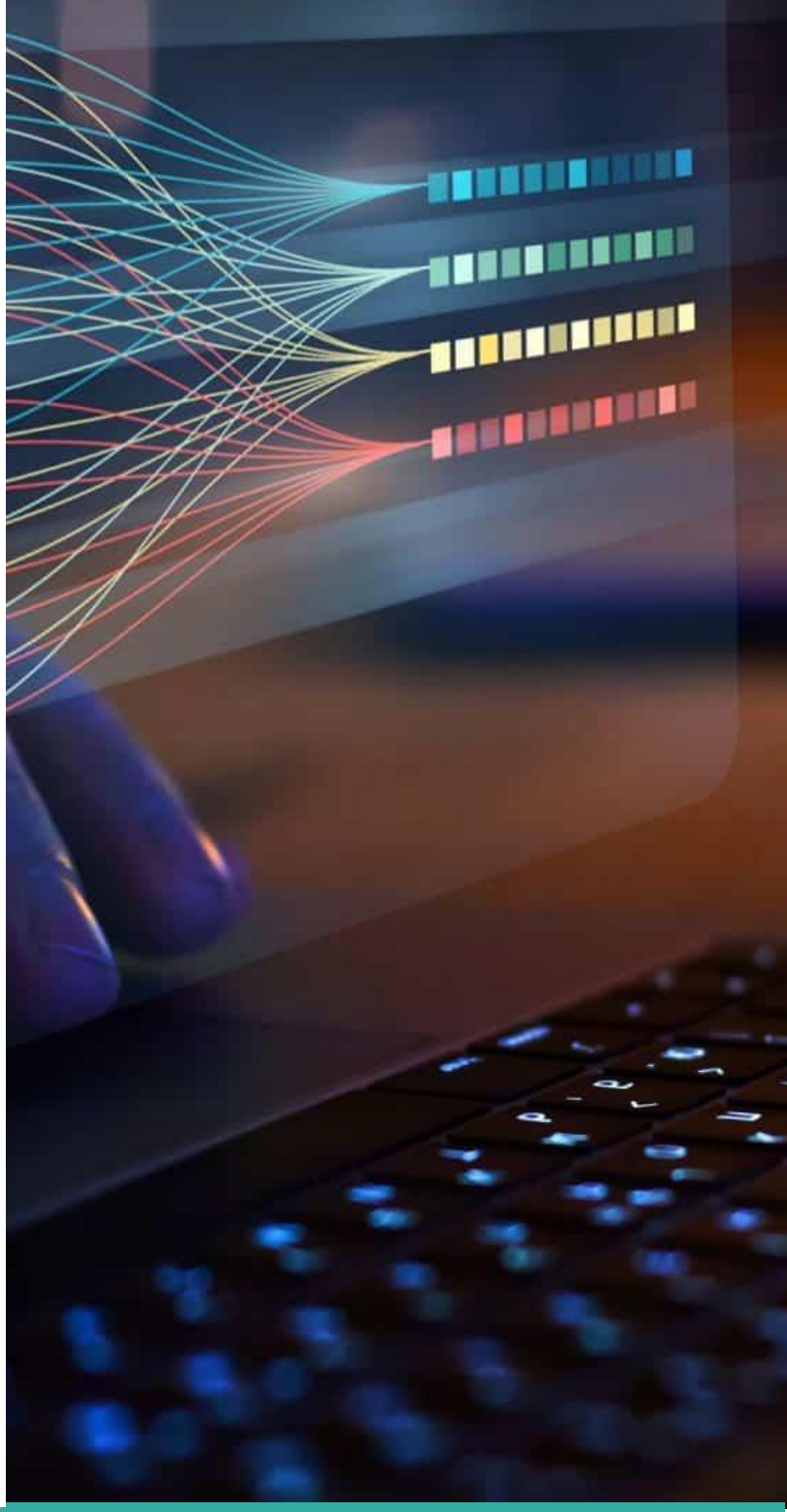
Taller 2

Visión por Computadora



3 MAYO

ELE068-B Inteligencia artificial

Creado por: Maciel Ripetti

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Visión por computadora

1. Dataset

Para este proyecto se utilizó un conjunto de 56 imágenes de perros de raza Golden Retriever. Estas imágenes fueron recolectadas desde Pinterest y posteriormente etiquetadas utilizando la herramienta LabelMe, la cual permite realizar etiquetas de segmentación sobre las imágenes en formato json.

- Cantidad total de imágenes: 56
- Imágenes de entrenamiento: 45 (aproximadamente 80%)
- Imágenes de validación: 11 (aproximadamente 20%)

Procedimiento de etiquetado con LabelMe:
Las imágenes fueron cargadas en LabelMe, donde se utilizó la herramienta de segmentación poligonal para marcar con precisión la silueta de los perros en cada imagen. Cada objeto etiquetado se guardó en formato. Json, que posteriormente fue convertido al formato YOLO para su uso con el modelo YOLOv8-seg. Este proceso incluyó generar máscaras binarizadas (segmentación semántica) y definir las clases correspondientes.

2. Modelos YOLO Utilizados

Para esta práctica se entrenaron y compararon dos modelos YOLOv8m-seg y YOLOv8s-seg

Modelo 1: YOLOv8m-seg (versión media)

Para esta tarea de segmentación se utilizó el modelo **yolov8m-seg.pt**. Se seleccionó la versión médium (m) optimizada para **segmentación de instancias,** lo cual permite no solo detectar objetos, sino también generar máscaras que delinean su contorno de manera precisa**.**

- **Estructura del modelo**
  Para el proceso de entrenamiento, la estructura del modelo fue entregada por consola:
  yolo task=segment mode=train epochs=30 data=dataset.yaml model=yolov8m-seg.pt imgsz=640 batch=2
  lo cual podemos observar en la imagen adjunta más abajo.

  La arquitectura general de YOLOv8 incluye:
  - **Backbone**: Extrae características visuales profundas (usualmente CSPDarknet).
  - **Neck**: Refina y fusiona características (como FPN o PAN).
  - **Head**: Produce las salidas finales para detección y segmentación.
  - **Segmentación**: En el modelo -seg, se agrega una rama dedicada a generar máscaras para cada objeto detectado.

- **Hiperparámetros del entrenamiento**
  Durante el entrenamiento se utilizaron más de **60 hiperparámetros**, de los cuales se destacan:
  Entrenamiento básico:
    epochs=30: número total de ciclos de entrenamiento.
    batch=2: tamaño de lote pequeño, útil para equipos sin GPU.
    imgsz=640: tamaño de las imágenes reescaladas.
  Optimización:
    lr0=0.01, lrf=0.01: tasa de aprendizaje inicial y final.
    momentum=0.937, weight_decay=0.0005: parámetros para estabilizar y regularizar el entrenamiento.
  Warmup y regularización:
    warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1
    dropout=0.0, label_smoothing=0.0: no se aplicó regularización adicional por dropout.
  Aumentos de datos
    translate=0.1, scale=0.5, fliplr=0.5: desplazamientos, escalados y espejeo.
    hsv_h=0.015, hsv_s=0.7, hsv_v=0.4: ajustes de color.
    auto_augment=randaugment, erasing=0.4: aumentos avanzados para mejorar la generalización.
  Segmentación específica:
    overlap_mask=True: permite que las máscaras se superpongan.
    mask_ratio=4: escala de las máscaras respecto a la imagen original.

Todo esto se encuentra en la siguiente imagen

```
(.venv) C:\Users\macie\Downloads\Taller_2_Maciel-Ripetti\imagen>yolo task=segment mode=train epochs=30 data=dataset.yaml
 model=yolov8m-seg.pt imgsz=640 batch=2
New https://pypi.org/project/ultralytics/8.3.121 available 🅿  Update with 'pip install -U ultralytics'
Ultralytics 8.3.15 🅿  Python-3.10.5 torch-2.6.0+cpu CPU (AMD Ryzen 5 4500U with Radeon Graphics)
engine\trainer: task=segment, mode=train, model=yolov8m-seg.pt, data=dataset.yaml, epochs=30, time=None, patience=100, b
atch=2, imgsz=640, save=True, save_period=-1, cache=False, device=None, workers=8, project=None, name=train4, exist_ok=F
alse, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=Fa
lse, close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, freeze=None, multi_scale=False, overlap_mask=
True, mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False, save_hybrid=False, conf=None, iou=0.7, max_det=30
0, half=False, dnn=False, plots=True, source=None, vid_stride=1, stream_buffer=False, visualize=False, augment=False, ag
nostic_nms=False, classes=None, retina_masks=False, embed=None, show=False, save_frames=False, save_txt=False, save_conf
=False, save_crop=False, show_labels=True, show_conf=True, show_boxes=True, line_width=None, format=torchscript, keras=F
alse, optimize=False, int8=False, dynamic=False, simplify=True, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01,
momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1
.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, sca
le=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, bgr=0.0, mosaic=1.0, mixup=0.0, copy_paste=0.0, copy_paste_m
ode=flip, auto_augment=randaugment, erasing=0.4, crop_fraction=1.0, cfg=None, tracker=botsort.yaml, save_dir=runs\segmen
t\train4
Overriding model.yaml nc=80 with nc=1
```

Modelo 2: YOLOv8s-seg (versión pequeña)

Para esta prueba se utilizó el modelo **yolov8s-seg.pt**, que corresponde a la versión small de la serie YOLOv8, específicamente diseñada para segmentación de instancias. Es una arquitectura ligera que busca un buen equilibrio entre velocidad y precisión, ideal para ejecutarse en CPU o en entornos con recursos limitados.

- **Estructura del modelo**

  El modelo fue entrenado con la siguiente instrucción lo cual podemos observar también en la imagen adjunta más abajo.:
  yolo task=segment mode=train epochs=30 data=dataset.yaml model=yolov8s-seg.pt imgsz=640 batch=2

  Esto indica:
  task=segment: se entrena para segmentación.
  model=yolov8s-seg.pt: se usa la versión pequeña del modelo segmentador.
  imgsz=640: todas las imágenes son redimensionadas a 640x640 píxeles.
  batch=2: el tamaño del lote es pequeño, apropiado para CPU.
  epochs=30: se realizan 30 ciclos completos de entrenamiento.

  La arquitectura base de YOLOv8 incluye:
  - **Backbone**: red convolucional que extrae características de la imagen.
  - **Neck**: estructura que combina y refina esas características (como FPN o PAN).
  - **Head de segmentación**: salida que genera máscaras para cada objeto detectado.

- **Hiperparámetros utilizados**

  Durante el entrenamiento, el modelo manejó más de **60 hiperparámetros**, agrupados en distintas categorías. Algunos importantes son:

  Entrenamiento
  - epochs=30, batch=2, imgsz=640
  - optimizer=auto: se selecciona automáticamente el mejor optimizador para el caso.
  - patience=100: tolerancia antes de detener por falta de mejora.

  Optimización
  - lr0=0.01, lrf=0.01: tasa de aprendizaje inicial y final.
  - momentum=0.937, weight_decay=0.0005: parámetros para estabilización y regularización.

  Aumentos de datos
  - hsv_h=0.015, hsv_s=0.7, hsv_v=0.4: alteraciones de color.
  - translate=0.1, scale=0.5, fliplr=0.5: transformaciones geométricas.
  - auto_augment=randaugment: estrategia automática de aumentos.
  - erasing=0.4: ocultamiento aleatorio de regiones para evitar sobreajuste.

Segmentación

- overlap_mask=True: permite máscaras que se superponen.
- mask_ratio=4: relación entre la resolución de las máscaras y la imagen.

Otros

- val=True: se evalúa el modelo en el conjunto de validación.
- save=True: guarda el mejor modelo (best.pt) al final del entrenamiento.

```
(.venv) C:\Users\macie\Downloads\Taller_2_Maciel-Ripetti\imagen>yolo task=segment mode=train epochs=30 data=dataset.yaml
model=yolov8s-seg.pt imgsz=640 batch=2
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8s-seg.pt to 'yolov8s-seg.pt'...
100%|████████████████████████████████████████████████████████████████████████████████████████| 22.8M/22.8M [00:03<00:00, 7.83MB/s]

New https://pypi.org/project/ultralytics/8.3.124 available 😃 Update with 'pip install -U ultralytics'
Ultralytics 8.3.15 🚀 Python-3.10.5 torch-2.6.0+cpu CPU (AMD Ryzen 5 4500U with Radeon Graphics)
engine\trainer: task=segment, mode=train, model=yolov8s-seg.pt, data=dataset.yaml, epochs=30, time=None, patience=100, ba
tch=2, imgsz=640, save=True, save_period=-1, cache=False, device=None, workers=8, project=None, name=train5, exist_ok=Fal
se, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=False
, close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, freeze=None, multi_scale=False, overlap_mask=True
, mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False, save_hybrid=False, conf=None, iou=0.7, max_det=300, ha
lf=False, dnn=False, plots=True, source=None, vid_stride=1, stream_buffer=False, visualize=False, augment=False, agnostic
_nms=False, classes=None, retina_masks=False, embed=None, show=False, save_frames=False, save_txt=False, save_conf=False,
 save_crop=False, show_labels=True, show_conf=True, show_boxes=True, line_width=None, format=torchscript, keras=False, op
timize=False, int8=False, dynamic=False, simplify=True, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, momentum=
0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=1
2.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, she
ar=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, bgr=0.0, mosaic=1.0, mixup=0.0, copy_paste=0.0, copy_paste_mode=flip, au
to_augment=randaugment, erasing=0.4, crop_fraction=1.0, cfg=None, tracker=botsort.yaml, save_dir=runs\segment\train5
Overriding model.yaml nc=80 with nc=1
```

3. Comparaciones

Se realizó una comparación entre ambos modelos utilizando la misma cámara web como fuente de video, se cambió la línea de código **cap = cv2.VideoCapture(1)** de 0 a 1 para usar una cámara externa, Ambos modelos fueron entrenados con el mismo dataset y los mismos parámetros, variando únicamente la arquitectura base (medium vs small).
La configuración usada para predicción fue:

results = model.predict(frame, imgsz=640, conf=0.90)

Ambas pruebas se realizaron con la misma configuración, ya que no fue necesario la modificación de la confianza, debido a que el valor de 0.90 permitió filtrar predicciones y mostró resultados satisfactorios especialmente en el modelo YOLOv8s-seg que fue bastante rápido en detectar la imagen. En el caso del modelo YOLOv8m-seg, no fue necesario reducir el umbral para que se mostraran resultados consistentes, pero si le costó unos segundos más detectar la imagen.
El modelo entrenado para **YOLOv8s-seg,** se llama **best_s.pt.** y para **YOLOv8m-seg** se llama **best_m.pt,** a continuación, se muestran las salidas de consola al momento de ejecutar el código con cada uno de los modelos entrenados.

Salida de consola de YOLOv8m-seg en la detección de perros Golden:

```
0: 480x640 1 Golden, 336.1ms
Speed: 2.0ms preprocess, 336.1ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 1 Golden, 294.1ms
Speed: 3.0ms preprocess, 294.1ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 1 Golden, 302.1ms
Speed: 2.0ms preprocess, 302.1ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)
```

Salida de consola de YOLOv8s-seg en la detección de perros Golden:

```
0: 480x640 1 Golden, 147.0ms
Speed: 2.0ms preprocess, 147.0ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 1 Golden, 147.0ms
Speed: 2.0ms preprocess, 147.0ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 1 Golden, 147.0ms
Speed: 1.0ms preprocess, 147.0ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)
```

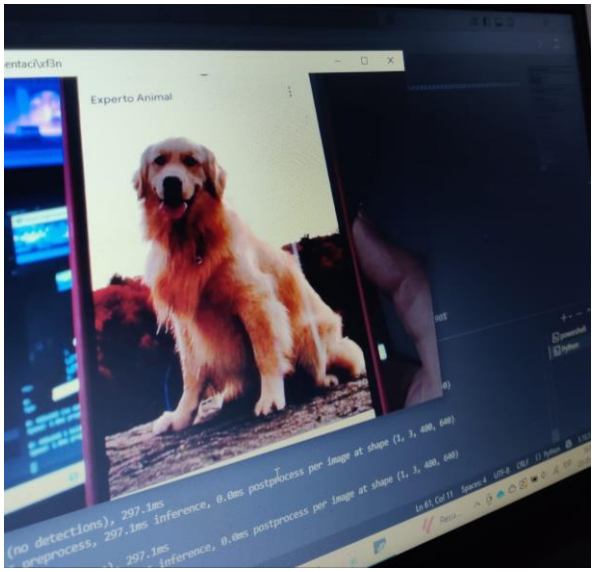
Comparación de ambas detecciones:

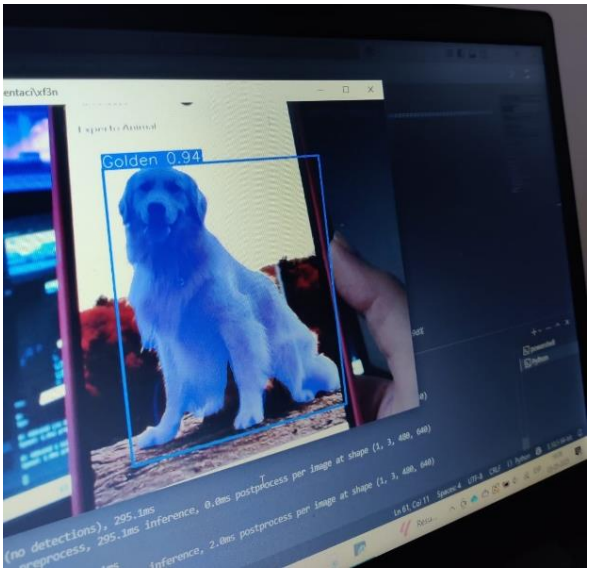
Métrica	YOLOv8m-seg	YOLOv8s-seg
Inferencia	~294–336 ms	147 ms
Total, por imagen	~298–340 ms	149–151 ms
Precisión esperada	Mayor	Menor

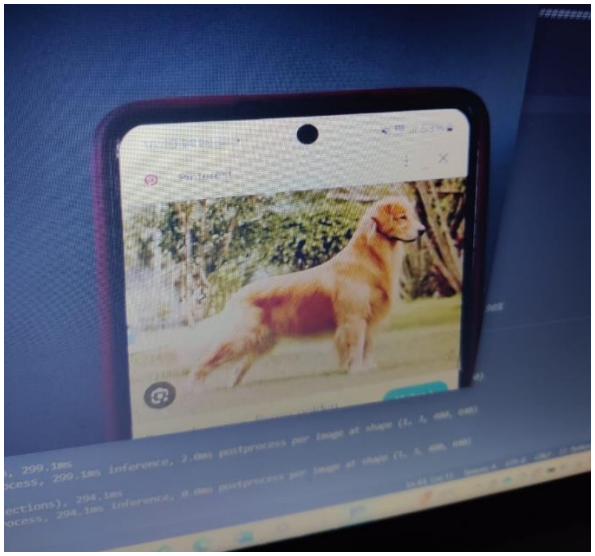
4. Resultados

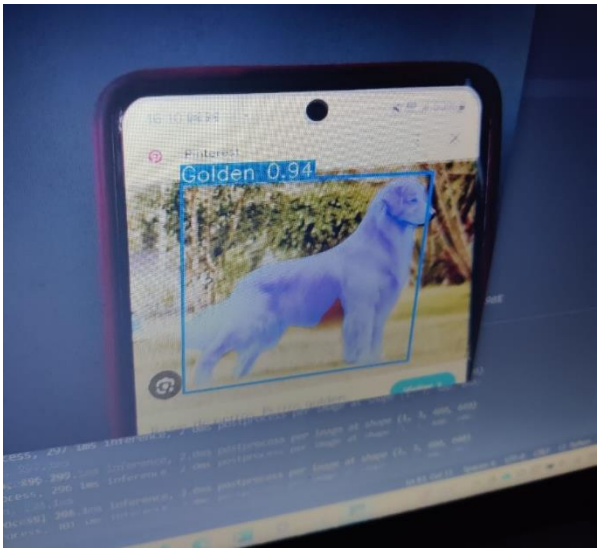
Ambos modelos fueron capaces de detectar y segmentar perros Golden en imágenes captadas desde la cámara.
Las pruebas se realizaron con imágenes del celular, sacadas de Google. El modelo medium fue más robusto ante variaciones de fondo e iluminación, pero tomo unos segundos más en detectar la imagen, en cambio el modelo small detecto de forma muy rápida y era muy preciso si es qué la imagen tenía fondo blanco.
A continuación, se mostrarán detecciones para ambos modelos.

Detección de perros Golden en YOLOv8m-seg :

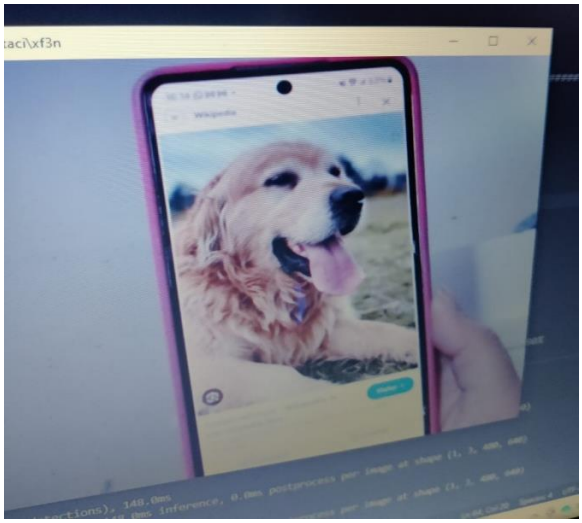


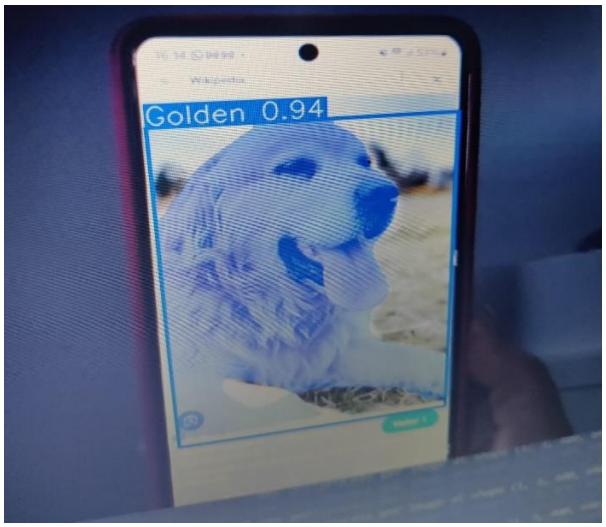


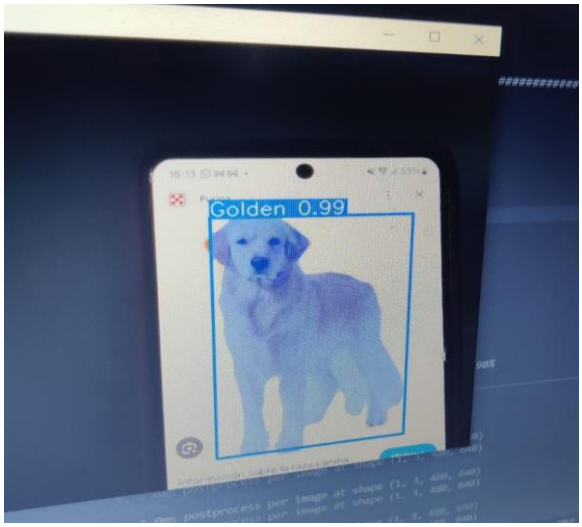


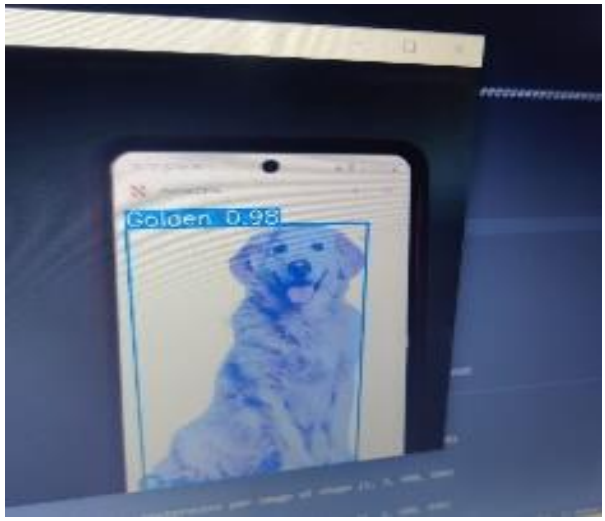


Detección de perros Golden en YOLOv8s-seg :









5. Conclusiones

Durante el desarrollo de esta práctica se aprendió a realizar un pipeline completo de segmentación de imágenes, desde la anotación con LabelMe hasta la evaluación en tiempo real con modelos YOLO. Se comprobó que el tamaño del modelo influye significativamente en la precisión de la segmentación, especialmente en tareas con objetos complejos como animales. Trabajar con un conjunto limitado de datos (56 imágenes) reforzó la importancia de una buena anotación y preprocesamiento.

En general, el modelo **YOLOv8m-seg** demostró ser el más adecuado para esta tarea, confirmando que modelos más complejos, aunque más pesados, pueden generar resultados confiables incluso con dataset pequeños por ende es mejor si tu prioridad es la **precisión**, aunque es más lento. En cambio, **YOLOv8s-seg** es preferible si necesitas **velocidad en tiempo real** con buena eficiencia y tienes recursos limitados, en este caso fue muy eficiente para imágenes con fondos blancos.