MÓDULO 2 SESIoN 3-4

PROCESADO DE IMAGEN Y VISIÓN POR COMPUTADOR



8 de noviembre de 2023

Ricardo Martínez Guadalajara

# Introducción

En esta sesión se va a trabajar con modelos YOLO

# Objetivo

Desarrollar los apartados del guion de prácticas como programas de Python

# Contenido

## Apartado 1

SE escogen tres categorías: train, book y zebra. En todos los casos se ha intentado trabajar sobre imágenes de alta complejidad para detección, ya sea por condiciones de iluminación, por cantidad de elementos o por intersección con otras imágenes.

En cuanto a los libros, el resultado ha sido bastante pobre. De las cuatro imágenes procesadas sólo en una ha detectado libros. El análisis que puedo hacer es que el modelo no recoge características de libros vistos de forma frontal y que las condiciones de iluminación influyen mucho.

El resultado con los trenes es algo mejor, pero tampoco muy satisfactorio. Se ha producido una mayor detección, pero en imágenes complejas no es capaz de separar elementos. Solo en una imagen donde se ven los trenes de forma clara, aunque sean seis bastante juntos es capaz de llevar a cabo una detección valiosa.

El caso de las cebras es muy distinto. Quizá porque tiene unas características muy marcadas con respecto al resto de elementos que puedan aparecer normalmente en una imagen. Pero es cierto que en imágenes altamente complejas por acumulación de animales es capaz de separar de manera satisfactoria.

**¿Qué dos parámetros de entrada incorpora la función de predicción? Justifique cómo afectan ambos parámetros en función de los resultados con las imágenes seleccionadas.**

La función de predicción `model.predict()` tiene dos parámetros de entrada: `conf` y `iou`. Estos parámetros afectan los resultados de la predicción en función de las imágenes seleccionadas.

1. El parámetro `conf` se refiere al umbral de confianza. Es un valor entre 0 y 1 que determina cuán seguro debe estar el modelo para considerar una detección como válida. Si estableces un valor alto para `conf`, como 0.8, el modelo solo considerará detecciones con una confianza del 80% o más. Esto puede ayudar a filtrar detecciones menos seguras y mejorar la precisión de las predicciones. Por otro lado, si estableces un valor bajo para `conf`, como 0.5, el modelo considerará detecciones con una confianza del 50% o más, lo que puede resultar en más detecciones, pero también puede incluir detecciones menos precisas.

2. El parámetro `iou` se refiere al umbral de intersección sobre unión. Es un valor entre 0 y 1 que determina cuánta superposición debe haber entre dos detecciones para considerarlas como la misma entidad. Si estableces un valor alto para `iou`, como 0.5, el modelo considerará que dos detecciones se superponen si su intersección sobre unión es del 50% o más. Esto puede ayudar a evitar la duplicación de detecciones y mejorar la coherencia de los resultados. Por otro lado, si estableces un valor bajo para `iou`, como 0.3, el modelo considerará que dos detecciones se superponen si su intersección sobre unión es del 30% o más, lo que puede resultar en más detecciones, pero también puede incluir detecciones redundantes.

En resumen, ajustar los valores de `conf` y `iou` te permite controlar el equilibrio entre la precisión y la cantidad de detecciones en tus resultados. Es importante encontrar un equilibrio adecuado para tu caso de uso específico, considerando la calidad deseada de las detecciones y la eficiencia computacional.

## Apartado 2

**Explique el cometido del siguiente bloque de código y el significado de las variables implicadas (track\_history, track y points).**

Para cada caja delimitadora (box) y su correspondiente identificador de seguimiento (track\_id):

* Se extraen las coordenadas x, y y las dimensiones w, h de la caja delimitadora.
* Se obtiene el seguimiento (track) correspondiente al track\_id del historial de seguimientos (track\_history).
* Se añade el punto central de la caja delimitadora al seguimiento. Las coordenadas x y y se convierten a float antes de añadirlas.
* Se convierten los puntos de seguimiento a un array de NumPy de tipo int32 y se reorganizan en un array de forma (-1, 1, 2). Esto se hace para que los puntos puedan ser utilizados por la función cv2.polylines().
* Finalmente, se dibujan las líneas de seguimiento en el fotograma anotado (annotated\_frame) utilizando la función cv2.polylines(). La línea de seguimiento no está cerrada (isClosed=False), el color de la línea es gris claro (color=(230, 230, 230)) y el grosor de la línea es 10.

**¿Cuál es el objetivo del parámetro fijado en un valor igual a 30? Pruebe con otros valores para ver cómo afecta a la representación.**

El parámetro fijado a 30 quiere decir que si el seguimiento tiene más de 30 puntos, se elimina el punto más antiguo. Esto mantiene un historial de los últimos 30 puntos de seguimiento.

**XXXX responder a todo tras ejecución de script instalar lap XXXXX**

## Apartado 3

El script yolov8\_pose.py se puede explicar en los siguientes pasos:

Primero, se importan las bibliotecas necesarias y se define una función compute\_angle para calcular el ángulo entre dos vectores.

Se inicializan los parámetros de umbral de confianza (confThreshold) e intersección sobre unión (iouThreshold).

Se carga un modelo preentrenado de YOLO con el archivo 'yolov8n-pose.pt'.

Se configura un analizador de argumentos para aceptar la ruta a un archivo de imagen o video como entrada.

Si se proporciona una imagen, se verifica que el archivo exista y se abre con cv.VideoCapture. Si se proporciona un video, se hace lo mismo. Si no se proporciona ninguna de las dos, se asume que se debe usar la cámara web.

Se crea una ventana de OpenCV para mostrar los resultados.

Si la entrada es un video o la cámara web, se inicializa un escritor de video para guardar el video de salida.

Luego, en un bucle, se lee cada fotograma del video o la imagen. Se detiene el programa si se llega al final del video.

Se convierte el fotograma a RGB y se pasa al modelo YOLO para la detección de objetos.

Se visualizan los resultados en el fotograma y se extraen los puntos clave y las cajas delimitadoras de los objetos detectados.

Finalmente, se guarda el fotograma con las cajas delimitadoras en el archivo de salida y se muestra en la ventana de OpenCV.

**Modifique el código para que mediante la función cv2.putText represente en la imagen el índice de cada keypoint con la finalidad de identificarlo.**

Se añade esta pieza de código

        # To do: represent indexes of key-points

        for j in range(len(kp\_obj\_i)):

            cv.circle(annotated\_frame, (int(kp\_obj\_i[j,0]\*width), int(kp\_obj\_i[j,1]\*height)), 5, (0,0,255), -1)

            cv.putText(annotated\_frame, str(j), (int(kp\_obj\_i[j,0]), int(kp\_obj\_i[j,1])), cv.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 255), 2, cv.LINE\_AA)

            #pintar por consola los keypoints y el texto

            print("keypoint ",j,": ",kp\_obj\_i[j,0],",",kp\_obj\_i[j,1])

Con ello recorremos los keypoints y pintamos un círculo rojo y añadimos un texto con su índice en color morado. Luego pintamos los keypoints por consola

**¿Cuántos puntos clave constituyen el modelo y a qué parte del cuerpo están asociados?**

Se puede observar claramente que son necesarios diecisiete puntos para determinar el mapa clave de la detección de posiciones en personas. Están asociados con las principales articulaciones: tobillos (2) rodillas (2) cadera (2) hombros(2) codos(2) manos(2), luego orejas (2) ojos(2) y nariz (2) otal 17.

**Puede intentar estimar el movimiento del corredor a partir de su patrón de marcha**

Se añade esta pieza de código

# Calcular los vectores

        hip\_to\_knee\_right = kp\_obj\_i[14] - kp\_obj\_i[12]

        knee\_to\_ankle\_right = kp\_obj\_i[16] - kp\_obj\_i[14]

        hip\_to\_knee\_left = kp\_obj\_i[13] - kp\_obj\_i[11]

        knee\_to\_ankle\_left = kp\_obj\_i[15] - kp\_obj\_i[13]

        # Calcular el ángulo

        angle\_right = compute\_angle(hip\_to\_knee\_right, knee\_to\_ankle\_right)

        angle\_left = compute\_angle(hip\_to\_knee\_left, knee\_to\_ankle\_left)

        # Dibujar el ángulo en la imagen

        cv.putText(annotated\_frame, f"Angle right leg: {angle\_right:.2f}", (50, 50), cv.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE\_AA)

        cv.putText(annotated\_frame, f"Angle left leg: {angle\_left:.2f}", (50, 100), cv.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2, cv.LINE\_AA)

Con ello se computa el ángulo de los vectores de la cadera a la rodilla y de la rodilla al tobillo de ambas piernas y lo pinta por pantalla. Para discernir el patrón de movimiento habría que guardar un histórico de estos valores y analizarlo en cuanto a la variación de apertura de la rodilla de ambas piernas y su variación en el tiempo podría indicar si está corriendo, trotando o andando, para lo cual habría que establecer umbrales de variación de estos ángulos y acotarlos a cada patrón de marcha. Pero este trabajo no es objeto de esta práctica, a mi entender.