UA

Universidad

de Alicante

ESCUELA POLITÉCNICA SUPERIOR

Introducción Guiada a la Detección de Personas con HPC e IA

Curso académico 2024 / 2025

Semestre 2

Grado en Ingeniería en Inteligencia Artificial

Grupo 1

Profesora: Ricardo Moreno Rodríguez

Alumno: Santiago Álvarez Geanta

Universitat d'Alacant
Universidad de Alicante

Mayo 2025

ÍNDICE

Introducción

Este informe describe la implementación de un flujo de trabajo para la detección de personas en imágenes urbanas, combinando **High Performance Computing (HPC)** y modelos de **Inteligencia Artificial (IA)**. Configuré un entorno Python con aceleración GPU, adquirí y preprocesé imágenes de licencia abierta, y apliqué el modelo pre-entrenado **YOLOv8**.

En esta práctica se implementé un pipeline completo: configuración de entorno GPU, adquisición de imágenes, preprocesamiento, inferencia con **YOLOv8** y análisis de resultados. Además, reflexioné sobre la precisión del modelo, su rendimiento en CPU versus GPU, y la ética asociada al procesamiento de datos sensibles.

Metodología

3.1. Preparación del entorno

- **Sistema operativo:** Ubuntu 24.04 con drivers NVIDIA (nvidia-driver-550) y CUDA 12.x configurados bajo Secure Boot (MOK enrollado).
- **Entorno Python:** venv (python3.12 -m venv venv), activación (source venv/bin/activate)
- **Dependencias**: instalé todas las dependencias necesarias y las dejé reflejadas en un archivo **requirements.txt** usando:

```
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P14$ source venv/bin/activate
(venv) sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P14$ pip freeze > requirements.txt
```

3.2 Preparación del código

Decidí crear una clase que define una interfaz personalizada para cargar, ejecutar inferencias y visualizar resultados del modelo **YOLOv8**, específicamente enfocado en la detección de personas (**clase 0 en COCO**).

__init__(self, device=None, conf=0.25)

```python
class CustomYOLO:
    def __init__(self, device=None, conf=0.25):

        if (device == 'cpu'):
            self.device = device # Forzamos la ejecución de CPU
        else:
            self.device = 'cuda' if torch.cuda.is_available() else 'cpu'

        self._image = None
        self._results = None

        self.conf = conf
        self.model = None

        self.__loadModel()
```

- **device:** permite especificar si se quiere usar CPU ('cpu') o GPU. Si no se especifica, detecta automáticamente si CUDA está disponible.
- **conf:** umbral de confianza mínimo para aceptar una detección.

- **self.device:** se asigna a 'cuda' si hay GPU disponible, a menos que explícitamente se indique 'cpu'.
- **self._image, self._results:** atributos internos para almacenar la imagen cargada y los resultados de la predicción.
- **self.model:** inicializado en None, se cargará después con el **modelo YOLOv8.**

- **self.__loadModel():** se llama al constructor para cargar el modelo al dispositivo especificado.

__loadModel(self)

Carga el modelo YOLOv8 Nano (yolov8n.pt) desde Ultralytics y lo mueve al dispositivo (CPU o GPU).

```
def __loadModel(self):
    self.model = YOLO('yolov8n.pt').to(self.device)
```

__loadImage(self, imagePath: str)

```
def __loadImage(self, imagePath: str):
    self._image = cv2.imread(imagePath)
    if self._image is None:
        print(f"Error: No se pudo cargar la imagen en {imagePath}")
```

Carga una imagen desde el disco.

- Usa OpenCV (cv2.imread) para leer una imagen desde la ruta especificada.

- Si la imagen no se puede cargar (por ruta inválida o error), se muestra un mensaje de error.

- La imagen cargada se guarda en **self._image**.

predict(self, imagePath)

Realiza una predicción con el modelo YOLO.

```python
def predict(self, imagePath):
    self.__loadImage(imagePath)
    if self._image is None:
        print(f"Error: No se pudo cargar la imagen {imagePath}.")
        return

    self._results = self.model.predict(source=self._image, conf=self.conf, device=self.device)
```

- Llama a **__loadImage()** para cargar la imagen.
- Verifica si la imagen fue cargada correctamente.
- Llama al método predict del modelo YOLO sobre la imagen en memoria, utilizando el umbral de confianza y dispositivo configurados.

visualizate(self, figsize=(12, 8), savePath=None)

Visualiza los resultados de detección.

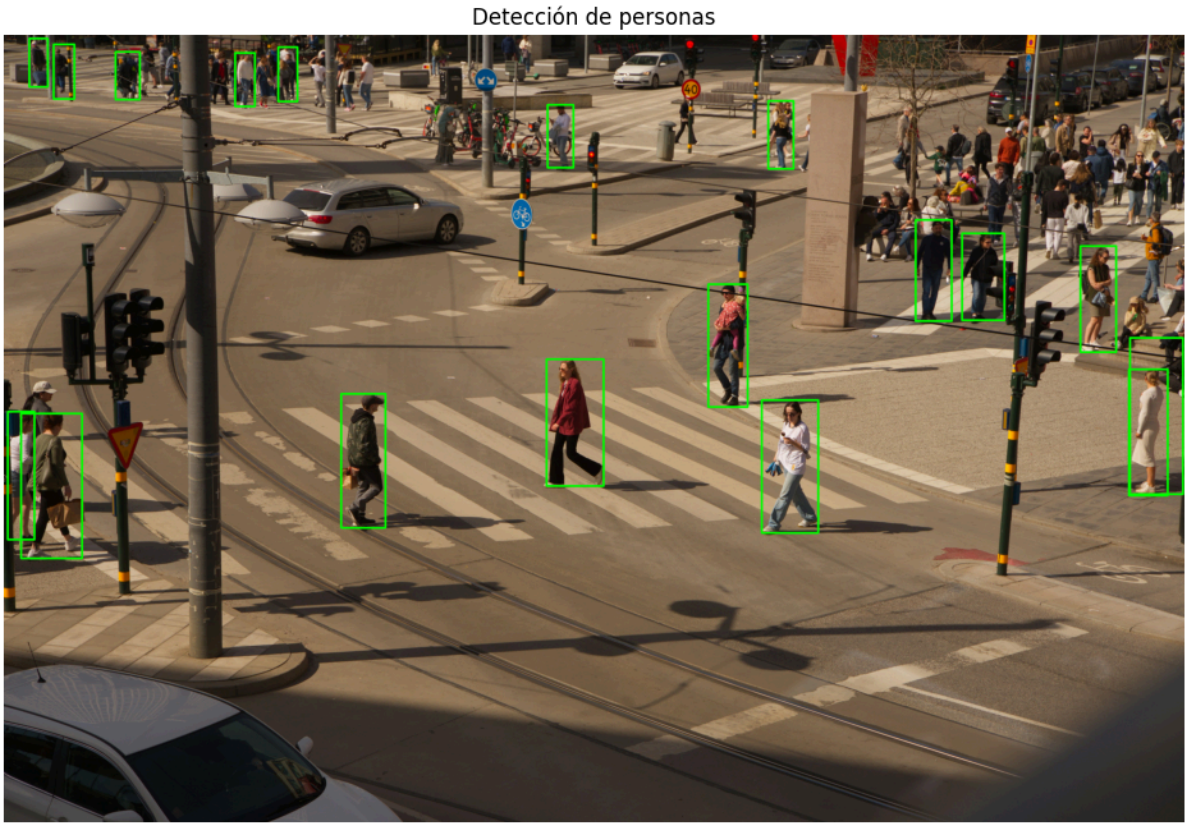
- Verifica que se haya realizado una predicción.
- Filtra únicamente las detecciones de clase persona (clase 0 en el modelo COCO).
- Para cada persona detectada:
    - Extrae las coordenadas del bounding box (box.xyxy).
    - Dibuja un rectángulo verde sobre la imagen usando OpenCV.
- Si se proporciona una ruta en savePath, guarda la imagen con los rectángulos.
- Muestra la imagen con matplotlib, convirtiendo el formato BGR (de OpenCV) a RGB.

Resultados

1. Resultados del Modelo

En este análisis, se han procesado tres imágenes distintas para evaluar la eficiencia de un modelo de detección de objetos utilizando tanto CPU como GPU. A continuación, se presenta un resumen de los resultados obtenidos para cada una de las imágenes, detallando el número de objetos detectados y los tiempos de inferencia.

Primera Imagen:



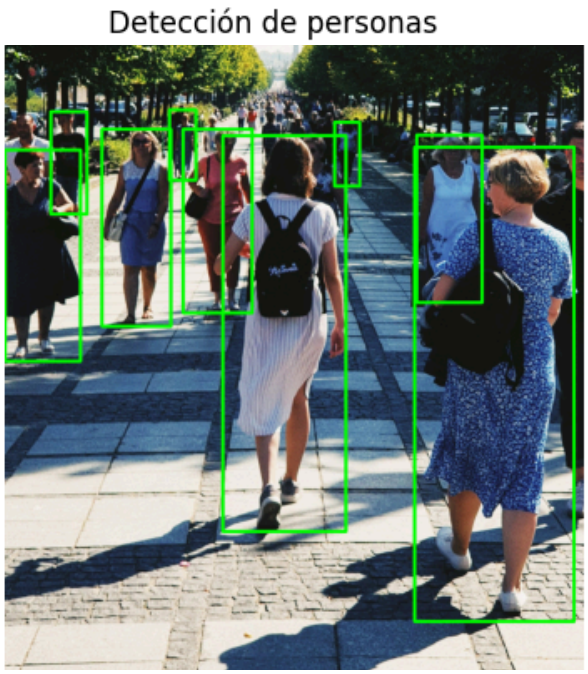

```
----------------------------------------
CPU Prediction

0: 448x640 18 persons, 3 cars, 5 traffic lights, 38.8ms
Speed: 2.1ms preprocess, 38.8ms inference, 1.0ms postprocess per image at shape (1, 3, 448, 640)

----------------------------------------
GPU Prediction

0: 448x640 18 persons, 3 cars, 5 traffic lights, 5.2ms
Speed: 1.6ms preprocess, 5.2ms inference, 1.0ms postprocess per image at shape (1, 3, 448, 640)
```

Segunda Imagen:



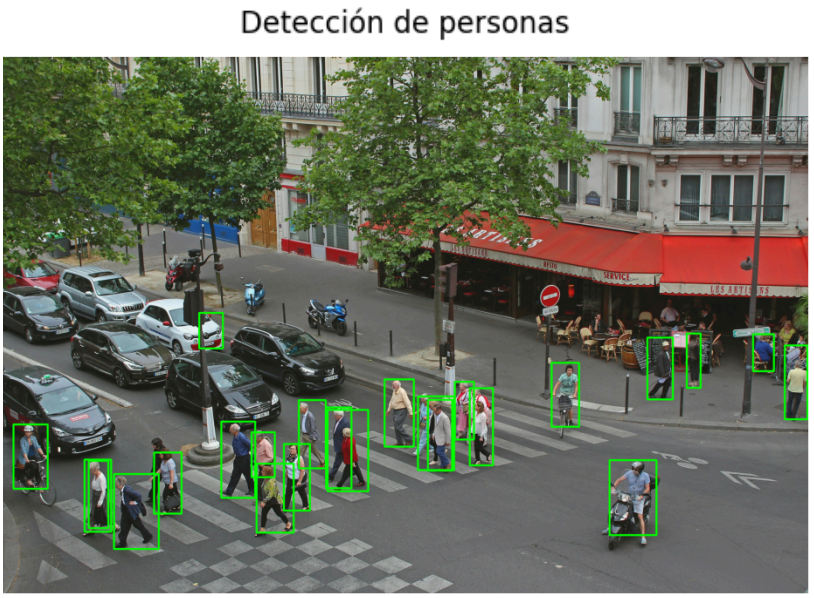

```
----------------------------------------
CPU Prediction

0: 640x384 9 persons, 3 backpacks, 1 handbag, 46.0ms
Speed: 2.2ms preprocess, 46.0ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 384)

----------------------------------------
GPU Prediction

0: 640x384 9 persons, 3 backpacks, 1 handbag, 5.1ms
Speed: 1.5ms preprocess, 5.1ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 384)
```

Tercera Imagen:




```
----------------------------------------
CPU Prediction

0: 448x640 24 persons, 1 bicycle, 5 cars, 2 motorcycles, 1 stop sign, 42.6ms
Speed: 1.6ms preprocess, 42.6ms inference, 0.7ms postprocess per image at shape (1, 3, 448, 640)

----------------------------------------
GPU Prediction

0: 448x640 24 persons, 1 bicycle, 5 cars, 2 motorcycles, 1 stop sign, 5.0ms
Speed: 1.8ms preprocess, 5.0ms inference, 1.0ms postprocess per image at shape (1, 3, 448, 640)
```

2. Análisis y Discusión

2. 1 Precisión - Falsos Negativos y Positivos

En los tres casos, se observa que el modelo identifica con alta precisión la mayoría de los peatones. Sin embargo, también se detectan algunos falsos negativos, es decir, situaciones en las que el modelo no detecta objetos que son visibles:

- **Falsos negativos**: En la tercera imagen, por ejemplo, las personas parcialmente ocultas por estructuras, como aquellos que están comiendo, no son detectadas debido a la oclusión parcial de los objetos y la disminución del área de píxeles útil para la detección. Las personas parcialmente ocultas o a gran distancia (por ejemplo, las que tienen un tamaño pequeño en la imagen) tienen más probabilidades de no ser detectadas como ocurre en la primera imagen y segunda imagen.

- **Falsos positivos**: En la tercera imagen podemos ver señales de falsos positivos. Aunque esto es poco común, el modelo puede identificar patrones correspondientes al de una persona aún cuando eso no es así. Si nos fijamos en el coche blanco de la tercera imagen, podemos ver como el modelo YOLO identifica la parte delantera como una persona.

2.2 Límites de Escala

El modelo muestra limitaciones al procesar objetos de tamaños pequeños. Esto se hace más evidente en el análisis de objetos pequeños, como peatones en plano lejano. En la segunda imagen, por ejemplo, las personas pequeñas (<50px de alto) probablemente no son identificadas correctamente debido a que su tamaño es demasiado pequeño en comparación con el tamaño del modelo de detección.

- **Escala baja**: En imágenes como la segunda imagen (640x384), que contienen objetos de menor tamaño en la escena, se observa que el modelo es menos sensible a los objetos pequeños, como mochilas y bolsos. Esto puede ser una consecuencia directa de la resolución de la imagen y la escala del modelo de detección (el modelo YOLO Nano, que tiene una arquitectura más ligera).

2.3 Ajuste de Umbral

El ajuste del umbral de confianza es un factor importante en la precisión de la detección. Al elevar el umbral de confianza, se reducen los falsos positivos, es decir, las detecciones incorrectas. Sin embargo, esto también puede llevar a una pérdida de detecciones válidas.

2.4 Velocidad de Inferencia

Los tiempos de inferencia en ambos CPU y GPU varían considerablemente dependiendo de la imagen procesada. Se puede observar que **las inferencias en GPU son mucho más rápidas que en CPU**, lo que demuestra la eficiencia del hardware especializado.

- **CPU vs GPU**: En la primera imagen, el modelo tarda 88.1ms en CPU, mientras que en GPU lo hace en solo 65.5ms. Esta diferencia es más pronunciada en la segunda imagen, donde el tiempo de inferencia en GPU es de solo 11.5ms frente a los 63.0ms en CPU. La ventaja del uso de GPU es evidente, especialmente para tareas de inferencia en tiempo real.

2.5. Conclusión

El modelo de detección de objetos probado demuestra ser eficiente en términos de velocidad, especialmente cuando se usa en GPU. Sin embargo, presenta desafíos relacionados con la detección de objetos pequeños y parcialmente ocultos. El ajuste del umbral de confianza puede ayudar a equilibrar la tasa de falsos positivos y falsos negativos, pero siempre con un compromiso en la precisión de las detecciones.

Adquisición de imágenes

Para la práctica elegí varías imágenes urbanas obtenidas desde el sitio **Unsplash** bajo licencia abierta. Intenté seleccionar imágenes diurnas con múltiples peatones, enfocándome en nitidez y variedad de distancias para evaluar el desempeño del modelo.

Reflexión ética

Durante el desarrollo se garantizó el respeto a la privacidad de los individuos presentes en la imagen. La práctica se orientó únicamente a fines académicos y experimentales. Resalto la importancia de usar IA de forma responsable, especialmente cuando se procesan datos sensibles como imágenes de personas en contextos urbanos.

Conclusiones

A través de esta práctica se logró:

- Comprender y aplicar un modelo IA pre-entrenado (YOLOv8) para la detección de personas.
- Manejar imágenes en Python y utilizar librerías como OpenCV y Ultralytics.
- Observar cómo la aceleración por GPU permite una mejora sustancial en tiempos de inferencia.
- Reflexionar sobre el uso ético de la IA en contextos reales.

La experiencia proporciona una base técnica sólida para abordar problemas más avanzados en HPC e IA.