MÓDULO 2 SESIoN 1

PROCESADO DE IMAGEN Y VISIÓN POR COMPUTADOR



8 de noviembre de 2023

Ricardo Martínez Guadalajara

# Introducción

En esta sesión se va a trabajar con detección de caras en imágenes

# Objetivo

Desarrollar los apartados del guion de prácticas como programas de Python

# Contenido

## Apartado 1

Se plantea las siguientes preguntas:

**¿Qué función concreta es la encargada de realizar la detección?**

La función que realiza la detección es cascade.detectMultiScale

**¿Qué parámetros recibe?**

Esta función recibe varios parámetros:

* img\_gray: la imagen en escala de grises.
* scaleFactor: el factor por el cual la ventana de búsqueda de la imagen se reduce en cada escala de la imagen.
* minNeighbors: el número mínimo de rectángulos vecinos que deben existir para retenerlo.
* minSize: el tamaño mínimo del objeto a detectar.
* flags: método de detección.

**¿Qué modelo de detector pre-entrenado se está utilizando?**

El modelo de detector pre-entrenado que se está utilizando es haarcascade\_frontalface\_alt.xml. Este es un clasificador de cascada Haar que ha sido entrenado para detectar caras frontales en imágenes.

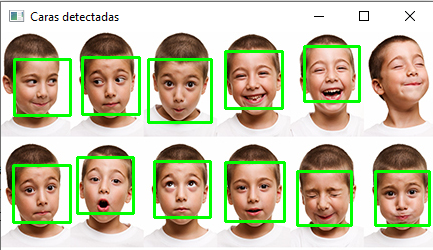
**¿Por qué cree que se realiza una ecualización de la imagen y una conversión de la misma escala de gris?**

La ecualización de la imagen se realiza para normalizar la iluminación en la imagen. Esto puede hacer que el detector de caras funcione mejor, ya que puede ser menos sensible a las variaciones de iluminación. La conversión de la imagen a escala de grises se realiza porque el detector de caras Haar funciona en imágenes en escala de grises, no en imágenes en color.

En este mismo punto se propone probar con distintos valores e ndistintas imágenes y ver los efectos que se producen sobre precisión y velocidad. Las conclusiones obtenidas, son las siguientes:

Tras probar el detector con las imágenes de prueba proporcionadas el resultado es el siguiente

Ejecución en: 1.123236894607544 segundos



Ejecución en: 2.8615427017211914 segundos



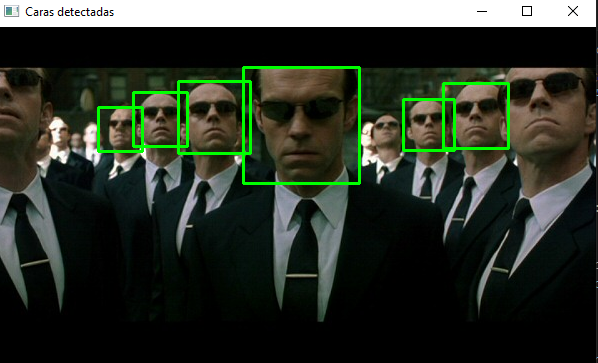
Ejecución en: 1.0867326259613037 segundos



Ejecución en: 0.6437904834747314 segundos



Ejecución en: 0.6754841804504395 segundos



En la imagen de ejemplo test4.jpg (imagen de la serie Perdidos) reduciendo en minSize se consigue detectar la cara de Jack, así que el problema en ese caso es el tamaño mínimo a detectar. En el resto de casos no se consigue detectar ninguna cara más.

Si se aumenta el minNeighbors aunque no se detectan más caras, aumenta el tiempo de ejecución a 1.2920806407928467 segundos

En cada imagen se ha puesto la salida del temporizador añadido.

Se aporta también el script detec\_demo-py modificado con el temporizador

## Apartado 2

Se proporciona el programa detectorVideo.py en el cual se implementa el flujo propuesto. Como problema sin resolver en dicho programa (ya que desde mi punto de vista no era objeto de esta práctica) es que en cada fotograma considera una nueva detección, provocando de esa manera que en cada frame el color sea distinto en caso de que haya más de una cara. Para paliar eso, habría que clasificar las caras, y reconocerlas en el siguiente frame, lo que no es objeto de esta práctica.

La salida de este script es el video output.avi, aportado también en la entrega.

Los cambios de iluminación bruscos influyen negativamente, y la baja iluminación empeora el rendimiento del software de detección

## Apartado 3

Se aporta el programa detectorVideoPixel.py en el cual se implementa la solución a la propuesta.

Se ha definido la función pixelado, a la cual se la pasa, la imagen original, la colección de rectángulos detectados con cara y por cada uno de ellos, se recorre vertical y horizontalmente en bloques de tamaño obtenido como parámetro y se calcula el valor promedio de dicho bloque. El cual a su vez sustituye al bloque recorrido, consiguiendo de esa manera un pixelado de la roi detectada

La salida, outputPixel.avi, se aporta como prueba de funcionamiento

## Apartado 4

En este apartado se aporta el programaDetetorVideoOjos.py. En el cual partiendo del código anterior, se detecta la cara. Las roi’s detectadas, se le pasan al siguiente detector de ojos el cual pinta el rectángulo en los ojos detectados con ese modelo.

Cabe destacar que funciona de manera poco robusta en condiciones de baja iluminación y encaras con gafas. Aumentando la iluminación y quitándome las gafas se comporta de una manera mucho más efectiva

La salida, outputOjo.avi, se aporta como prueba de funcionamiento