

        #Obtenemos los FPS
        fps = cap.get(cv2.CAP_PROP_FPS)

        #Definimos tamaño a redimensionar
        frame_width = 352#
        frame_height = 288#

        #Le damos un nombre al video que tendremos como salida
        nombreSalida = "salida"+str(contador)+"_"+str(fps)+"fps"

        # Definimos el codec y guardamos el video
        out = cv2.VideoWriter(directorioSalida+nombreSalida+'.avi',cv2.Video
Writer_fourcc('M','J','P','G'), int(fps), (frame_width,frame_height))
        #Asignamos el contador de activación en 0
        contadorActivacion = 0

```

```

    #Asignamos un tiempo de inicio
    inicio = time.time()
    #Este ciclo 'While' nos dice si los cuadros de un video fueron ya p
    rocesados
    while ban:
        #Se lee un cuadro del video
        r, frame = cap.read()
        #Se aumenta el contador de cuadros
        contadorCuadros = contadorCuadros + 1
        #se verifica si el vídeo aún tiene cuadros por procesar
        if r == True:
            #Este proceso es el mismo que se vio anteriormente
            frame = cv2.resize(frame, (frame_width, frame_height))

            #Activamos el detector con el cuadro seleccionado
            boxes, scores, classes, num = odapi.processFrame(frame)
            #El detector nos arroja coorenadas con lo que podría una per
            sona

            #Recordemos que este detector puede detectar una gran varied
            ad de cosas

            for i in range(len(boxes)):
                # la clase 1, es la que representa 'humanos'
                if classes[i] == 1 and scores[i] > threshold:
                    box = boxes[i]
                    #Se procede a pintar un rectangulo en el grame
                    cv2.rectangle(frame,(box[1],box[0]),(box[3],box[2]),
                    (255,0,0),2)

                    #Se aumenta el contador de activación, pues el algor
                    itmo encontró una persona
                    contadorActivacion = contadorActivacion + 1

            #Mostramos el frame
            cv2.imshow('Faster R CNN',frame)
            #Guardamos ese frame
            out.write(frame)

            key = cv2.waitKey(1)
            if key & 0xFF == ord('q'):
                ban = False
            else:
                #En caso de que no tenga cuadros por procesar se sale del ci
                clo

                ban = False

    contador = contador + 1

```

```
        fin = time.time()
        segundos = fin - inicio
        fpsNuevos = contadorCuadros/segundos
        #Se guardan los resultados en los archivos
        f.write(str(nombreEntrada) + ',' + str(segundos) + ',' + str(fpsNuevos) + '\n')
        f2.write(str(nombreEntrada) + ',' + str(contadorCuadros) + ',' + str(contadorActivacion) + '\n')
        #Se imprimen por pantalla los resultados
        print(nombreEntrada + " FPS: " + str(fpsNuevos))
        print(nombreEntrada + " Tiempo total: " + str(segundos))
        #Se cierran los capturadores
        cap.release()
        out.release()

    #Se cierran los archivos
    f.close()
    f2.close()
    cv2.destroyAllWindows()
```



Nuestro algoritmo final que convida Haar Cascade y MTCNN sería el siguiente:

```
from mtcnn.mtcnn import MTCNN
import cv2
import time
import os

#HAAR CASCADE
person_cascade = cv2.CascadeClassifier( '../..\\xmls\\haarcascade_fullbody.xml' )
#MTCNN
detector = MTCNN()

rutaDatasets = '../..\\Videos\\Datasets/'
nombreDataset = 'CamNet 352 x 288/'
#Asigamos un directorio donde se guardarán nuestros archivos
directorioSalida = '../..\\Videos\\Resultados\\Salida Haar-MTCNN/'

#Archivo

try:
    # Create target Directory
    os.mkdir(directorioSalida)
    print( directorioSalida , " Creado ")
except FileExistsError:
    print( directorioSalida , " Ya existia")

try:
    #Creamos archivo
    f = open(directorioSalida+"Salida HAAR-MTCNN.csv", "x")
    f2 = open(directorioSalida+"Conteo de activacion HAAR-MTCNN.csv", "x")
except:
    f = open(directorioSalida+"Salida HAAR-MTCNN.csv", "w")
    f2 = open(directorioSalida+"Conteo de activacion HAAR-MTCNN.csv", "w")

contador = 1
f.write("nombre, tiempo, fps \n")
f2.write("nombre, cuadros Totales, cuadrosActivados HC, cuadrosActivados MTCNN \n")
for i in range(8): #CAMBIAR NÚMERO DE ELEMENTOS
    ban = True
    contadorCuadros = 0
    nombreEntrada = "video_"+str(contador)
    print(nombreEntrada)
    cap=cv2.VideoCapture(rutaDatasets+nombreDataset+nombreEntrada+'.mp4')
    #Sacamos los FPS
```

```

fps = cap.get(cv2.CAP_PROP_FPS)

#Definimos tamaño
frame_width = 352#
frame_height = 288#

nombreSalida = "salida"+str(contador)+"_"+str(fps)+"fps"
# Definimos el codec y guardamos el video
out = cv2.VideoWriter(directorioSalida+nombreSalida+'.avi',cv2.VideoWriter_fourcc('M','J','P','G'), int(fps), (frame_width,frame_height))
contadorActivacion = 0
contadorActivacion2 = 0
inicio = time.time()
while ban:
    r, frame = cap.read()

    contadorCuadros = contadorCuadros + 1
    if r == True:
        #REESCALAR
        frame = cv2.resize(frame, (frame_width, frame_height))
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY) #El Haar-cascade classifier necesita escala de grises
        rects = person_cascade.detectMultiScale(gray_frame)

        if len(rects):
            x = 1

            contadorActivacion = contadorActivacion + 1

            for (x, y, w, h) in rects:
                cv2.rectangle(frame, (x,y), (x+w,y+h),(0,255,0),2)

            face_locations = detector.detect_faces(frame)

            #print("Activación de haar")

            if len(face_locations):
                #print("Activación MTCNN")
                contadorActivacion2 = contadorActivacion2 + 1
                for face in zip(face_locations):
                    (x, y, w, h) = face[0]['box']
                    contadorActivacion = contadorActivacion + 1
                    cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)

```

```

        cv2.imshow("preview", frame)
        out.write(frame)
        key = cv2.waitKey(1)
        if key & 0xFF == ord('q'):
            ban = False
    else:
        ban = False

    contador = contador + 1
    fin = time.time()
    segundos = fin - inicio
    fpsNuevos = contadorCuadros/segundos
    f.write(str(nombreEntrada) + ',' + str(segundos) + ',' + str(fpsNuevos)+
'\n')
    f2.write(str(nombreEntrada) + ',' + str(contadorCuadros) + ',' + str(con
tadorActivacion)+ ','+str(contadorActivacion2) +'\n')
    print(nombreEntrada + " FPS: " +str(fpsNuevos))
    print(nombreEntrada + " Tiempo total: " + str(segundos))
    cap.release()
    out.release()

f.close()
f2.close()
cv2.destroyAllWindows()

```