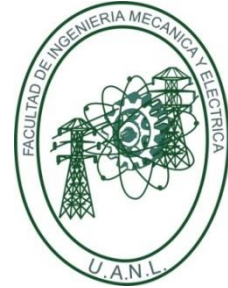




Universidad Autónoma
de Nuevo León



FACULTAD DE INGENIERÍA
MECÁNICA Y ELÉCTRICA

Visión computacional

EXAMEN ORDINARIO - PROYECTO FINAL

Nombre	Matrícula
MARQUEZ ALONSO LUIS FERNANDO	1614658
MARTÍNEZ RAMÍREZ JORGE ANTONIO	1617998
AGUILAR GUERRERO ALAN EDUARDO	1639244
SALAS CHAVEZ BRAYAN	1619199
AVILA HERRERA EMMANUEL DE JESUS	1719376

Catedrático

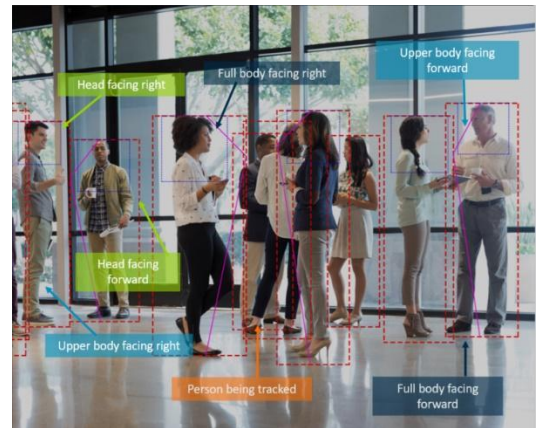
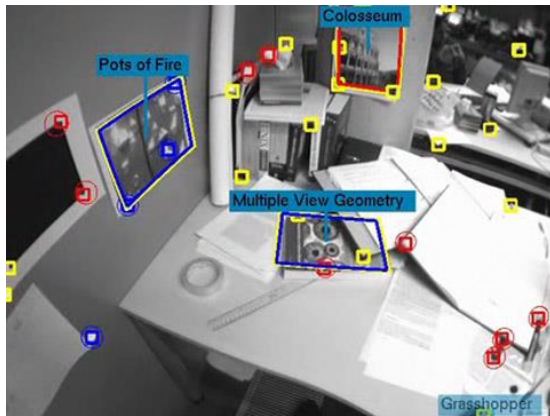
M. C. Laura Patricia del Bosque Vega

¿Por qué se seleccionó el proyecto?

El proyecto seleccionado actualmente no era el que teníamos en mente, se había visto proyecto llamado BlinkToText el cual constaba de un reconocimiento facial el cual detectaba cuando estaban los ojos cerrados y posteriormente seleccionaba letras de la pantalla, esto para personas que no pueden utilizar sus extremidades. Pero al no poder realizarlo ya que otro equipo lo dio de alta primero optamos por buscar detección de objetos.

El proyecto que encontramos para realizar fue el YOLO o como su traducción dice solo se ve una vez, el cual fue desarrollado por Pjreddie un usuario de GitHub como cualquier otro el cual creo detección de ciertos objetos.

En sí nos pareció interesante el cómo poder reconocer objetos solamente con mostrarselos a la computadora y fue la principal causa para seleccionar dicho proyecto.



El proyecto en sí se le encontró como **detección de objetos** el cual consta de detectar personas y objetos a partir de una imagen en el lenguaje de programación Python ya sea en su versión 3.5 o 2.7 que es más estable.



¿Cómo se realizó el proyecto?

Primero se necesitó echarle un vistazo al artículo del proyecto el cual nos hablaba sobre la detección de objetos con ayuda de Open CV, Python, en cualquier editor de texto.

Para esto se necesitó de instalar las librerías necesarias para Python (3.5 o 2.7 que fue las que probamos para realizar dicho proyecto) como lo son numpy y Open CV cada uno con sus comandos necesarios, el programa se probó tanto en Ubuntu como en Windows.

En **Windows** se obtuvo con:

- Open CV desde: <https://sourceforge.net/projects/opencvlibrary/files/>
- Numpy con: `python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose`



En **Ubuntu** se obtuvo con:

- Open CV con: `sudo apt-get install python-opencv`
- Numpy con: `sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook python-pandas python-sympy python-nose`

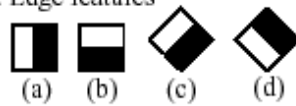


Una vez teniendo las librerías listas se pasó a probar dicho proyecto el cual consta de utilizar archivos HAAR, que son clasificados por clasificador HAAR, pero;

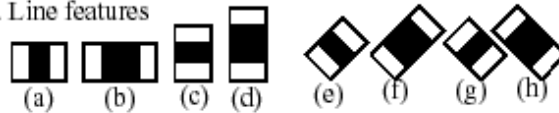
¿Qué es el Clasificador HAAR?

El clasificador HAAR es un algoritmo de machine learning creado por Paul Viola y Michael Jones en el cual se trata de contener imágenes positivas en una carpeta y en otra carpeta imágenes negativas de lo que se va a realizar, en este caso caras, en la carpeta positiva solo contendrá eso, y en la otra lo que sea menos caras.

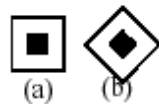
1. Edge features



2. Line features



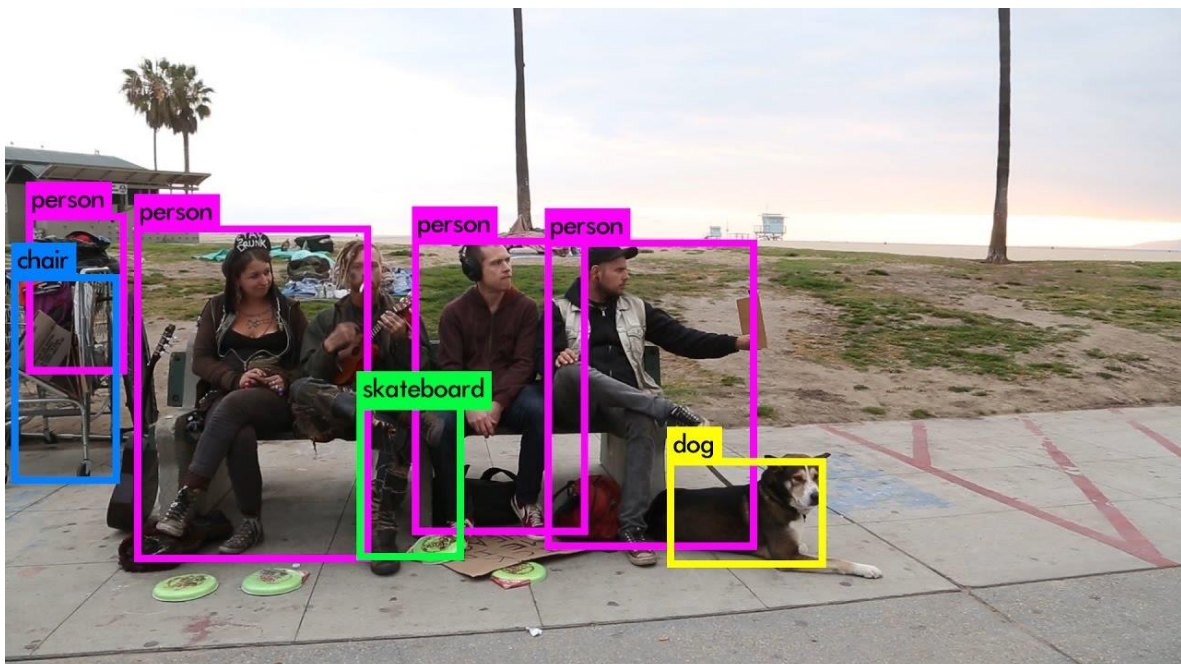
3. Center-surround features



También se observó que existe otro algoritmo llamado LBP el cual también es bueno pero tiene diversas desventajas con el HAAR, el cual por ejemplo como ventajas el HAAR tiene una puntuación de acertar muy alta con un bajo falso positivo, pero en lo que es lo computacional, LBP es muchísimo más fácil, rápido y simple, pero por lo tanto menos eficiente para detectar objetos.

Del proyecto podemos analizar como se obtiene el rostro de una persona, con la diferencia que ellos utilizaron la librería matplotlib pero ellos mismos explican que es opcional porque no es necesariamente útil.

Ejemplo con YOLO):

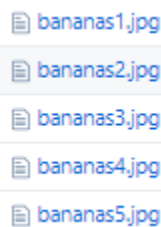


En dicho proyecto no se menciona el cómo utilizarlo para videos ya sean en webcam o grabados en formato mp4 o avi, nosotros tomamos su framework para utilizar el reconocimiento para lo que son ojos, sonrisa, bananas, plumas y carros, en tiempo real o en videos.

¿Cómo lo logramos?

Siguiendo distintos tutoriales de los cuales nos decían que al tener una carpeta con fotos positivas y otras con fotos negativas se podría hacer todo el archivo xml para que fuera detectando cosas como lo fue con los rostros.

En una carpeta teníamos archivos de bananas por ejemplo:



bananas1.jpg
bananas2.jpg
bananas3.jpg
bananas4.jpg
bananas5.jpg

Mientras que en otras imágenes aleatorias que no tienen nada que ver con bananas, y una vez dicho esto se corre el programa el cual debe ser clonado desde github con:

```
git clone https://github.com/mrnugget/opencv-haar-classifier-training
```

Después de bajar dicho repositorio ya se tienen los archivos para entrenar, para la carpeta de positivos se debe tener un archivo txt con la información de cada imagen y dentro de cuales pixeles sale el objeto, y en las negativas solo que se tenga un archivo de información txt con sus imágenes. Esto se hace con los comandos:

```
find ./positive_images -iname "*.jpg" > positives.txt  
find ./negative_images -iname "*.jpg" > negatives.txt
```

Al finalizar esto se corre lo siguiente:

```
brew tap homebrew/science  
brew install --with-tbb opencv
```

Se crean los samples con:

```
perl bin/createsamples.pl positives.txt negatives.txt samples 1500\  
"opencv_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1\  
-maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 80 -h 40"
```

Por último se copia con el comando siguiente y se ejecuta para obtener los samples:

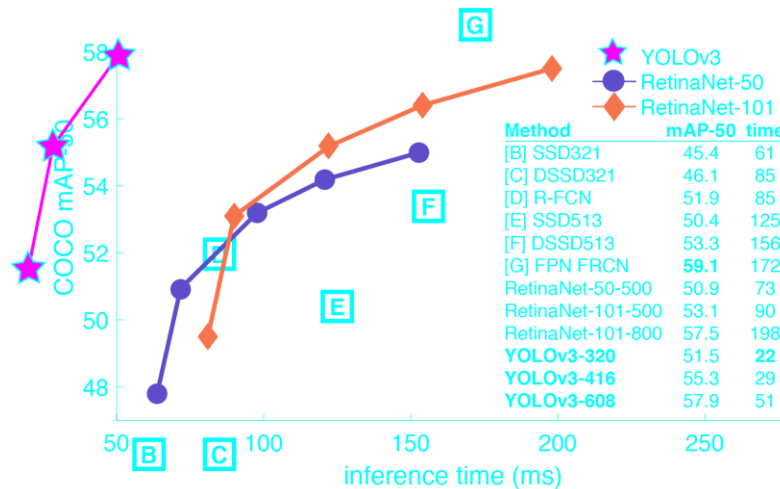
```
cp src/mergevec.cpp ~/opencv-2.4.5/apps/haarttraining  
cd ~/opencv-2.4.5/apps/haarttraining  
g++ `pkg-config --libs --cflags opencv` -l. -o mergevec mergevec.cpp\  
cvboost.cpp cvcommon.cpp cvsamples.cpp cvhaarclassifier.cpp\  
cvhaarttraining.cpp\  
-lopencv_core -lopencv_calib3d -lopencv_imgproc -lopencv_highgui -lopencv_objdetect
```

Creando los samples:

```
find ./samples -name '*.vec' > samples.txt  
./mergevec samples.txt samples.vec
```

¿Cómo funciona YOLO V3?

YOLO V3 es extremadamente rápido y preciso. Además, puede sacrificar la velocidad y la precisión simplemente cambiando el tamaño del modelo (multimedia) sin requerir ningún tipo de reentrenamiento.



Gráfica de pérdida focal contra velocidad.

Los sistemas de detección previa reutilizan clasificadores o localizadores para realizar la detección. Aplican el modelo a una imagen en múltiples ubicaciones y escalas. Las regiones de alta puntuación de la imagen se consideran detecciones.

En YOLO se usa un enfoque totalmente diferente. Se aplica una red neuronal única a la imagen completa. Esta red divide la imagen en regiones y predice cuadros de límite y probabilidades para cada región. Estos cuadros delimitadores están ponderados por las probabilidades pronosticadas.

Este modelo tiene varias ventajas sobre los sistemas basados en clasificadores. Este modelo mira la imagen completa, de modo que sus predicciones se basan en el contexto global de la imagen.

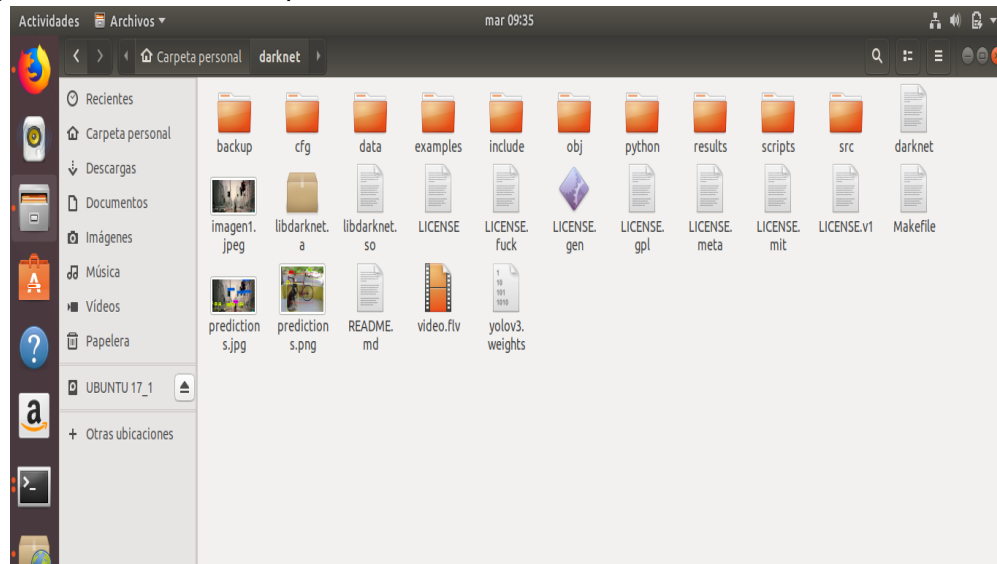
YOLO puede compilar con dos formas:

- OpenCV: Al usar esta compilación se usa el CPU y se pueden usar muchos más formatos de imágenes.
- CUDA: Esta compilación usa el GPU de tecnología Nvidia y hace que las detecciones sean mucho más rápidas, a costa de menos variedad de formatos de imágenes.

Darknet (YOLO) imprime los objetos que detectó, su confianza y cuánto tiempo tomó encontrarlos. Cuando Darknet compila con OpenCV las detecciones se hacen mediante CPU y tardan entre 6 y 12 segundos por imagen. Si se usa la versión de CUDA, sería mucho más rápido.

Implementación:

El proyecto está constituido por esta estructura



El cual incluye todas las librerías necesarias de openCV , así como los datos de entrenamientos para el reconocimiento de objetos, algunos scripts para su ejecución en ubuntu, los scripts en python y los archivos del framework de YOLO.

Una vez teniendo todo el proyecto, ejecutamos el comando make en la terminal bash, esto creará objetos y nos descargara las dependencias necesarias para comenzar a trabajar:

```
markintoch@Markintoch: ~/darknet
Archivo Editar Ver Buscar Terminal Ayuda
markintoch@Markintoch:~/darknet$ make
gcc -Iinclude/ -Isrc/ -DOPENCV 'pkg-config --cflags opencv' -Wall -Wno-unused-r
esult -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -c ./src/gemm.c
-o obj/gemm.o
gcc -Iinclude/ -Isrc/ -DOPENCV 'pkg-config --cflags opencv' -Wall -Wno-unused-r
esult -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -c ./src/utlis.c
-o obj/utlis.o
gcc -Iinclude/ -Isrc/ -DOPENCV 'pkg-config --cflags opencv' -Wall -Wno-unused-r
esult -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -c ./src/cuda.c
-o obj/cuda.o
gcc -Iinclude/ -Isrc/ -DOPENCV 'pkg-config --cflags opencv' -Wall -Wno-unused-r
esult -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -c ./src/deconvo
lutional_layer.c -o obj/deconvolutional_layer.o
gcc -Iinclude/ -Isrc/ -DOPENCV 'pkg-config --cflags opencv' -Wall -Wno-unused-r
esult -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -c ./src/convolu
tional_layer.c -o obj/convolutional_layer.o
gcc -Iinclude/ -Isrc/ -DOPENCV 'pkg-config --cflags opencv' -Wall -Wno-unused-r
esult -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -c ./src/list.c
-o obj/list.o
gcc -Iinclude/ -Isrc/ -DOPENCV 'pkg-config --cflags opencv' -Wall -Wno-unused-r
esult -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -DOPENCV -c ./src/image.c
-o obj/image.o
```

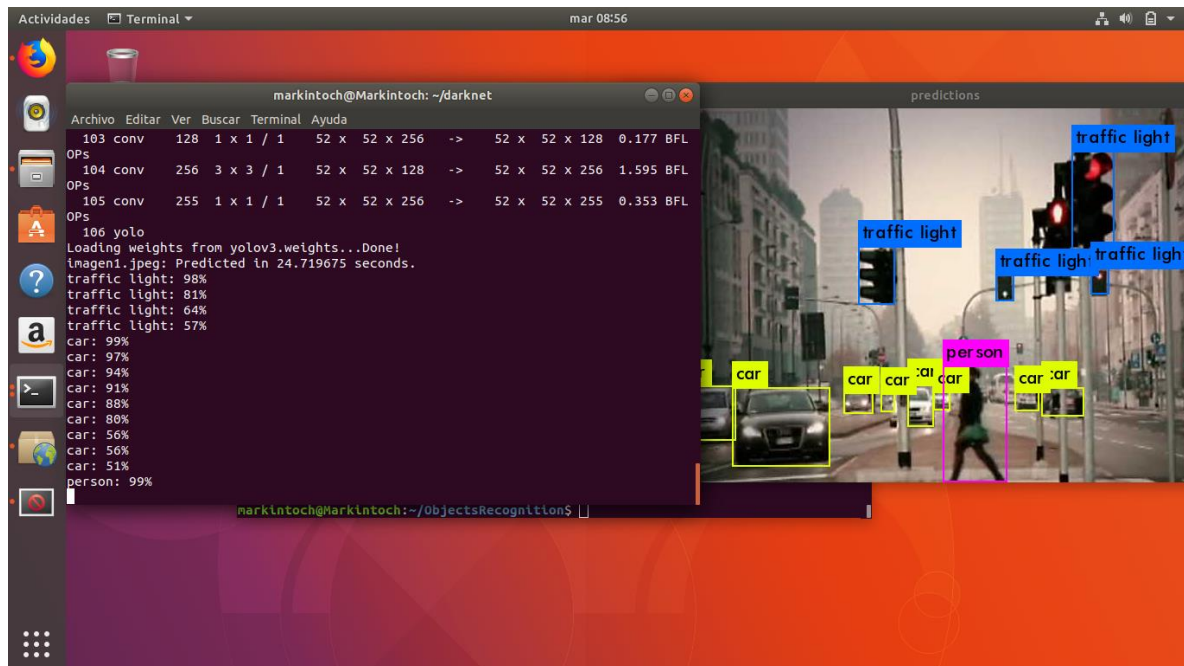
Ya descargadas las dependencias se necesitan los archivos de entrenamiento de los datos, los pesos de los valores para la detección de los objetos con el framework, las cuales se descargan con este comando:

wget <https://pjreddie.com/media/files/yolov3.weights>

Descargando los pesos, hacemos uso de estos y ejecutamos el archivo detect, el cual recibirá como parámetro la imagen de la cual detectará los objetos:

`./darknet detect cfg/yolov3.cfg yolov3.weights <file-name>`

Arrojando un resultado de porcentaje de confiabilidad y la imagen editada con los objetos detectados y sus respectivas etiquetas:



Observando que el índice de confiabilidad en la detección es casi perfecto, es muy alto.

Aquí es donde se hace uso de OpenCV, ya que en este caso, es la librería encargada de manipular la imagen de entrada. Así como que el proyecto puede detectar entre los siguientes objetos:



Y el funcionamiento total esta en manos de darknet.py, que es el que se encarga de detectar los objetos y clasificarlos, para después pasarlos a listas y esas listas sobre escribirlas en la imagen original, dándole etiqueta y detección a la figura:

```
Actividades Editor de textos mar 10:09
darknet.py
~/darknet/python
Guardar

predict_image.restype = POINTER(c_float)

def classify(net, meta, im):
    out = predict_image(net, im)
    res = []
    for i in range(meta.classes):
        res.append((meta.names[i], out[i]))
    res = sorted(res, key=lambda x: -x[1])
    return res

def detect(net, meta, image, thresh=.5, hier_thresh=.5, nms=.45):
    im = load_image(image, 0, 0)
    num = c_int(0)
    pnum = pointer(num)
    predict_image(net, im)
    dets = get_network_boxes(net, im.w, im.h, thresh, hier_thresh, None, 0, pnum)
    num = pnum[0]
    if (nms): do_nms_obj(dets, num, meta.classes, nms);

    res = []
    for j in range(num):
        for i in range(meta.classes):
            if dets[j].prob[i] > 0:
                b = dets[j].bbox
                res.append((meta.names[i], dets[j].prob[i], (b.x, b.y, b.w, b.h)))
    res = sorted(res, key=lambda x: -x[1])
    free_image(im)
    free_detections(dets, num)
    return res

if __name__ == "__main__":
    #net = load_net("cfg/densenet201.cfg", "/home/pjreddie/trained/densenet201.weights", 0)
    #in = load_image("data/wolf.jpg", 0, 0)
    #meta = load_meta("cfg/imagenet1k.data")
    #r = classify(net, meta, in)
    #print r[:10]
    net = load_net("cfg/tiny-yolo.cfg", "tiny-yolo.weights", 0)
    meta = load_meta("cfg/tiny-yolo.data")
```

Funcionamiento del archivo main.py

El archivo main.py es un archivo que creamos a partir del framework de YOLO, el cual está basado en el uso de archivos xml con el algoritmo de HAAR para lo cual tuvimos que realizar dichos entrenamientos con los archivos xml.

Código

```
1 #LIBRARIES
2 import cv2
3 import numpy as np
4 #HAARCASCADE FILES, REPLACE YOUR SOURCE
5 face_cascade = cv2.CascadeClassifier('C:/Users/braya/opencv/sources/data/haarcascades_cuda/haarcascade_frontalface_default.xml')
6 eye_cascade = cv2.CascadeClassifier('C:/Users/braya/opencv/sources/data/haarcascades_cuda/haarcascade_eye.xml')
7 smile_cascade = cv2.CascadeClassifier('C:/Users/braya/opencv/sources/data/haarcascades_cuda/haarcascade_smile.xml')
8 pen_cascade = cv2.CascadeClassifier('C:/Users/braya/opencv/sources/data/haarcascades_cuda/haarcascade_pen.xml')
9 banana_cascade = cv2.CascadeClassifier('C:/Users/braya/opencv/sources/data/haarcascades_cuda/haarcascade_banana.xml')
10 car_cascade = cv2.CascadeClassifier('C:/Users/braya/opencv/sources/data/haarcascades_cuda/haarcascade_cars.xml')
11
12 #OPEN WEBCAM (0 IS WEBCAM, YOU CAN REPLACE 0 FOR OTHER NUMBER IF YOU HAVE ANOTHER WEBCAM OR SOME VIDEO)
13 #EXAMPLE cam = cv2.VideoCapture(1), cam = cv2.VideoCapture(video.mp4)
14 cam = cv2.VideoCapture(0)
15
16 #FONT FOR PUT TEXT
17 font = cv2.FONT_HERSHEY_SIMPLEX
18
19 #WHILE FROM DETECT BANANAS, PENS, CARS, FACES, SMILES, EYES
20 while(True):
21     ret, img = cam.read()
22     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
23     cars = car_cascade.detectMultiScale(gray, 1.9, 1)
24     for (x,y,w,h) in cars:
25         cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
26         cv2.putText(img, 'Carro', (x,y), font, 0.8, (0, 0, 255), 2, cv2.LINE_AA)
27     pens = pen_cascade.detectMultiScale(gray, scaleFactor=1.24, minNeighbors=5, minSize=(30, 30), flags = cv2.CASCADE_SCALE_IMAGE)
28     for (x,y,w,h) in pens:
29         cv2.rectangle(img,(x,y),(x+w,y+h), (255,255,0), 2)
30         cv2.putText(img, 'Pluma', (x,y), font, 0.8, (255, 255, 0), 2, cv2.LINE_AA)
31     bananas = banana_cascade.detectMultiScale(gray, scaleFactor=1.24, minNeighbors=5, minSize=(30, 30), flags = cv2.CASCADE_SCALE_IMAGE)
32     for (x,y,w,h) in bananas:
33         cv2.rectangle(img,(x,y),(x+w,y+h), (0,255,0), 2)
```

Como cualquier programa en la parte superior comienza con las importaciones y despues de la relación de los archivos HAAR los cuales estan almacenados en la

carpeta de instalación del OpenCV, los incluimos allí para tener un orden con los archivos.

Luego de esto se paso a encender la cámara web, esto puede cambiarse solo si sustituimos el 0 por una ruta en la computadora como se explica en los comentarios puede visualizar un video, o en cambio si se tiene una cámara externa al poner 1 o 2 dependiendo de cual número detecte la pc puede hacer que se visualice.

```
cam = cv2.VideoCapture(1), cam = cv2.VideoCapture(video.mp4)
```

Por último se ejecuta lo que encuentra de los archivos que tomamos del framework y los ponemos con un rectángulo de colores diferentes para verificar diferencias, y también con una etiqueta en español.

```
34 cv2.putText(img, 'Banana', (x,y), font, 0.8, (0, 255, 0), 2, cv2.LINE_AA)
35 faces = face_cascade.detectMultiScale(gray, 1.3, 5)
36 for(x,y,w,h) in faces:
37     cv2.rectangle(img,(x,y), (x+w, y+h), (255,0,0), 2)
38     cv2.putText(img, 'Cara', (x,y), font, 0.8, (255, 0, 0), 2, cv2.LINE_AA)
39     roi_gray = gray[y:y+h, x:x+w]
40     roi_color = img[y:y+h, x:x+w]
41     eyes = eye_cascade.detectMultiScale(roi_gray)
42     for(ex,ey,ew,eh) in eyes:
43         cv2.rectangle(roi_color, (ex,ey), (ex+ew, ey+eh), (0,255,0), 2)
44         cv2.putText(roi_color, 'Ojo', (ex, ey), font, 0.8, (0, 255, 0), 2, cv2.LINE_AA)
45     smiles = smile_cascade.detectMultiScale(roi_gray,scaleFactor = 1.7,minNeighbors = 22,minSize = (25, 25),flags = cv2.CASCADE_SCALE_IMAGE)
46     for(zx,zy,zw,zh) in smiles:
47         cv2.rectangle(roi_color, (zx,zy), (zx+zw, zy+zh), (0,0,255), 2)
48         cv2.putText(roi_color, 'Sonrisa', (zx, zy), font, 0.8, (0, 0, 255), 2, cv2.LINE_AA)
49
50 #SHOW IMAGE OF WEBCAM IN SCREEN
51 cv2.imshow("img", img)
52 #IF YOU PRESS ESC BUTTON, EXIT
53 k = cv2.waitKey(30) & 0xff
54 if k == 27:
55     break
56 #DESTROY ALL WINDOWS CREATED BY OPENCV
57 cv2.destroyAllWindows()
```

¿Porque va todo dentro de un for?, eso va dentro de un for porque va checando en los pixeles en qué sección ve el objeto detectado, por eso algunas veces se confunde y da falsos positivos, al principio se tuvo problemas con la detección de autos, pero moviendo parametros que estan en el comando multiScale logramos darle una configuración óptima.

La sonrisa y los ojos van dentro del mismo for de la detección de la cara, ¿por qué?, porque los ojos nunca van a estar fuera del rango de la cara, tampoco la sonrisa, por eso al reconocer la cara, el for solo pasa por el área de la cara.

Para finalizar el programa al detectar que se presionó la tecla ESC finaliza el programa y cierra las ventanas de opencv.