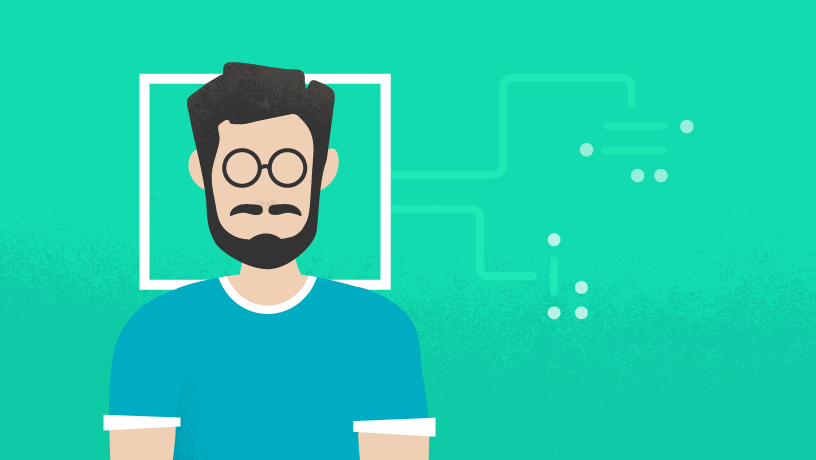
**Tarea 7 – Haar Cascades para Detección y Entrenamiento de Detectores**

**Detección Facial**



La Detección Facial en base a Haar Cascades también se le conoce “Viola-Jones Detector”. En el 2001 era estado del arte para detección.

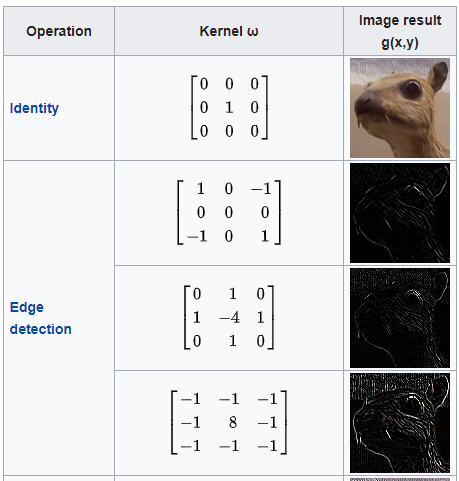
El objetivo de este laboratorio es el de utilizar python y la librería de OpenCV para realizar en primera instancia un detector facial basado en un extractor de características o el “Haar Feature-based cascade classifier”

**Introducción**

El método de detección de clasificación de cascadas es un método efectivo de detección de objetos. Este está basado en el paper de Paul Viola y Michael Jones llamado “Rapid Object Detection using a Boosted Cascade of Simple Features” en 2001. Prácticamente es un tipo de aprendizaje supervisado en donde obtenemos un grupo de muestras positivas y negativas. Este era estado del arte para estos tiempos y la malloría de los dispositivos como cámaras digitales fotográficas que aplicaban detección facial, tenían este algoritmo implementado en su “core”.

En principio el algoritmo inicialmente necesita una gran cantidad de imágenes en muestras positivas, estas “muestras positivas” son las imágenes que utilizará el algoritmo de entrenamiento para detección y además necesitamos muchas muestras negativas que son imágenes, en este caso, donde no haya caras.

Para extraer las características, el clasificador haar utiliza las imágenes que se muestran a continuación como kernel convolucional. Hablando un poco más de que representa un kernel, este en procesamiento de imágenes es una máscara que se le aplica a una o un grupo de imágenes y cuyo resultado es “filtrar”, detectar bordes, aplicar niveles. El kernel se pasará sobre la imagen y la multiplicación matricial generará una nueva imagen “filtrada”.



Ejemplo de diferentes kernels. Se pueden aplicar a una imagen para detectar bordes, afinar, suavizar, remover ruido, etc.

Cada caraterística es extraida sustrayendo la suma de los pixeles debajo de la región blanca de la suma de pixeles de la región negra del rectángulo.



Ejemplo de detección de características.

Se realizan muchas posibilidades o extracción de características. A medida que la imagen crece las características aumentan, por ejemplo, para una imagen de 24x24 son 160k posibles características. Para cada una de estas se necesita entonces encontrar los pixeles como comentamos anteriormente. Para reducir este problema se introdujo el concepto de imagen integral que reduce la dimensionalidad de cálculo a medida que el tamaño de la imagen aumenta reduciendo el cálculo a solo cuatro pixeles.

A pesar de parecer ventajoso quizá la mayoría de las características calculadas son irrelevantes. Considerando la siguiente imagen inferior; las filas superiores muestran dos buenas características sobre los ojos en la parte oscura y la separación de ojos y nariz. Para el siguiente kernel quizá solamente es relevante la característica en esta zona pues muestra igualemente los ojos, pero es irrelevante para las siguientes características.

El algoritmo utiliza Adaboost para seleccionar las mejores características de las 160k.



Características a extraer. Ojos para kernel horizontal y vertical.

Adaboost lo que realiza es aplicar para cada característica el mejor umbra en el cual se clasifica la cara como positiva o negativa. Obviamente habrán errores de clasificación, por consiguiente se debe seleccionar características con el mínimo porcentaje de error, es decir, las más probables a ser caras (u otro objeto que queramos clasificar).

El paper dice que aún con 200 características se pueden tener 95% de precisión y el final es de 6k características de las 160k, lo que conlleva a que el algoritmo haga rápido su labor. Aún así, es algo ineficiente que en el paper se cubre.

La propuesta se basa en que como hay regiones donde no aparecen el elemento a detectar (cara en este caso), no procesar esta región y seguir a la siguiente, solo concentrándose en las que hay una cara, por ejemplo. Es aquí a donde llegamos a los clasificadores de cascada, que en vez de aplicar las 6k características a la ventana estas son agrupadas en diferentes estados o cascadas y aplicadas una por una.

Al solo considerar las características que se requieren detectar entonces procesa el siguiente estado en dooden está la cara. De las 6000 cracterísticas con 38 estados se dan 1, 10, 25, 25 y 50 características para los primeros estados. Y aseguran que alrededor de 10 características son evaluadas de acorde a cada sub entana.

**Implementando Haar-Cascades en OpenCV**

Para nuestra suerte OpenCV tiene un detector ya pre-entrenado incorporado, si desea entrenar su propio detector, en la siguiente sección después de esta se da un ejemplo de como hacer su propio detector.

Veamos el ejemplo en Python de como detectar caras y ojos en OpenCV

import numpy as np

import cv2 as cv

face\_cascade = cv.CascadeClassifier('haarcascade\_frontalface\_default.xml')

eye\_cascade = cv.CascadeClassifier('haarcascade\_eye.xml')

img = cv.imread(“faces.jpg”)

gray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY)

faces = face\_cascade.detectMultiScale(gray, 1.3, 5)

for (x,y,w,h) in faces:

cv.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

roi\_gray = gray[y:y+h, x:x+w]

roi\_color = img[y:y+h, x:x+w]

eyes = eye\_cascade.detectMultiScale(roi\_gray)

for (ex,ey,ew,eh) in eyes:

cv.rectangle(roi\_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv.imshow('img',img)

cv.waitKey(0)

cv.destroyAllWindows()

Las primeras líneas importan librerías como numpy y opencv necesarias para trabajar

Luego importamos los detectores pre-entrenados de cara y ojos.

Seguido entonces buscamos la región de caras, si existe, extraemos esta región de interes y volvemos a verificar sobre esta imagen si existen ojos para finalmente enmarcarlos.

Lo ultimo despliega nuestro resultado en una ventana para verifiar el resultado.

Ejecútelo como sigue: python haar\_cascade.py

**Entrenando nuestro propio Haar Cascades classifier**

Ahora crearemos nuestro propio detector de imágenes basado en “haar cascades”. Principalmente debe obtener la siguiente estructura de archivos como sigue a continuación

|Project\_Folder|

|

|--- haar

|--- negatives

|--- positives

|--- samples

|--- createTrainData.py

|--- detect\_obj\_image.py

|--- detect\_obj\_video.py

|--- download\_images.py

|--- mergeVec.py

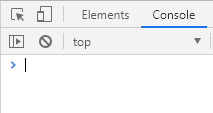
|--- savedirtofile.py

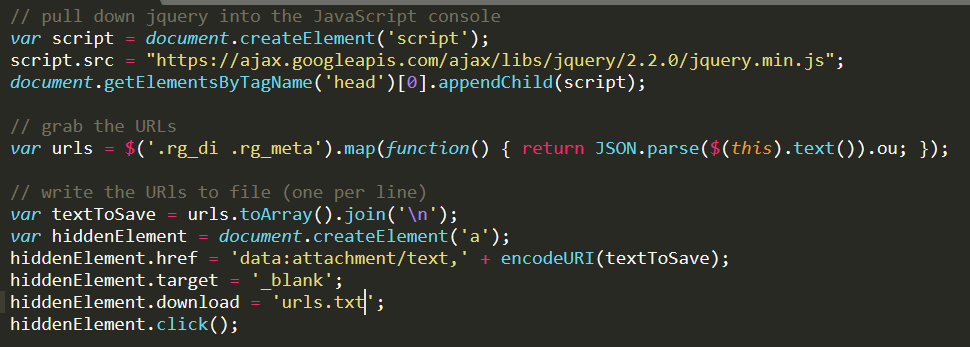
|--- js\_console.js

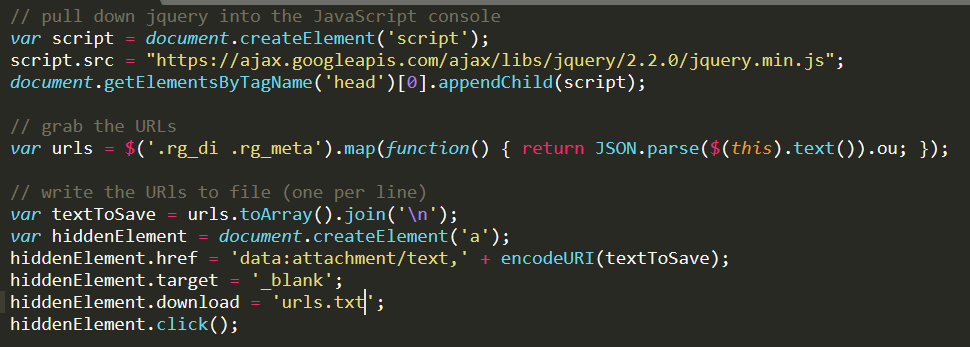
En donde haar, negatives, positives, samples son directorios vacíos en su primer uso.

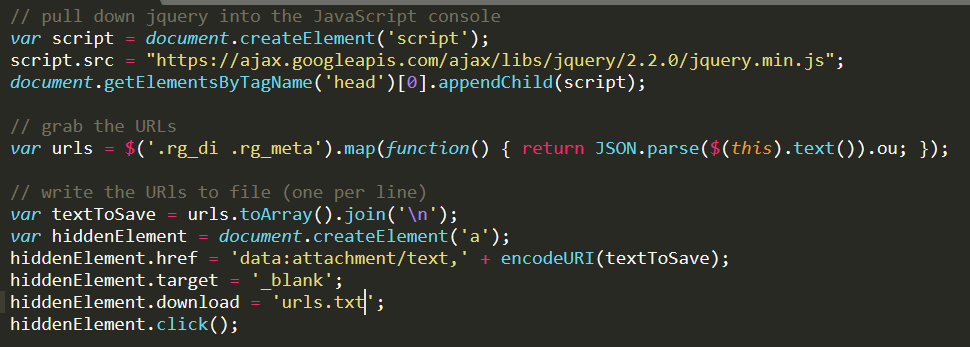
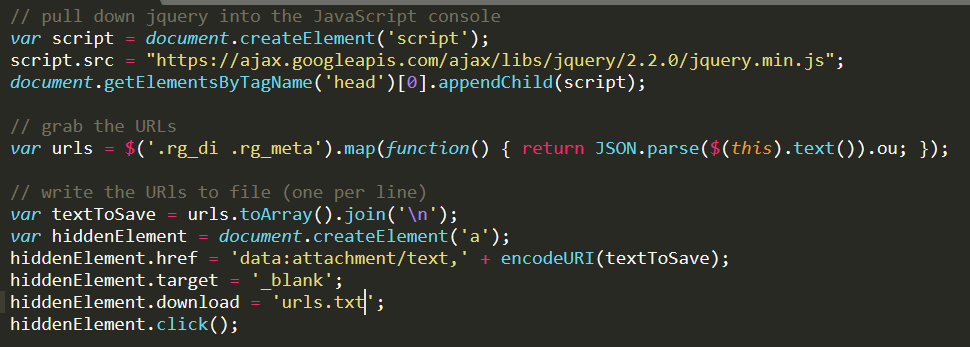
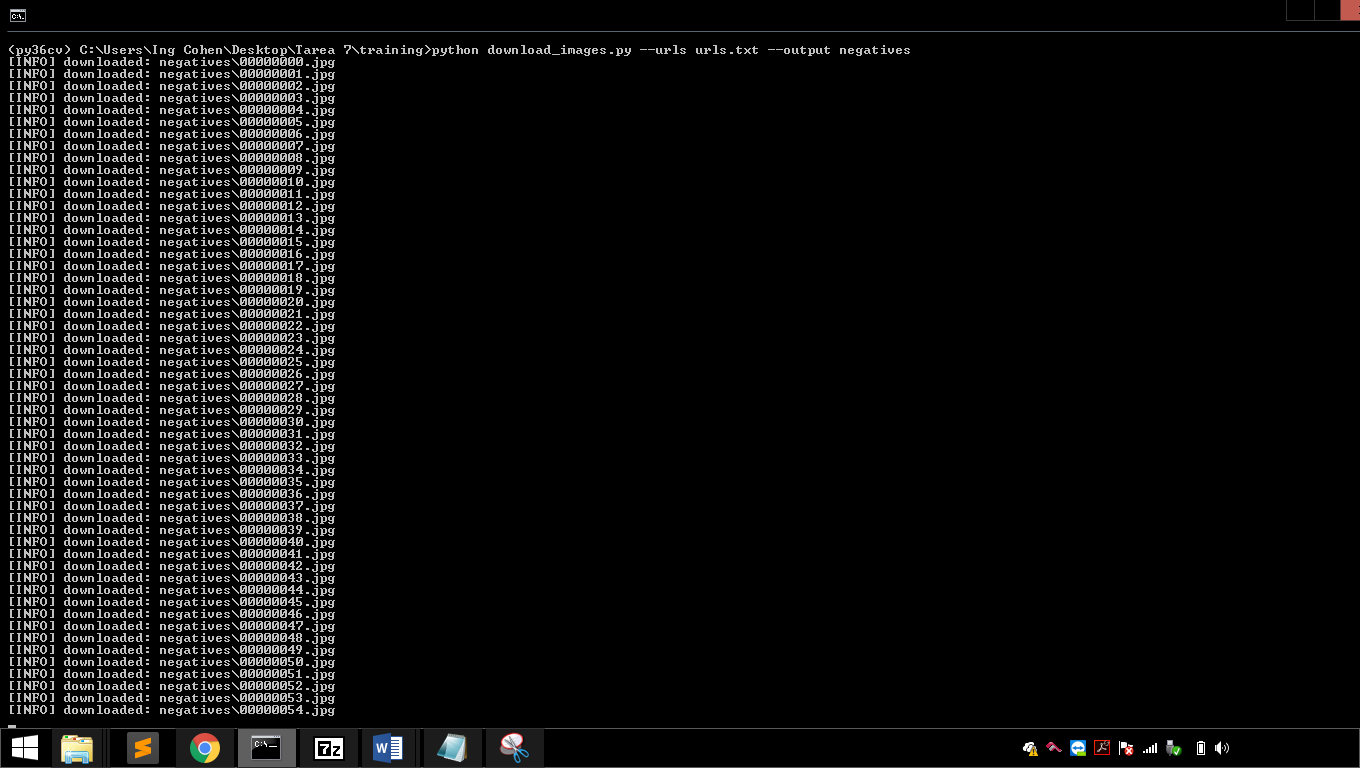
Paso 1: Colectar imágenes positivas o negativas

* Las imágenes positivas en este caso se utiliza el dataset de CalTech
  + <http://www.vision.caltech.edu/Image_Datasets/Caltech101/>
* Luego de descargar el dataset llamado 101\_ObjectCategories seleccione por ejemplo un grupo de imagenes que desea detectar, por ejemplo, en nuestro caso elegimos imágenes de leopardo
* Incluya las imágenes de leopardo en el folder llamado “positives”
* Para las imágenes negativas buscaremos estas imágenes buscando en google y con el código de Adrian Rosebrock (pyimagesearch) para descarga de imágenes de google.
* Buscar imágenes que no sean de leopardo en google
* Hacer scroll hacia abajo hasta la cantidad de imágenes que usted desee desplegar
* Presionar Shift+Control+I
* Hacer click en el tab llamado “Console”



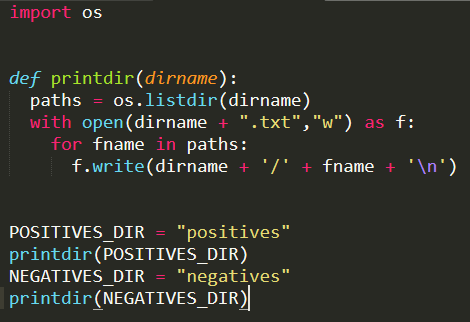
* Abrir el archivo llamado js\_console.js
*  Copiar el contenido del primer párrafo a la consola y presione Enter



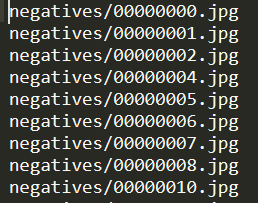
* Copiar el contenido del segundo párrafo a la consola y presionar Enter
* 
* Copiar el contenido del tercer párrafo a la consola y presionar Enter}
* Si lo que realizó fue correcto se debió de descargar un archivo llamado urls.txt que contiene todas las ubicaciones web de los archivos. Ubique este archivo en la carpeta de proyectos.
* Abra su ambiente virtual de trabajo, por ejemplo, si se llama py3, workon py3
* Ahora instale la librería imutils como sigue: pip install imutils
* Adicionalmente instale la librería requests como sigue: pip install requests
* Ejecute el siguiente comando para descargar las imágenes de los vínculos web
  + python download\_images.py --urls urls.txt --output negatives
* Si realizó lo anterior apropiadamente debe verse una imagen como sigue, esta imagen informa el trayecto de descarga
* 
* Al finalizar la descarga, verifique que en su conjunto de datos negativos no existan imágenes de leopardo, debido a que estas son las muestras que no queremos detectar.

Paso 2: Generar los archivos positives.txt y negatives.txt

* Abra el archivo savedirtofile.py
* Examinando el archivo observamos que este pequeño script de python toma las carpetas “positives” y “negatives” e imprime el contenido en un archivo de texto del mismo nomre

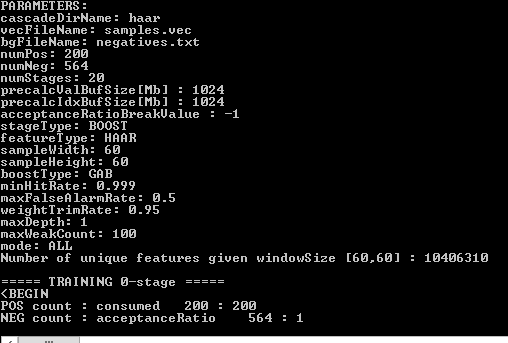


* Cierre el archivo y ejecútelo como sigue dentro de la carpeta del proyecto
  + python savedirtofile.py
* Examine por ejemplo el contenido de la carpeta de negativos, debe tener como resultado lo que sigue a continuación

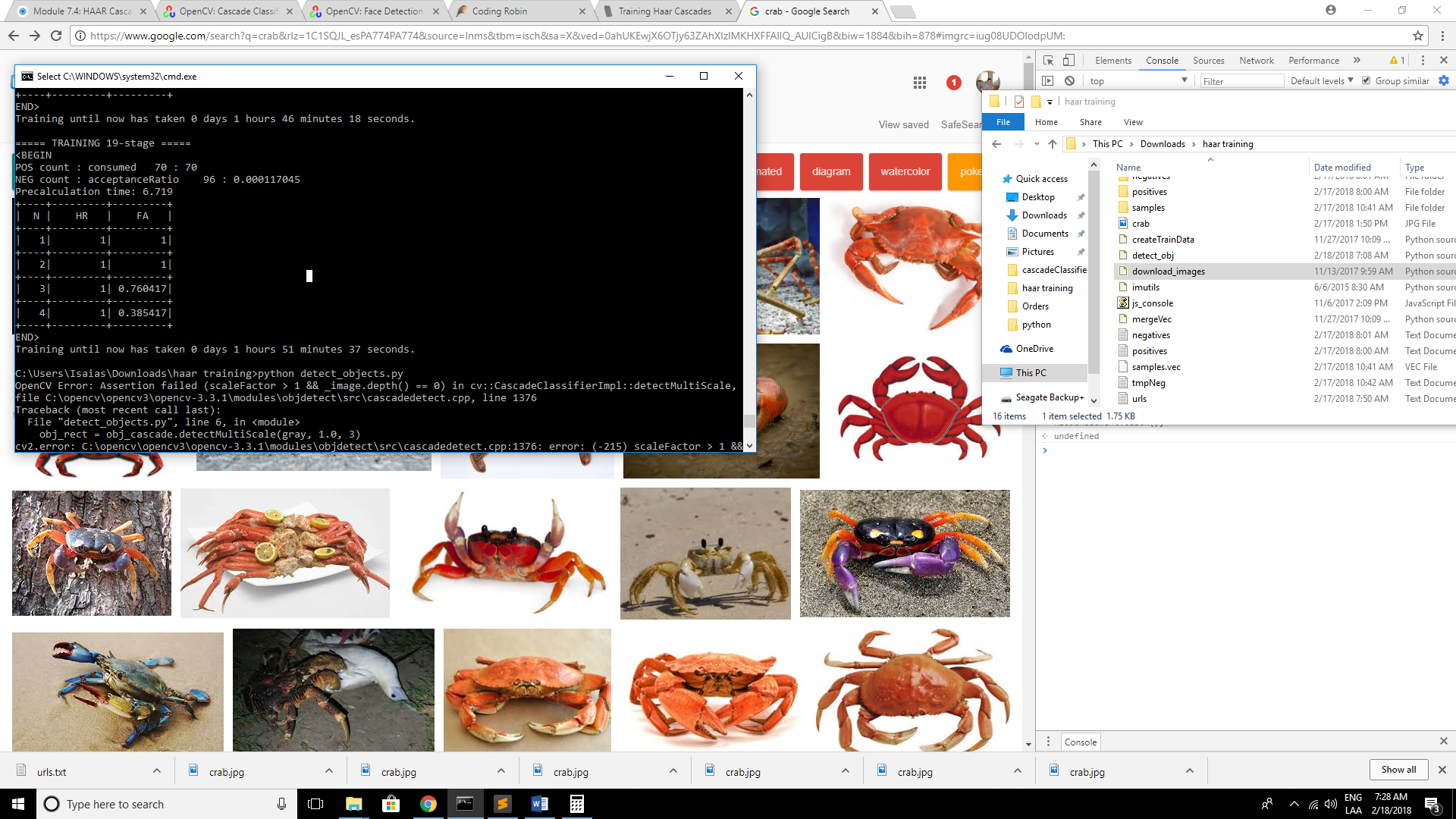


Paso 3: Creando los archivos samples.vec y vectors.vec

* Cuente cuantas muestras positivas hay en su carpeta
* Antes de ejecutar el siguiente comando cambie el número que viene después de samples a el número total de muestras positivas, luego ejecutar:
  + python createTrainData.py positives.txt negatives.txt samples 200 "opencv\_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1 -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 60 -h 60"
* Genere el archivo vectors.vec corriendo el siguiente comando
  + python mergeVec.py -v samples/ -o samples.vec
* Ahora corra el siguiente comando dentro de la carpeta de proyecto para entrenar el sistema. Note que debe matchar los positivos y los negativos con las cantidades reales (numPos y numNeg) para que funcione. Adicionalmente –w y –h deben ser en magnitud iguales de tamaño que en el paso anterior
  + opencv\_traincascade -data haar -vec samples.vec -bg negatives.txt -numStages 20 -minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos 200 -numNeg 564 -w 60 -h 60 -mode ALL -precalcValBufSize 1024 -precalcIdxBufSize 1024
* Si todo funciona correctamente debe empezar el entrenamiento y debe observarse la pantalla que sigue



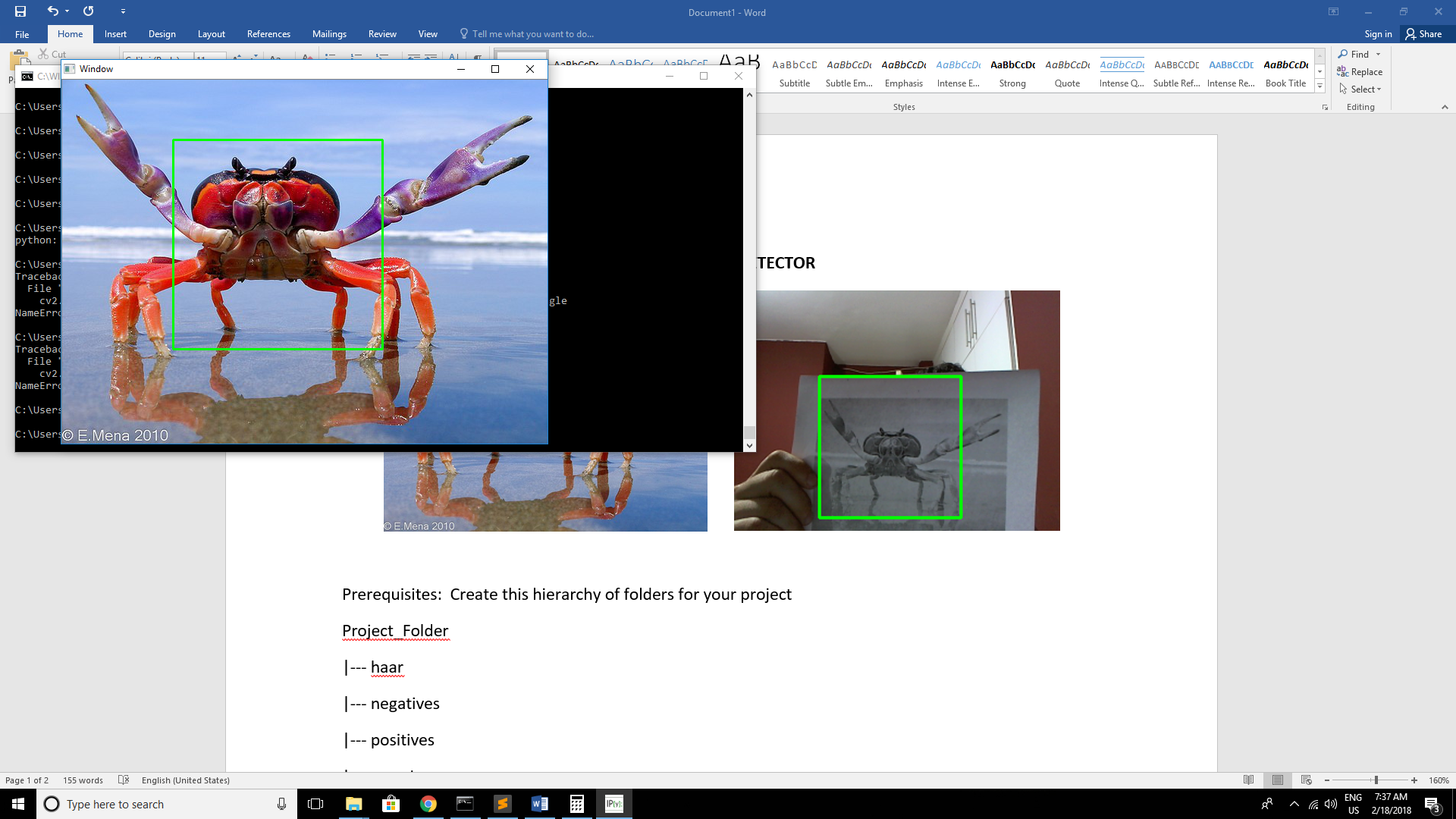
* Dependiendo de la data de entrenamiento, capacidad del CPU o si se está entrenando en un GPU, esto puede tardar de varios segundos, minutos, horas o días por entrenar
* Un buen modelo con buena capacidad de dataset debe tomar algunas horas, este ejemplo demoró alrededor de 2 horas. Debajo muestro los resultados del entrenamiento para propósitos de demostración
* Cuando el entrenamiento ha de terminar debe verse algo parecido a esto



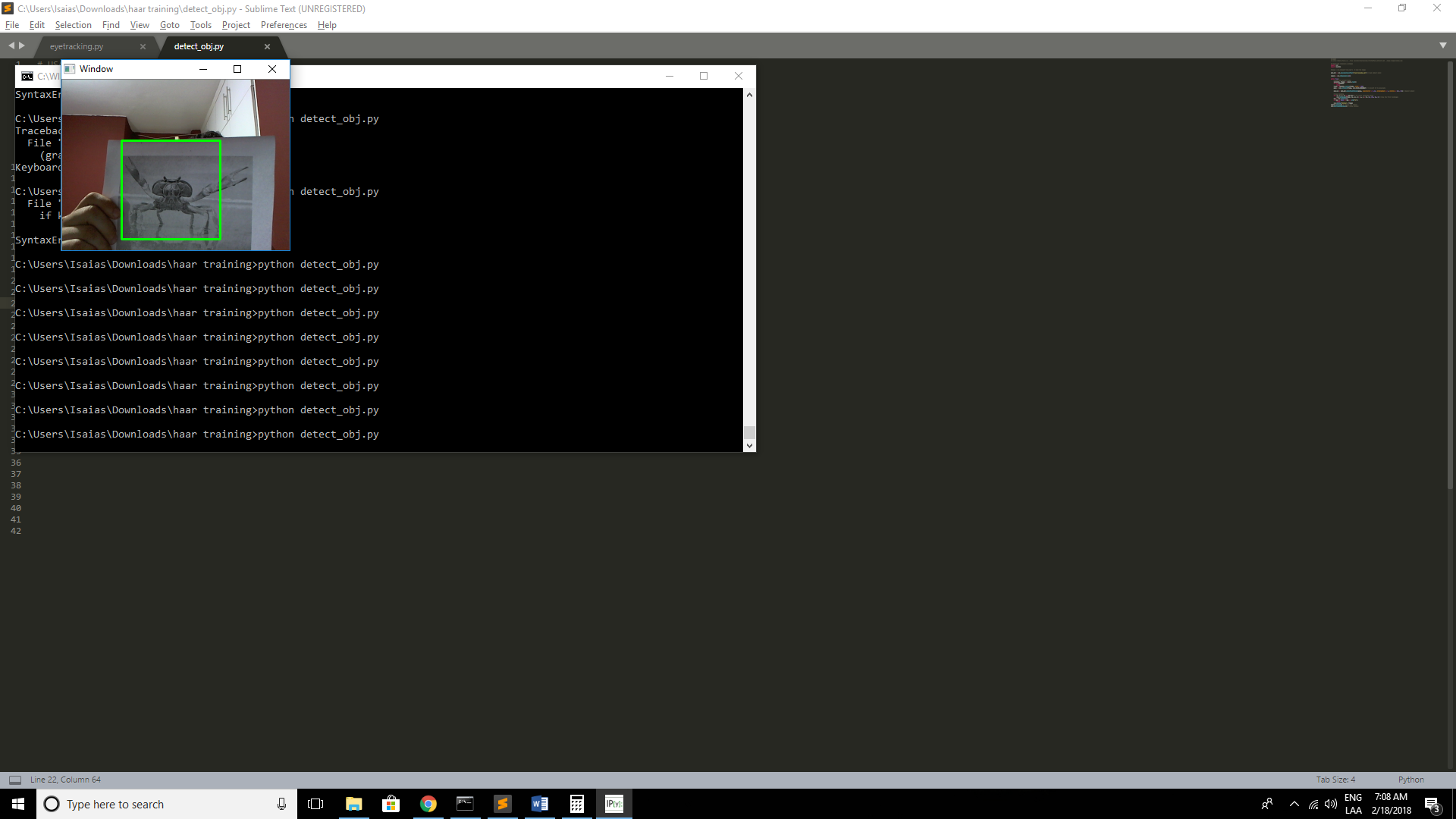
* Para la figura superior, esta es la salida a un entrenamiento de 4 estados

Paso 4: Corriendo el clasificador.

* El modelo final está en el folder de haar y es llamado cascades.xml
* Para prueba de una imagen solo corra el código de la siguiente manera
  + python detect\_obj\_image.py –-image cougar.jpg
  + --image <nombre\_de\_la\_imagen> es el argumento de entrada de la función
* Paara prueba de un video solo llame al siguiente script
  + Python detect\_obj\_video.py
  + Intentará detectar el objeto en tiempo real
* Importante es saber que el objeto no debe pertenecer al training set, es decir, no debe ser igual a alguno de las imágenes positivas utilizadas para entrenar el clasificador.
* En mi caso, aún no ha terminado de correr el entrenamiento y debido al tiempo pondré resultados de un detector de cangrejos que se realizó anteriormente trabajando imágenes



* Y también el mismo resultado, pero trabajando video



Referencias

<http://www.vision.caltech.edu/Image_Datasets/Caltech101/>

<https://www.pyimagesearch.com/2017/12/04/how-to-create-a-deep-learning-dataset-using-google-images/>

<https://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html>

<http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html>

<https://memememememememe.me/post/training-haar-cascades/>

<https://www.learnopencv.com/training-better-haar-lbp-cascade-eye-detector-opencv/>

Calificaciones

50% - Hacer lo mismo descrito en el documento

50% - Entrenar a otro clasificador utilizando la tabla que se da a continuación, favor referirse a su número para realizar el training

* Para el test debe usar una fotografía no proveniente del training set y apuntar a la cámara
* Debe detectarse por video

|  |  |
| --- | --- |
| ID | Objeto de Entrenamiento |
| 1 | Camaras |
| 2 | Cocodrilos |
| 3 | Dalmatas |
| 4 | Elefantes |
| 5 | Camellos |
| 6 | Canguros |
| 7 | Motocicletas |
| 8 | Carros |
| 9 | Palomas |
| 10 | Rinocerontes |
| 11 | Silla de Ruedas |
| 12 | Revolver |
| 13 | Pizza |
| 19 | Escorpiones |