Universitat d'Alacant
Universidad de Alicante

Práctica 0: Introducción Guiada a la Detección de Personas con HPC e IA

1. Introducción

La **Computación de Alto Rendimiento (HPC)** y la **Inteligencia Artificial (IA)** forman una combinación poderosa que permite abordar problemas complejos de forma eficiente. En particular, la **detección de personas en imágenes** es un ejemplo representativo: requiere modelos de IA avanzados y puede beneficiarse enormemente de los recursos de HPC (por ejemplo, GPUs o clusters) para procesar gran cantidad de datos en poco tiempo. Esta práctica introductoria tiene como objetivo principal familiarizar al estudiante con las herramientas y conceptos básicos para detectar personas en imágenes utilizando técnicas modernas de visión por computador, todo ello sirviéndose de la infraestructura de computación de alto rendimiento disponible.

Figura 1. Ejemplo de un cruce urbano concurrido captado en imagen. La detección de peatones en entornos reales, como el que se ilustra, es un desafío donde las técnicas actuales de IA (p. ej. el modelo YOLO) permiten identificar automáticamente a cada persona en la escena.

En esta práctica 0 (de carácter obligatorio) realizarás una **introducción guiada** paso a paso. A lo largo de la misma aprenderás a descargar imágenes de cámaras urbanas o de fuentes abiertas, a procesar esas imágenes con Python, y a aplicar un modelo pre-entrenado de detección de personas (*You Only Look Once* – YOLO, en su versión más reciente YOLOv8). La práctica está diseñada para ser asequible y comprensible, sirviendo de preparación técnica y conceptual para la Práctica 1. Siguiendo las indicaciones, podrás completar las tareas con autonomía creciente, sentando las bases para afrontar la práctica voluntaria siguiente con más confianza.

Asignatura: Computación de Alto Rendimiento (CAR)

Profesor: Ricardo Moreno Rodríguez

Universitat d'Alacant
Universidad de Alicante

Nota sobre el uso responsable de la IA: Durante el desarrollo de esta práctica es importante recordar que el análisis de imágenes que contienen personas conlleva responsabilidades éticas. Asegúrate de **respetar la privacidad** de las personas que puedan aparecer en las fotografías y de cumplir con la normativa de protección de datos vigente. Los sistemas de IA deben usarse de forma transparente y segura; en un contexto urbano real, la detección automatizada de personas no debe derivar en un uso indebido (por ejemplo, vigilancia no autorizada). Esta práctica educativa se realiza con fines académicos y de investigación, fomentando siempre un **uso responsable de la IA**.

2. Objetivos

Los objetivos de esta Práctica 0 son:

- **Familiarizarse con la adquisición de datos visuales urbanos:** aprender a acceder a imágenes provenientes de cámaras de tráfico u otras fuentes públicas que muestran entornos con peatones.
- **Manipular y preprocesar imágenes con Python:** utilizar librerías como OpenCV o PIL para cargar, visualizar y realizar operaciones básicas sobre imágenes, preparando los datos para su análisis.
- **Aplicar un modelo de detección de personas pre-entrenado:** entender el funcionamiento general de un detector como **YOLOv8**, usarlo para identificar personas en una imagen dada (sin requerir entrenar el modelo desde cero) y obtener resultados interpretables (p. ej. imágenes con recuadros alrededor de cada persona detectada).
- **Introducir conceptos de HPC en visión por computador:** reconocer cómo los recursos de computación de alto rendimiento (por ejemplo, la aceleración por GPU) pueden acelerar el procesamiento de imágenes y la detección en tiempo real, sentando bases para experimentos más complejos.
- **Preparar al estudiante para la Práctica 1:** tras completar esta práctica introductoria, el alumno contará con la base técnica necesaria (manejo de imágenes, uso de modelos IA pre-entrenados, consideraciones de rendimiento) para afrontar con mayor autonomía la práctica voluntaria 1, que profundizará en un caso más complejo de HPC e IA.

3. Tareas a Realizar

A continuación se detallan las tareas guiadas que debes completar para llevar a cabo la práctica. Sigue el orden propuesto y las indicaciones en cada paso. Aunque se ofrecen ayudas y ejemplos, procura interpretar los resultados y entender cada acción, ya que esto te preparará para trabajos posteriores más abiertos.

1. Preparación del entorno de trabajo: Asegúrate de tener un entorno Python funcional con las librerías necesarias. Es recomendable usar Jupyter Notebook o un entorno similar donde puedas ejecutar código y visualizar resultados cómodamente. Deberás contar con:

- **Python 3** instalado (idealmente en un entorno virtual o conda para la asignatura).
- Librerías básicas de procesamiento de imágenes como **OpenCV** (`cv2`) o **PIL/Pillow** para cargar y manipular imágenes. OpenCV será especialmente útil para dibujar detecciones sobre las imágenes.
- La biblioteca o script para **YOLOv8 pre-entrenado**. Para simplificar, puedes usar la implementación de Ultralytics: `pip install ultralytics`. Esta librería proporciona una interfaz sencilla para cargar modelos YOLO pre-entrenados (por ejemplo, `yolov8n.pt` o `yolov8s.pt`, que son versiones pequeñas del modelo entrenadas en el conjunto COCO). *Nota:* Si no puedes instalar nuevas librerías en local, considera usar Google Colab o los recursos HPC de la universidad, donde podrás acceder a GPUs y ya hay entornos preparados con estas herramientas.

Asegúrate de tener acceso a Internet si vas a descargar librerías o pesos pre-entrenados la primera vez. Consulta con el profesor o los técnicos de laboratorio si necesitas ayuda para configurar el entorno HPC con los paquetes necesarios.

2. Obtención de imágenes desde cámaras urbanas: El primer paso práctico es conseguir una o varias imágenes donde haya personas en un entorno urbano (calles, plazas, transporte, etc.). Se sugieren dos opciones principales:

- **a. Imágenes de cámaras de tráfico en tiempo real:** Muchas ciudades ofrecen acceso público a imágenes actualizadas de sus cámaras de tráfico. Por ejemplo, el Ayuntamiento de Madrid y la DGT disponen de portales donde se pueden ver fotografías actualizadas cada pocos minutos de distintas ubicaciones. Busca en Internet si tu ciudad local o alguna ciudad de interés tiene un portal de *"cámaras de tráfico"* o *"tráfico en directo"*. Suelen ser imágenes JPEG accesibles vía URL. *Consejo:* Si encuentras la URL directa de la imagen (por ejemplo, terminada en `.jpg`), puedes descargarla programáticamente con Python (usando `requests.get` o `urllib`) para incorporarla a tu entorno de trabajo. En esta práctica **no necesitas automatizar una descarga continua**, basta con obtener una instantánea estática para analizar.
- **b. Imágenes abiertas o de ejemplo:** Si no logras obtener una imagen de cámara de tráfico real, puedes usar imágenes de repositorios abiertos. Por ejemplo, sitios como Pixabay, Unsplash u OpenImageDataset ofrecen fotografías urbanas con transeúntes bajo licencias abiertas. También puedes tomar una fotografía tú mismo en la vía pública (asegurándote de respetar la privacidad y de que sea un lugar público). Lo importante es que en la imagen haya **personas claramente visibles** (de cuerpo completo o al menos de la cintura para arriba) para que el detector las pueda encontrar. Idealmente, elige imágenes de día y relativamente nítidas, para no complicar en exceso la detección en este primer intento.

Una vez elegida la fuente, **descarga la imagen** y guárdala en tu directorio de trabajo con un nombre reconocible (por ejemplo, `imagen_urbana.jpg`). Si vas a probar con varias imágenes, organízalas en una carpeta o con nombres numerados para facilitar luego la iteración.

Ayuda: Para comprobar que la imagen se ha obtenido correctamente, puedes abrirla manualmente (por ejemplo, haciendo doble click en el archivo descargado) o insertar una celda de código en tu notebook que la muestre. Con OpenCV podrías hacer: `img = cv2.imread('imagen_urbana.jpg')` seguido de `cv2.imshow('Vista previa', img)` (en entornos locales con interfaz gráfica) o usar funciones de matplotlib/Notebook (`plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))`) para visualizarla inline. Asegúrate de que la imagen se ve correctamente y tiene las dimensiones esperadas.

3. Carga y preprocesamiento de la imagen en Python: Ahora procederemos a trabajar con la imagen desde código. Los subpasos recomendados son:

- **Lectura de la imagen:** Utiliza OpenCV (`cv2.imread`) o PIL (`Image.open`) para cargar la imagen desde el archivo. Verifica las dimensiones (ancho, alto) imprimiendo `img.shape` si usas OpenCV, para confirmar que no está vacía y se leyó bien.
- **Conversión de color (si es necesaria):** Ten en cuenta que OpenCV carga las imágenes en formato BGR (canales azul y rojo invertidos respecto al estándar RGB). Si vas a mostrar la imagen con matplotlib u otra librería, conviene convertirla a RGB con `cv2.cvtColor(img, cv2.COLOR_BGR2RGB)`. Este detalle no afecta a la detección en sí, pero sí a la visualización con colores correctos.
- **Redimensionado u otras transformaciones (opcional):** Para esta práctica introductoria no es estrictamente necesario cambiar el tamaño ni aplicar filtros a la imagen. No obstante, ten presente que los modelos de detección suelen requerir que la imagen de entrada tenga cierta resolución o incluso ser cuadrados de tamaño fijo. En el caso de YOLOv8, la propia librería se encarga de escalar la imagen de entrada al tamaño de red de la IA (por defecto 640x640 píxeles) conservando la relación de aspecto. Puedes imprimir el tamaño original de la imagen; si es muy grande (por ejemplo, 4000x3000 px de una cámara HD), el procesamiento será más lento. Por ahora no recortes nada (no queremos perder personas en la escena), pero mentalmente observa si la imagen contiene muchas áreas irrelevantes (cielo, edificios lejanos) ya que eso influirá en la rapidez y resultados de detección.
- **Visualización rápida (opcional pero recomendable):** Muestra la imagen en el notebook para confirmar que todo está correcto antes de aplicar el modelo. Esto te asegurará que no haya problemas con la lectura de datos. Además, así podrás identificar a simple vista cuántas personas *esperas* que sean detectadas por la IA, lo cual luego podrás comparar con el resultado obtenido.

4. Introducción al modelo YOLOv8 y configuración: YOLO (*You Only Look Once*) es una familia de modelos de detección de objetos en imágenes que se caracterizan por su alta velocidad y buena precisión. En esta práctica usaremos **YOLOv8**, la última versión publicada por la comunidad Ultralytics. No te preocupes por entrenar un modelo desde cero: utilizaremos un modelo YA ENTRENADO en un amplio conjunto de datos (COCO, que incluye la clase "personas" entre 80 tipos de objetos). Esto significa que simplemente cargándolo podremos buscar personas en nuestra imagen sin más entrenamiento.

- **Carga del modelo pre-entrenado:** Gracias a la librería Ultralytics YOLO, la carga es sencilla. Por ejemplo, en tu código Python puedes hacer:

```
from ultralytics import YOLO
model = YOLO('yolov8n.pt')  # también puedes usar 'yolov8s.pt' para mayor precisión a costa de algo más de tiempo
```

- La primera vez que ejecutes esto, si no tienes el archivo de pesos `yolov8n.pt` en tu sistema, la librería lo descargará automáticamente (asegúrate de tener conexión o de haber colocado previamente el archivo en el directorio). El sufijo `n` se refiere a **Nano** (un modelo muy ligero). Existe también `yolov8s.pt` (**Small**), `yolov8m.pt` (Medium), `yolov8l.pt` (Large) y `yolov8x.pt` (Extra Large). Para nuestras pruebas iniciales, YOLOv8-Nano o Small son adecuados ya que son rápidos incluso sin GPU.

- **Configuración adicional (opcional):** Por defecto, el modelo cargado sabe detectar múltiples clases de objetos (personas, coches, semáforos, etc.). En esta práctica nos centraremos en las **personas**, pero no hace falta cambiar nada en el modelo porque luego filtraremos resultados. Asegúrate de que el modelo está en modo *inferencia* (por defecto lo está, mientras no llames a `.train()`). También puedes establecer parámetros como el umbral de confianza si lo deseas. Por ejemplo, `results = model.predict(img, conf=0.5)` usaría 0.5 como umbral de confianza mínimo en lugar del 0.25 predeterminado. En un primer intento, no es necesario ajustar esto; usa los valores por defecto y luego, si ves que detecta muchos *falsos positivos* (objetos marcados como persona que claramente no lo son), podrías subir el umbral.

- **Comprensión básica de la salida del modelo:** Cuando ejecutamos la predicción, por ejemplo:

```
results = model.predict(source=img)  # también podrías pasar directamente el path del archivo
```

- la variable `results` contendrá la información de detección. En la implementación de Ultralytics, `results` es una lista de objetos result (uno por cada imagen

procesada; en nuestro caso lista de longitud 1). Puedes acceder a `results[0].boxes` para obtener las cajas delimitadoras detectadas. Cada *box* tiene coordenadas `(x_min, y_min, x_max, y_max)` en píxeles sobre la imagen original, además de una etiqueta de clase (por ejemplo, "person") y una confianza. Ultralytics YOLO facilita métodos para dibujar o exportar estas detecciones (p.ej. `results[0].plot()` devuelve una imagen con las detecciones dibujadas). En el siguiente paso utilizaremos esta información para marcar las personas en la imagen.

5. Detección de personas en la imagen: ¡Ha llegado el momento de ejecutar el detector! Usando el modelo YOLOv8 cargado en el paso anterior, aplicarlo a la imagen y obtener resultados:

- **Ejecución de la inferencia:** Como se mostró, con `results = model.predict(source='imagen_urbana.jpg')` (o pasando directamente el array de imagen) obtenemos las detecciones. Esta llamada realizará internamente varias operaciones: escalado de la imagen a la dimensión de entrada de la red (p. ej. 640x640), inferencia mediante la red neuronal convolucional para producir predicciones de cajas, filtrado por confianza y no-maxima supresión para quitar duplicados, etc. No es necesario profundizar ahora en cada etapa, pero ten en cuenta que, gracias a HPC, todo esto puede ocurrir en fracciones de segundo para una imagen si se usa hardware adecuado.
- **Revisión de los resultados crudos:** Para entender lo obtenido, imprime o inspecciona `results[0]`. Dependiendo de la biblioteca, podrías hacer `print(results[0].boxes)` o incluso iterar:

```
for box in results[0].boxes:
  print(box.xyxy, box.conf, box.cls)
```

- Esto listaría las coordenadas de cada caja, la confianza y el índice de clase. El índice de clase para "person" en COCO suele ser 0 (siendo 0=persona, 1=bicycle, 2=car, etc.), pero la librería también ofrece `box.name` o similar para dar la etiqueta directamente. Comprueba cuántas detecciones de clase "person" aparecen y con qué confianza. Idealmente, debería detectar a la mayoría de los peatones visibles en la imagen.

- **Filtrado por clase (opcional):** Si el modelo detectó otros objetos (por ejemplo, coches, semáforos) y no te interesan para esta práctica, puedes filtrar las cajas para quedarte solo con las personas. Por ejemplo:

```
person_boxes = [b for b in results[0].boxes if int(b.cls) == 0]
```

- asumiendo que la clase 0 es person. Así trabajarás solo con esas. En muchos casos no hará falta filtrar manualmente porque en la propia imagen de ejemplo los únicos objetos

destacables son personas, pero tenlo en cuenta si usaste una imagen donde salen coches u otros elementos prominentes.

- **Dibujo de las detecciones sobre la imagen:** Una vez tenemos las coordenadas de las cajas de personas detectadas, el siguiente paso es visualizar el resultado marcándolas. Usando OpenCV, puedes dibujar rectángulos:

```
for box in person_boxes:
  x1, y1, x2, y2 = map(int, box.xyxy[0])  # coordenadas de esquina (asegurando que sean enteros)
  cv2.rectangle(img, (x1, y1), (x2, y2), (0,255,0), 2)
```

- Esto dibujará un rectángulo verde de grosor 2 en la imagen `img` alrededor de cada detección de persona. También podrías agregar el texto "Persona" encima de cada recuadro usando `cv2.putText`. En esta práctica no es obligatorio etiquetar con el texto, pero puede ser útil si hubiera múltiples clases; como aquí solo nos interesan personas, el recuadro ya es suficientemente informativo.
- **Visualizar/guardar el resultado:** Muestra la imagen resultante con los recuadros. Ahora deberías ver a cada persona rodeada por un rectángulo. ¿Corresponde con lo que esperabas visualmente? Es posible que el modelo detecte incluso personas que están parcialmente visibles (p.ej. alguien caminando a medias fuera del encuadre) o muy pequeñas a lo lejos; y también puede que alguna persona no haya sido detectada debido a ángulos difíciles, oclusiones o movimiento. Toma nota mental de estos casos. Guarda la imagen anotada resultante, por ejemplo: `cv2.imwrite('resultado_deteccion.jpg', img)`. Este archivo podrás incluirlo en tu informe.

Ayuda: La librería de Ultralytics YOLO tiene un método más directo para guardar resultados: si llamaste a `model.predict(..., save=True)` te creará automáticamente una carpeta `runs/detect/predict` con la imagen marcada. Sin embargo, es instructivo dibujarlo manualmente al menos una vez para entender el formato de salida.

6. Experimentación adicional (opcional, para profundizar): Hasta aquí habrás logrado realizar una detección básica de personas en una imagen con ayuda de HPC/IA. Si aún cuentas con tiempo y curiosidad, puedes ampliar un poco el experimento:

- **Probar con más imágenes:** Intenta con 2-3 imágenes diferentes (por ejemplo, distintas ubicaciones urbanas, o distintos niveles de ocupación de personas). Puedes automatizar que el código procese una lista de archivos y genere salidas para cada uno. Observa cómo se comporta el detector en cada caso: ¿falla más cuando las personas están muy lejos? ¿o si hay poca iluminación? ¿detecta a

personas en movimiento borrosas? Estas observaciones te darán intuición sobre las capacidades y límites de los modelos pre-entrenados.

- **Medir tiempos de ejecución:** Un aspecto importante en HPC es el rendimiento. Puedes usar la función `time()` del módulo `time` o `%timeit` en Jupyter para medir cuánto tarda la llamada `model.predict` en procesar una imagen. Prueba con y sin GPU (si tu entorno HPC dispone de GPU, por ejemplo en Colab puedes activar GPU desde Runtime > Change runtime type). Compara los tiempos: probablemente en CPU tarde del orden de varios cientos de milisegundos o incluso más de 1 segundo en imágenes grandes, mientras que en GPU pueda ser varias veces más rápido. Documenta esta comparación si la haces, pues es un **punto clave de la motivación HPC**: acelerar la inferencia de IA.
- **Ajustar parámetros del detector:** Podrías intentar bajar el umbral de confianza para ver si detecta más personas (a costa de más falsos positivos) o subirlo para que sea más estricto. Incluso podrías probar otro modelo YOLOv8 de distinto tamaño (por ejemplo, `yolov8m.pt`) y comparar si detecta alguna persona que el modelo más pequeño pasó por alto. Ten en cuenta que los modelos más grandes requieren más recursos; en un entorno HPC con buena GPU podrías hacerlo sin problemas.
- **Explorar otros enfoques o librerías:** OpenCV, por ejemplo, tiene clasificadores pre-entrenados como el detector de personas basado en HOG + SVM (`cv2.HOGDescriptor_getDefaultPeopleDetector`). No es tan preciso como YOLOv8, pero podrías probarlo para comparar resultados. Sin embargo, esto es completamente opcional y solo con fines de conocimiento adicional.

7. Conclusiones y cierre: Para finalizar la práctica, reflexiona brevemente sobre lo realizado. Asegúrate de **responder a las preguntas planteadas** (si en el enunciado se solicita alguna respuesta específica, por ejemplo, indicar cuántas personas se detectaron en tu imagen, o comentar qué dificultades surgieron).

Resume qué aprendiste: por ejemplo, "cómo leer imágenes con Python, cómo funciona un modelo pre-entrenado de detección de personas, y la importancia de la potencia de cálculo para procesar imágenes rápidamente". Si realizaste las partes opcionales, puedes mencionar hallazgos interesantes (p.ej., "el modelo detectó 9 de 10 personas, fallando en una muy oculta detrás de un objeto"). Esta sección de conclusiones te ayudará a consolidar el conocimiento y será útil para enlazar con la Práctica 1.

Finalmente, recuerda guardar todos tus resultados (código fuente, imágenes originales y resultantes, gráficas de tiempos si las hiciste). Procede según las indicaciones de entrega para enviar tu trabajo.

4. Entrega y Evaluación

Forma de entrega: Al tratarse de una práctica obligatoria, deberás entregar tu trabajo para evaluación. La entrega consistirá en un **informe breve en formato PDF** junto con el código empleado:

- El **informe** debe incluir una descripción concisa de cada paso realizado, con capturas de pantalla o imágenes resultantes de las detecciones. No hace falta que sea extenso; con 2 o 3 páginas bien organizadas suele ser suficiente. Asegúrate de incluir al menos una imagen donde se vean las personas detectadas con sus recuadros (resultado final). También comenta brevemente los resultados: por ejemplo, cuántas personas hay en la imagen vs cuántas detectó el modelo, y posibles razones de aciertos/errores. Incluye tu nombre, grupo y título de la práctica en la portada del informe.
- El **código fuente** usado (script .py, notebook .ipynb, etc.) debe adjuntarse. Si utilizaste un notebook, puedes exportarlo a HTML o PDF aparte, pero es importante que el texto principal esté en el informe escrito de manera coherente (no solo código comentado). El profesor podría ejecutar tu código para reproducir resultados, así que procura que esté limpio y funcione (puedes comentar partes que sean costosas, indicando los resultados obtenidos en vez de ejecutarlas de nuevo).
- Sube ambos elementos (informe y código) a la plataforma Moodle antes de la fecha límite establecida.

Criterios de evaluación: A continuación se presenta la rúbrica de evaluación con los criterios que se tendrán en cuenta y su ponderación. La puntuación máxima total es de **10 puntos**. Para superar la práctica se requiere una comprensión básica de cada apartado y un funcionamiento correcto del proceso de detección implementado.

Universitat d'Alacant
Universidad de Alicante

Rúbrica de evaluación de la práctica

Criterio	Descripción	Nivel básico (0–0,5 pt)	Nivel intermedio (0,6–1 pt)	Nivel excelente (1,1–1,5 pt) (*requiere elaboración personal clara*)
Informe técnico	Claridad, estructura y calidad del documento entregado.	Entrega incompleta, desorganizada o confusa.	Informe comprensible y estructurado correctamente.	Redacción clara y ordenada, con decisiones explicadas de forma personal (se aprecia que el alumno ha vivido el proceso).
Tratamiento de la imagen	Elección, obtención y lectura de imágenes urbanas.	Imagen incorrecta o sin justificar.	Imagen válida con tratamiento básico.	Imagen adecuada, con criterios argumentados y justificación técnica propia.
Aplicación del modelo YOLOv8	Uso funcional del modelo de detección y su comprensión.	Modelo incorrecto o sin resultados.	Aplicación funcional sin análisis profundo.	Aplicación correcta, con interpretación razonada del funcionamiento y decisiones técnicas.
Análisis e interpretación	Reflexión crítica sobre los resultados obtenidos.	Reflexión mínima o generalista.	Análisis correcto con observaciones relevantes.	Análisis detallado, juicio técnico y conexión con contexto real.
Código y replicabilidad	Calidad técnica del código y posibilidad de reproducir resultados.	Código incompleto o desordenado.	Código funcional con estructura básica.	Código claro y comentado que demuestra comprensión del proceso.
Elaboración personal y uso crítico de IA	Grado de reelaboración propia y reflexión ética en el uso de herramientas externas.	Redacción impersonal o dependiente de IA.	Trabajo correcto pero genérico.	Trabajo reflexivo, adaptado y con uso responsable de herramientas.

Comentarios clave sobre los criterios de evaluación y el uso responsable de la IA

La rúbrica anterior incluye no solo una valoración técnica del trabajo entregado, sino también una valoración transversal del pensamiento personal, la reflexión crítica y la elaboración autónoma del estudiante. Este enfoque se ha adoptado porque consideramos que en el proceso de aprendizaje universitario, especialmente en una asignatura como Computación de Alto Rendimiento, el verdadero valor no está únicamente en obtener un resultado funcional, sino en cómo se ha llegado a él.

Por ello, para alcanzar la máxima puntuación en cada apartado no basta con presentar un trabajo formalmente correcto o funcional, sino que debe reflejarse que el alumno ha comprendido lo que hace, ha razonado sus decisiones, ha analizado críticamente los resultados y ha aportado su perspectiva. Esto implica que la presentación de resultados debe estar personalizada, justificada y conectada con la experiencia directa del estudiante al realizar la práctica.

Asignatura: Computación de Alto Rendimiento (CAR)

Profesor: Ricardo Moreno Rodríguez

¿Qué entendemos por uso responsable de la inteligencia artificial?

En este contexto académico, el uso responsable de herramientas de IA como asistentes de redacción, generación de código o generación de contenidos gráficos se define así:

- **Es responsable cuando acompaña al proceso de aprendizaje**, ayudando al estudiante a comprender mejor los conceptos y a producir sus propios resultados razonados.
- Es responsable cuando, una vez adquirido el conocimiento y la competencia propia, el estudiante utiliza herramientas de IA **para mejorar la presentación o comunicación de su trabajo personal**.

Por el contrario, no es responsable el uso de IA para sustituir la reflexión y el aprendizaje personal, ni para generar entregas sin comprensión real de su contenido. Un trabajo generado por IA sin reelaboración humana, sin justificación de decisiones ni reflexión técnica, podrá ser formalmente correcto pero académicamente insuficiente, y **será evaluado en consecuencia con una calificación limitada**.

Objetivo formativo: evitar la sustitución del pensamiento propio

El objetivo de esta política no es restringir el uso de herramientas modernas, sino asegurar que su utilización no sustituya el proceso formativo. Durante el periodo de aprendizaje, lo más importante es que el estudiante desarrolle su pensamiento propio, practique el análisis crítico y sea capaz de generar ideas, soluciones y conclusiones a partir de lo aprendido.

5. Contenidos y Competencias trabajadas

Esta práctica de introducción abarca varios **contenidos conceptuales** de la asignatura y permite desarrollar **competencias** del plan de estudios, conectando la teoría de HPC con una aplicación práctica de IA:

Contenidos (según plan docente y CENAE) trabajados:

- **Uso de la computación de altas prestaciones en inteligencia artificial:** se experimenta con un modelo de IA (YOLO) que puede aprovechar aceleración por GPU, ilustrando cómo HPC permite **acelerar el procesamiento de imágenes** para detección en tiempo real.
- **Procesamiento y análisis de datos visuales a gran escala:** se cubre la lectura y manipulación de **imágenes** (un tipo de dato frecuentemente masivo) y la aplicación de un modelo para extraer información (personas detectadas), sentando base para escalarlo a conjuntos mayores o secuencias de vídeo.
- **Aplicación de modelos pre-entrenados y transferencia de aprendizaje:** aunque no se entrena un modelo desde cero, el uso de YOLOv8 pre-entrenado ejemplifica el concepto de **transferir un modelo existente a un problema concreto**, un contenido relevante en IA de alto rendimiento.

- **Consideraciones de rendimiento y escalabilidad:** mediante la medición opcional de tiempos en CPU vs GPU y la discusión, se introduce la idea de **escalabilidad** (¿qué pasa si tuviéramos 1000 imágenes o un vídeo 24 fps? cómo HPC ayuda a escalar) y de eficiencia en el uso de recursos computacionales.
- **Aspectos éticos y sociales en sistemas HPC/IA:** la práctica incluye la reflexión sobre la **privacidad y el uso responsable** de sistemas de visión en ciudades inteligentes, alineado con el estudio de los retos actuales en computación de alto rendimiento aplicada (ética, privacidad, impacto social).

Competencias desarrolladas:

- **CE15 – Seleccionar y aplicar sistemas computacionales de altas prestaciones** para acelerar la ejecución de métodos y algoritmos de IA. (En la práctica, el estudiante identifica la utilidad de GPUs/ HPC para la tarea de detección y aprende a usar un modelo optimizado).
- **CG1 – Analizar, diseñar e implementar soluciones completas de IA** utilizando las infraestructuras, tecnologías y herramientas adecuadas. (La práctica guía por todo el flujo: obtención de datos, aplicación de modelo IA, interpretación de resultados con la herramienta apropiada, integrando conocimientos de diferentes áreas).
- **CG3 – Mantenerse actualizado en los últimos avances de IA** y determinar su aplicabilidad para resolver problemas. (YOLOv8 es un modelo de vanguardia en visión; al usarlo, el alumno toma contacto con un avance reciente y evalúa cómo aplicarlo a un caso concreto).
- **CG8 – Concebir y desarrollar sistemas de IA** atendiendo a criterios de calidad, seguridad, uso **ético y eficiente** de recursos e información. (Se trabaja la eficiencia vía HPC y se hace hincapié en el uso ético de la información visual de personas).
- **CT02 – Capacidad de comunicar de forma clara y profesional** soluciones técnicas. (Mediante la elaboración del informe de la práctica, explicando el procedimiento y resultados de forma comprensible).
- **CB2 – Aplicar los conocimientos a la práctica** en contextos reales y resolver problemas en su área de estudio. (La tarea de detectar personas en imágenes urbanas refleja un problema real de "smart cities"; el estudiante aplica conocimientos de programación, IA y HPC para resolverlo, integrando teoría y práctica).

¡Enhorabuena! Con la finalización de esta Práctica 0, habrás dado tus primeros pasos en el uso combinado de HPC e IA para abordar un problema de visión por computador. Estos fundamentos te servirán de base para emprender desafíos más complejos en la **Práctica 1** y futuras actividades de la asignatura. Recuerda que el verdadero poder de la Computación de Alto Rendimiento se manifiesta cuando escalamos nuestros experimentos: más datos, modelos más grandes, requerimientos de tiempo real… estás comenzando a adquirir las habilidades para enfrentarte a esos retos.