



**Tecnicatura Universitaria en Inteligencia Artificial**  
**Facultad de Ciencias Exactas, Ingeniería y Agrimensura**  
**Universidad Nacional de Rosario**

---

# Trabajo Práctico Final

---

Cátedra: **VISIÓN POR COMPUTADORA**

**2025**

**Profesores:**

- Juan Pablo Manson
- Constantino Ferrucci
- Lucas Brugé

**Alumna:**

- Rizzotto, María Camila

## Contenido

Resumen .....	3
Introducción .....	4
Etapa 1: Buscador de Imágenes por Similitud .....	5
1 – Creación de la Base de Datos Vectorial .....	5
2 – Desarrollo de la Aplicación en Gradio y Clasificación basada en Similitud .....	5
3 – Métrica de Evaluación .....	6
Etapa 2: Entrenamiento y Comparación de Modelos de Clasificación .....	7
1 – Entrenamiento de Modelos .....	7
2 – Integración y Selección en la App de Gradio .....	8
Etapa 3: Pipeline de Detección y Clasificación en Escenas Complejas .....	10
1 – Detección de Objetos .....	10
2 – Creación del Pipeline Completo .....	11
Etapa 4: Evaluación, Optimización y Herramientas de Anotación .....	12
1 – Evaluación del Pipeline .....	12
2 – Optimización del Modelo .....	13
3 – Script de Anotación Automática .....	14
Conclusiones .....	16



## **Resumen**

A lo largo de este proyecto se logró diseñar, entrenar, integrar y optimizar un sistema completo para el reconocimiento de razas de perros en imágenes del mundo real, combinando técnicas modernas de visión por computadora con herramientas interactivas de anotación y visualización.

En la Etapa 1, se abordó la creación de un buscador por similitud visual, utilizando embeddings extraídos con modelos preentrenados y herramientas como FAISS y Gradio para construir una interfaz funcional y flexible.

En la Etapa 2, se entrenó un modelo personalizado de clasificación de razas mediante transfer learning en PyTorch, alcanzando métricas de desempeño superiores al 97% y validando la capacidad del sistema con estrategias de evaluación como F1 macro y matriz de confusión.

La Etapa 3 integró detección y clasificación en un pipeline robusto, utilizando YOLOv8 como detector general y ResNet18 como clasificador específico, logrando predicciones multiobjeto en imágenes reales con resultados consistentes.

La Etapa 4 se dedicó a evaluar el pipeline sobre un conjunto de imágenes anotadas manualmente, superando dificultades iniciales en el uso de métricas convencionales mediante el diseño de evaluaciones manuales. Las métricas obtenidas reflejaron precisión, cobertura y calidad espacial en los resultados.

En la Etapa de Optimización, se aplicó cuantización dinámica sobre el modelo de clasificación, comprobando que esta técnica reduce el tiempo de inferencia sin comprometer el rendimiento del modelo.

Finalmente, se desarrolló una herramienta automática de generación de anotaciones, capaz de procesar imágenes no etiquetadas y producir archivos en formato YOLO y COCO, extendiendo la aplicabilidad del sistema a tareas de anotación masiva o armado de nuevos datasets.

Este trabajo integró diversas técnicas de visión computacional, aprendizaje profundo e ingeniería de software, logrando un pipeline sólido, modular y extensible. Las decisiones tomadas en cada etapa reflejan una búsqueda por el equilibrio entre rendimiento, simplicidad y aplicabilidad práctica.



**Tecnicatura Universitaria en Inteligencia Artificial**  
**Facultad de Ciencias Exactas, Ingeniería y Agrimensura**  
**Universidad Nacional de Rosario**

## **Introducción**

En el presente trabajo práctico se desarrollará un sistema completo de visión por computadora para la detección y clasificación de razas de perros en imágenes. A lo largo del informe, se irá construyendo un pipeline que abordará distintas etapas de complejidad creciente, combinando técnicas de búsqueda por similitud, aprendizaje profundo, detección de objetos y optimización del rendimiento.

El lector se encontrará primero con el desarrollo de un buscador por similitud, basado en modelos preentrenados y una base de datos vectorial que permitirá recuperar imágenes similares. Luego, se entrenarán modelos propios de clasificación utilizando técnicas como fine-tuning, evaluando su desempeño con métricas tradicionales. En una tercera etapa, se implementará un pipeline de detección en escenas complejas con múltiples objetos, integrando modelos como YOLO para identificar perros en imágenes reales. Finalmente, se realizará una evaluación exhaustiva del sistema, se explorarán técnicas de optimización para mejorar la velocidad de inferencia, y se desarrollará un script de anotación automática para generar datasets anotados de manera eficiente.

Este informe detallará las decisiones técnicas que se tomarán en cada instancia, permitiendo al lector seguir la evolución del sistema, comprender las elecciones realizadas y visualizar los desafíos involucrados en el desarrollo de un sistema robusto de clasificación de razas de perros mediante visión por computadora.

## **Etapas 1: Buscador de Imágenes por Similitud**

La primera instancia del trabajo consistió en la construcción de una herramienta capaz de recuperar imágenes similares a una imagen de entrada y predecir su raza mediante mecanismos de búsqueda visual. Para ello, se comenzó con la carga del dataset “70 Dog Breeds Image Dataset” de Kaggle, accediendo con usuario y clave personal. Una vez descargado, se procedió a inspeccionar su estructura y contenido.

Durante esta revisión, se detectaron inconsistencias en las etiquetas: el dataset presentaba 71 razas únicas en vez de 70, debido a una duplicación generada por un error tipográfico en el nombre “American Spaniel”, que aparecía tanto escrito correctamente como con doble espacio entre las palabras. Asimismo, se corrigió la mala escritura de la raza dálmata. Estas modificaciones permitieron homogeneizar las etiquetas y garantizar la coherencia en los pasos posteriores.

Se realizó luego una visualización del conjunto de entrenamiento, evidenciando un claro desbalance entre clases: razas como el Shih-Tzu o el Labrador contaban con muchas más imágenes que otras como el Saint Bernard o el Yorkie. Este análisis, junto con observaciones complementarias sobre los conjuntos de validación y prueba —ambos con 10 imágenes por raza y 700 registros en total—, puede consultarse en detalle accediendo al notebook de Colab a través de mi repositorio de [GitHub](#).

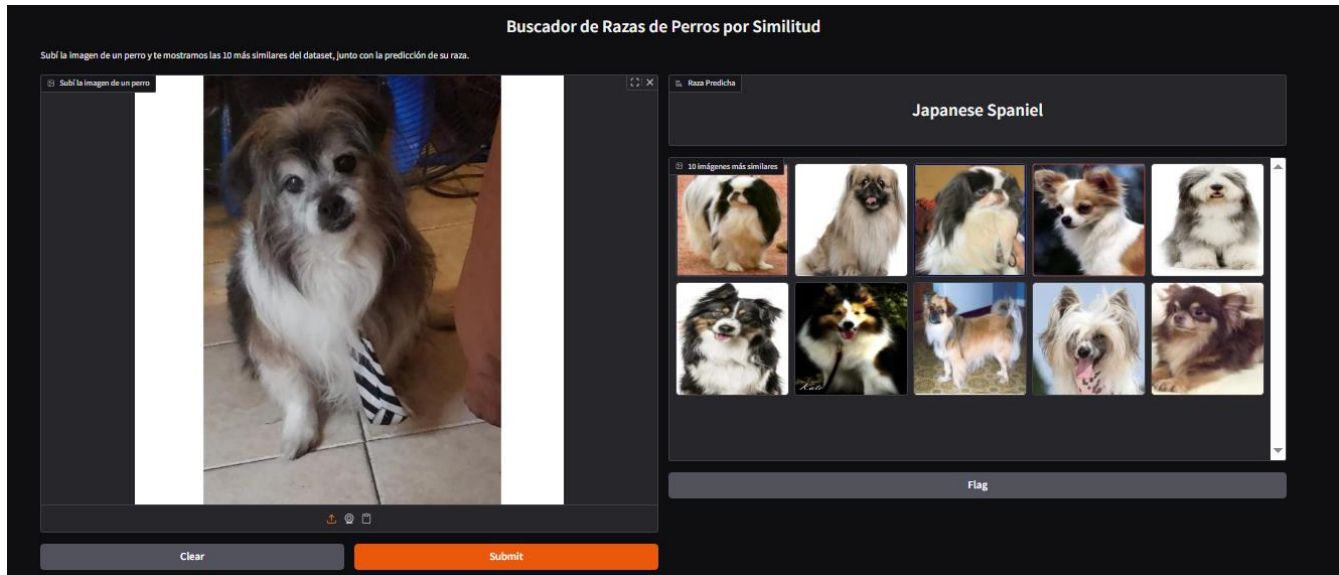
### **1 – Creación de la Base de Datos Vectorial**

Avanzando hacia la creación de la base vectorial, se empleó el modelo preentrenado ResNet50 mediante la librería TensorFlow. Se desarrolló una función personalizada ‘*get\_embedding*’, encargada de procesar cada imagen y extraer su correspondiente vector de características. El cálculo de estos embeddings fue costoso en tiempo de ejecución, pero su posterior indexación se realizó de manera eficiente utilizando la herramienta **FAISS**, seleccionada por su velocidad y solidez comprobada en contextos similares.

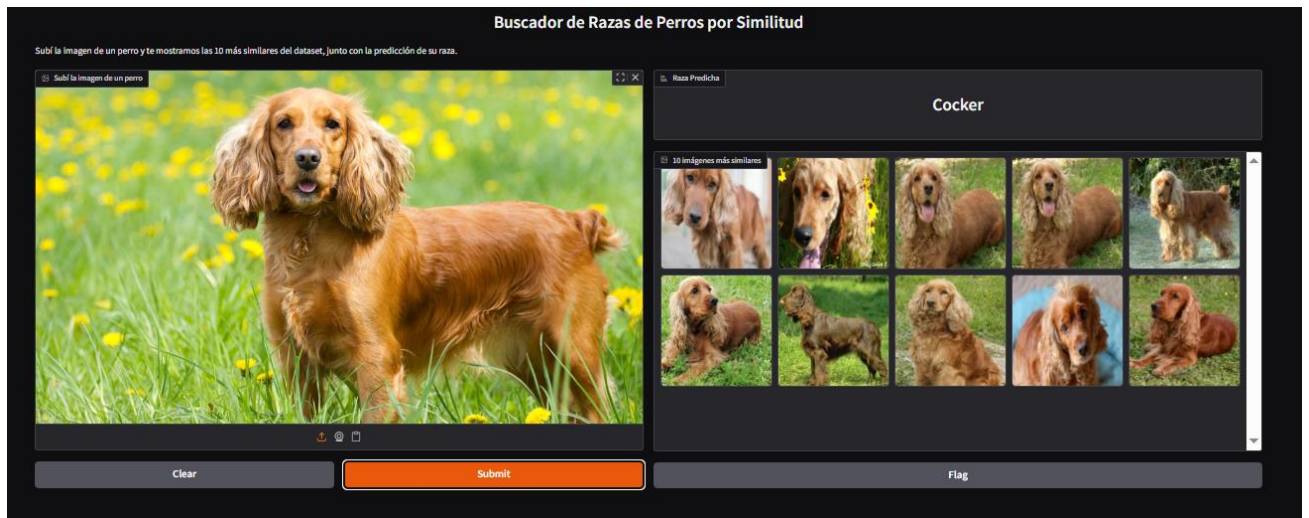
### **2 – Desarrollo de la Aplicación en Gradio y Clasificación basada en Similitud**

Con los vectores ya indexados, se procedió al desarrollo de una interfaz interactiva utilizando Gradio. Esta aplicación permite al usuario subir una imagen, obtener sus embeddings, ejecutar la búsqueda de las 10 imágenes más similares a través de FAISS, y visualizar los resultados junto a una predicción de raza calculada por voto mayoritario. La lógica de la búsqueda se concentró en una función que integró procesamiento, recuperación y visualización de resultados, logrando una respuesta ágil y efectiva.

Adjuntamos una foto del uso de la aplicación:



Dirigirse a Colab para ver comentarios al respecto.



### 3 – Métrica de Evaluación

Finalmente, se evaluó la calidad del sistema de búsqueda mediante la métrica  $NDCG@10$ , que mide la relevancia y el orden de los resultados recuperados. La implementación personalizada de esta métrica arrojó un valor promedio de 0.9505 sobre el conjunto de prueba, lo cual reflejó un desempeño sólido y satisfactorio en esta primera etapa del pipeline.

## **Etapas 2: Entrenamiento y Comparación de Modelos de Clasificación**

### **1 – Entrenamiento de Modelos**

En esta segunda etapa, se abordó el entrenamiento de modelos de clasificación para mejorar la capacidad de predicción del sistema. Se optó por utilizar técnicas de transfer learning, empleando la arquitectura ResNet18 preentrenada sobre ImageNet, adaptada para la clasificación de las 70 razas presentes en el dataset.

El proceso comenzó con la construcción de una clase personalizada DogDataset en PyTorch, diseñada para preparar los datos a partir del archivo dogs.csv, codificar las etiquetas con LabelEncoder, y aplicar transformaciones específicas para el entrenamiento y validación. La estructura del dataset se respetó según las divisiones train, valid y test, configurando los correspondientes dataloaders con un tamaño de batch de 32 y shuffling en entrenamiento.

Para el entrenamiento del modelo, se redefinió la capa final (fc) de ResNet18 para ajustar la cantidad de salidas a 70 clases. Se utilizó CrossEntropyLoss como función de pérdida y el optimizador Adam con una tasa de aprendizaje de  $1e-4$ . Como criterio de evaluación principal se eligió el puntaje F1 macro, dada la presencia de desbalance vista en el conjunto de entrenamiento, lo cual permitía tener una métrica más representativa del rendimiento general por clase.

El entrenamiento se desarrolló con una estrategia de early\_stopping, monitorizando el desempeño en validación durante un máximo de 20 épocas. Se observó una rápida mejora en las primeras iteraciones, alcanzando un F1 macro de 0.9681 en la cuarta época. A partir de allí, no se observaron mejoras significativas, y el entrenamiento se detuvo automáticamente en la época 9.

Los resultados obtenidos fueron altamente satisfactorios:

- *Accuracy*: 0.9757
- *F1 macro*: 0.9753
- *F1 micro*: 0.9757
- *Precision macro*: 0.9798
- *Recall macro*: 0.9757
- *Especificidad promedio*: 0.9996

Además, la matriz de confusión mostró una fuerte diagonal, indicando una clasificación precisa y consistente en casi todas las clases, con muy pocas confusiones inter-clase. Se documentaron observaciones complementarias en el entorno de desarrollo de Colab.

Estos resultados consolidaron al modelo fine-tuned de ResNet18 como la opción preferida para etapas posteriores del pipeline, por su rendimiento y estabilidad.

## **2 – Integración y Selección en la App de Gradio**

Una vez finalizado el entrenamiento del modelo ResNet18, se procedió a su integración dentro de la aplicación desarrollada en Gradio. Para que el nuevo modelo pudiera formar parte del sistema de búsqueda por similitud, fue necesario generar sus embeddings a partir del conjunto de imágenes del dataset, replicando la lógica utilizada previamente con el modelo ResNet50.

Se definió la función `'get_embedding_pytorch'`, encargada de cargar cada imagen, transformarla adecuadamente, y extraer su vector de características con el modelo entrenado. Estos embeddings se almacenaron y fueron indexados nuevamente utilizando FAISS, aprovechando su eficiencia y compatibilidad con la estructura anterior.

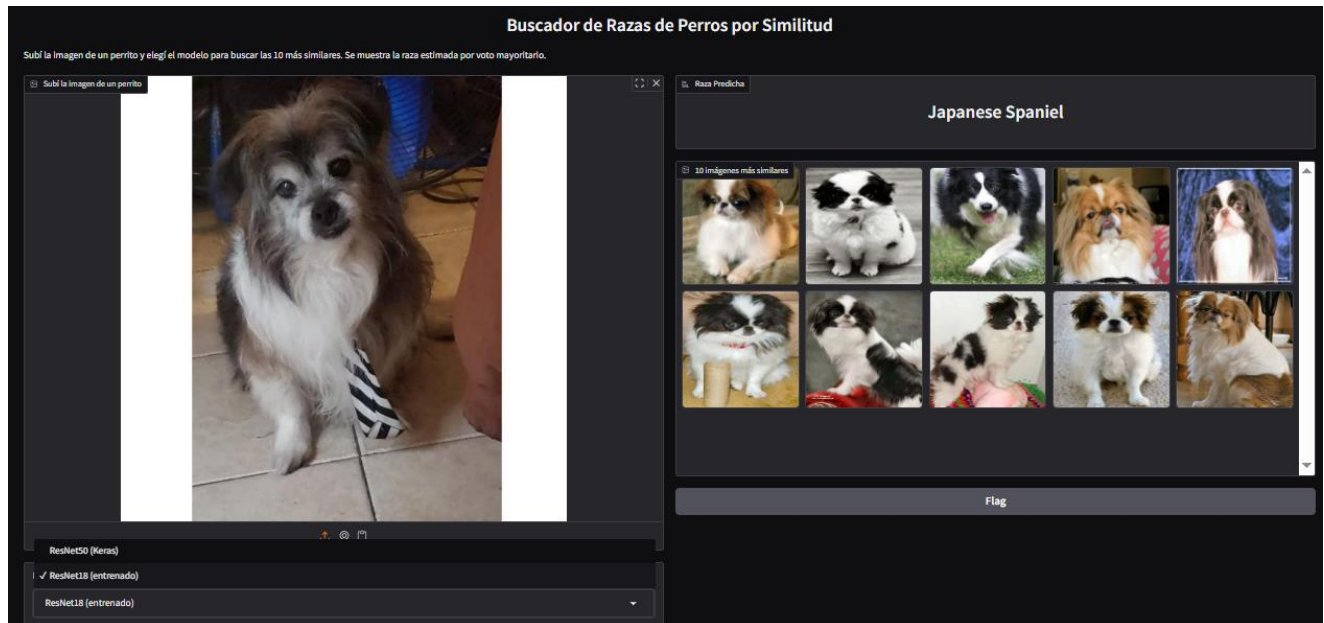
Con ambas variantes de embeddings disponibles —ResNet50 (preentrenado en Keras) y ResNet18 (fine-tuned en PyTorch)— se rediseñó la interfaz interactiva de Gradio, incorporando un selector de modelo que permitiera al usuario elegir con qué arquitectura realizar la búsqueda. La función `'extraer_embedding'` gestionó la extracción de vectores según el modelo seleccionado, y `'buscar_y_predecir'` se adaptó para utilizar el índice FAISS correspondiente.

La nueva interfaz ofreció:

- Subida de imagen de entrada.
- Selección de modelo ("ResNet50 (Keras)" o "ResNet18 (entrenado)").
- Visualización de las 10 imágenes más similares.
- Predicción de raza por voto mayoritario.

Este rediseño de la app fortaleció la flexibilidad y la interacción con el sistema, permitiendo comparar el rendimiento de distintos modelos directamente desde la interfaz.





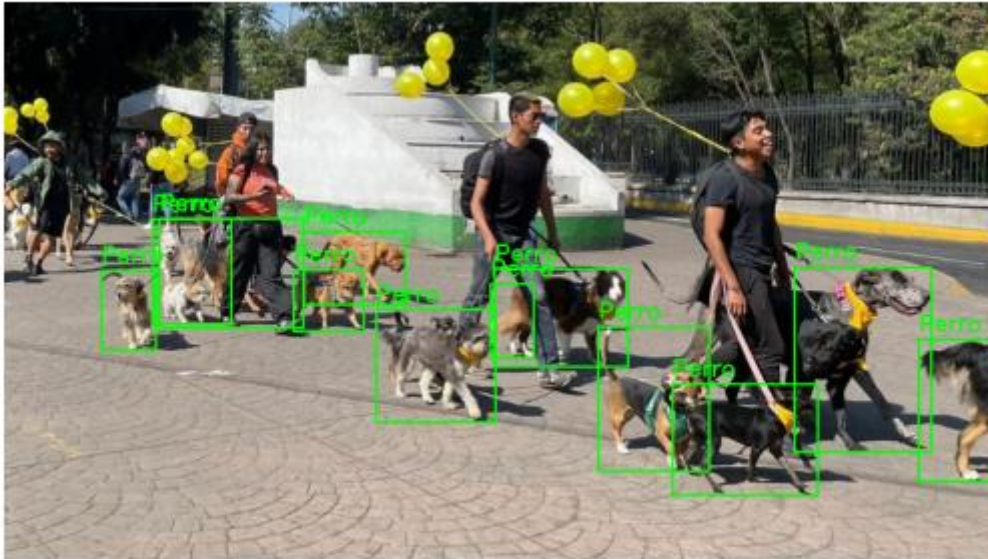
Sigue con problemas para definir una de las razas padre de este perrito mestizo, pero trajo imágenes más similares que el otro modelo.

## Etapa 3: Pipeline de Detección y Clasificación en Escenas Complejas

Con el objetivo de adaptar el sistema a imágenes del mundo real, se avanzó hacia la implementación de un pipeline capaz de detectar múltiples perros en una misma escena y clasificar su raza de forma automática. Para ello, se integró un modelo de detección y uno de clasificación en una misma solución.

### 1 – Detección de Objetos

Como detector se empleó el modelo YOLOv8m, que es más potente que la configuración mínima de YOLOv8n. Probé con una imagen compleja y tuve muy buenos resultados:



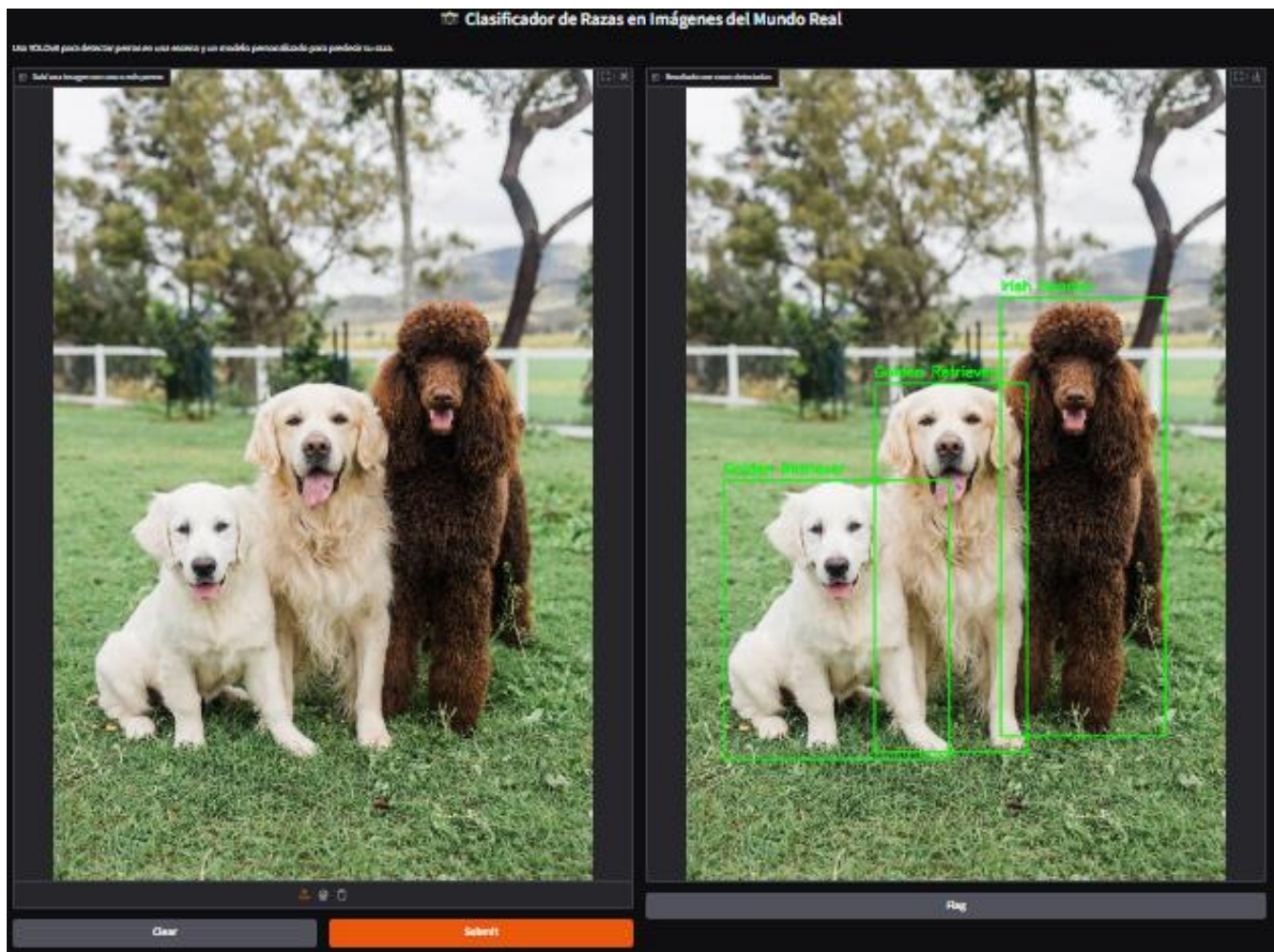
Primero había probado ésta imagen en el modelo más pequeño y noté que ya tenía dificultades para realizar la tarea: reconocía casi a la mitad de los perros nada más y con bounding boxes confusos. Ahí fue cuando decidí implementar un modelo un poco más grande y me fue mejor: detectó 12 perros junto con otras clases como personas, mochilas y pelotas deportivas, indicando mayor poder para enfrentar escenarios del mundo real sin dificultades. Además, el tiempo total de inferencia resultante (1227.7ms) fue muy razonable, y de hecho muy similar al del modelo anterior que daba peores resultados.

El modelo se cargó directamente desde Ultralytics sin necesidad de entrenamiento adicional.

## 2 – Creación del Pipeline Completo

A continuación, se desarrolló el pipeline completo. Dada una imagen de entrada, el sistema convertía la imagen a formato OpenCV, ejecutaba la detección mediante YOLO, extraía los bounding boxes para cada perro identificado y realizaba el recorte de cada región detectada. Cada crop se pasaba por el modelo ResNet18 entrenado previamente en la Etapa 2, el cual devolvía el índice de la raza correspondiente.

Para facilitar el uso interactivo del sistema, se diseñó una nueva interfaz en Gradio, que permite al usuario subir una imagen con uno o más perros y recibir como salida la misma imagen con los bounding boxes dibujados y las etiquetas de raza predicha para cada perro. Esta solución logró procesar imágenes complejas —incluso con múltiples objetos en escena— y mantener una respuesta rápida y precisa. Se muestran resultados:



Este módulo constituyó la base para las tareas de evaluación, optimización y anotación automática abordadas en la etapa siguiente.

## **Etapla 4: Evaluación, Optimización y Herramientas de Anotación**

### **1 – Evaluacion del Pipeline**

Para evaluar el rendimiento del sistema desarrollado en la Etapa 3, se confeccionó un conjunto de prueba compuesto por 10 imágenes anotadas manualmente. Navegué por las imágenes de Google hasta obtener 10 imágenes en formato .jpg que fueran lo suficientemente complejas para usarlas de prueba. Luego, las anotaciones se realizaron utilizando la herramienta makesense.ai, que es práctica y sencilla, empleando el formato de anotaciones YOLO, en el cual cada archivo .txt contiene coordenadas normalizadas de los bounding boxes junto con el índice de la clase correspondiente.

Las imágenes recopiladas se almacenaron en una carpeta llamada conjunto\_prueba\_perritos, y sus anotaciones respectivas en anotaciones\_perritos, con nomenclatura coincidente entre ambos conjuntos (prueba-1.jpg y prueba-1.txt, etc.). Las etiquetas utilizadas en makesense.ai abarcaron un subconjunto de 26 razas, seleccionadas del total de clases disponibles en el modelo de clasificación entrenado en la etapa anterior.

Inicialmente, se intentó evaluar el sistema utilizando las funciones tradicionales de sklearn.metrics, como precision\_score, recall\_score y f1\_score. Sin embargo, los resultados obtenidos fueron erróneamente nulos o constantes en cero, lo cual impedía una evaluación fiable del pipeline. Esto se debió a limitaciones de estas funciones en contextos con detecciones por región e índice, y emparejamientos malos entre predicciones y anotaciones manuales.

Ante esta situación, cambié de estrategia y procedí a implementar una evaluación manual para calcular las métricas de manera personalizada. Para cada imagen se computaron:


- TP (True Positives): predicciones correctamente emparejadas con una anotación.
- FP (False Positives): detecciones sin anotación real asociada.
- FN (False Negatives): anotaciones reales que no fueron detectadas por el sistema.
- IoU Promedio: media del coeficiente de intersección sobre unión entre cada predicción correcta y su anotación.
- Precision, Recall y F1-score: calculadas directamente en función de TP, FP y FN.
- Estimación de mAP (mean Average Precision) por imagen como métrica integrada.



A continuación, se presenta una tabla resumen que condensa todos los resultados por imagen:

	Foto	TP	FP	FN	IoU	Precision	Recall	F1-Score	mAP (imagen)
0	prueba-1.jpg	2	0	2	0.824	1.000	0.500	0.667	0.500
1	prueba-10.jpg	9	2	1	0.785	0.818	0.900	0.857	0.750
2	prueba-2.jpg	4	1	0	0.679	0.800	1.000	0.889	0.800
3	prueba-3.jpg	2	1	0	0.920	0.667	1.000	0.800	0.667
4	prueba-4.jpg	1	1	2	0.915	0.500	0.333	0.400	0.250
5	prueba-5.jpg	3	0	0	0.741	1.000	1.000	1.000	1.000
6	prueba-6.jpg	6	1	0	0.807	0.857	1.000	0.923	0.857
7	prueba-7.jpg	5	1	0	0.859	0.833	1.000	0.909	0.833
8	prueba-8.jpg	3	0	0	0.737	1.000	1.000	1.000	1.000
9	prueba-9.jpg	2	1	0	0.699	0.667	1.000	0.800	0.667

Y los siguientes promedios generales:



Promedios generales:	
TP	3.700
FP	0.800
FN	0.500
IoU	0.797
Precision	0.814
Recall	0.873
F1-Score	0.825
mAP (imagen)	0.732

Este enfoque permitió obtener métricas buenas, con valores promedio globales que evidencian un sistema preciso y bastante robusto. Para mayor detalle sobre la implementación del cálculo manual, y las observaciones sobre los casos particulares y la visualización de los resultados por imagen, se pide consultar el entorno de desarrollo.

## 2 – Optimización del Modelo

Con el objetivo de acelerar la inferencia del sistema sin comprometer su precisión, decidí utilizar la técnica de cuantización dinámica sobre el modelo ResNet18 previamente entrenado. Reduje la precisión numérica de los pesos del modelo aprovechando las herramientas nativas de PyTorch, sin requerir reentrenamiento ni modificación del pipeline.

Una vez cuantizado, se evaluó el rendimiento del modelo optimizado sobre el conjunto de validación, utilizando como métrica principal el F1 macro y midiendo el tiempo promedio de inferencia por batch.

Los resultados fueron los siguientes:

Modelo	F1 Macro	Tiempo Promedio de Inferencia
ResNet18 Original	0.9681	3.1385 segundos
ResNet18 Cuantizado	0.9696	3.1326 segundos

Se observó que el modelo cuantizado no solo mantuvo la precisión esperada, sino que elevó la métrica de comparación, quedando levemente superior al modelo original. Además, logró una reducción en el tiempo de inferencia, aunque fue mínima, pero cumpliendo así con el objetivo de esta etapa: optimizar el desempeño computacional sin degradar la calidad de clasificación.

Estos resultados validan la incorporación de cuantización como técnica eficaz para mejorar la eficiencia del pipeline propuesto.

### 3 – Script de Anotación Automática

Con el propósito de extender la aplicabilidad del sistema desarrollado hacia tareas de etiquetado automático, se implementó un script en Python que permite generar anotaciones de perros detectados y clasificados en imágenes del mundo real. Este módulo utiliza el pipeline completo, integrando el modelo YOLOv8m como detector y el modelo ResNet18 fine-tuned como clasificador de razas.

El script procesa de forma iterativa una carpeta de imágenes de entrada, ejecutando la detección de objetos y clasificando cada perro detectado. A partir de estas predicciones, genera las anotaciones en dos formatos:

1. **YOLOv5 (.txt)**: se produce un archivo de texto por imagen, con coordenadas normalizadas de bounding boxes y el índice de la clase correspondiente. Estos archivos se almacenan en la carpeta anotaciones\_yolo.
2. **COCO (.json)**: se construye un archivo único con estructura jerárquica, incluyendo el listado de imágenes procesadas, categorías disponibles y anotaciones con bounding boxes en formato absoluto. El resultado se guarda como anotaciones\_coco.json.

Ambos formatos son compatibles con plataformas de entrenamiento, conjuntos públicos de evaluación, y herramientas de visualización como FiftyOne.



**Tecnicatura Universitaria en Inteligencia Artificial**  
**Facultad de Ciencias Exactas, Ingeniería y Agrimensura**  
**Universidad Nacional de Rosario**

El script cumple así un rol fundamental como etiquetador automático, permitiendo aplicar el pipeline sobre nuevos conjuntos de imágenes sin anotación previa, y facilitando el armado de datasets personalizados para posteriores instancias de entrenamiento o refinamiento.



**Tecnicatura Universitaria en Inteligencia Artificial**  
**Facultad de Ciencias Exactas, Ingeniería y Agrimensura**  
**Universidad Nacional de Rosario**

## **Conclusiones**

En este proyecto se logró cumplir con los objetivos: el desarrollo de un sistema completo para la identificación de razas de perros en imágenes del mundo real, combinando técnicas de clasificación, detección y búsqueda por similitud, sorteando los obstáculos que se fueron presentando a lo largo del mismo. Se implementaron con éxito modelos basados en transfer learning (ResNet18), detección con YOLOv8, optimización por cuantización, evaluación de métricas, y generación automática de anotaciones en formatos estándar (YOLOv5 y COCO). La aplicación interactiva se logró integrar en Gradio, permitiendo experimentación directa con modelos seleccionables. Los resultados obtenidos demuestran buena precisión, capacidad de adaptación a escenas complejas y potencial para usos prácticos en tareas de etiquetado, búsqueda visual y clasificación automatizada. Por lo que considero que fue un trabajo desafiante y fructífero en la misma medida, habiendo adquirido y afianzado una gran cantidad de conceptos y herramientas cruciales en el marco de la materia Visión por Computadora.