Portada

Contenido

[Evaluación Diagnostica Inteligencia Artificial 3](#_Toc191019058)

[Semana 1 Instalación 5](#_Toc191019059)

[Segunda practica detector de color 9](#_Toc191019060)

[¿Qué es OpenCV? 11](#_Toc191019061)

[Operaciones básicas en el procesamiento de imágenes con OpenCV 11](#_Toc191019062)

[Semana 2 – Modelo HaarCascade 14](#_Toc191019063)

# Evaluación Diagnostica Inteligencia Artificial

Nombre: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fecha: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Instrucciones: Responde las siguientes preguntas de manera clara y concisa. Esta evaluación tiene como objetivo identificar tu conocimiento previo sobre Inteligencia Artificial (IA).

1. ¿Qué es la Inteligencia Artificial (IA)?

(Define el concepto en tus propias palabras).

2. Menciona dos ejemplos de aplicaciones de la Inteligencia Artificial en la vida cotidiana.

(Describe brevemente cómo se utiliza la IA en estos casos).

a) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. ¿Cuál es la diferencia entre Inteligencia Artificial, Machine Learning y Deep Learning?

(Explica brevemente cómo se relacionan estos conceptos).

4. ¿Qué éticas o desafíos consideras que plantea el uso de la Inteligencia Artificial?

(Menciona al menos dos aspectos éticos o desafíos).

a) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. ¿Qué crees que es necesario para desarrollar un sistema de Inteligencia Artificial?

(Menciona al menos dos elementos clave, como datos, algoritmos, hardware, etc.).

a) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Comentarios: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Semana 1 Instalación

Elementos de la instalación:

1. Visual estudio Code: Extensiones **Pylance, Python, Python Debugger y OpenCV-intellisense.**
2. **Instalar la ultima verison de Python 3.11.1**
3. **Instalar con pip desde la consola como administrador pip install opencv-python**

Primera practica

Cargar Imágenes

En OpenCV, la función cv.imread se utiliza para leer imágenes desde un archivo. El segundo parámetro de esta función (denominado flags) determina cómo se leerá la imagen. A continuación, se explica la diferencia entre los valores 1, 0 y -1:

**1. cv.imread('cocacola.png', 1)**

* **Descripción**: Lee la imagen a color en su modo predeterminado (3 canales: BGR).
* **Uso**: Este es el valor por defecto si no se especifica ningún flag.
* **Resultado**:
  + La imagen se carga con tres canales (azul, verde, rojo, en el orden BGR).
  + Se conserva toda la información de color de la imagen.
* **Propósito**: Útil cuando se necesita procesar o trabajar con imágenes a color.

python

CopiarEditar

import cv2 as cv

myImage = cv.imread('cocacola.png', 1)

print(myImage.shape) # Ejemplo: (altura, ancho, 3)

**2. cv.imread('cocacola.png', 0)**

* **Descripción**: Lee la imagen en escala de grises (1 canal).
* **Uso**: Convierte automáticamente la imagen a escala de grises al cargarla.
* **Resultado**:
  + La imagen resultante tiene un solo canal.
  + Cada píxel se representa con un valor de intensidad de gris (0 a 255).
* **Propósito**: Ideal para algoritmos que no requieren información de color, como detección de bordes o procesamiento más rápido.

python

CopiarEditar

import cv2 as cv

myImage = cv.imread('cocacola.png', 0)

print(myImage.shape) # Ejemplo: (altura, ancho)

**3. cv.imread('cocacola.png', -1)**

* **Descripción**: Lee la imagen tal cual está, incluyendo el canal alfa si la imagen tiene transparencia.
* **Uso**: Se conserva toda la información de la imagen, incluidos los posibles canales adicionales como el canal alfa (RGBA en lugar de solo BGR).
* **Resultado**:
  + Si la imagen tiene un canal alfa (transparencia), este se carga junto con los otros tres canales (resultando en 4 canales).
  + Si no tiene canal alfa, se carga igual que con el flag 1 (3 canales).
* **Propósito**: Útil para trabajar con imágenes que necesitan transparencia o información completa.

python

CopiarEditar

import cv2 as cv

myImage = cv.imread('cocacola.png', -1)

print(myImage.shape) # Ejemplo: (altura, ancho, 4) si hay transparencia

Código

#Libreria de opencv

import cv2 as cv;

#Libreria para cargar imagenes

myImage = cv.imread('cocacola.png', 1)

#Libreria para mostrar imagenes

cv.imshow('CocaCola', myImage)

#Pausar una pantalla

cv.waitKey(0)

#Terminar la ejecucion de mi programa

cv.destroyAllWindows()

Texto

Descripción generada automáticamente

Código Completo

#Libreria de opencv

import cv2 as cv;

#Libreria para cargar imagenes

myImage = cv.imread('cocacola.png', 1)

#Cambiar tamaño de la imagen

myImage = cv.resize(myImage, (200, 200))

#Libreria para mostrar imagenes

cv.imshow('CocaCola Imagen Original', myImage)

#Convertir la imagen a RGB

myImage\_RGB = cv.cvtColor(myImage, cv.COLOR\_BGR2RGB)

myImage\_RGB = cv.resize(myImage\_RGB, (200, 200))

#Mostrar la imagen RGB

cv.imshow('CocaCola Imagen RGB', myImage\_RGB

#convertir imagen a escala de grises

myImage\_GRAY = cv.cvtColor(myImage, cv.COLOR\_BGR2GRAY)

myImage\_GRAY = cv.resize(myImage\_GRAY, (200, 200))

#MOSTRAR IMAGEN GRAY

cv.imshow('CocaCola Imagen GRAY', myImage\_GRAY)

#Convertir imagen en HSV

myImage\_HSV = cv.cvtColor(myImage, cv.COLOR\_BGR2HSV)

myImage\_HSV

cv.imshow('CocaCola Imagen HSV', myImage\_HSV)

#Pausar una pantalla

cv.waitKey(0)

#Terminar la ejecucin de mi programa

cv.destroyAllWindows()

## Segunda practica detector de color

Captura de pantalla de computadora

Descripción generada automáticamente

Código:

import cv2 as cv;

import numpy as np;

#Libreria para cargar imagenes

myImage = cv.imread('marisco.jpg', 1)

#Para este programa de decteción de color rojo

#cambiamos el formato de la imagen a hsv

myImage\_HSV = cv.cvtColor(myImage, cv.COLOR\_BGR2HSV)

#Definimos el rango de color rojo con np

lower\_red1 = np.array([0, 150, 100])

upper\_red1 = np.array([5, 255, 255])

lower\_red2 = np.array([175, 150, 100])

upper\_red2 = np.array([180, 255, 255])

#Creamos una mascara para el color rojo

mask1 = cv.inRange(myImage\_HSV, lower\_red1, upper\_red1)

mask2 = cv.inRange(myImage\_HSV, lower\_red2, upper\_red2)

#Unimos las dos mascaras esto es para la interface

mask = cv.add(mask1, mask2)

#Encontramos los contornos de la mascara

contours, \_ = cv.findContours(mask, cv.RETR\_TREE, cv.CHAIN\_APPROX\_SIMPLE)

#dibujamos los contornos en la imagen original

for contour in contours:

    cv.drawContours(myImage, contour, -1, (0, 0, 0), 3)

#mostrar resultado

cv.imshow('CocaCola Imagen Original', myImage)

cv.imshow('Detector de Color Rojo', mask)

cv.waitKey(0)

cv.destroyAllWindows()

## ¿Qué es OpenCV?

OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto especializada en visión por computadora y procesamiento de imágenes. Fue desarrollado originalmente por Intel y ahora es mantenido por una comunidad de desarrolladores en código abierto.

**Características principales de OpenCV**

* **Procesamiento de imágenes y vídeo:**pe
* **Reconocimiento facial y de objetos:**incluye alg
* **Visión artificial y realidad aumentada:** permite la
* **Soporte para múltiples lenguajes y plataformas:**funciona con C++,

**Aplicaciones de OpenCV**

* Detección
* Sistemas
* Control de Actuadores
* Vehículos autónomos (detección de objetos y señales)
* Aplicaciones de realidad aumentada

## Operaciones básicas en el procesamiento de imágenes con OpenCV

**1️ Cargar y mostrar una imagen**

import cv2

# Cargar imagen desde un archivo

imagen = cv2.imread("imagen.jpg")

# Mostrar la imagen en una ventana

cv2.imshow("Imagen Original", imagen)

cv2.waitKey(0) # Espera una tecla para cerrar

cv2.destroyAllWindows()

**2️ Convertir una escala de grises**

gris = cv2.cvtColor(imagen, cv2.COLOR\_BGR2GRAY)

cv2.imshow("Imagen en escala de grises", gris)

cv2.waitKey(0)

cv2.destroyAllWindows()

**3️ Redimensionar una imagen**

redimensionada = cv2.resize(imagen, (400, 300)) # Ancho x Alto

cv2.imshow("Imagen Redimensionada", redimensionada)

cv2.waitKey(0)

cv2.destroyAllWindows()

**4️ Dibujar formas en una imagen**

import numpy as np

# Crear una imagen en negro

imagen\_negra = np.zeros((500, 500, 3), dtype="uint8")

# Dibujar una línea (imagen, punto inicial, punto final, color, grosor)

cv2.line(imagen\_negra, (50, 50), (450, 50), (0, 255, 0), 3)

# Dibujar un rectángulo

cv2.rectangle(imagen\_negra, (100, 100), (400, 300), (255, 0, 0), 2)

# Dibujar un círculo

cv2.circle(imagen\_negra, (250, 250), 50, (0, 0, 255), -1)

# Mostrar la imagen con las figuras

cv2.imshow("Formas en OpenCV", imagen\_negra)

cv2.waitKey(0)

cv2.destroyAllWindows()

**5️ Detección de bordes con Canny**

bordes = cv2.Canny(gris, 100, 200) # Límite inferior y superior

cv2.imshow("Detección de Bordes", bordes)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Semana 2 – Modelo HaarCascade

Activar Cámara

import cv2 as cv

import numpy as np

#1 Decalar una variable = webcam

camara = cv.VideoCapture(1)

#Crear un ciclo while para mantener la captura de la camara

while True:

    ret, frame = camara.read()

    if not ret:

        print("La esta inactiva")

        break

    #Mostrar en una ventana los frame capturados

    cv.imshow('WebCam Live', frame)

    if cv.waitKey(1) & 0xFF == ord('q'):

        break

#Nota despues de ocupar la camara hay que vaciar el buffer

camara.release()

cv.destroyAllWindows()

Importar Modelo

import cv2 as cv

import numpy as np

face\_cascade = cv.CascadeClassifier(cv.data.haarcascades + 'haarcascade\_frontalface\_alt.xml')

cam = cv.VideoCapture(1)

while True:

    ret, frame = cam.read()

    # Convertir el frame a escala de grises (mejora la precisión de detección)

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

    # Detectar rostros en el frame

    faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    print(faces)

    #Mostar Video

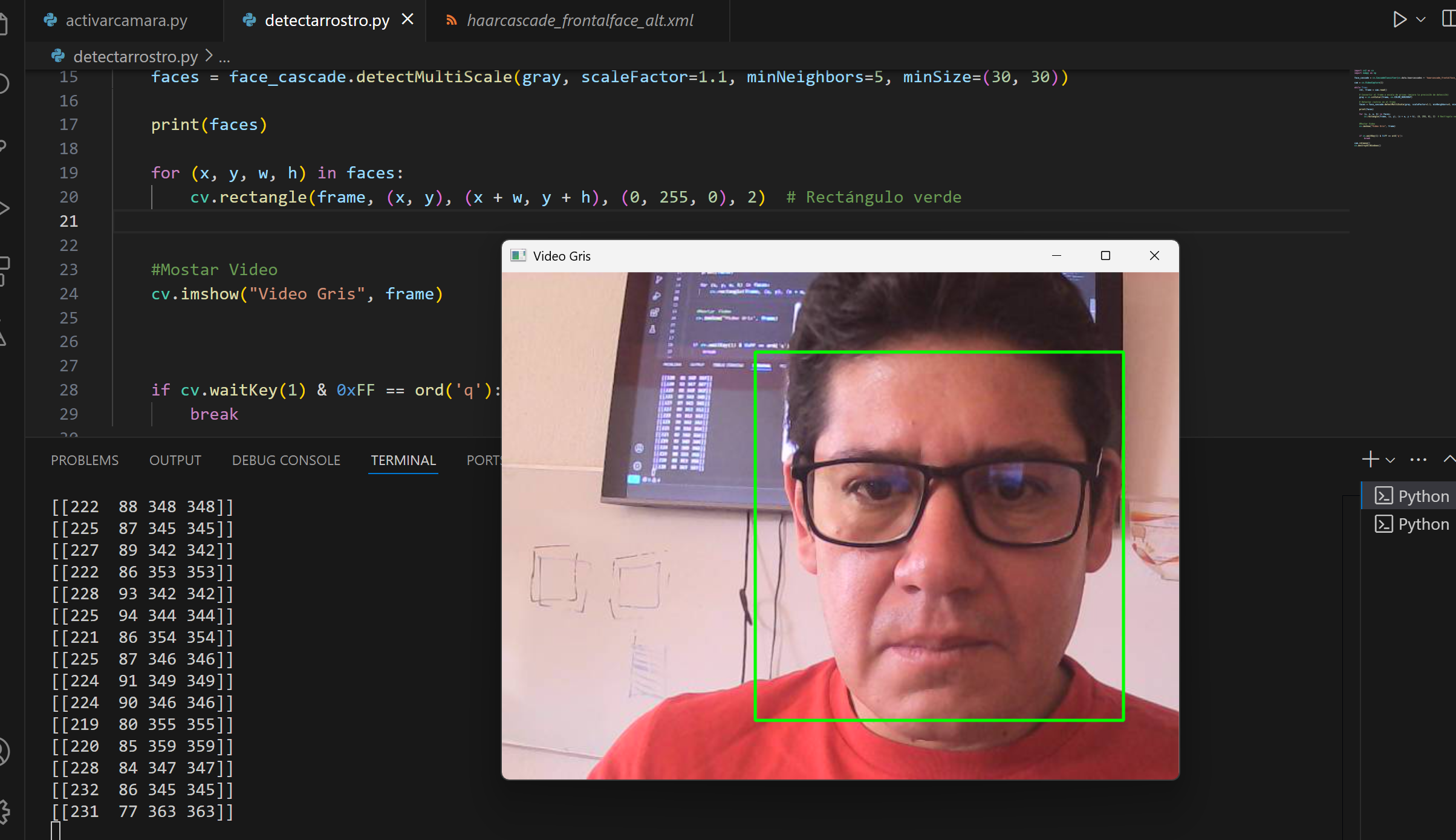
    cv.imshow("Video Gris", gray)

    if cv.waitKey(1) & 0xFF == ord('q'):

        break

cam.release()

cv.destroyAllWindows()



import cv2 as cv

import numpy as np

face\_cascade = cv.CascadeClassifier(cv.data.haarcascades + 'haarcascade\_frontalface\_alt.xml')

cam = cv.VideoCapture(1)

while True:

    ret, frame = cam.read()

    # Convertir el frame a escala de grises (mejora la precisión de detección)

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

    # Detectar rostros en el frame

    faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    print(faces)

    for (x, y, w, h) in faces:

        cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)  # Rectángulo verde

    #Mostar Video

    cv.imshow("Video Gris", frame)

    if cv.waitKey(1) & 0xFF == ord('q'):

        break

cam.release()

cv.destroyAllWindows()

import cv2 as cv

import numpy as np

face\_cascade = cv.CascadeClassifier(cv.data.haarcascades + 'haarcascade\_frontalface\_alt.xml')

cam = cv.VideoCapture(0)

while True:

    ret, frame = cam.read()

    # Convertir el frame a escala de grises (mejora la precisión de detección)

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

    # Detectar rostros en el frame

    faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    print(faces)

    for (x, y, w, h) in faces:

        #cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)  # Rectángulo verde

         # Centro del rostro

        center\_x = x + w // 2

        center\_y = y + h // 2

        # Definir el tamaño de las líneas de la cruz

        line\_length = w-x

        # Dibujar las dos líneas que forman una cruz

        cv.line(frame, (center\_x - line\_length, center\_y), (center\_x + line\_length, center\_y), (0, 255, 0), 2)  # Línea horizontal

        cv.line(frame, (center\_x, center\_y - line\_length), (center\_x, center\_y + line\_length), (0, 255, 0), 2)  # Línea vertical

    #Mostar Video

    cv.imshow("Video Gris", frame)

    if cv.waitKey(1) & 0xFF == ord('q'):

        break

cam.release()

cv.destroyAllWindows()

Semana 3 Media Pipe

¿Qué es mediapipe?

Google Open Source ha desarrollado más de 2000 proyectos Open Source que podemos explorar en su página oficial (https://opensource.google/). Precisamente uno de esos proyectos es el que veremos como instalar hoy, MediaPipe.

Este es un framework multimodal y multiplataforma que aplica machine learning, permitiéndonos desarrollar proyectos o aplicaciones en dispositivos móviles, de escritorio o en la web.

Pero, ¿qué tiene de bueno este framework?, pues posee modelos de machine learning, para la detección de rostro, trancking de manos, segmentación de cabello, detección y trackeo de objetos e incluso detección y tracking de objetos 3D. Y estos son solo algunos.

Además, uno pensaría que al aplicar estos modelos, el resultado de las detecciones sería lento. Pero no, de hecho es de lo más sorprendente, ya que nos permite correr programas desde la CPU, y con un muy buen desempeño.

MediaPipe está disponible para Android, iOS, c++, Python, Javascript y Coral. Pero hay que tomar en cuenta que no todas las soluciones están disponibles para todos ellos, de hecho a continuación tenemos la tabla de información que nos provee la web oficial hasta el momento.

Figura 1: Soluciones de MediaPipe a las que podemos acceder según el lenguaje de programación. (Fuente)

Para Python, que es el lenguaje de programación con el que estaremos usando este framework en los próximos tutoriales, tenemos disponible:

Face Detection

Figura 2: Ejemplo de la detección de rostro con MediaPipe.

Face Mesh

Figura 3: Ejemplo de face mesh con MediaPipe.

Hands

Figura 4: Ejemplo de detección de manos y hand landmarks con MediaPipe.

Pose

Holistic

Figura 5: Ejemplo de Pose y Holistic con MediaPipe.

Objectron

Figura 6: Ejemplo de Objectron con MediaPipe.

¿Cómo instalar MediaPipe?

Para instalar MediaPipe en nuestro computador, necesitamos tener previamente instalado Python 3.

En mi caso voy a instalarlo en un entorno virtual. Si no has manejado antes entornos virtuales y quieres probarlos, échale un ojo a este post: ENTORNOS VIRTUALES DE PYTHON | Usando virtualenv

Pero si en realidad no quieres instalarlo usando un entorno virtual pues tranquilo, igualmente puedes seguir los mismos pasos.

Vamos al símbolo del sistema, en mi caso he creado un entorno virtual llamado mp. Y como puedes ver, ya lo tengo activado, entonces voy a digitar: pip install mediapipe.

Figura 7: Instalación de MediaPipe.

Esperamos a que se realice la instalación y listo. 😀

Podemos usar pip freeze para ver que hayamos instalado corectamente, y como puedes ver, al instalar mediapipe, también se han instalado otros módulos, tales como OpenCV. Esto lo podemos visualizar a continuación:

Figura 8: Uso de pip freeze, para asegurarnos de que mediapipe se haya instalado.

Como puedes ver, la instalación de este framework es bastante sencilla usando pip. Y esto ha sido todo por el post de hoy, espero que te haya gustado y nos vemos en el siguiente.

Semana 4 OCR Reconocimiento de Caracteres **Pytesseract**

Ya se han tratado varios temas en este blog sobre visión por computador empleando OpenCV, por lo que me he sentido impulsada (basándome en el contenido previo de las publicaciones/videos) a crear este post relacionado con la detección de placas de autos y a identificar los caracteres que aparecen en ellas.

Debo decir que lo que quisiera mostrarte hoy es una aplicación que engloba detección de bordes, detección de figuras geométricas, entre otros, que no necesita por ahora de machine learning (que es algo que espero desarrollar en un futuro). Y claro, como no tiene un proceso de aprendizaje puede que esta aplicación únicamente se pueda aplicar bajo cierta iluminación o ángulo y distancia de la cámara al auto, entre otros. Sin embargo, si se te permitiera controlar estas u otras condiciones, puede que tengas muy buenos resultados.

En gran medida en este programa se usará contenido que ya hemos visto, por lo tanto cuando sea necesario lo citaré para que puedas profundizar algún tema (si es que fuera necesario), de igual manera lo haré si se presenta algún tema nuevo en el blog. ¡Así que abre tu editor de código favorito y empecemos a realizar la detección de placas vehiculares junto con sus caracteres!.

Imagen de la pantalla de un video juego de un carro

El contenido generado por IA puede ser incorrecto.

**CONTENIDO**

* ¿Qué es el Reconocimiento Automático de Matrículas (ANPR)?
* ¿Qué es el Reconocimiento Óptico de Caracteres (OCR)?
  + ¿Cómo usar OCR en Python?

1. Empezando la aplicación
   1. Leyendo la imagen y transformarla a binaria
   2. Encontrando contornos
      1. Determinando el área de los contornos
      2. Encontrar el rectángulo de la matrícula
      3. Discriminando contornos y obteniendo el aspect ratio
   3. Una vez obtenida la placa… ¡Es hora de aplicar OCR con pytesseract!
   4. Finalmente, ¡visualización!

* Programa completo
* Resultados

**¿Qué es el Reconocimiento Automático de Matrículas (ANPR)?**

Imagen de la pantalla de un video juego de un carro

El contenido generado por IA puede ser incorrecto.

**Figura 1:** Reconocimiento de matrículas ([Fuente](https://trends.directindustry.es/lontrend-corporation/project-197078-179363.html)).

[ANPR](https://es.wikipedia.org/wiki/Reconocimiento_autom%C3%A1tico_de_matr%C3%ADculas) pos sus siglas en inglés *Automatic number plate recognition,*también llamado *Licence Plate Recognition (LPR)*, es un método de vigilancia en masa que permite detectar dentro de una imagen a la placa de un auto y posteriormente utilizando Reconocimiento Óptico de Caracteres (OCR) determinar cada uno de los alfanuméricos que componen a dicha matrícula, de tal modo que esta información pueda ser usada con algún fin.

Si buscas información sobre reconocimiento de matrículas en internet, podrás encontrar varios videos en donde se puede apreciar el registro de entrada y salida a ciertos lugares como estacionamientos, unidades educativas, residenciales o el uso de esta tecnológica en peajes, entre otros. Incluso como a mí me pasó al buscar información para desarrollar este tutorial, te podrías topar con varias empresas como [esta](http://www.arhungary.hu/101125/doc/arh_anpr12_flyer.pdf), que ofrecen servicios y/o productos que permiten reconocer placas con gran precisión y velocidad, sin importar el país o la cámara que se use.

Por ello si estás desarrollando alguna aplicación sobre este tema, te recomiendo que analices muy bien las características que encuentres en cada uno de los servicios que ofertan distintas empresas, o estudios desarrollados sobre este reconocimiento. De tal modo que tomes dichas características como referencia, y a su vez, que estas puedan guiarte a agregar robustez al programa que estés desarrollando.

**¿Qué es el Reconocimiento Óptico de Caracteres (OCR)?**

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

**Figura 2:** Reconocimiento óptico de caracteres ([Fuente](https://medium.com/@fatih_yildizli/tesseract-ocr-implementation-in-net-core-spring-boot-6f876a5d4ae5)).

[*Optical Character Recognition (OCR)*](https://es.wikipedia.org/wiki/Reconocimiento_%C3%B3ptico_de_caracteres)*,* es un proceso automático mediante el cual se extrae texto de una imagen y se transforma a un formato digital, almacenando caracteres, números y símbolos en forma de datos, a los cuales podremos acceder y manipular.

Si pensamos en alguna aplicación, podríamos por ejemplo usar OCR para digitalizar el contenido de un documento, de este modo nos ahorraríamos el pasar manualmente todo ese contenido. Podríamos también extraer los caracteres y dígitos de placas de autos, certificados, registros, entre otros. Cabe destacar que la imagen de entrada puede contener no solo texto impreso sino también escrito a mano.

**¿Cómo usar OCR en python?**

El motor de reconocimiento óptico de caracteres que emplearemos es [tesseract](https://es.wikipedia.org/wiki/Tesseract_OCR" \t "_blank), cuyo desarrollo ha sido financiado por Google desde el 2006, es de código abierto y considerado como uno de los más precisos. Si deseas más información puedes acceder a su [manual de usuario](https://github.com/tesseract-ocr/tessdoc), en donde encontrarás su documentación, los [idiomas con los que trabaja](https://github.com/tesseract-ocr/tessdoc/blob/master/Data-Files-in-different-versions.md) según la versión que instales o como [mejorar la extracción de caracteres](https://github.com/tesseract-ocr/tessdoc/blob/master/ImproveQuality.md#improving-the-quality-of-the-output), entre otros.

Para acceder a tesseract usando Python necesitamos del paquete *[pytesseract](https://github.com/madmaze/pytesseract" \t "_blank)*. Si aún no lo has instalado te recomiendo seguir los pasos de este [post de Pyimagesearch](https://www.pyimagesearch.com/2017/07/03/installing-tesseract-for-ocr/). También puedes encontrar información sobre su instalación en su [repo en github](https://github.com/madmaze/pytesseract#installation).

**1. Empezando la aplicación**

La imagen que escogí para poder realizar la detección de placas y sus caracteres contiene un auto el cual está visto de frente, donde se puede apreciar claramente su placa. Esta no es una imagen tomada de la realidad, sino que es una imagen tratada de un spot publicitario. Sin embargo al final haré pruebas del código con imágenes descargadas de internet que si corresponden a un entorno real.

La imagen con la que vamos a trabajar es la siguiente:

Un coche deportivo de color rojo

El contenido generado por IA puede ser incorrecto.

**Figura** 3: Imagen de entrada.

**1.1 Leyendo la imagen y transformarla a binaria**

import cv2

import pytesseract

placa = []

image = cv2.imread('auto001.jpg')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

gray = cv2.blur(gray,(3,3))

canny = cv2.Canny(gray,150,200)

canny = cv2.dilate(canny,None,iterations=1)

**Línea 1:** Importamos openCV.

**NOTA:** Si usas Windows, es posible que necesites añadir

pytesseract.pytesseract.tesseract\_cmd = r'C:\Program Files\Tesseract-OCR\tesseract'

 en la **línea 3**.

**Línea 2:** Importamos

pytesseract

 que será el paquete que nos ayudará a extraer los caracteres de la placa.

**Línea 4:**Declaramos

placa

 un array vacío que nos permitirá almacenar la imagen a la matrícula encontrada. Esto lo veremos luego.

**Línea 6:** Leemos la imagen de entrada que corresponde a la figura 3.

**Linea 7 y 8:** Transformamos la imagen de BGR a escala de grises, mientras que en la siguiente línea aplicamos

cv2.blur

 para disminuir el ruido que pueda presentar la imagen, puedes asignar otros valores dependiendo de la imagen sobre la que estés trabajando (¿más información sobre cv2.blur?, aquí te dejo[la documentación de OpenCV](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html)).

**Línea 9 y 10:** Aplicamos

cv2.Canny

 a la imagen a escala de grises para poder encontrar los bordes que presente la imagen. He establecido

150

 y

200

 como primer y segundo umbral, luego de algunas pruebas ([aquí](https://omes-va.com/contando-objetos-aplicando-deteccion-de-bordes-con-canny-en-python-opencv/) tienes otro tutorial en donde apliqué canny). En la **línea 10** aplicamos dilatación ([doc de OpenCV](https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html" \t "_blank)) a la imagen, esto permitirá engrosar las líneas blancas que obtuvimos al aplicar la detección de bordes, lo que nos servirá para cerrar el contorno correspondiente a la placa en caso de que haya estado abierto.

Hasta aquí debemos asegurarnos que exista área en blanco que esté rodeando a la placa, con ello ya podremos trabajar en los siguientes pasos.

Si visualizamos la imagen, tendríamos:

Texto

El contenido generado por IA puede ser incorrecto.

**Figura 4:** Imagen obtenida de la variable canny.

**1.2 Encontrarndo contornos**

En un principio vamos a aplicar la función cv2.findContours y visualizaremos los contornos, luego procederemos  a determinar la matrícula.

#\_,cnts,\_ = cv2.findContours(canny,cv2.RETR\_LIST,cv2.CHAIN\_APPROX\_SIMPLE)#OpenCV 3

cnts,\_ = cv2.findContours(canny,cv2.RETR\_LIST,cv2.CHAIN\_APPROX\_SIMPLE)#OpenCV 4

cv2.drawContours(image,cnts,-1,(0,255,0),2)

**Línea 12 y 13:** Procedemos a encontrar los contornos de

canny

 que es la imagen binaria que habíamos conseguido antes. Puedes emplear la **línea 12 o 13** dependiendo de la versión de OpenCV que tengas instalado. Esta vez he usado

cv2.RETR\_LIST

 para obtener todos los contornos de la imagen, si tienes dudas sobre este argumento y en sí sobre

cv2.findContours

 puedes visitar este post: [?‍? CONTORNOS y como DIBUJARLOS en OpenCV y Python](https://omes-va.com/contornos/).

**Línea 14:** Para dibujar todos los contornos encontrados vamos a ayudarnos dela función

cv2.drawContours

. Si visualizamos obtendremos lo siguiente:

Imagen de la pantalla de un video juego de un carro

El contenido generado por IA puede ser incorrecto.

**Figura 5:** Visualización de los contornos encontrados.

Como podemos apreciar en la figura 5, se han dibujado un motón de contornos, y eso puede preocuparnos porque el único que realmente necesitábamos era el que rodea la placa.

*¿Qué podemos hacer?*, no te preocupes, en la figura 5 podemos apreciar que sí se ha encerrado el contorno correspondiente a la placa, lo que necesitamos ahora es desechar aquellos contornos no deseados Para ello podemos diferenciarlos basándonos en su área, intentando de encontrar la forma rectangular dela matrícula, entre otros. Veamos.

**1.2.1 Determinando el área de los contornos**

**for** c **in** cnts:

area = cv2.contourArea(c)

**Línea 16:**Como necesitamos analizar  cada contorno almacenado en

cnts

, nos ayudamos de un for.

**Línea 17:**Para determinar el área de un contorno usaremos la función

cv2.contourArea

. Te recomiendo imprimir en cada momento la variable

area

, para que puedas darte cuenta en que rango está la correspondiente a la matrícula.

**NOTA:**Puedes determinar un área mínima y/o máxima para que el contorno sea considerado como una placa. La cantidad que especifiques para ello corresponderá también a la distancia a la que esté ubicada la matrícula de la cámara, dicho de otro modo, mientas más lejos esté la cámara del auto, más pequeña se va a apreciar la matrícula, mientras que si se acerca, la matrícula se visualizará más grande.

Incluso podrías tomar esos datos y establecer a qué distancia de la cámara-auto puede funcionar tu aplicación. Así eliminaras también aquellos contornos pequeños que no corresponden al área de interés.

**1.2.2 Encontrar el rectángulo de la matrícula**

¿Recuerdas el post en el que hablamos  de como detectar figuras geométricas simples?, Si tu respuesta es no o un «más  o menos», te dejo este post: [Detectando FIGURAS GEOMÉTRICAS (??⬛) con OpenCV – Python](https://omes-va.com/detectando-figuras-geometricas-con-opencv-python/). En este habíamos hablado de como detectar un rectángulo en una imagen y eso es precisamente lo que vamos a realizar en esta aplicación.

x,y,w,h = cv2.boundingRect(c)

epsilon = 0.09\*cv2.arcLength(c,**True**)

approx = cv2.approxPolyDP(c,epsilon,**True**)

**Línea 19:**Obtenemos los puntos

x

,

y

,

w

 y

h

 que rodean a cada contorno, esto nos ayudará para luego determinar el aspect ratio del contorno y determinar que se trate de un rectángulo.

**Línea 20 y 21:**Obtenemos

approx

, que es en donde se almacenará el número de vértices del contorno, para ello usamos la función

cv2.approxPolyDP

 que a su vez usa

epsilon

de la **línea 20**. Aquí el porcentaje que he usado es de 9%, pero tu podrías cambiarlo para experimentar. Si estás batallando un poco para entender lo que hacen estas líneas te recomiendo [este tutorial](https://omes-va.com/detectando-figuras-geometricas-con-opencv-python/).

**1.2.3 Discriminando contornos y obteniendo el aspect ratio**

**if** len(approx)==4 and area>9000:

print('area=',area)

#cv2.drawContours(image,[approx],0,(0,255,0),3)

**Línea 23:** Ahora vamos a usar un condicional. Si el número de vértices es 4 y el contorno tiene un área mayor a 9000 es potencialmente una placa.

**Línea 24:**Puedes imprimir el área de los contornos que estan siendo potencialmente considerados como una placa, para modificar la condición de la **línea 23** y adaptarlo a tu aplicación.

**Línea 25:**Si descomentas esta línea, podrás dibujar el contorno que está siendo considerado como placa bajo la condición de la**línea 23.**Obtendrías algo así:

Un coche deportivo de color rojo

El contenido generado por IA puede ser incorrecto.

**Figura 6:** Contorno dibujado luego de aplicar las condiciones de la línea 23.

Hasta ahora todo va de maravilla, pero podríamos añadir más criterios para detectar la placa, como por ejemplo el aspect ratio del contorno para determinar que sea un rectángulo, pero además… también podrías averiguar el alto y ancho de las placas vehiculares de tu país y trabajar también con su aspect ratio.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Figura 7:** Medidas de las placas de autos de Ecuador ([fuente](https://ecuadorec.com/placas-vehiculos-ecuador-tipos-letras-provincia/))  
.

En la figura 7 por ejemplo tenemos las medidas de las placas de Ecuador, entonces podríamos determinar su aspect ratio. ¿Recuerdas cómo obtener el aspect ratio de un contorno?.

Texto, Carta

El contenido generado por IA puede ser incorrecto.

Entonces el código sería:

aspect\_ratio = float(w)/h

Bien, si calculamos el aspect ratio de la placa es decír 404/154 obtendríamos 2.62, entonces podríamos comparar con un valor cercano a este. Para dejar un margen de error, concretamente usaré para este tutorial 2.4.

**if** aspect\_ratio > 2.4:

**NOTA:** El criterio de aplicar el aspect ratio según las medidas de la placa vehicular es una sugerencia que podrías o no aplicar. Si deseas no aplicar esto, únicamente debes comparar con que el aspect ratio sea mucho mayor a 1, para que el contorno sea considerado un rectángulo.

**1.3 Una vez obtenida la placa… ¡Es hora de aplicar OCR con pytesseract!**

Ahora que establecimos condiciones para poder encontrar el área en donde se encuentra la placa, deberemos trabajar sobre ese área para obtener sus caracteres y dígitos.

placa = gray[y:y+h,x:x+w]

text = pytesseract.image\_to\_string(placa,config='--psm 11')

print('PLACA: ',text)

**Línea 29:** Almacenamos en

placa

 el área donde está presente la matrícula, de la imagen en escala de grises.

Debo decir que realicé pruebas tanto con imágenes binarias de placas, así como en grises, y con la segunda obtuve un mejor reconocimiento de caracteres (en este caso), es por ello que en este tutorial tomo el área de la placa en la imagen en escala de grises.

**Línea 30:** Como ya tenemos la región de interés lista aplicaremos

pytesseract.image\_to\_string

, como primer argumento estará

placa

, mientras que el segundo corresponde al modo de segmentación de página

config='--psm 11'

. ¿Qué quiere decir esto?, te dejo a continuación una lista de los modos de segmentación de página que soporta tesseract (Si pruebas con otros, el resultado de reconocimiento de caracteres puede cambiar).

Texto

El contenido generado por IA puede ser incorrecto.

Lista de modos de segmentación de página compatibles con tesseract ([Fuente](https://github.com/tesseract-ocr/tessdoc/blob/master/ImproveQuality.md#page-segmentation-method))

**Línea 31:** Imprimimos la informacion obtenida luego de aplicar tesseract a la placa.

**1.4 Finalmente, ¡visualización!**

cv2.imshow('PLACA',placa)

cv2.moveWindow('PLACA',780,10)

cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),3)

cv2.putText(image,text,(x-20,y-10),1,2.2,(0,255,0),3)

cv2.imshow('Image',image)

cv2.moveWindow('Image',45,10)

cv2.waitKey(0)

**Línea 33:** Visualizamos la placa detectada.

**Línea 34:** Con

cv2.moveWindow

, indicamos que la ventana

PLACA

 se visualice en la posición

(780,10)

 de la pantalla.

**Línea 35:** Dibujamos un rectángulo que rodeará a la placa, para ello usamos la información obtenida en la **línea 19**.

**Línea 36:** Visualizamos la información obtenida de la **línea 30**.

**Línea 38 a 40:** Visualizamos la imagen contenida en

image

, luego hacemos que la ventana correspondiente a esta imagen se ubique en la posición

(45,10)

 de la pantalla, y finamente usamos

cv2.waitKey(0)

 para visualizar hasta que cualquier tecla sea presionada.

**Programa completo**

import cv2

import pytesseract

pytesseract.pytesseract.tesseract\_cmd = r'C:\Program Files\Tesseract-OCR\tesseract'

placa = []

image = cv2.imread('auto001.jpg')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

gray = cv2.blur(gray,(3,3))

canny = cv2.Canny(gray,150,200)

canny = cv2.dilate(canny,None,iterations=1)

#\_,cnts,\_ = cv2.findContours(canny,cv2.RETR\_LIST,cv2.CHAIN\_APPROX\_SIMPLE)

cnts,\_ = cv2.findContours(canny,cv2.RETR\_LIST,cv2.CHAIN\_APPROX\_SIMPLE)

#cv2.drawContours(image,cnts,-1,(0,255,0),2)

**for** c **in** cnts:

area = cv2.contourArea(c)

x,y,w,h = cv2.boundingRect(c)

epsilon = 0.09\*cv2.arcLength(c,**True**)

approx = cv2.approxPolyDP(c,epsilon,**True**)

**if** len(approx)==4 and area>9000:

print('area=',area)

#cv2.drawContours(image,[approx],0,(0,255,0),3)

aspect\_ratio = float(w)/h

**if** aspect\_ratio>2.4:

placa = gray[y:y+h,x:x+w]

text = pytesseract.image\_to\_string(placa,config='--psm 11')

print('PLACA: ',text)

cv2.imshow('PLACA',placa)

cv2.moveWindow('PLACA',780,10)

cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),3)

cv2.putText(image,text,(x-20,y-10),1,2.2,(0,255,0),3)

cv2.imshow('Image',image)

cv2.moveWindow('Image',45,10)

cv2.waitKey(0)

**Resultados**

Luego de todo este procedimiento, obtenemos lo siguiente:

Un coche deportivo de color rojo

El contenido generado por IA puede ser incorrecto.

**Figura 8:** Identificación de la placa vehicular, junto a la imagen de la placa

En la figura 8 muestra la imagen de salida, en la que podemos ver como se ha encontrado exitosamente la placa, además de los caracteres que aparecene en ella.

He probado también con otras imágenes que encontré navegando por internet, y esta vez son imágenes reales, veamos los resultados:

Un coche deportivo de color gris

El contenido generado por IA puede ser incorrecto.

**Figura 9:** Resultados al aplicar el programa sobre imágenes reales.

En los autos de las imagenes de la parte superior de la figura 9 hemos obtenido realmente buenos resultados, sin embargo si nos fijamos en los autos de la parte inferior en especial en la ‘Q’, tenemos que el carro de la izquierda si se pudo identificar, sin embago en el auto de la derecha en vez de «Q» hemos obtenido «0», además si notas el último dígito es 0, pero dado a que tiene un vacío en la parte superior derecha ha sido reconocido como un ‘6’.

En otras también probadas, a pesar de que se detectaban muy bien las placas, al aplicar OCR no obtenía ningún caracter u obtenía caracteres totalmente diferentes. Te recomiendo realizar muchas pruebas de funcionamiento ya que el reconocimiento de caracteres podría fallar.

Dado que este programa no está realizado con machine learning, es más propenso a que falle ante cambios de iluminación, ángulo de vista, entre otros a menos que, puedas controlar el ambiente donde trabaje. Y bien, espero que este tutorial sirva para repasar el contenido visto en anteriores publicaciones y también espero que te hayas divertido mucho. Nos vemos en un siguiente tutorial de visión por computador.