UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES

CARRERA: COMPUTACIÓN		**ASIGNATURA**: VISIÓN POR COMPUTADOR
NRO. PRÁCTICA:	4-1	**TÍTULO PRÁCTICA**: Clasificación de Objetos Usando Técnicas de Deep Learning: fine tuning empleando YOLO v12 y ejecución de redes detección de objetos y super resolución en unidades gráficas de procesamiento (GPU)

OBJETIVO:
Reforzar los conocimientos adquiridos en clase sobre detección de objetos usando deep learning, mejora de imágenes a través de redes neuronales para super resolución y su ejecución en unidades gráficas de procesamiento en tarjetas NVIDIA con CUDA.

INSTRUCCIONES:	**1.** Revisar el contenido teórico del tema
	2. Profundizar los conocimientos revisando los libros guías, los enlaces contenidos en los objetos de aprendizaje y la documentación disponible en fuentes académicas en línea
	3. Deberá desarrollar un cuaderno en Google Colab® o similar para realizar la detección de objetos usando técnicas de deep learning a través de aprendizaje por transferencia de conocimiento en la red YOLOv11.
	4. Deberá probar dos redes de aprendizaje profundo, una para detectar objetos en vídeo y otra para aplicar super resolución considerando la ejecución sobre unidades gráficas de procesamiento (GPU) con CUDA.

ACTIVIDADES POR DESARROLLAR

PARTE 1A (individual o en parejas)

- Emplear una red YOLOv11 o YOLOv12 (como se ha visto en clase) o similar a la que se haya realizado un proceso de ***transfer learning*** para **segmentar** objetos de imágenes capturadas a través de una webcam. Deberá realizar pruebas de rendimiento usando GPUs vs CPU (para ello puede emplear las computadoras del laboratorio de Cómputo 8). **El entrenamiento y validación de resultados de la red puede hacerlo en Google Colab o en cualquier entorno de su preferencia**. Por ejemplo, podría realizar una tarea similar a la segmentación de personas como se describe en el cuaderno de Kaggle ("Human Segmentation Dataset - TikTok Dances", ver Ilustración 1). Se sugiere emplear imágenes que se puedan encontrar en repositorios para realizar la tarea. **No puede haber ningún grupo que tenga las mismas imágenes ni usar un código previamente desarrollado en Kaggle u otra plataforma similar.**

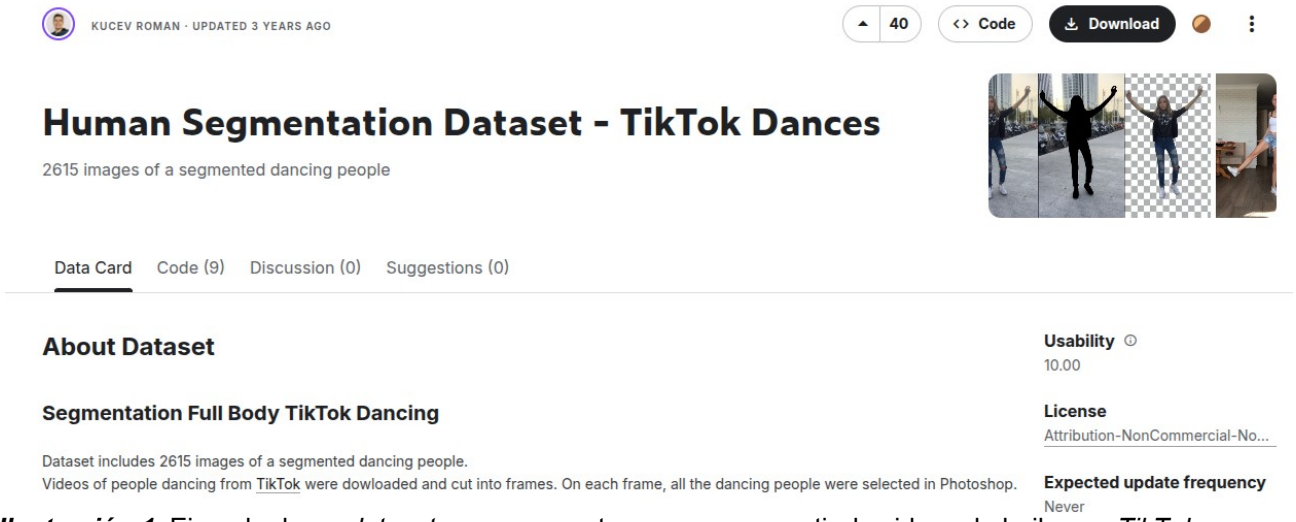
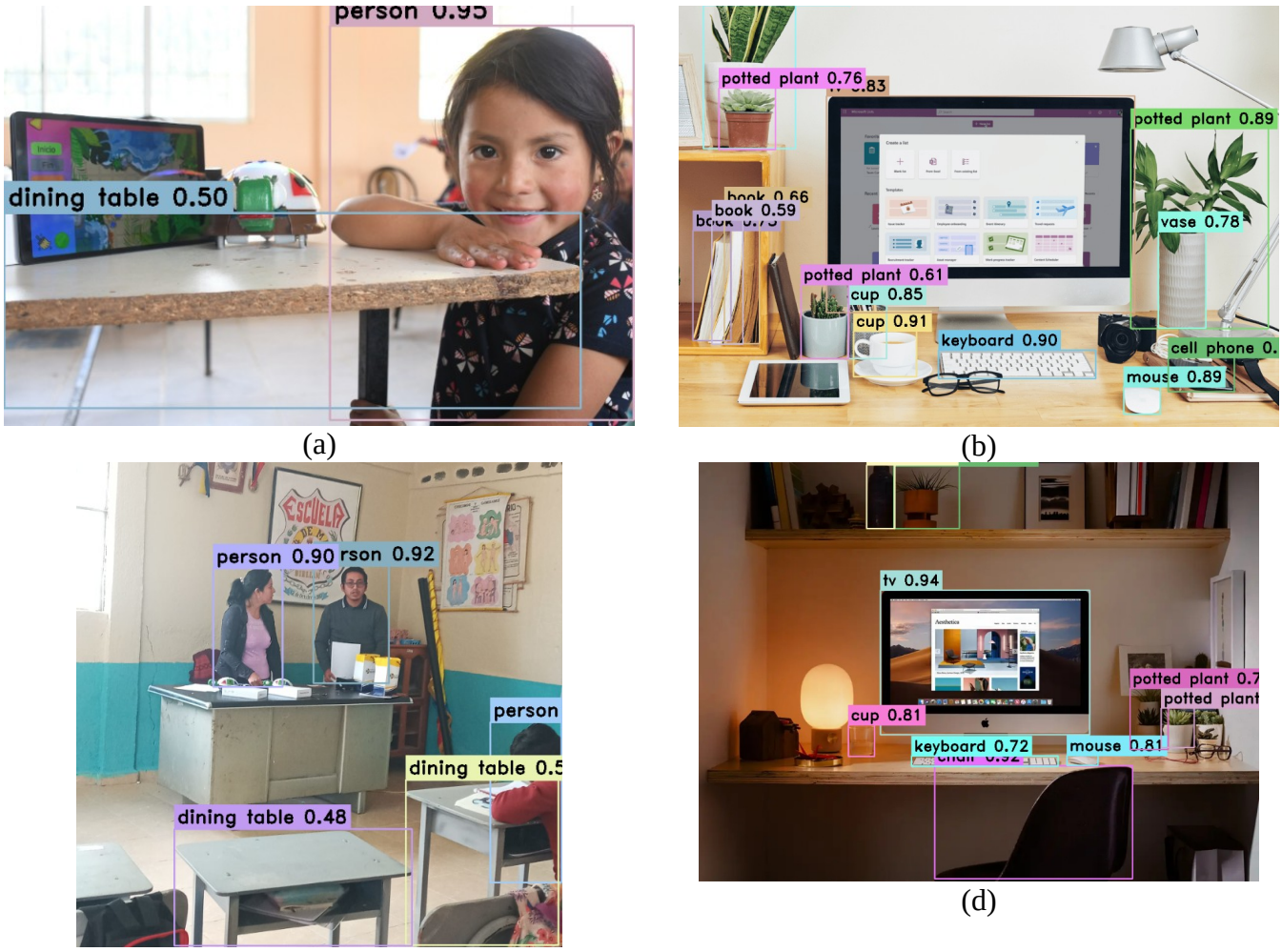


Ilustración 1. Ejemplo de un *dataset* para segmentar personas a partir de videos de bailes en *TikTok*.

- Por otra parte, como podemos apreciar en la Ilustración 2, se ha realizado un proceso de detección de objetos usando la red neuronal YOLOv8l (versión large) en diferentes imágenes:

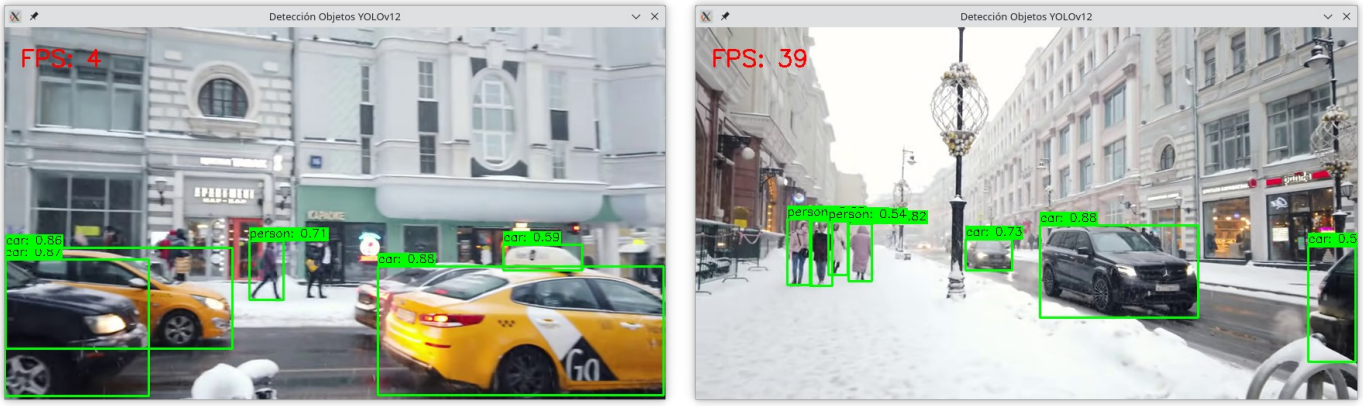


(a) (b) (c) (d)

Ilustración 2. Ejemplos de reconocimiento de objetos usando YOLO8l en una imagen de una niña (a), en imágenes de escritorios (b) y (d) y en un aula en Biblián (c).

PARTE 1B (individual)

- Realizar pruebas de rendimiento con dos redes neuronales: **(a) YOLOv11 o YOLOv12** (como se ha visto en clase) y **(b) una red para aplicar super resolución en vídeo** (**actualizada de no más de 1 año de existencia**), comparando GPUs vs CPU (para ello debe emplear las computadoras del laboratorio de Cómputo 8). Para ello, deberá visualizar la cantidad de Frames por Segundo que se tiene en CPU frente a GPU (ver Ilustración 2), como se explica en el ejemplo visto en clase (vídeo y código adjunto):



(a) (b)

Ilustración 3. Ejemplos de detección de objetos usando la red neuronal YOLOv12 en CPU (a) y en GPU (b) de un video de recorrido por las calles de Moscú.

- Cada estudiante deberá usar un computador distinto (con una Mac Address Única)
- Ningún estududiante puede
- Deberá grabar un vídeo del resultado que se obtiene tanto en CPU como en GPU y mostrar la siguiente información:
  - Uso de la memoria con el comando ***nvidia-smi***
  - Número de FPS (*frames per second*) en GPU vs CPU
  - Uso de memoria RAM en GPU vs CPU
  - Mac Address del Computador

PARTE 1C (individual)

- Debe realizar un código en OpenCV C++ para preprocesamiento de imágenes en CPU vs GPU. Para ello, deberá realizar las siguientes operaciones de cualquier ejemplo que desee (aplicación de imágenes médicas, detección de bordes, etc.):
  - Suavizado (Filtro Gaussiano)
  - Operaciones morfológicas (Erosión / Dilatación)
  - Detección de bordes (Canny)
  - Ecualización del histograma
- Por ejemplo, para ejecutar estas operaciones (como se indicó en clase), OpenCV implementa una librería para que el código corra sobre GPU (Tabla 1):

Tabla 1: Aplicación del filtro Gaussiano en CPU vs GPU.

CPU:	***GPU***:

cv::Mat blurred_cpu;
cv::GaussianBlur(frame, blurred_cpu, cv::Size(5, 5), 1.5);
	cv::cuda::GpuMat d_frame, d_blurred;
	d_frame.upload(frame);
	auto gaussian = cv::cuda::createGaussianFilter(d_frame.type(), d_frame.type(), cv::Size(5, 5), 1.5);
	gaussian->apply(d_frame, d_blurred);

- Deberá realizar un análisis de resultados, considerando lo siguiente:
  - Comparación cualitativa de los resultados visuales.
  - Comparación cuantitativa (tiempo de procesamiento por frame).
  - Reflexión: ¿cuál es la diferencia entre pipeline CPU ↔ GPU y Pipeline GPU-only? y ¿cuándo vale la pena usar la GPU?

- Considerar que debe usar Pipeline GPU-only para realizar esta tarea y debe incluir en el informe capturas de pantalla.

Tabla 2: Ejemplo de uso de pipeline GPU-only (eficiente)

```
cv::cuda::GpuMat d_frame, d_gray, d_blur, d_edges;

cap.read(frame);                // CPU
d_frame.upload(frame);          // CPU → GPU

cv::cuda::cvtColor(d_frame, d_gray, cv::COLOR_BGR2GRAY); // GPU
auto gauss = cv::cuda::createGaussianFilter(d_gray.type(), d_gray.type(), cv::Size(5,5), 1.5);
gauss->apply(d_gray, d_blur);                            // GPU

auto canny = cv::cuda::createCannyEdgeDetector(50, 150);
canny->detect(d_blur, d_edges);                          // GPU

cv::Mat result;
d_edges.download(result);       // Solo al final
cv::imshow("GPU Result", result);
```

Documentación de Soporte:

- El enlace donde se encuentra el código de ejemplo para crear aplicaciones que usan YOLOv5 y YOLOv8 es el que se detalla a continuación:
  - https://github.com/vlarobbyk/yolov5-and-yolov8-object-detection-OpenCV-C-
- Tutorial "**Object Detection using YOLOv5 OpenCV DNN in C++ and Python**" de LearnOpenCV: https://learnopencv.com/object-detection-using-yolov5-and-opencv-dnn-in-c-and-python/
- Artículo Científico "**A fine-tuned YOLOv5 deep learning approach for real-time house number detection**": https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10403189/
- Tutorial "**Intersection over Union (IoU) for object detection**": https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
- Vídeo tutorial (YouTube) "Train Yolov8 object detection on a custom dataset": https://www.youtube.com/watch?v=m9fH9OWn8YM

RESULTADO(S) OBTENIDO(S):

Al finalizar esta práctica, los estudiantes lograron aplicar de forma efectiva técnicas avanzadas de **detección de objetos mediante redes neuronales profundas**, tales como **YOLOv11** y **MobileNetv3**, así como redes para **super resolución** y preprocesamiento de imágenes con **aceleración GPU utilizando CUDA y OpenCV**. Fueron capaces de realizar un proceso de **fine-tuning** sobre datasets personalizados, implementaron mecanismos para evaluar el rendimiento entre CPU y GPU (FPS, uso de memoria y eficiencia), y desarrollaron tanto scripts en Python (Colab) como módulos nativos en C++ para pruebas comparativas. Asimismo, se evidenció una adecuada comprensión de los conceptos de **pipeline GPU-only**, optimización computacional y diseño de flujos eficientes en tiempo real, aplicados en tareas reales como clasificación de alfabetos extranjeros y procesamiento de vídeo en alta resolución.

CONCLUSIONES:

Esta práctica permite consolidar los conocimientos teóricos y prácticos adquiridos durante el curso de

Visión por Computador, integrando componentes esenciales de **aprendizaje profundo, visión artificial y procesamiento acelerado en GPU**. Se puede comprobar que el uso de GPUs modernas (con soporte CUDA) permite mejorar significativamente el rendimiento en tareas de detección y preprocesamiento de imágenes, logrando hasta 5x–10x más velocidad comparado con ejecuciones en CPU. Además, es factible demostrar la importancia del uso de **pipelines GPU-only** para evitar los cuellos de botella causados por transferencias innecesarias entre memoria CPU y GPU. El desarrollo de soluciones personalizadas con redes como YOLOv11 permite a los estudiantes adaptar modelos de última generación a contextos específicos, promoviendo el pensamiento crítico, la experimentación responsable y la innovación tecnológica en entornos educativos reales.

RECOMENDACIONES:

- **Revisar previamente el entorno técnico**: Asegurarse de tener las dependencias necesarias configuradas (CUDA, OpenCV con soporte GPU, versiones correctas de Python o C++) para evitar contratiempos durante la práctica.
- **Realizar pruebas progresivas y controladas**: Comenzar con datasets pequeños y realizar pruebas incrementales para evaluar el comportamiento de la red antes de escalar a conjuntos más complejos.
- **Fomentar la documentación del código y del proceso experimental**: Anotar resultados, cambios en los hiperparámetros y observaciones para facilitar el análisis comparativo y la elaboración de informes.
- **Aprovechar las herramientas de visualización como nvidia-smi y monitores de recursos** para entender mejor el uso de la GPU y tomar decisiones informadas sobre la arquitectura de los modelos y la distribución de la carga computacional.
- **Profundizar en métricas de evaluación** como mAP, IoU, recall y precisión para una evaluación más completa del rendimiento de los modelos entrenados.
- **Fomentar la ética académica y la originalidad** en la elección de datasets y scripts, evitando el uso de soluciones ya desarrolladas en plataformas públicas sin un análisis propio o mejora significativa.

Docente / Técnico Docente: Ing. Vladimir Robles, Bykbaev

Firma: _____