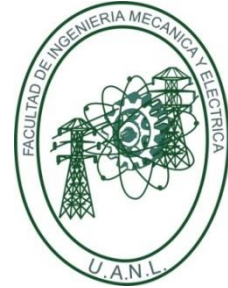




Universidad Autónoma
de Nuevo León



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

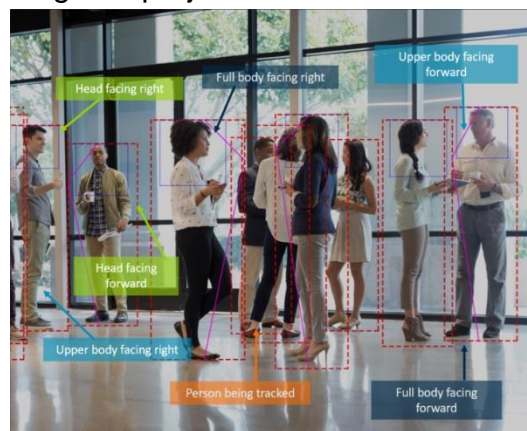
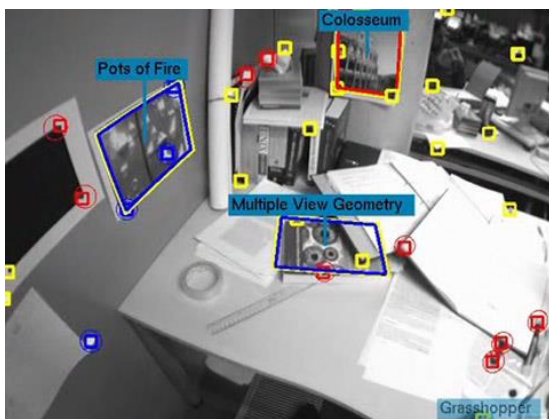
M. C. Laura Patricia del Bosque Vega

Why was the project chosen?

The project currently selected was not the one we had in mind, we had seen a project called BlinkToText which consisted of a facial recognition which detected when closed eyes were closed and then selected letters on the screen, this for people who could not use their extremities. But not being able to do it since another team discharged it first we decided to look for objects detection.

The project we found to perform was one called YOLO or as its translation says it is only seen once, which was developed by Pjreddie a user of GitHub which created detection of certain objects.

We found it interesting how to recognize objects only by showing them to the computer and was the main cause for selecting that project.



The project itself was found as object detection which consists of detecting people and objects from an image in the Python programming language, either in version 3.5 or 2.7, which is more stable.



How was the project created?

First it was necessary to take a look at the article of the project which talked about the detection of objects with the help of Open CV, Python, in any text editor.

For this it was necessary to install the necessary libraries for Python (3.5 or 2.7 that were the ones that we tried to carry out this project) as they are numpy and Open CV each one with its necessary commands, the program was tested in both Ubuntu and Windows.

Windows

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



Ubuntu

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

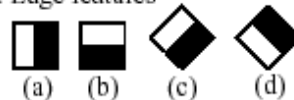


Once the libraries were ready, they went on to test the project, which consists of using HAAR files, which are classified by the HAAR classifier, but;

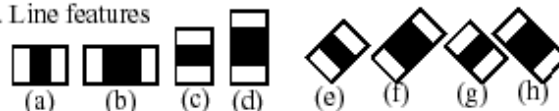
What is the HAAR classifier?

The HAAR classifier is a machine learning algorithm created by Paul Viola and Michael Jones in which it is about containing positive images in a folder and in another folder negative images of what is going to be done, in this case faces, in the folder Positive will only contain that, and in the other, whatever is less expensive.

1. Edge features



2. Line features



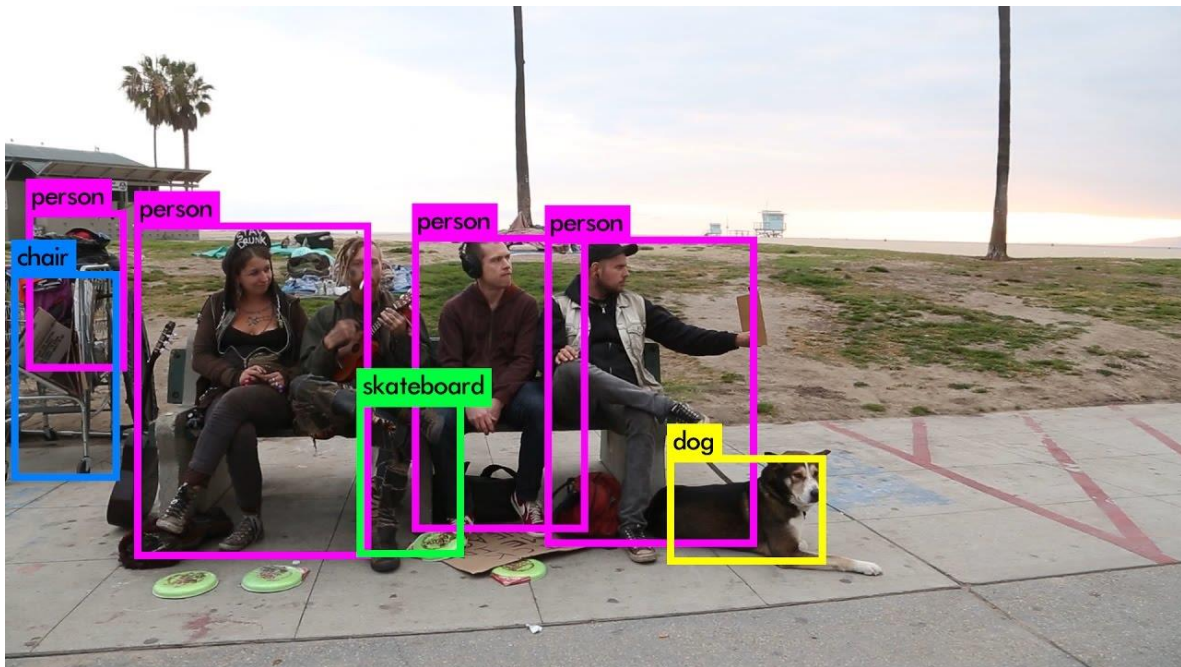
3. Center-surround features



It was also noted that there is another algorithm called LBP which is also good but has several disadvantages with HAAR, which for example as HAAR has a very high score with a false positive low, but in what is computational, LBP is much easier, faster and simpler, but therefore less efficient to detect objects.

From the project we can analyze how the face of a person is obtained, with the difference that they used the matplotlib library but they explain that it is optional because it is not necessarily useful.

Example



In this project is not mentioned how to use it for videos either on webcam or recorded in mp4 or avi format, we take its framework to use the recognition for what are eyes, smile, bananas, pens and cars, in real time or in videos.

How do we do it?

Following different tutorials of which they told us that having a folder with positive photos and others with negative photos could make the entire xml file to detect things as it was with the faces.

In a folder we had banana files for example:

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

While in other random images that have nothing to do with bananas, and having said that, the program is run which must be cloned from github with:

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

After downloading said repository you already have the files to train, for the folder of positives you must have a txt file with the information of each image and within which

pixels the object comes out, and in the negative ones you only have a file of txt information with your images. This is done with the commands:

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

When this is finished, the following is done:

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

The samples were created with:

```
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"
```

Finally, it is copied with the following command and executed to obtain the samples:

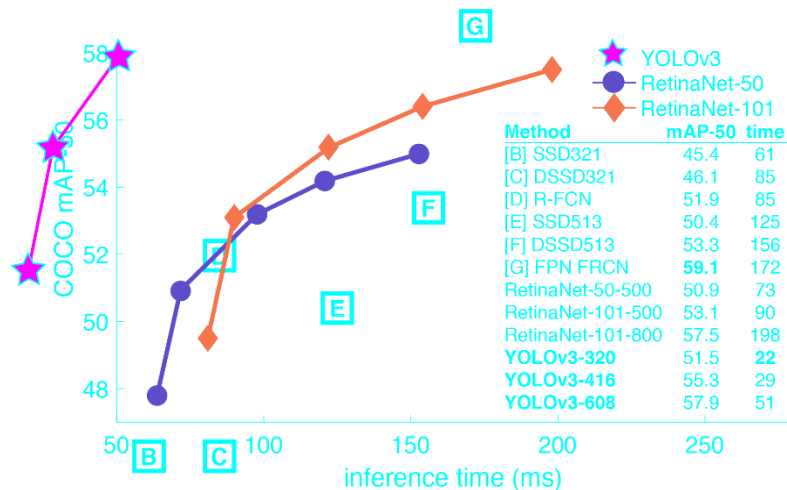
```
cp src/mergevec.cpp ~/opencv-2.4.5/apps/haartesting  
cd ~/opencv-2.4.5/apps/haartesting  
g++ `pkg-config --libs --cflags opencv` -I. -o mergevec mergevec.cpp\  
cvboost.cpp cvcommon.cpp cvsamples.cpp cvhaarclassifier.cpp\  
cvhaartesting.cpp\  
-lopencv_core -lopencv_calib3d -lopencv_imgproc -lopencv_highgui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

Creating the samples:

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

How does YOLO V3 work?

YOLO V3 is extremely fast and precise. In addition, you can sacrifice speed and accuracy simply by changing the size of the model (multimedia) without requiring any type of retraining.



Gráfica de pérdida focal contra velocidad.

Pre-detection systems reuse classifiers or locators to perform the detection. Apply the model to an image in multiple locations and scales. High-scoring regions of the image are considered detections.

In YOLO, a totally different approach is used. A single neural network is applied to the whole image. This network divides the image into regions and predicts limit and probability tables for each region. These delimiter boxes are weighted by the predicted probabilities.

This model has several advantages over classifier-based systems. This model looks at the whole image, so that its predictions are based on the global context of the image.

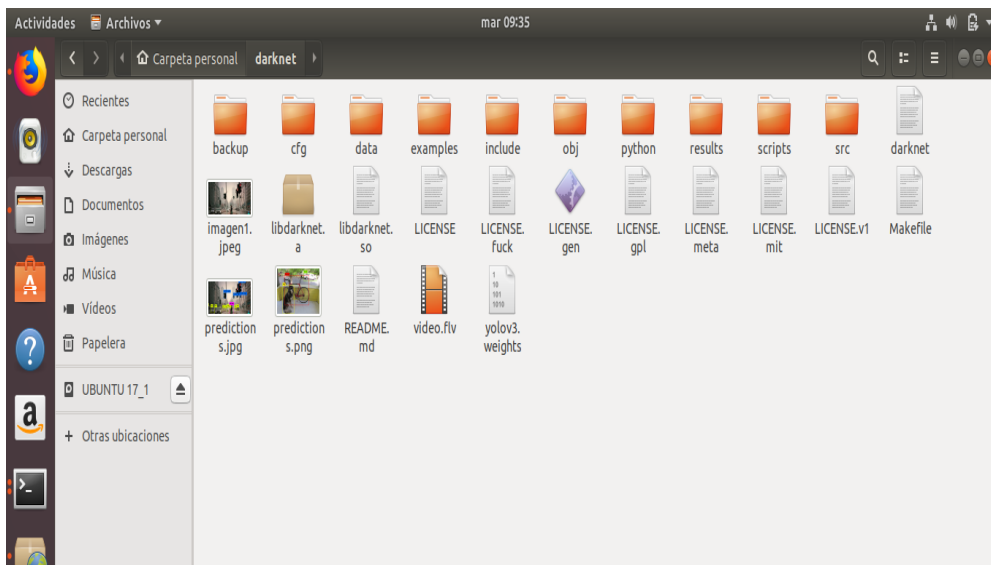
YOLO can compile with two forms:

- OpenCV: By using this compilation the CPU is used and many more image formats can be used.
- CUDA: This compilation uses the Nvidia technology GPU and makes the detections much faster, at the expense of less variety of image formats.

Darknet (YOLO) prints the objects you detected, your confidence and how long it took to find them. When Darknet compiles with OpenCV the detections are made by CPU and take between 6 and 12 seconds per image. If you use the CUDA version, it would be much faster.

Implementation

The project is constituted by this structure



Which includes all the necessary openCV libraries, as well as the training data for the recognition of objects, some scripts for its execution in ubuntu, the scripts in python and the files of the YOLO framework.

Once we have the whole project, we execute the make command on the bash terminal, this will create objects and download the necessary dependencies to start working:

```
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
```

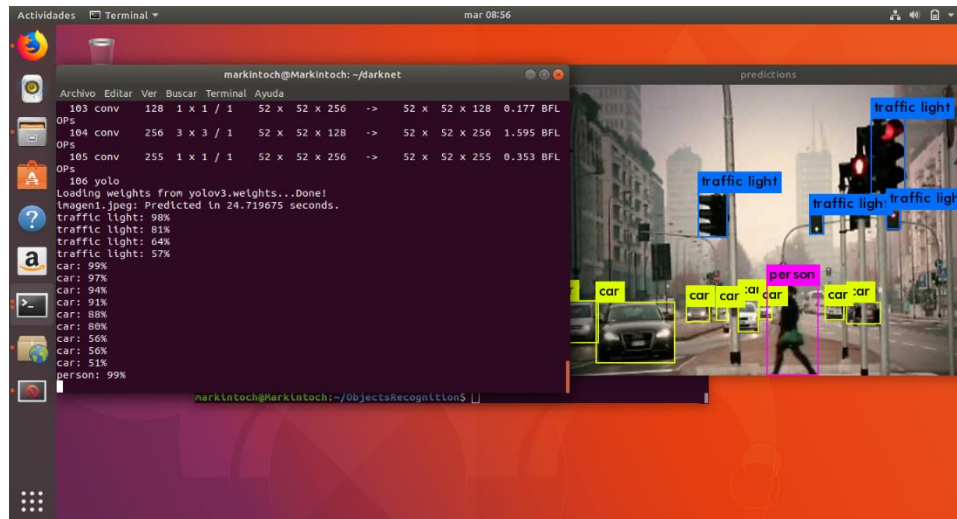
Once the dependencies have been downloaded, the training files of the data, the weights of the values for the detection of the objects with the framework, which are downloaded with this command, are needed:

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

By downloading the weights, we make use of these and execute the detect file, which will receive as a parameter the image from which it will detect the objects:

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

Throwing a result of percentage of reliability and the image edited with the detected objects and their respective labels:



Noting that the detection reliability index is almost perfect, it is very high.

This is where OpenCV is used, since in this case, it is the library in charge of manipulating the input image. As well as that the project can detect between the following objects:

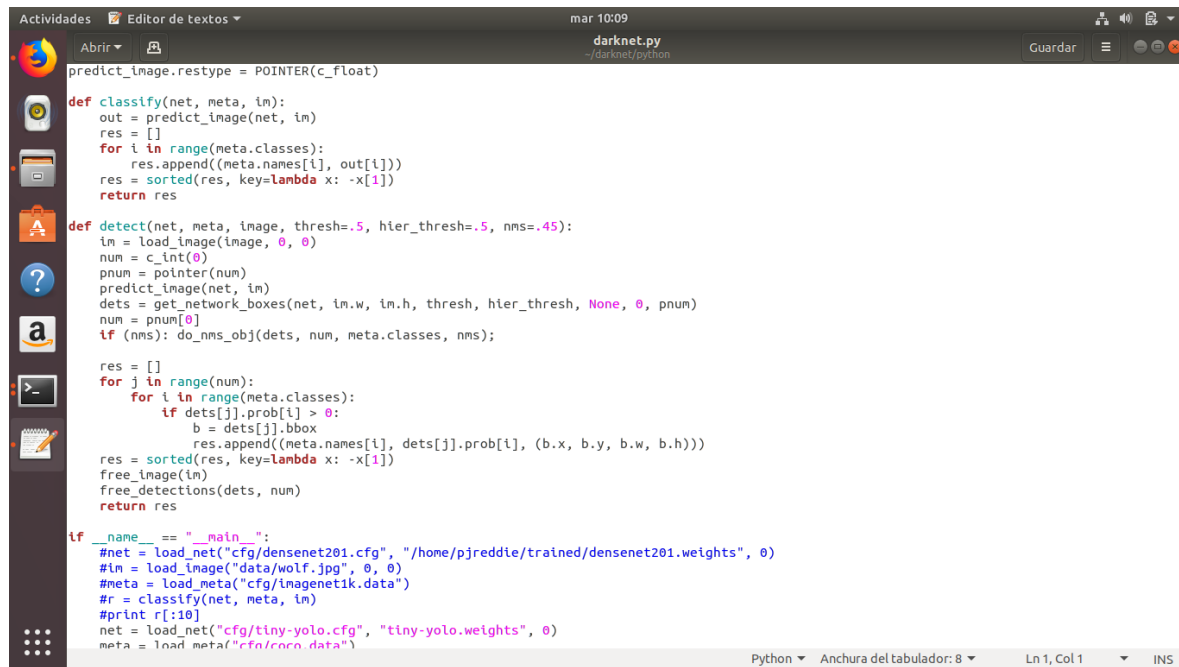
```
import xml.etree.ElementTree as ET
import pickle
import os
from os import listdir, getcwd
from os.path import join

sets=[('2012', 'train'), ('2012', 'val'), ('2007', 'train'), ('2007', 'val'), ('2007', 'test')]

classes = ["aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike",
"person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

def convert(size, box):
    dw = 1./size[0]
    dh = 1./size[1]
    x = (box[0] + box[1])/2.0 - 1
    y = (box[2] + box[3])/2.0 - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    y = y*dh
    w = w*dw
    h = h*dh
```


And the total operation is in the hands of darknet.py, which is responsible for detecting the objects and classify them, then pass them to lists and those lists about writing them in the original image, giving label and detection to the figure:



```

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)
    #im = load_image("data/wolf.jpg", 0, 0)
    #meta = load_meta("cfg/inagenet1k.data")
    #r = classify(net, meta, im)
    #print r[1:10]
    net = load_net("cfg/tiny-yolo.cfg", "tiny-yolo.weights", 0)
    meta = load_meta("cfg/tiny-yolo.data")

```

Operation of the main.py file

The main.py file is a file that we created from the YOLO framework, which is based on the use of xml files with the HAAR algorithm, for which we had to carry out such training with the xml files.

Code

```

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)

```

As any program at the top starts with the imports and after the relation of the HAAR files which are stored in the OpenCV installation folder, we include them there to have an order with the files.

After this happened to turn on the webcam, this can be changed only if we replace the 0 by a route on the computer as explained in the comments can watch a video, or instead if you have an external camera to put 1 or 2 Depending on which number the PC detects, it can be displayed.

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

Finally, what is found from the files that we took from the framework is executed and we put them with a rectangle of different colors to verify differences, and also with a Spanish label.

```
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()
```

Why is everything in a cycle for?, that goes inside a for because it is checking in the pixels in which section it sees the detected object, so sometimes it gets confused and gives false positives, at the beginning it had problems with the detection of cars, but moving parameters that are in the multiScale command we can give you an optimal configuration.

The smile and the eyes go within the same for the detection of the face, why? because the eyes are never going to be outside the range of the face, neither the smile, that's why when recognizing the face, the for it goes through the face area.

To end the program when detecting that the ESC key was pressed, the program ends and the opencv windows are closed.