UNIVERSIDAD MODELO
INGENIERÍA

Detección de celular con YOLOv8

Visión por Computadora

Jorge Humberto Sosa García

Ingeniería Mecatrónica

Séptimo Semestre

(Agosto / diciembre 2025)

03 de noviembre de 2025

Universidad Modelo Campus Mérida

ÍNDICE

1 INTRODUCCIÓN

El reconocimiento en tiempo real de objetos mediante visión por computadora representa un componente clave en aplicaciones modernas de automatización, seguridad e interacción hombre-máquina. Esta práctica se centra en la detección de un teléfono celular utilizando un modelo de inteligencia artificial basado en YOLOv8, ejecutado sobre un entorno Python controlado.

Aprovechando modelos previamente entrenados sobre el conjunto de datos COCO, se logró identificar dispositivos móviles en video en vivo capturado por la cámara del sistema. Este tipo de implementación permite entender el funcionamiento de los sistemas de inferencia visual, evaluar su precisión y explorar su aplicabilidad en entornos reales como sistemas de vigilancia, clasificación automática o control por gestos.

La práctica forma parte del curso de Visión por Computadora y busca fortalecer la comprensión de modelos convolucionales para detección de objetos, así como la integración de frameworks como Ultralytics con flujos en tiempo real.

Ilustración 1. Imagen representativa de YOLOv8

2 OBJETIVOS

2.1 Objetivo General

Implementar un sistema de visión por computadora capaz de detectar un teléfono celular en tiempo real utilizando el modelo YOLOv8 y una cámara conectada al equipo.

2.2 Objetivos Específicos

- Configurar un entorno virtual con Python 3.10 para asegurar compatibilidad con las dependencias necesarias.
- Instalar y utilizar la librería Ultralytics para aprovechar modelos preentrenados de YOLO.
- Ejecutar detección de objetos en una fuente de video en vivo (webcam).
- Identificar la clase “cell phone” entre las predicciones y reportarla durante la ejecución.
- Evaluar la precisión del modelo y reflexionar sobre su aplicabilidad en escenarios reales.

3 MATERIALES Y HERRAMIENTAS

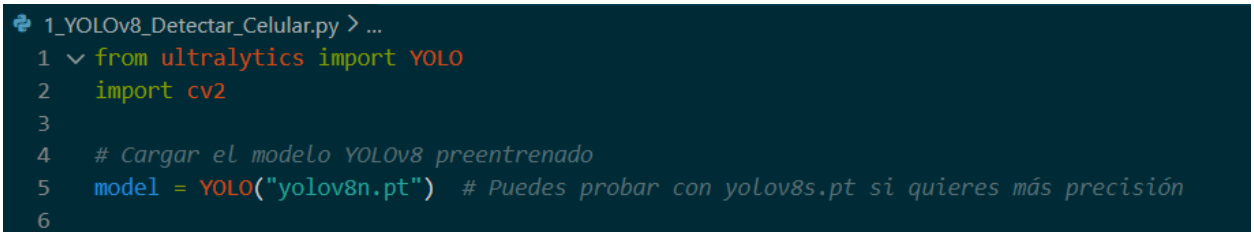
3.1 Hardware:

- Laptop personal con cámara integrada
- Teléfono celular como objeto de prueba

3.2 Software

- Sistema Operativo: Windows 11 (64-bit)
- Visual Studio Cod
- Python 3.10.9 (gestor de versiones con py)
- Entorno virtual (venv
- Librerías:
    - ultralytics
    - opencv-python

4 CÓDIGO



```
1_YOLOv8_Detectar_Celular.py > ...
from ultralytics import YOLO
import cv2

# Cargar el modelo YOLOv8 preentrenado
model = YOLO("yolov8n.pt")  # Puedes probar con yolov8s.pt si quieres más precisión

# Abrir cámara (0 = cámara por defecto)
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("No se pudo abrir la cámara.")
    exit()

print("Presiona 'q' para salir...")
```

Ilustración 2. Fragmento 1 del Código

En esta sección se importan dos librerías fundamentales para la práctica: ultralytics, que permite utilizar modelos YOLOv8 de forma directa, y cv2 (OpenCV), utilizada para capturar video desde la cámara y manipular imágenes. A continuación, se instancia el modelo YOLOv8 utilizando el archivo preentrenado yolov8n.pt. Esta versión del modelo (nano) es ligera y está optimizada para tareas en tiempo real, sacrificando algo de precisión en favor de velocidad. El modelo se carga en memoria y queda listo para realizar inferencias sobre imágenes o video.

Ilustración 3. Fragmento 2 del Código

Aquí se configura el acceso a la cámara web del equipo utilizando cv2.VideoCapture(0), donde el parámetro 0 representa la cámara principal. Se incluye una verificación para comprobar si la cámara fue correctamente abierta. En caso contrario, el programa muestra un mensaje de error y finaliza la ejecución. Este control es importante para asegurar que la entrada de video esté disponible antes de iniciar el procesamiento. También se imprime un mensaje en consola indicando que se puede cerrar el programa presionando la tecla q.

```python
while True:
    ret, frame = cap.read()
    if not ret:
        print("No se pudo leer el frame de la cámara.")
        break

    # Ejecutar detección
    results = model(frame)

    # Dibujar resultados
    annotated_frame = results[0].plot()

    # Mostrar ventana
    cv2.imshow("Detección en tiempo real", annotated_frame)
```

Ilustración 4. Fragmento 3 del Código

Este bloque representa el núcleo del programa. Se inicia un bucle while que se ejecuta de forma continua mientras la cámara esté activa. En cada iteración, se captura un fotograma (frame) desde la cámara. Luego, se pasa ese fotograma al modelo YOLO para realizar la detección de objetos. El resultado de la detección se procesa y se genera una versión anotada del fotograma, donde se dibujan las cajas delimitadoras y etiquetas sobre los objetos identificados. Finalmente, el fotograma anotado se muestra en una ventana usando cv2.imshow() para visualización en tiempo real.

```python
    # Verificar si se detecta un celular
    for box in results[0].boxes:
        cls_id = int(box.cls[0])
        cls_name = model.names[cls_id]
        if cls_name == "cell phone":
            print("📱 ¡Celular detectado!")

    # Salir si presiona 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Liberar recursos
cap.release()
cv2.destroyAllWindows()
```

Ilustración 5. Fragmento 4 del Código

Este último bloque complementa la detección realizando un análisis sobre los resultados. Recorre cada objeto detectado en el fotograma y verifica si su clase corresponde a "cell

phone", que es el identificador utilizado en el conjunto de datos COCO. Si se detecta un celular, se imprime un mensaje de alerta en la consola. Además, se monitorea la entrada del teclado: si el usuario presiona la tecla q, el bucle se detiene y se cierran tanto la cámara como la ventana de visualización. Esta estructura permite un cierre seguro y ordenado del programa.

5 RESULTADOS

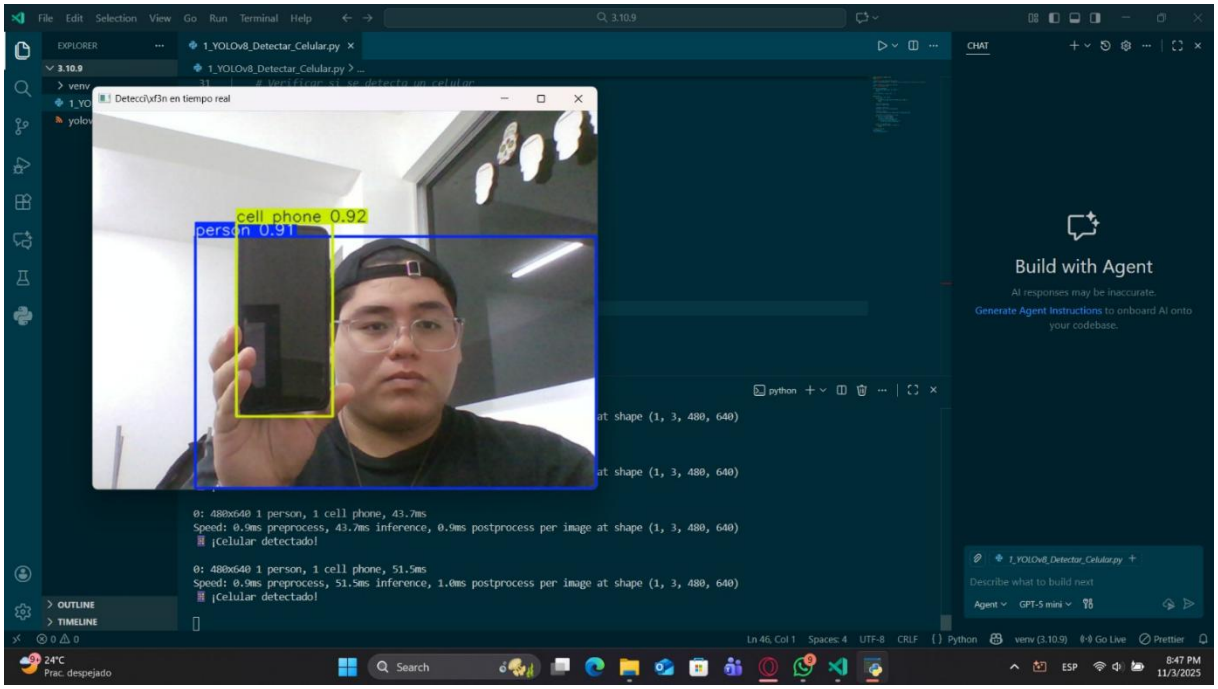


Ilustración 6. Prueba con vista de cara del celular

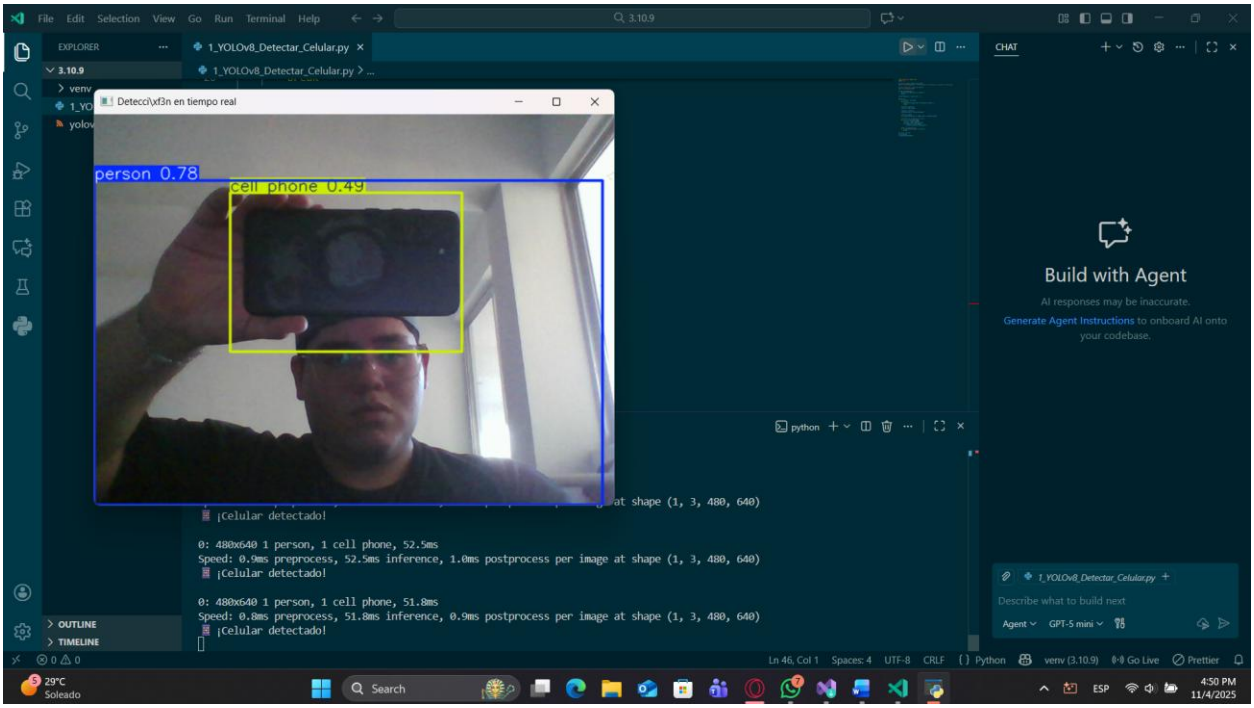


Ilustración 7. Pruba con vista de espalda del celular

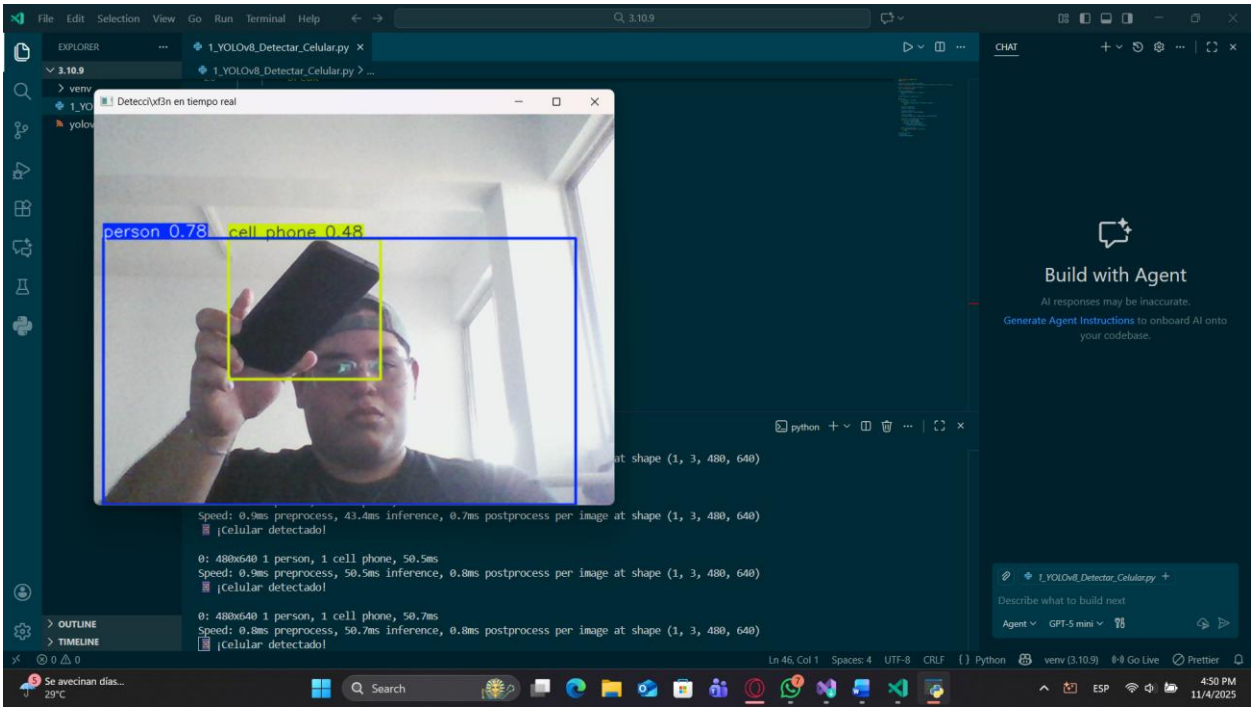


Ilustración 8. Prueba con vista irregular del celular (inclinación 45°)

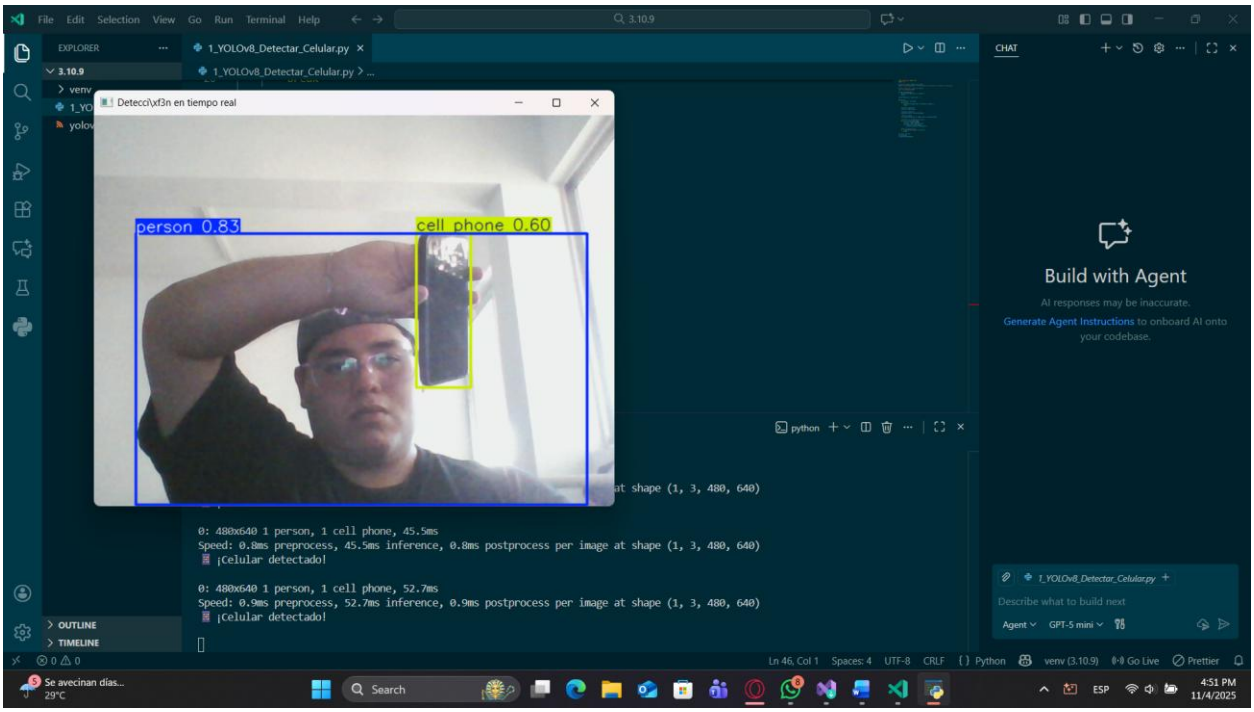


Ilustración 9. Prueba con vista irregular del celular (vertical)

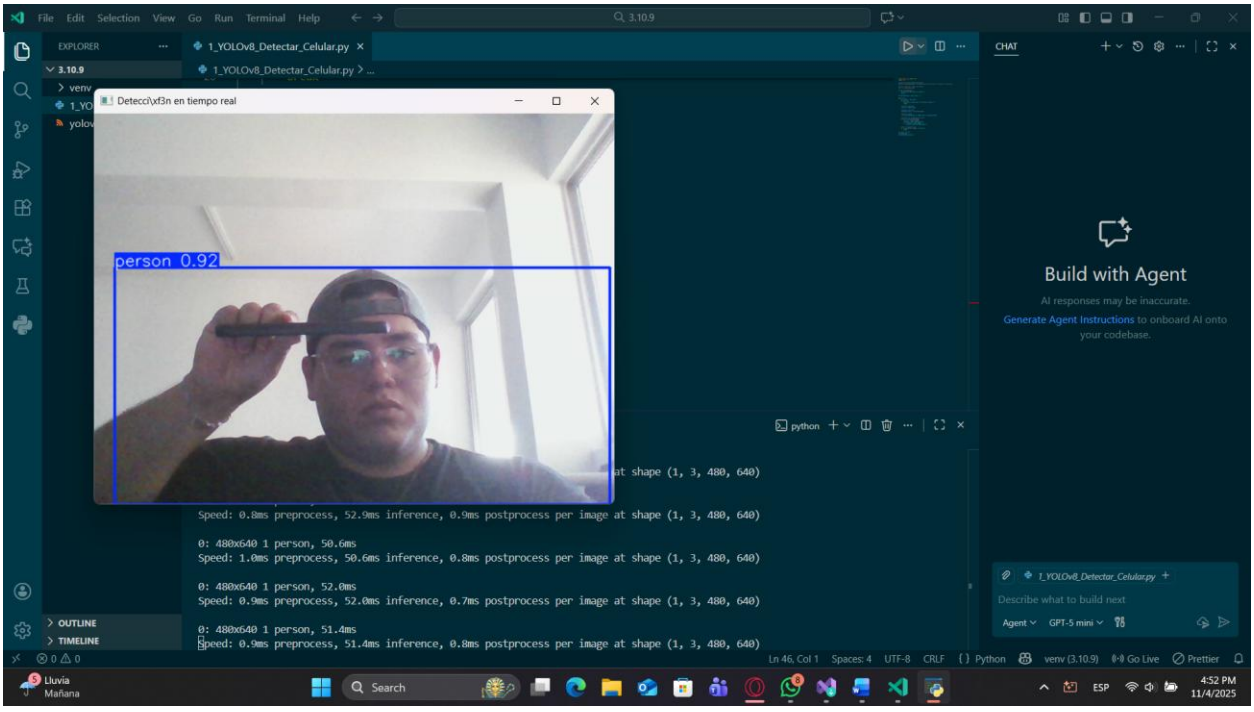


Ilustración 10. Prueba con vista lateral plana del celular

6 CONCLUSIONES

La implementación de un sistema de detección de objetos en tiempo real utilizando YOLOv8 fue exitosa, cumpliendo con el objetivo principal de identificar un celular frente a la cámara. El modelo preentrenado mostró una alta eficiencia al reconocer la clase “cell phone” en distintos ángulos y condiciones de iluminación moderada.

Uno de los aspectos más destacables fue la capacidad del sistema para procesar video en vivo con un rendimiento fluido, gracias al uso del modelo yolov8n.pt, que equilibra precisión y velocidad.

Sin embargo, se identificó una limitación notable: cuando el celular se encuentra completamente plano (de canto o alineado con el plano de la cámara), como se observa en la imagen 10 del documento, el modelo pierde la capacidad de detectarlo correctamente. Esto resalta la importancia del ángulo de exposición y la visibilidad de las características del objeto para una detección exitosa.

En general, la práctica permitió comprender a fondo la integración de modelos de visión por computadora con flujos de video en tiempo real, además de identificar fortalezas y debilidades de los sistemas de inferencia basados en redes neuronales convolucionales.

7 REFERENCIAS

- Jocher, G., et al. (2023). YOLO by Ultralytics. Ultralytics. https://docs.ultralytics.com/

- Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767. https://arxiv.org/abs/1804.02767

- OpenCV. (2023). Open Source Computer Vision Library. https://opencv.org/

- Python Software Foundation. (2023). Python Language Reference, version 3.10. https://docs.python.org/3.10/

- COCO Dataset. (2017). Common Objects in Context (COCO) Dataset. https://cocodataset.org/

- Ultralytics. (2023). YOLOv8 Release Notes and GitHub Repository. https://github.com/ultralytics/ultralytics