

HAAR CASCADE

Es un enfoque de aprendizaje automático donde la función de cascada está entrenada a partir de muchas imágenes positivas y negativas. Luego se usa para detectar objetos en otras imágenes.

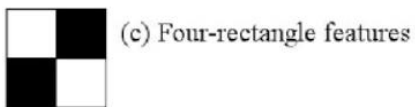
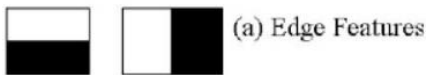
El haar cascade tienen algoritmo se puede explicar en cuatro etapas:

- Cálculo de las características de Haar
- Creación de imágenes integrales
- Usando Adaboost
- Implementación de clasificadores en cascada

Es importante recordar que este algoritmo requiere muchas **imágenes positivas** de objetos / rostros e **imágenes negativas** de no objetos / rostros para entrenar al clasificador, similar a otros modelos de aprendizaje automático.

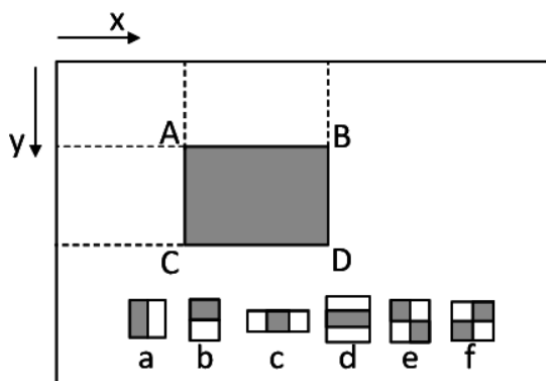
Cálculo de las características de Haar

El primer paso es recopilar las características de Haar. Una **característica de Haar** es esencialmente cálculos que se realizan en regiones rectangulares adyacentes en una ubicación específica en una ventana de detección. El cálculo implica sumar las intensidades de píxeles en cada región y calcular las diferencias entre las sumas. A continuación, se muestran algunos ejemplos de las funciones de Haar.



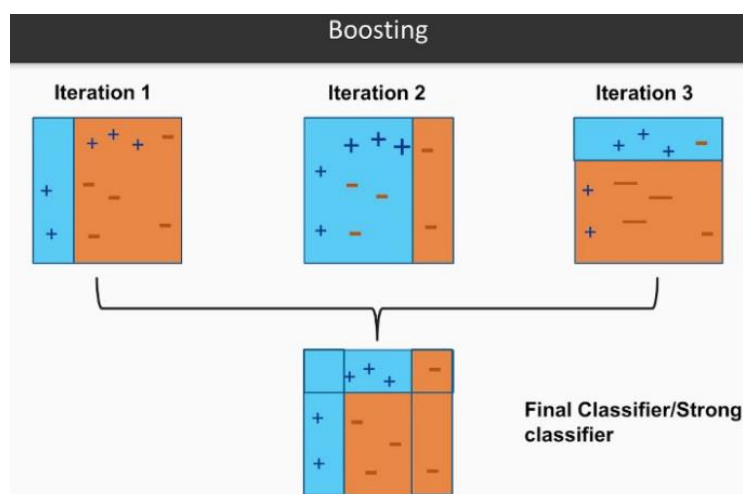
Creación de imágenes integrales

Las imágenes integrales esencialmente aceleran el cálculo de estas características de Haar. En lugar de calcular en cada píxel, crea sub-rectángulos y crea referencias de matriz para cada uno de esos sub-rectángulos. A continuación, se utilizan para calcular las características de Haar.

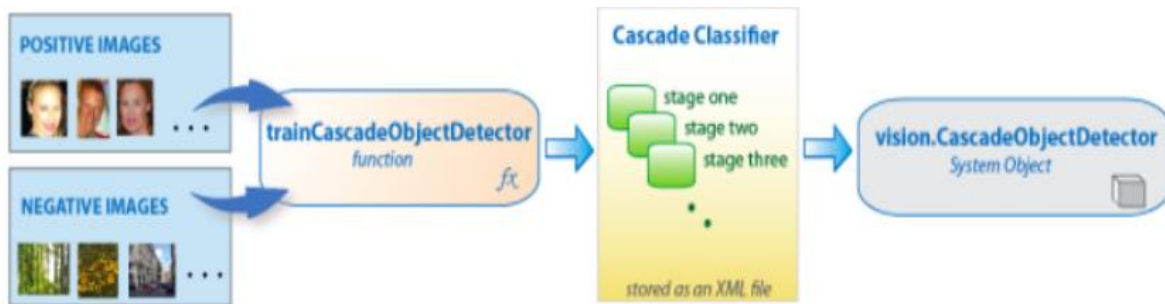


Entrenamiento Adaboost

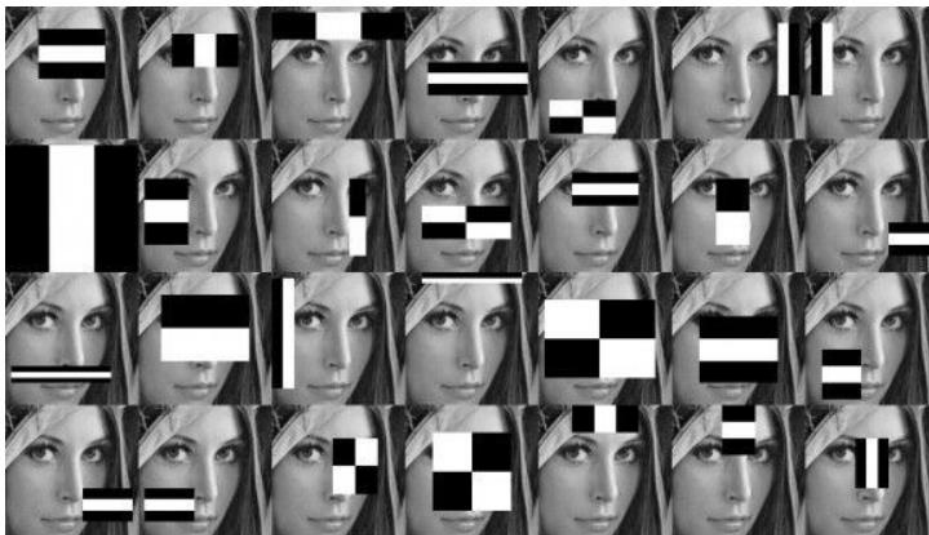
Adaboost esencialmente elige las mejores características y entrena a los clasificadores para que las utilicen. Utiliza una combinación de "**clasificadores débiles**" para crear un "**clasificador fuerte**" que el algoritmo puede usar para detectar objetos.



El clasificador en cascada se compone de una serie de etapas, donde cada etapa es una colección de estudiantes débiles. Los estudiantes débiles se entrenan mediante el refuerzo, lo que permite un clasificador de alta precisión a partir de la predicción media de todos los estudiantes débiles. Según esta predicción, el clasificador decide indicar que se encontró un objeto (positivo) o pasar a la siguiente región (negativo). Las etapas están diseñadas para rechazar muestras negativas lo más rápido posible, porque la mayoría de las ventanas no contienen nada de interés.



Ejemplo de como funciona



Cascade Trainer GUI

<https://amin-ahmadi.com/cascade-trainer-gui/>

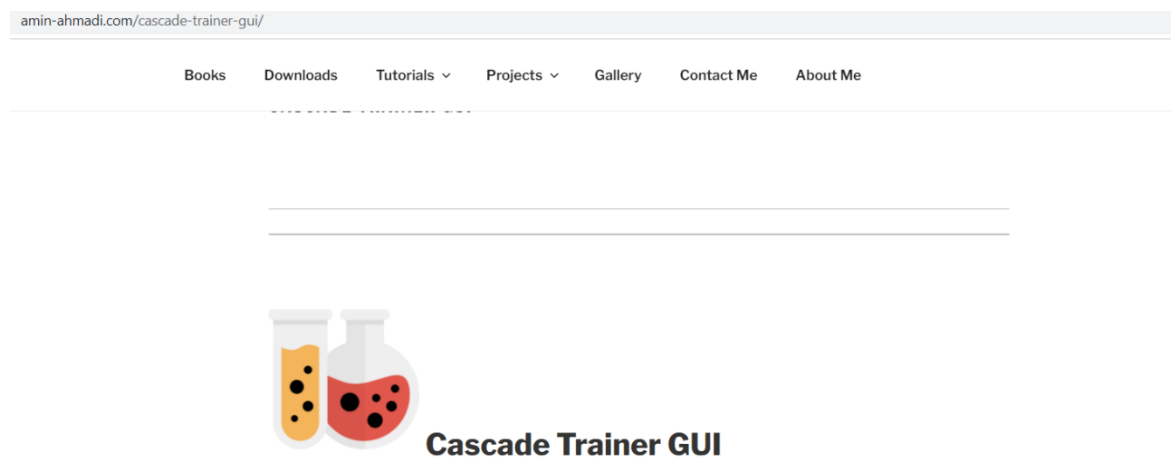
Es un programa que se puede utilizar para entrenar, probar y mejorar modelos de clasificadores en cascada. Utiliza una interfaz gráfica para establecer los parámetros y facilitar el uso de herramientas OpenCV para entrenar y probar clasificadores.

DETECTOR DE OBJETOS con Haar Cascade | Python y OpenCV

Lo primero que vamos a hacer es descargar el software Cascade Trainer GUI

<https://amin-ahmadi.com/cascade-trainer-gui/>

En la página encontraremos toda la información respectiva del software,



Nos vamos al final de la página y descargamos la versión para Windows que se requiera

4. Download

Updated: July 05, 2017 (Bug fixes and cropper improvements – version 3.3.1)

Updated: March 13, 2017 (Updated Windows versions to 3.1.1.0 which includes fixed cropper tool)

Updated: November 09, 2016 (Updated Ubuntu version to 2.1.10)

Updated: November 07, 2016 (Many enhancements to cropper, training and testing utilities. Version x64 for Windows is built using Intel TBB and works much faster now for training. It is also the recommended version as of now.)

Updated: October 27, 2016 (Added two more parameters to UI. Neg count and Pos usage percentage.)

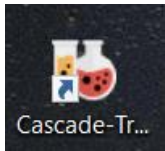
Updated: October 19, 2016 (Fixed an issue with file names in positive and negative images folders.)



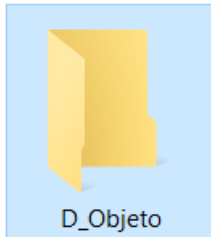
Cascade Trainer GUI 3.3.1 (Windows x86)



Instalamos el programa y coloca un icono en escritorio



Luego creamos una carpeta con el nombre del proyecto en mi caso es D_objeto



Vamos a hacer el programa para recolectar la información para entrenar el modelo, requerimos imágenes positivas del objeto e imágenes del fondo donde estará el objeto. La idea es tomar las imágenes en el mismo fondo que se va a reconocer, hay q tener presente fondo claro que no tenga mucho contraste de igual forma la luz

Luego vamos a nuestro IDE y creamos un archivo llamado Objetos_PO.py con el siguiente código este guardara en la carpeta "p" las imágenes positivas del objeto

```
import cv2
import numpy as np
import imutils
import os
#librerias que se requieren

Datos = 'p'
if not os.path.exists(Datos):
    print('Carpeta creada: ', Datos)
    os.makedirs(Datos)
    #se crea la carpeta "p" para guardar imagenes positivas

cap = cv2.VideoCapture(0,cv2.CAP_DSHOW) # para abrir video streaming webcam

#x1, y1 dibujamos rectangulo en el centro de la Imagen
x1, y1 = 190, 80
x2, y2 = 450, 398
```

```

count = 0# variable para guardar varias imagenes
while True:

    ret, frame = cap.read()
    if ret == False: break
    imAux = frame.copy()#dibuje el recuadro sin borde azul
    cv2.rectangle(frame,(x1,y1),(x2,y2),(255,0,0),2)#cargamos el rectangulo
    en la imagen

    objeto = imAux[y1:y2,x1:x2]#la parte central se guarda en variable
    objeto = imutils.resize(objeto, width=38)#colocar la escala de la imagen
    en 38
    # print(objeto.shape)

    k = cv2.waitKey(1)

    #Si presionamos tecla s guardaremos la imagen en la carpeta
    #que corresponde la variable datos P, con el nombre de Objeto_ la variabl
    e count,
    #también se imprimirá un mensaje en consola y finalmente aumentamos en 1 el
    valor
    #de count para poder seguir almacenando las otras imágenes
    if k == ord('s'):
        cv2.imwrite(Datos+'/objeto_{}.jpg'.format(count),objeto)
        print('Imagen almacenada: ', 'objeto_{}.jpg'.format(count))
        count = count + 1
    if k == 27:
        break

    cv2.imshow('frame',frame)
    cv2.imshow('objeto',objeto)

cap.release()
cv2.destroyAllWindows()

```

Luego creamos otro archivo llamado Objetos_NE.py con el siguiente código este guardara en la carpeta "n" las imágenes negativas del objeto.

```
import cv2
import numpy as np
import imutils
import os
#librerias que se requieren

Datos = 'n'
if not os.path.exists(Datos):
    print('Carpeta creada: ', Datos)
    os.makedirs(Datos)
    #se crea la carpeta "p" para guardar imagenes positivas

cap = cv2.VideoCapture(0,cv2.CAP_DSHOW) # para abrir video striming webcam

#x1, y1 dibujamos rectangulo en el centro de la Imagen
x1, y1 = 190, 80
x2, y2 = 450, 398

count = 0# variable para guardar varias imagenes
while True:

    ret, frame = cap.read()
    if ret == False: break
    imAux = frame.copy()#dibuje el recuadro sin borde azul
    cv2.rectangle(frame,(x1,y1),(x2,y2),(255,0,0),2)#cargamos el rectangulo
    en la imagen

    objeto = imAux[y1:y2,x1:x2]#la parte central se guarda en variable
    objeto = imutils.resize(objeto, width=38)#colocar la escala de la imagen
    en 38
    # print(objeto.shape)

    k = cv2.waitKey(1)

    #Si presionamos tecla s guardaremos la imagen en la carpeta
    #que corresponde la variable datos P, con el nombre de Objeto_ la variabl
    e count,
    #también se imprimirá un mensaje en consola y finalmente aumentamos en 1 el
    valor
    #de count para poder seguir almacenando las otras imágenes
```

```

if k == ord('s'):
    cv2.imwrite(Datos+'/objeto_{}.jpg'.format(count),objeto)
    print('Imagen almacenada: ', 'objeto_{}.jpg'.format(count))
    count = count + 1
if k == 27:
    break

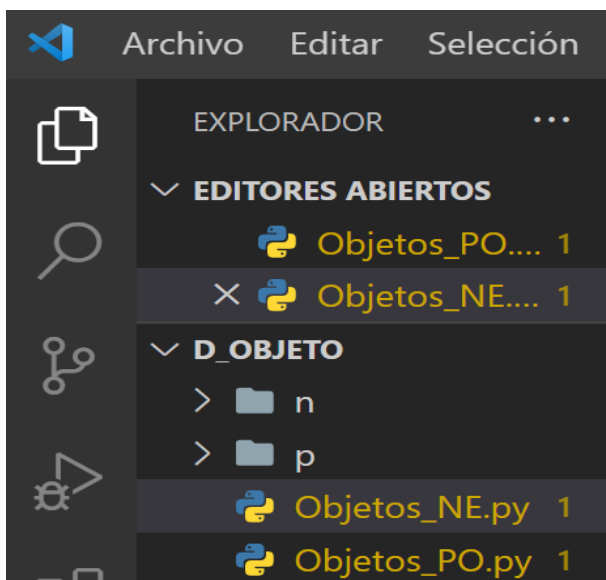
cv2.imshow('frame',frame)
cv2.imshow('objeto',objeto)

cap.release()
cv2.destroyAllWindows()

```

← → ∨ ↑ > Este equipo > Escritorio > IA > cnca > D_Objeto

Nombre	Fecha de modificación	Tipo
n	06/10/2021 17:42	Carpet
p	06/10/2021 17:39	Carpet
Objetos_NE.py	06/10/2021 17:33	Python
Objetos_PO.py	06/10/2021 17:33	Python



Luego ejecutamos el archivo Objetos_PO.py desde la terminal de Visual Code en mi caso lo hago desde cmd

```
C:\Users\Redes Cableadas\Desktop\IA\cnca\D_Objeto>python Objetos_PO.py  
Carpeta creada: p
```

Se crea la carpeta y muestra la ventana donde se ve la cam y queda preparada para capturar la imágenes

Al darle click a la tecla "S", guarda imagen en la carpeta definida. Se capturan 50 imágenes positivas

```
Imagen almacenada: objeto_43.jpg  
Imagen almacenada: objeto_44.jpg  
Imagen almacenada: objeto_45.jpg  
Imagen almacenada: objeto_46.jpg  
Imagen almacenada: objeto_47.jpg  
Imagen almacenada: objeto_48.jpg  
Imagen almacenada: objeto_49.jpg  
Imagen almacenada: objeto_50.jpg
```

Luego ejecutamos el archivo Objeto_NE.py para capturar las imágenes negativas

```
C:\Users\Redes Cableadas\Desktop\IA\cnca\D_Objeto>python Objetos_NE.py  
Carpeta creada: n
```

Se crea la carpeta y muestra la ventana donde se ve la cam y queda preparada para capturar las imágenes

Al darle click a la tecla "S", guarda imagen en la carpeta definida. Se capturan 350 imágenes negativas

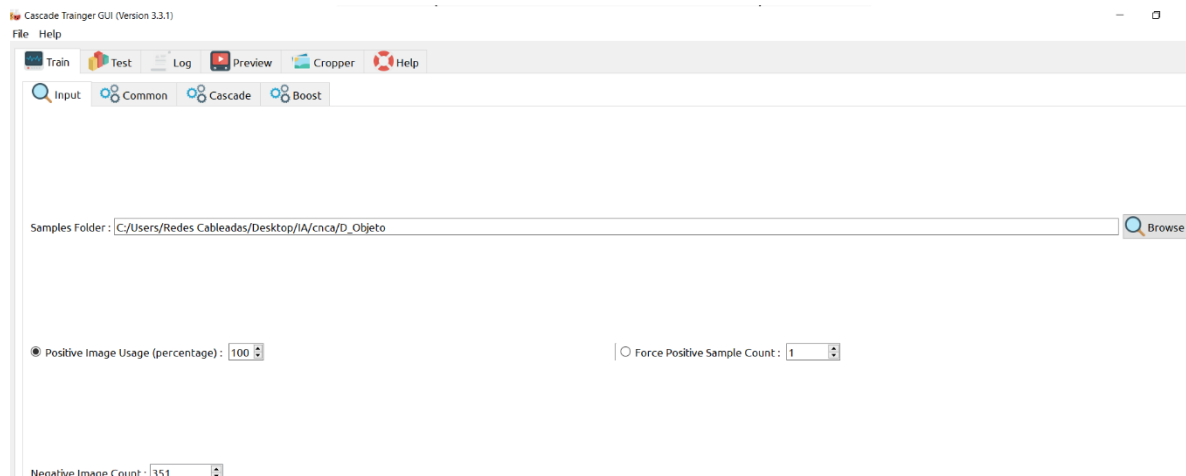
```
Imagen almacenada: objeto_344.jpg  
Imagen almacenada: objeto_345.jpg  
Imagen almacenada: objeto_346.jpg  
Imagen almacenada: objeto_347.jpg  
Imagen almacenada: objeto_348.jpg  
Imagen almacenada: objeto_349.jpg  
Imagen almacenada: objeto_350.jpg
```

Ya teniendo las dos carpetas con las imágenes nos vamos al programa Cascade Trainer GUI que esta en escritorio y lo ejecutamos

NOTA

recuerden mientras más imágenes mejor será el modelo, imágenes positivas podrían ser 100 y negativas podrían ser 500 o mas

Nos abre el programa y nos vamos a pestaña INPUT



Dentro de INPUT nos vamos a BROWSE

Buscamos la carpeta del proyecto, donde están las carpetas (n,p)

Samples Folder : C:/Users/Redes Cableadas/Desktop/IA/cnca/D_Objeto

En imágenes positiva colocamos 100%

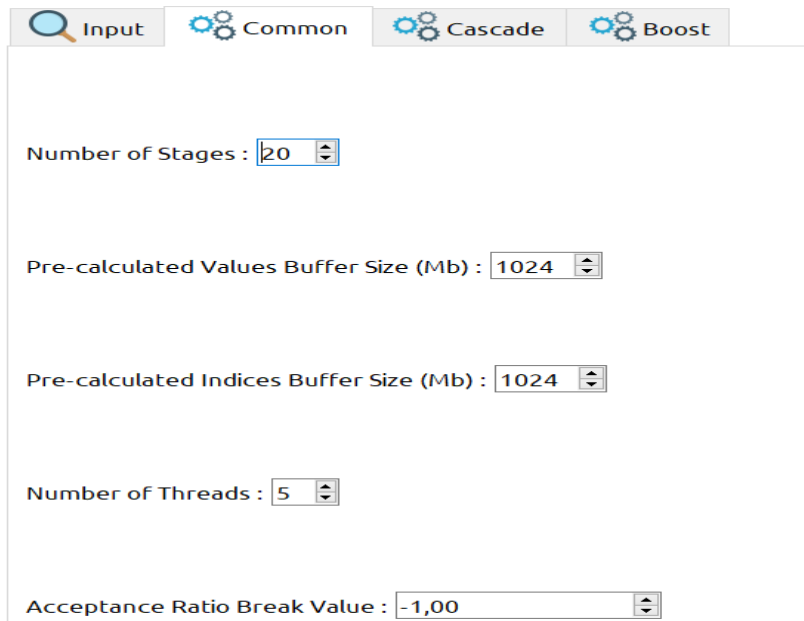
En imágenes negativas colocamos la cantidad de imágenes negativas que hay dentro la carpeta N

Positive Image Usage (percentage) : 100

Negative Image Count : 351

Luego nos vamos a la pestaña COMMON

Aquí dejamos todo igual como esta

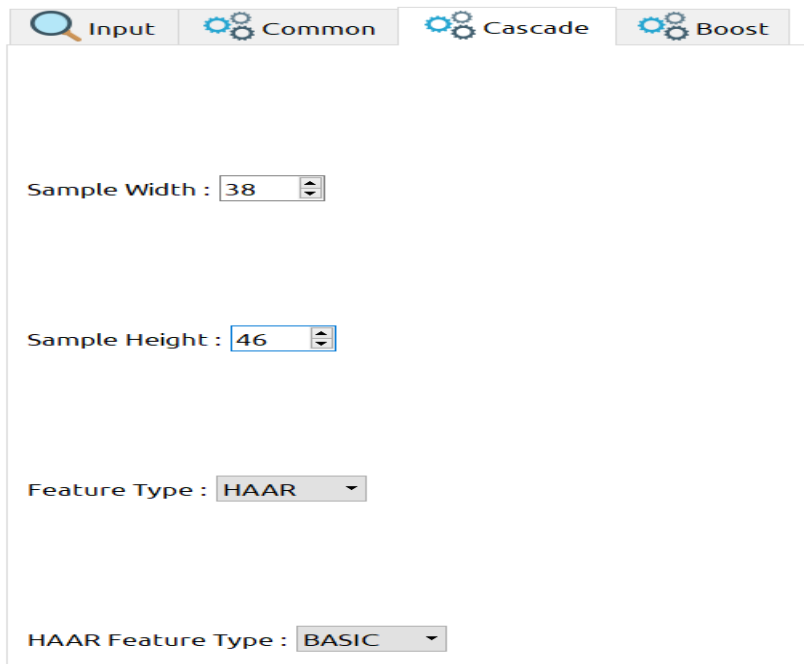


The image shows a software interface with four tabs: 'Input', 'Common', 'Cascade', and 'Boost'. The 'Common' tab is selected. Below the tabs, there are five settings, each with a label and a numeric input field with up/down arrows:

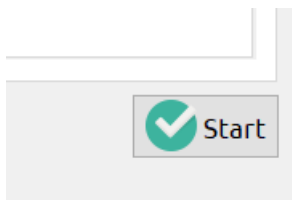
- Number of Stages : 20
- Pre-calculated Values Buffer Size (Mb) : 1024
- Pre-calculated Indices Buffer Size (Mb) : 1024
- Number of Threads : 5
- Acceptance Ratio Break Value : -1,00

Luego nos vamos a la carpeta CASCADE

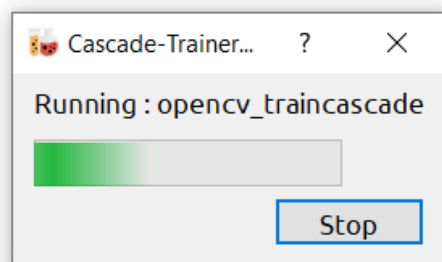
Aquí colocamos las dimensiones de las imágenes guardadas las cuales por código son 38x46, lo demás queda igual



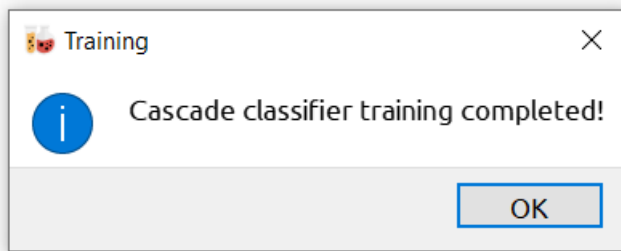
Luego nos vamos al botón Start que esta en la esquina inferior derecha



El programa comienza a correr y entrenar las imágenes para crear el archivo cascade.xml



Al finalizar nos muestra el siguiente mensaje, el tiempo depende de los característica del equipo



Al terminar el programa crea la carpeta Classifier la cual tiene el archivo cascade.xml.
El cual debemos copiar y colocarlo fuera de la carpeta

📁 classifier

📄 cascade.xml
📄 log.txt
📄 params.xml
📄 stage0.xml

Posteriormente se borran los archivos neg, los archivos pos

Quedando de la siguiente forma

Nombre ^

📁 classifier
📁 n
📁 p
📄 cascade.xml
📄 Objetos_NE.py
📄 Objetos_PO.py

Luego creamos el archivo detectando.py con el siguiente código

```
import cv2

cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)# utilizar la wecam
```

```

majinBooClassif = cv2.CascadeClassifier('cascade.xml')#para utilizar el modelo que genera el soft

while True:

    ret,frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    toy = majinBooClassif.detectMultiScale(gray,
    scaleFactor = 9,
    minNeighbors = 91,
    minSize=(70,78))#se convierte la imagen en escala de gris y se guarda en variable toy

    for (x,y,w,h) in toy:
        cv2.rectangle(frame, (x,y),(x+w,y+h),(0,255,0),2)#dibujar el rectangulo
        cv2.putText(frame,'MUG OF COFFE',(x,y-10),2,0.7,(0,255,0),2,cv2.LINE_AA)#colocar el nombre del objeto

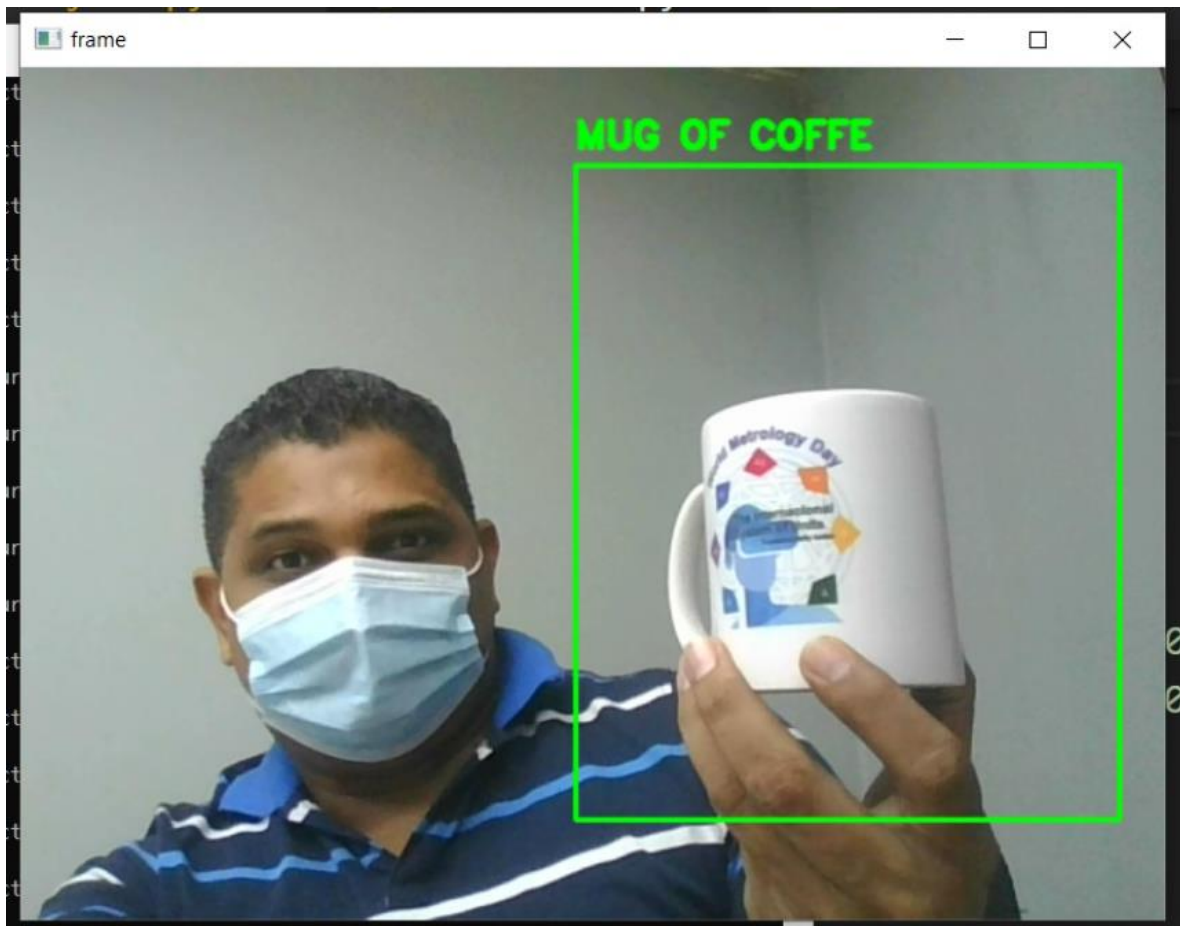
    cv2.imshow('frame',frame)

    if cv2.waitKey(1) == 27:
        break
cap.release()
cv2.destroyAllWindows()

```

luego colocamos a correr el archivo

```
C:\Users\Redes Cableadas\Desktop\IA\cnca\D_Objeto>python detectando.py
```



El sistema inicialmente trata de calcular la imagen acá influyen varios factores, la luz la calidad de la cam, el fondo claro que no tenga mucho contraste entre otras cosas.

Es importante que juegues con los valores de las líneas 13,14,15 del archivo detectando.py

Para encontrar mejores resultados.

```
12 toy = majinBooClass
13 scaleFactor = 9,
14 minNeighbors = 91,
15 minSize=(70,78))#se
```

DETECTMULTISCALE

Para emplear la detección de rostros con haar cascade en OpenCV vamos a necesitar del módulo detectMultiScale que ayudará a detectar los objetos de acuerdo al clasificador que se utilice. Este nos permitirá obtener un rectángulo delimitador en donde se encuentre el objeto que se desea encontrar dentro de una imagen, para ello se deben especificar algunos argumentos que veremos a continuación

Image: Es la *imagen* en donde va a actuar el detector de rostros.

ScaleFactor: Este parámetro *especifica que tanto va a ser reducida la imagen*. Por ejemplo si se ingresa 1.1, quiere decir que se va a ir reduciendo la imagen en 10%, con 1.3 se reducirá 30%, creando de esta manera una pirámide de imágenes. Hay algo que debemos tener en cuenta y es que si damos un número muy alto, se pierden algunas detecciones. Mientras que para valores muy pequeños como por ejemplo 1,01 (es decir reducir en un 1% la imagen), llevará mayor tiempo de procesamiento, ya que se tendrán más imágenes a analizar, además de que pueden incrementar los falsos positivos (que son detecciones presentadas como objetos u rostros, pero que en realidad no lo son).

MinNeighbors: *Este parámetro especifica cuántos vecinos debe tener cada rectángulo candidato para retenerlo.*

Una pequeña ventana que va a ir pasando por una imagen buscando rostros, entonces puede que te encuentres que al final de todo el proceso ha identificado varios rostros, pero puede que muchos de ellos correspondan a la misma persona. Entonces este parámetro, hace relación a todos esos rectángulos delimitadores de un mismo rostro. Por lo cual minNeighbors especifica el número mínimo de cuadros delimitadores o vecinos, que debe tener un rostro para que detectado como tal. Ahora, debemos tener en cuenta que entre más alto sea el valor que pongamos, menos caras se detectaran, mientras que si se da un valor muy bajo se pueden presentar falsos positivos.

MinSize: Este parámetro indica el tamaño mínimo posible del objeto. Objetos más pequeños son ignorados.

MaxSize: Este parámetro indica el tamaño máximo posible del objeto. Objetos más grandes son ignorados.

NOTA: Recuerda que la combinación de los valores que le des a cada parámetro (especialmente ScaleFactor y MinNeighbors) afectará a las detecciones. Y por cierto este

procedimiento puede aplicarse a cualquier otro detector de objetos que use esta técnica, como los clasificadores pre entrenados que te comenté o podrías realizar el tuyo propio.