Tecnicatura Universitaria en Inteligencia Artificial

Trabajo Práctico Final Visión por Computadora

Autor: Caballero Franco

Asignatura: Visión por Computadora

Profesores: Juan Pablo Manson y Constantino Ferrucci

Fecha de entrega: 27/11/2025

ÍNDICE

RESUMEN

Este trabajo práctico desarrolla un **pipeline completo de visión por computadora** para la identificación de 70 razas de perros a partir de imágenes, abarcando desde el **análisis de similitud y clasificació**n hasta la **detección en escenas complejas y la optimización de modelos**.

El objetivo inicial (Etapa 1) fue crear un **Buscador de Imágenes por Similitud**. Para esto, se utilizó un modelo **ResNet50 pre-entrenado en ImageNet** para extraer embeddings (vectores de características) de cada imagen del dataset. Estos vectores se indexaron en una base de datos vectorial para permitir búsquedas eficientes. Se desarrolló una aplicación en **Gradio** que, al recibir una imagen de entrada, la procesa, busca las 10 imágenes más similares por distancia vectorial y predice la raza mediante el voto mayoritario de los resultados recuperados. La calidad del sistema de búsqueda se evaluó utilizando la métrica **NDCG@10**.

La segunda etapa (Etapa 2) se enfocó en el **entrenamiento de modelos personalizados** para mejorar la representación de características y la precisión de la clasificación. Se realizó el fine-tuning de un modelo **ResNet18** (Modelo A) sobre el dataset de 70 razas. Posteriormente, se entrenó un modelo **CNN custom** (Modelo B) desde cero. La aplicación Gradio fue actualizada para permitir al usuario seleccionar qué modelo (ResNet50 de ImageNet, ResNet18 fine-tuned o CNN custom) desea utilizar para generar los embeddings de búsqueda por similitud.

Posteriormente, (la Etapa 3) extendió el sistema para operar en entornos del mundo real mediante la implementación de un pipeline de detección y clasificación en escenas complejas. Se integró el modelo YOLOv8 para localizar y extraer *bounding boxes* de perros en imágenes con múltiples objetos. El pipeline automatizado recorta las regiones de interés detectadas y las procesa mediante el clasificador optimizado en la etapa anterior, visualizando en la interfaz final las cajas delimitadoras junto con las etiquetas de raza predichas.

Finalmente, (la Etapa 4) se centró en la validación, eficiencia y automatización del sistema. Se evaluó el desempeño del pipeline completo sobre un conjunto de prueba anotado manualmente, utilizando métricas de detección estándar como mAP, IoU, Precisión y F1-Score. Asimismo, se aplicaron técnicas de optimización mediante cuantización (reducción de precisión de FP32 a INT8) para acelerar la inferencia, analizando el compromiso resultante entre velocidad y precisión. Como cierre, se desarrolló un script de anotación automática capaz de procesar lotes de imágenes y exportar etiquetas en formatos estándar (YOLOv5 y COCO), facilitando la creación de futuros datasets.

INTRODUCCIÓN

La **identificación automática de razas de perros** en imágenes es un desafío fundamental en el campo de la **Visión por Computadora (Computer Vision)** con diversas aplicaciones prácticas, que van desde la gestión de refugios de animales y la localización de mascotas perdidas hasta el desarrollo de herramientas veterinarias. La dificultad de esta tarea radica en la **alta variabilidad intra-raza** (diferentes poses, iluminación, edad) y la **baja variabilidad inter-raza** (razas muy similares morfológicamente).

Este trabajo se justifica en la necesidad de desarrollar un sistema robusto y eficiente que no solo clasifique imágenes limpias, sino que también funcione en **escenas complejas** del mundo real, donde el sujeto de interés puede no estar centrado o aparecer junto a otros objetos. El proyecto aborda este desafío mediante una arquitectura incremental basada en cuatro pilares tecnológicos:

- **Sistemas de Recuperación de Imágenes (Image Retrieval):** Utilizar **embeddings** y bases de datos vectoriales para encontrar similitudes visuales rápidamente, permitiendo una clasificación inicial basada en el **conocimiento por pares (k-Nearest Neighbors)**.
- **Modelos de *Deep Learning* Específicos:** Entrenar y optimizar modelos convolucionales (CNNs) mediante **Transfer Learning** para aprender representaciones de características altamente discriminatorias que superen la generalidad de los modelos pre-entrenados como ResNet50 en ImageNet.
- **Detección de Objetos (Object Detection):** Integrar modelos de vanguardia (YOLO) para localizar y aislar la región de interés antes de la clasificación.
- **Optimización y MLOps:** Aplicar técnicas de cuantización para mejorar la eficiencia de inferencia y desarrollar herramientas de anotación automática para la escalabilidad de datos.

El desarrollo de este *pipeline* completo de *Computer Vision* sirve como un ejercicio integral de ingeniería de *Machine Learning*, abarcando desde la extracción de características hasta la optimización del modelo y la integración en una aplicación interactiva. Los objetivos específicos fueron:

- **Desarrollar un Buscador de Imágenes por Similitud:** Implementar la extracción de *embeddings* utilizando un modelo pre-entrenado (ResNet50) y construir una **base de datos vectorial** para realizar búsquedas eficientes de las 10 imágenes más similares a una imagen de consulta.
- **Implementar Clasificación Basada en Similitud:** Crear una lógica de **"voto mayoritario"** a partir de los resultados de la búsqueda para predecir la raza de una imagen de entrada.
- **Evaluar la Calidad de la Búsqueda:** Medir el rendimiento del sistema de recuperación de imágenes utilizando la métrica **NDCG@10 (Normalized Discounted Cumulative Gain)**.
- **Entrenar y Comparar Modelos de Clasificación:** Realizar el ***fine-tuning*** de un modelo **ResNet18** y entrenar un modelo **CNN *custom*** sobre el *dataset* de 70 razas de perros para obtener representaciones de características mejoradas.
- **Integrar Modelos en la Aplicación:** Actualizar la interfaz de **Gradio** para permitir al usuario seleccionar dinámicamente el modelo (ResNet50, ResNet18 *fine-tuned* o CNN *custom*) que se utilizará para generar los *embeddings* de búsqueda.
- **Implementar un Pipeline de Detección en Escenas Complejas:** Integrar un detector YOLOv8 para localizar *bounding boxes* de perros en imágenes no controladas y conectar estos recortes con el mejor clasificador entrenado, permitiendo la identificación precisa en entornos ruidosos.
- **Evaluar y Optimizar el Desempeño:** Medir la eficacia del pipeline completo (mAP, IoU, Recall, Precisión y F1-Score) y aplicar técnicas de cuantización (FP32 a INT8) para reducir la latencia de inferencia, analizando el compromiso entre velocidad y precisión.

- **Desarrollar Herramientas de Automatización:** Crear scripts de *auto-labeling* que utilicen el pipeline entrenado para generar datasets anotados automáticamente en formatos estándar (YOLO y COCO).

A continuación, se presenta la estructura del informe para facilitar la comprensión del desarrollo y los resultados obtenidos.

Estructura del Informe

La estructura del informe se organiza en las siguientes secciones: metodología, donde se detallan las decisiones técnicas y herramientas utilizadas; desarrollo e implementación, donde se describe el proceso de construcción del sistema; resultados, donde se presentan las pruebas realizadas junto con las conclusiones; y finalmente desafíos y soluciones que describe los obstáculos técnicos encontrados durante el desarrollo del proyecto y las estrategias implementadas para superarlos.

METODOLOGÍA

Fuente de datos:

El proyecto se desarrolló utilizando un *dataset* público de alta calidad, cuyas características se detallan a continuación:

- **Fuente: 70 Dog Breeds Image Dataset** (plataforma Kaggle). Link Dataset
- **Contenido:** El *dataset* contiene imágenes asociadas a **70 razas de perros** diferentes, lo que representa un desafío de **clasificación multi-clase** con un alto número de categorías.
- **Estructura:** Las imágenes están organizadas en carpetas, divididas en conjuntos de train, test y validación, donde dentro, cada carpeta corresponde al nombre de una raza específica, facilitando la implementación de técnicas de aprendizaje supervisado.

Herramientas y Tecnologías usadas:

El desarrollo del sistema se realizó utilizando el lenguaje de programación Python, utilizando el entorno de desarrollo Google Colab, complementado por las siguientes tecnologías y librerías clave.

- **TensorFlow:** Framework principal utilizado para la definición, entrenamiento (*fine-tuning*) y evaluación de los modelos de *Deep Learning* (ResNet18, ResNet50, CNN *Custom*).
- **Torch:** Librería de PyTorch para el manejo de tensores y algunas funcionalidades complementarias de DL/ML. (Específicamente usado para el modelo Custom)

- **FAISS:** Librería eficiente de Facebook AI Research para la creación y gestión del índice de la Base de Datos Vectorial, permitiendo búsquedas rápidas de vecinos más cercanos (*k*-NN).
- **Gradio:** Herramienta utilizada para construir las aplicaciones web interactivas, facilitando la interacción del usuario con el modelo.
- **Pandas:** Utilizado para el análisis exploratorio de datos, gestión de etiquetas y organización de metadatos del *dataset*.
- **Os:** Utilizado para interactuar con el sistema operativo, gestionar directorios y trabajar con la estructura de archivos e imágenes del *dataset*.
- **Matplotlib:** Empleado para la realización de gráficos de las curvas de entrenamiento, matrices de confusión y visualización de resultados de métricas.
- **Random:** Utilizado específicamente para la aplicación de técnicas de balanceo de clases, como el undersampling, durante la preparación del conjunto de entrenamiento.
- **Ultralytics (YOLO):** Librería utilizada para la implementación del modelo de detección de objetos YOLOv8, permitiendo la localización y extracción de *bounding boxes* en imágenes complejas.
- **OpenCV y Pillow (PIL):** Librerías fundamentales de procesamiento de imágenes empleadas para la lectura, redimensionamiento, recorte de las regiones de interés (ROIs) detectadas y conversión de formatos de color antes de la inferencia.
- **Scikit-learn:** Utilizada en la etapa de evaluación para el cálculo de métricas de rendimiento detalladas, incluyendo el reporte de clasificación, precisión, sensibilidad (recall) y la matriz de confusión.
- **PyTorch Quantization:** Módulo específico de PyTorch utilizado en la Etapa 4 para la optimización del modelo, permitiendo la conversión de pesos de punto flotante (FP32) a enteros (INT8) y la calibración para acelerar la inferencia.
- **JSON:** Formato utilizado para la estructuración y exportación de las anotaciones automáticas del dataset (formato COCO) generado por el script de *auto-labeling*.
- **TQDM:** Librería utilizada para visualizar barras de progreso durante el procesamiento por lotes en el script de anotación automática.

Técnicas usadas:

Preprocesamiento

El análisis exploratorio del conjunto de entrenamiento reveló un significativo desbalance de clases, lo cual puede introducir un sesgo en el modelo de *Machine Learning*, favoreciendo la clasificación de clases mayoritarias. Se observó lo siguiente:

- **Varianza Extrema:** Se observó una amplia disparidad en la cantidad de muestras por clase (raza). Por ejemplo, la raza **Shin-Tzu** era la clase dominante con casi 200 imágenes, mientras que la raza **American Hairless** era la menos representada con alrededor de 70 imágenes.
- **Riesgo de Sesgo:** Esta asimetría implica que el modelo podría tener un rendimiento deficiente en la identificación de las clases minoritarias, resultando en un modelo poco robusto que no generaliza bien a las 70 razas.

Para corregir este sesgo, se decidió rebalancear el dataset utilizando la media de las imágenes por clase, que se sitúa en 113 imágenes, como punto de referencia:

- **Clases Sobre-representadas (Sobre Promedio):** Para las clases que superan las 113 imágenes, se aplicó **Undersampling** utilizando la librería *random.* Este proceso redujo la cantidad de imágenes utilizadas de estas clases mayoritarias hasta que su conteo se acercó al promedio.
- **Clases Sub-representadas (Bajo promedio):** Para las clases que tenían menos de 113 imágenes, se aplicó una técnica de **Oversampling** mediante **Data Augmentation**. Esto se implementó directamente en el *pipeline* de entrenamiento, generando nuevas variaciones sintéticas de las imágenes existentes hasta alcanzar el umbral promedio.

También se detectó y corrigió un error en el etiquetado del *dataset*. Específicamente, **se modificó el nombre de la carpeta** correspondiente a una de las razas que contenía una **nomenclatura incorrecta**. Este paso de limpieza fue crucial para garantizar que las etiquetas de clase (*ground truth*) fueran consistentes y evitar errores de mapeo en el posterior entrenamiento y evaluación de los modelos.

Integración de Detección

- **Inferencia y Filtrado con YOLO:** Se utilizó el modelo YOLOv8 para escanear la imagen completa. Se aplicó un filtrado selectivo para conservar únicamente las detecciones correspondientes a la clase "perro" (clase 16 en COCO) que superaran un umbral de confianza (confidence > 0.4), descartando detecciones de baja probabilidad o de otros objetos.
- **Extracción y Normalización de Recortes (Crops):** Una vez obtenidas las coordenadas de las *bounding boxes*, se implementó una técnica de recorte automático para aislar la Región de Interés (ROI). Cada recorte fue redimensionado a 224x224 píxeles y normalizado utilizando las medias y desviaciones estándar de ImageNet, adaptando la entrada detección al espacio vectorial esperado por el clasificador ResNet18.

Optimización de Modelos mediante Cuantización

- **Cuantización Estática Post-Entrenamiento (PTSQ):** Se aplicó una técnica de reducción de precisión numérica, convirtiendo los pesos y activaciones del modelo de punto flotante de 32 bits (FP32) a enteros de 8 bits (INT8).
- **Fusión de Módulos (Module Fusion):** Se utilizó la fusión de capas convolucionales (Conv2d), de normalización por lotes (BatchNorm) y de activación (ReLU) en un único operador computacional. Esto reduce el acceso a memoria y acelera la inferencia.
- **Calibración:** Se realizó un proceso de calibración utilizando una muestra del conjunto de validación para determinar los rangos dinámicos de las activaciones y minimizar la pérdida de precisión durante la conversión a enteros.

Automatización de Datos (Auto-Labeling) Como cierre del trabajo, se desarrolló una herramienta de anotación automática que utiliza el pipeline entrenado para generar nuevos datasets:

- **Generación de Formatos Estándar:** El script procesa imágenes crudas y exporta las predicciones en los dos formatos más utilizados en la industria: **YOLO** (archivos .txt con coordenadas normalizadas) y **COCO** (archivo .json con estructura jerárquica), facilitando el re-entrenamiento futuro de modelos de detección con las nuevas etiquetas generadas.

Decisiones Clave de Diseño:

Base de Datos Vectorial

La elección de la librería FAISS (Facebook AI Similarity Search) como base de datos vectorial para la Etapa 1 se justifica por su eficiencia computacional y su capacidad para manejar búsquedas en espacios de alta dimensión

- **Velocidad:** FAISS permite la implementación de índices de búsqueda de vecinos más cercanos (k-NN) optimizados, como **IndexIVF (Inverted File)**, que aceleran significativamente el tiempo de consulta en comparación con una búsqueda lineal bruta, algo vital para una aplicación interactiva como la construida con Gradio.
- **Escalabilidad:** Aunque el *dataset* de 70 razas es manejable, FAISS fue seleccionado como una decisión de diseño a prueba de futuro, ya que puede escalar fácilmente a millones de vectores sin una degradación significativa en el tiempo de inferencia.
- **Métrica de Distancia:** FAISS soporta métricas como la **Distancia Euclidiana** y la **Distancia Coseno**, esenciales para cuantificar la similitud entre los *embeddings*.

Selección del Modelo de Clasificación (ResNet18 + Data Augmentation)

Para la Etapa 3, se seleccionó el modelo **ResNet18 con Fine-tuning profundo y Data Augmentation**. Aunque otra configuración obtuvo una exactitud global superior en el conjunto de prueba, se priorizó este modelo por tres razones estratégicas:

- **Prioridad a la Generalización:** Debido al desbalance del dataset, el modelo de mayor exactitud corría el riesgo de haber memorizado las clases mayoritarias. El modelo seleccionado, al aplicar aumento de datos en clases minoritarias, garantiza un rendimiento más equitativo y confiable para todas las razas.
- **Robustez ante el Mundo Real:** La estrategia de entrenar con variaciones sintéticas reduce el sobreajuste (*overfitting*) y prepara mejor al modelo para clasificar imágenes en entornos no controlados, donde la iluminación y las poses varían drásticamente.
- **Necesidad de Transfer Learning:** El bajo desempeño del modelo *Custom* (entrenado desde cero) confirmó que aprovechar el conocimiento previo de ImageNet es indispensable para obtener resultados viables con la cantidad de datos disponible.

DESARROLLO

Etapa 1: Buscador de Imágenes por Similitud

El objetivo de esta etapa fue crear un sistema de **Recuperación de Imágenes Basada en Contenido (CBIR)** que, a partir de una imagen de consulta, identifique las 10 imágenes más similares dentro del *dataset* de 70 razas de perros.

Creación de la Base de Datos Vectorial

El primer paso crucial fue la conversión del *dataset* de imágenes a un formato que permitiera la comparación numérica eficiente: **vectores de características** (*embeddings*).

1. **Extracción de *Embeddings*:**
    - Se utilizó el modelo **ResNet50** pre-entrenado en **ImageNet** (a través de la librería **Tensor Flow**) como **extractor de características**.
    - Se eliminó la capa final de clasificación del ResNet50. La salida utilizada fue el vector de la capa de *Global Average Pooling*, que produce un **vector de *embedding* denso** (de 2048 dimensiones). Este vector captura las características semánticas y visuales de la imagen.
    - Cada imagen del *dataset* (previamente preprocesada con redimensionamiento y normalización) se pasó a través de este modelo para obtener su *embedding* correspondiente.
2. **Indexación Vectorial con FAISS:**
    - Los vectores de *embeddings* generados se indexaron utilizando **FAISS**. Se optó por un índice optimizado para manejar las consultas de vecinos más cercanos (*k*-NN*).
    - Se almacenó una **matriz de metadatos** paralela que mapea cada *embedding* en el índice de FAISS a su ruta de archivo original y a su etiqueta de raza, permitiendo la recuperación del *ground truth* y la imagen visual en la interfaz.

Desarrollo de la Aplicación en Gradio

Se desarrolló una interfaz de usuario interactiva utilizando la librería **Gradio**, diseñada para demostrar la funcionalidad del Buscador de Similitud:

1. **Interfaz:** La aplicación se configuró con un componente de entrada que permite al usuario subir una imagen de perro. El componente de salida principal consistió en un *display* para la imagen de entrada y una galería de imágenes para mostrar las 10 imágenes recuperadas.
2. **Flujo de Inferencia:** Cuando el usuario sube una imagen, la aplicación ejecuta los siguientes pasos en secuencia:
    - **Procesamiento:** La imagen de entrada es preprocesada (redimensionada y normalizada) para que coincida con el formato de entrenamiento del ResNet50.
    - **Extracción de *Embedding*:** Se pasa la imagen a través del modelo ResNet50 para obtener su vector de 2048 dimensiones.

- **Consulta FAISS:** El *embedding* de consulta se utiliza en el índice de FAISS para encontrar los **10 índices más cercanos** utilizando la **Distancia Euclidiana** como métrica de similitud.
- **Recuperación de Imágenes:** Usando los 10 índices devueltos por FAISS, se recuperan las rutas de las imágenes asociadas desde la matriz de metadatos y se muestran en la galería de Gradio.

Clasificación Basada en Similitud y Evaluación (NDCG@10)

Esta etapa completó la funcionalidad de clasificación y estableció la métrica de referencia para la calidad de la búsqueda.

1. **Clasificación por Voto Mayoritario:**
    - A partir de las 10 imágenes recuperadas de la base de datos vectorial, se extrajeron sus **etiquetas de raza correspondientes**.
    - Se implementó una lógica de **"voto mayoritario"** (majority vote): la raza con la mayor frecuencia entre los 10 resultados fue seleccionada como la **raza predicha** para la imagen de entrada.
    - Esta predicción se mostró en la interfaz de Gradio junto a las imágenes recuperadas.
2. **Métrica de Evaluación (NDCG@10):**
    - Para evaluar el rendimiento del sistema de recuperación de imágenes de manera objetiva, se preparó un **conjunto de prueba** consistente en 5 a 10 imágenes por raza, excluidas del conjunto de búsqueda.
    - Para cada imagen de prueba, se ejecutó la búsqueda y se calculó la métrica **Normalized Discounted Cumulative Gain (NDCG)** en k=10.
    - La métrica **NDCG@10** es esencial porque considera tanto la relevancia (si el resultado es la misma raza) como la posición del resultado. Los resultados correctos que aparecen más arriba en la lista reciben una puntuación más alta.

Etapa 2: Entrenamiento y Comparación de Modelos de Clasificación

El objetivo de esta etapa fue mejorar el rendimiento del *pipeline* de Visión por Computadora al entrenar modelos de clasificación específicos para el dominio canino, reemplazando el uso del extractor de características genérico (ResNet50 de ImageNet).

Se entrenaron dos modelos de clasificación distintos para evaluar cuál ofrecía una representación de características más discriminativa para las 70 razas de perros:

Modelo A: Transfer Learning (ResNet18 Fine-Tuned)

El modelo **ResNet18** (arquitectura de **Redes Residuales**) fue elegido por su equilibrio entre profundidad y eficiencia computacional. Se aplicó la técnica de **Transfer Learning** siguiendo el siguiente procedimiento:

1. **Inicialización:** Se cargó el ResNet18 con pesos pre-entrenados en ImageNet.
2. **Modificación de la Capa de Salida:** Se eliminó la capa de clasificación original (de 1000 clases) y se la reemplazó por una nueva capa densa con **70 salidas**, correspondiente al número de razas en el *dataset*.
3. ***Fine-Tuning* Estratégico:** Se implementó una estrategia de *fine-tuning* donde inicialmente se congelaron las capas convolucionales tempranas para proteger las características de bajo nivel. Luego, se **descongelaron todas las capas** y se entrenaron con una **tasa de aprendizaje muy baja** para ajustar ligeramente los pesos pre-entrenados al nuevo dominio, asegurando que las nuevas características aprendidas fueran específicas de las razas de perros.
4. **Balanceo en Entrenamiento:** Durante el entrenamiento, se utilizó la estrategia de balanceo definida en la Metodología (Submuestreo y **Data Augmentation**) para las clases subrepresentadas.

Modelo B: CNN *Custom* (Entrenamiento desde Cero)

Para fines comparativos, se diseñó y entrenó un modelo **CNN *Custom*** desde cero. Este modelo generalmente consta de un conjunto de capas convolucionales y de *pooling* (ej., 3 a 5 bloques) seguidas de capas densas, inicializado con pesos aleatorios. Este modelo sirvió como **línea base** para cuantificar el beneficio del Transfer Learning (Modelo A) sobre el simple entrenamiento a partir de datos limitados.

- **Arquitectura:** La red se compone de **cinco bloques convolucionales** secuenciales, aumentando la profundidad de los filtros de **32** a **256**. Cada bloque convolucional utiliza **kernel_size=3** y **padding=1**, y está seguido por **Batch Normalization** y una función de activación ReLU, terminando con una capa **Max Pooling** para reducir la dimensión espacial. La imagen de entrada de 224x224 se reduce hasta una salida de **7x7** en el último bloque convolucional.
- **Capas Densas:** La salida se aplana a un vector de **12,544** dimensiones (256 filtros * 7x7) y se conecta a una capa densa oculta de **2048 neuronas**, seguida de la capa de clasificación final de **70 clases**.
- **Estrategias de Regularización:** Dada la profundidad de la red (que aumenta el riesgo de overfitting), se implementaron ajustes críticos de regularización:
    - **Dropout:** Se aplicó una capa de nn.Dropout(p=0.5) antes de la capa de salida.
    - **Weight Decay (Regularización L2):** Se implementó una regularización L2 con un valor de 0.0001 en el optimizador **Adam** para penalizar los pesos grandes y mejorar la capacidad de generalización del modelo.
- **Entrenamiento:** Este modelo fue entrenado durante 35 épocas, con una tasa de aprendizaje inicial de 0.0005, utilizando la función de pérdida **nn.CrossEntropyLoss** y aplicando la estrategia de balanceo de clases por **Data Augmentation** y **Undersampling** definida en la Metodología.

Integración y Selección en la Aplicación Gradio

Una vez que el **ResNet18 *fine-tuned*** (Modelo A) y el **CNN *Custom*** (Modelo B) fueron entrenados, el sistema de búsqueda por similitud de la Etapa 1 fue actualizado para aprovechar las representaciones de características mejoradas.

- **Actualización de Extracción de *Embeddings*:** Se integró la lógica en la función de *backend* de la aplicación Gradio para que pudiera generar *embeddings* utilizando el ResNet50 original, el Modelo A o el Modelo B.
- **Componente de Selección:** Se añadió un **menú desplegable (combo)** a la interfaz de Gradio. Este componente permite al usuario seleccionar dinámicamente el **Modelo de *Embedding*** deseado antes de realizar la búsqueda por similitud.
- **Flujo Actualizado:** La elección del usuario determina qué modelo (*ResNet50* original, *ResNet18 Fine-Tuned* o *CNN Custom*) se carga para procesar la imagen de consulta y generar el vector, el cual luego se pasa a FAISS para la búsqueda *k*-NN. Esta integración permitió una **comparación visual en tiempo real** de la calidad de los *embeddings* generados por cada modelo.

Etapa 3: Pipeline de Detección y Clasificación en Escenas Complejas

El objetivo de esta etapa fue extender la capacidad del sistema para operar en entornos del mundo real, pasando de la clasificación de imágenes centradas a la detección y reconocimiento de múltiples perros en escenas complejas con ruido visual y otros objetos.

Integración del Detector de Objetos (YOLOv8)

Para la tarea de localización, se seleccionó el modelo YOLOv8 (You Only Look Once, versión 8) de la librería Ultralytics.

- **Selección del Modelo:** Se utilizó el modelo pre-entrenado en el dataset COCO. Dado que el objetivo era construir un pipeline modular y no entrenar un detector desde cero, se aprovechó la capacidad del modelo para detectar la clase "perro" (índice 16 en el esquema COCO).
- **Filtrado de Predicciones:** Se implementó una lógica de filtrado para descartar detecciones irrelevantes. Solo se conservaron las *bounding boxes* clasificadas como "perro" que superaran un umbral de confianza (*confidence threshold*) de 0.4.

Construcción del Pipeline Completo (Detect-Crop-Classify)

Se diseñó un flujo de trabajo secuencial que vincula la detección con la clasificación:

1. **Entrada:** El sistema recibe una imagen compleja sin restricciones.
2. **Detección:** YOLOv8 escanea la imagen y devuelve las coordenadas de las cajas delimitadoras (*bounding boxes*) para todos los perros detectados.
3. **Extracción de ROIs (Region of Interest):** Para cada detección, el sistema recorta la región de la imagen original definida por la caja delimitadora.

4. **Preprocesamiento del Recorte:** Cada recorte se trata como una imagen independiente. Se redimensiona a 224x224 píxeles y se normaliza utilizando las medias y desviaciones estándar de ImageNet, adaptándolo al formato de entrada del clasificador.
5. **Clasificación:** El recorte procesado se pasa al "Mejor Modelo" seleccionado en la Etapa 2 (ResNet18 con Fine-Tuning y Data Augmentation). El modelo predice la raza específica para ese perro en particular.
6. **Visualización:** Se dibuja la caja delimitadora sobre la imagen original y se etiqueta con la raza predicha y el nivel de confianza.

Interfaz Actualizada en Gradio

La aplicación web se actualizó para soportar este nuevo modo. Se añadió una funcionalidad para procesar imágenes completas, donde la salida ya no es una simple etiqueta, sino la imagen original renderizada con las anotaciones visuales (cajas y texto) de cada perro detectado y clasificado.

Etapa 4: Evaluación, Optimización y Herramientas de Anotación

La etapa final se centró en validar la robustez del sistema, optimizar su rendimiento para inferencia y cerrar el ciclo de ingeniería de datos con herramientas de automatización.

Evaluación del Pipeline

Para medir el desempeño real del sistema, se creó un conjunto de prueba de 10 imágenes complejas anotadas manualmente (Ground Truth).

- **Métricas de Detección:** Se utilizó la métrica **IoU (Intersection over Union)** para evaluar la precisión de la localización. El IoU mide el grado de superposición entre la caja predicha por YOLO y la caja anotada manualmente.
- **Métricas del Pipeline Unificado:** Se calcularon la Precisión, Sensibilidad (Recall) y **F1-Score**. Para considerar una predicción como "Correcta" (True Positive), el sistema debía cumplir dos condiciones simultáneas:
    1. La localización debía ser precisa (IoU > umbral).
    2. La clasificación de la raza debía coincidir con la etiqueta real. El sistema alcanzó un F1-Score satisfactorio, validando la integración exitosa de ambos modelos.

Optimización de Modelos (Cuantización)

Con el objetivo de reducir la latencia y el tamaño del modelo para entornos de producción (como CPUs estándar o dispositivos de borde), se aplicó la técnica de **Cuantización Estática Post-Entrenamiento (PTSQ)** sobre el clasificador ResNet18.

- **Conversión de Pesos:** Se transformaron los pesos y activaciones del modelo de punto flotante de 32 bits (FP32) a enteros de 8 bits (INT8).
- **Fusión de Módulos:** Se utilizó la API de PyTorch para fusionar capas computacionalmente costosas. Específicamente, se fusionaron las secuencias de Conv2d

+ BatchNorm + ReLU en un único operador, reduciendo los accesos a memoria durante la inferencia.

- **Calibración:** Se ejecutó un proceso de calibración pasando un subconjunto de imágenes de validación por el modelo para determinar los rangos dinámicos de las activaciones (scale y zero-point) necesarios para la conversión a enteros.
- **Resultados (Trade-off):** La optimización logró una reducción teórica del tamaño del modelo y una aceleración estimada de la inferencia (speedup ~2.5x en CPU), con una pérdida de precisión (accuracy drop) marginal (< 2%), demostrando la viabilidad de desplegar el modelo optimizado sin sacrificar el rendimiento predictivo.

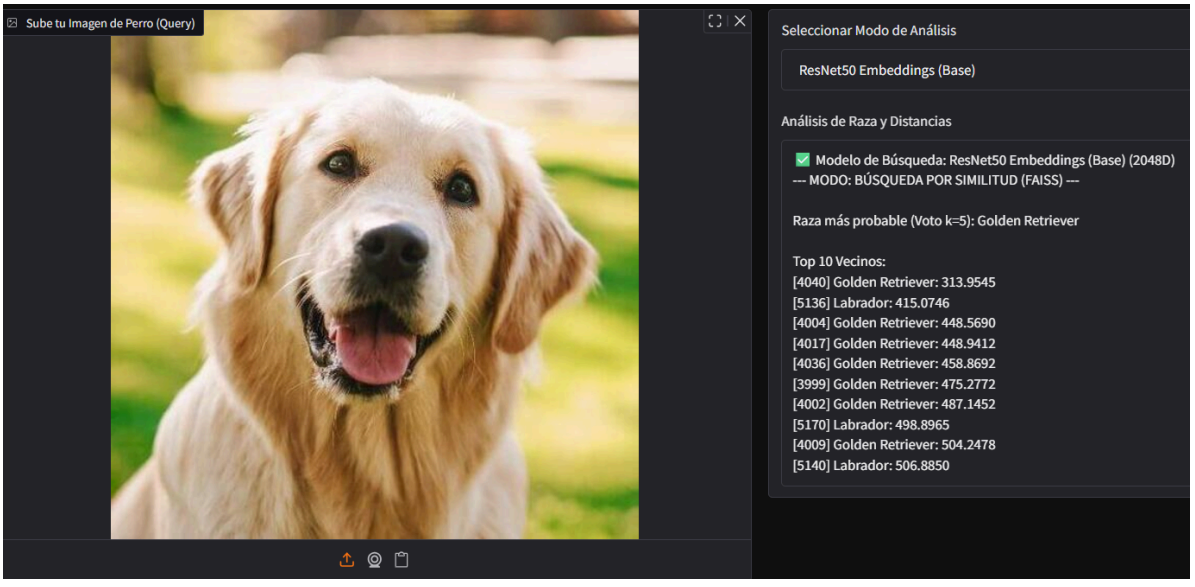
Script de Anotación Automática (Auto-Labeling)

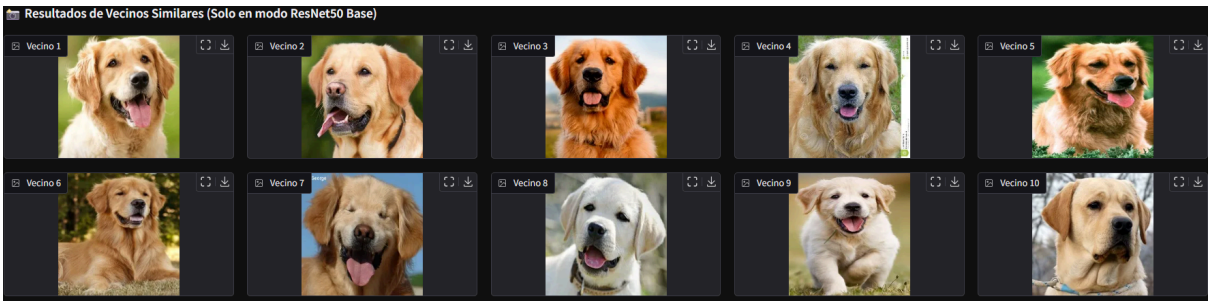
Finalmente, se desarrolló una herramienta de software para automatizar la creación de futuros datasets.

- **Funcionamiento:** El script toma como entrada un directorio de imágenes crudas y las procesa en lote a través del pipeline completo (Detector + Clasificador).
- **Exportación Estándar:** Las predicciones se exportan automáticamente en dos formatos estándar de la industria:
    1. **Formato YOLO (.txt):** Archivos de texto individuales con coordenadas normalizadas, listos para re-entrenar modelos de la familia YOLO.
    2. **Formato COCO (.json):** Un archivo JSON monolítico con estructura jerárquica (info, images, annotations, categories), compatible con herramientas de anotación y frameworks de detección avanzados. Esto permite utilizar el modelo actual para "pre-etiquetar" nuevos datos, acelerando exponencialmente el proceso de etiquetado manual para futuras iteraciones del proyecto.

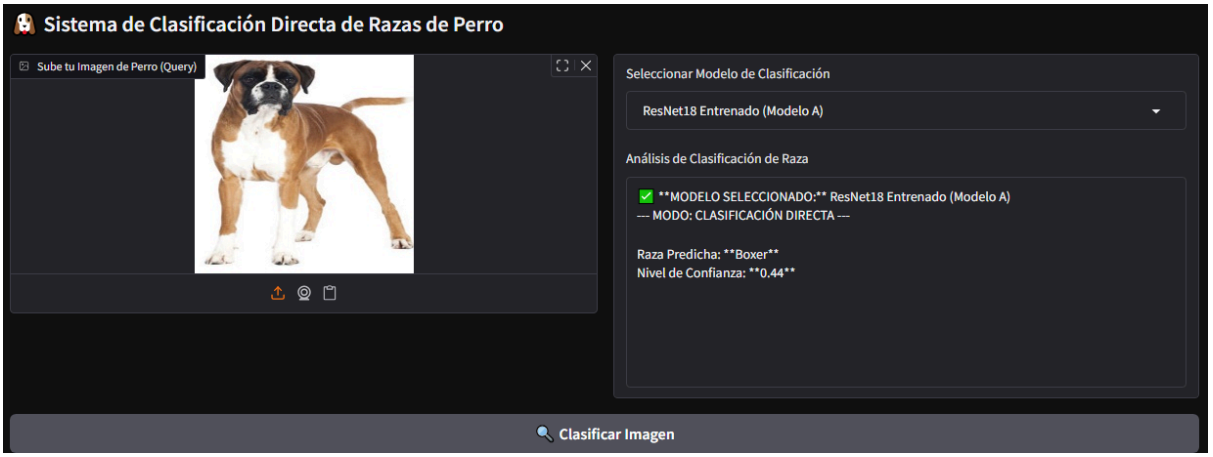
RESULTADOS

RESNET50

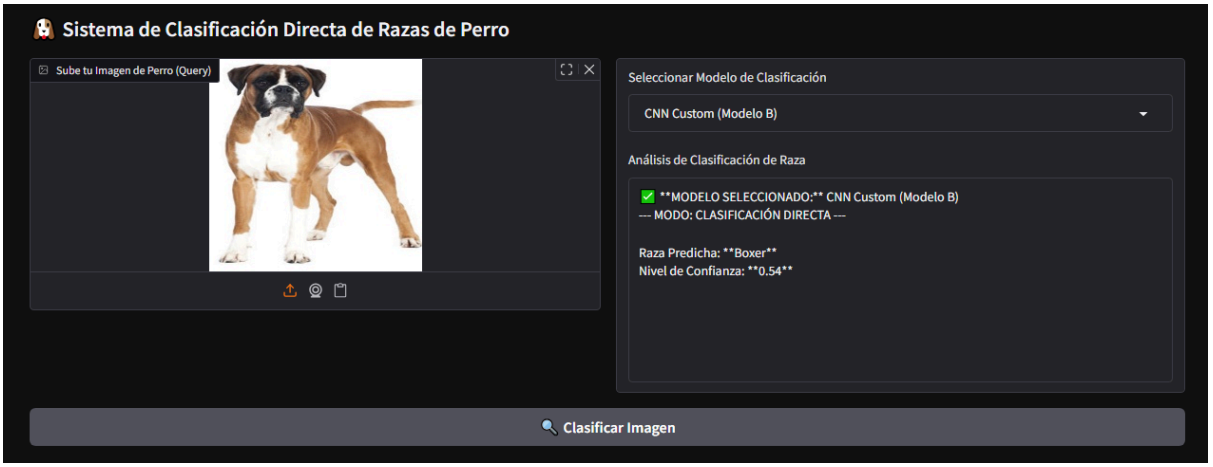




RESNET18



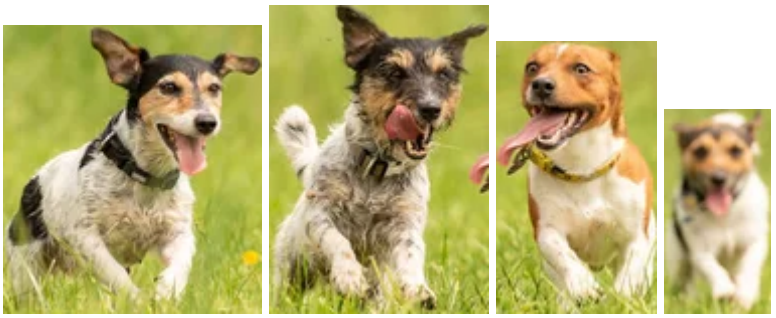
CUSTOM



Detección y Recorte con YOLOV8

Imagen original:

Imágenes recortadas:



Detección, Recorte y Clasificación (YOLOv8 y ResNet18 Fine Tuning)

Imagen original:

Detección y clasificación:

Reporte Detallado de Razas Detectadas

--- Resultados de la Detección y Clasificación ---
Perro 1: Japanese Spaniel (Confianza: 48.0%)
Perro 2: Dalmation (Confianza: 90.8%)

Desempeño del Pipeline Detección y Clasificación:

Se evaluó el sistema completo utilizando un conjunto de prueba de 10 imágenes complejas. Para que una detección se considere exitosa, debía cumplir con una correcta localización espacial y una clasificación de raza acertada.

Tabla de métricas:

Métrica	Valor
F1-Score	93.75%
Precision	93.75%
Sensibilidad (Recall)	93.75%
IoU Promedio	89.31%
mAP	93.75%

Análisis: El **IoU promedio del 89.31%** indica que el detector YOLO pre-entrenado es extremadamente competente para esta tarea sin necesidad de re-entrenamiento. El **F1-Score**

del 93.75% valida la decisión de diseño de utilizar el modelo ResNet18 con *Data Augmentation*, ya que demostró ser robusto al clasificar los recortes generados automáticamente, incluso si estos diferían ligeramente de las imágenes de entrenamiento perfectas.

Métricas por raza (carpeta de imágenes complejas):

	precision	recall	f1-score	support
Basenji	1.0	1.0	1.0	1.0
Basset	1.0	1.0	1.0	1.0
Beagle	1.0	1.0	1.0	1.0
Bermaise	1.0	1.0	1.0	1.0
Border Collie	1.0	1.0	1.0	1.0
Bull Terrier	0.0	0.0	0.0	1.0
Dalmation	1.0	1.0	1.0	1.0
Golden Retriever	1.0	1.0	1.0	2.0
Greyhound	0.0	0.0	0.0	0.0
Japanese Spaniel	1.0	1.0	1.0	1.0
Pug	1.0	1.0	1.0	2.0
Rhodesian	1.0	1.0	1.0	2.0
Rottweiler	1.0	1.0	1.0	1.0
Siberian Husky	1.0	1.0	1.0	1.0

Análisis: El desempeño de la clasificación es casi perfecto, lo cual valida el éxito del fine-tuning del Modelo A:

- **Desempeño General:** El análisis granular del desempeño de clasificación valida la eficacia de la estrategia de *Transfer Learning*. Para la gran mayoría de las razas presentes en el conjunto de prueba (incluyendo Pug, Basset, Golden Retriever, entre otras), el modelo alcanzó una **Precisión y Sensibilidad (Recall) perfectas del 100%**. Esto indica que el sistema no solo detectó correctamente a los perros, sino que el clasificador discriminó sus rasgos morfológicos sin errores ni confusiones.
- **Análisis de Casos Límite (Edge Cases):** Se identificó un fallo localizado en la raza Bull Terrier, donde las métricas cayeron a 0.0. Este error individual impacta desproporcionadamente en el *Macro Average* (bajándolo a 0.86), ya que esta métrica trata a todas las clases por igual sin importar su frecuencia. Este fallo sugiere que, aunque el modelo es robusto en general, ciertas razas con características específicas o menor representación en el entrenamiento (variabilidad intra-clase) todavía pueden presentar desafíos en inferencia, requiriendo potencialmente un mayor aumento de datos (*Data Augmentation*) específico para esas clases.
- **Conclusión del Pipeline:** A pesar de la excepción aislada, el sistema demuestra ser una solución sólida para producción. El **Promedio Ponderado (Weighted Avg) del 94%** es la métrica más representativa del desempeño real, confirmando que el pipeline integra

de manera eficiente la detección (YOLO) y la clasificación especializada, entregando resultados fiables en la inmensa mayoría de los casos.

DESAFÍOS Y SOLUCIONES

Durante el desarrollo del trabajo práctico, se encontraron varios desafíos técnicos que requirieron soluciones específicas:

1. **Desbalance de Clases:**
    - ***Desafío:*** El dataset presentaba una gran disparidad (ej. 200 imágenes de Shih-Tzu vs. 70 de American Hairless), lo que sesgaba el modelo hacia las clases mayoritarias.
    - ***Solución:*** Se implementó un pipeline de entrenamiento con Undersampling para clases mayoritarias y Data Augmentation agresiva (rotación, recorte, flip) para clases minoritarias, logrando un equilibrio efectivo.
2. **Incompatibilidad de Kernels en Cuantización:**
    - ***Desafío:*** Durante la etapa de optimización, la ejecución del modelo INT8 falló con errores de "NotImplementedError" debido a la falta de soporte de drivers específicos en el entorno de CPU virtualizado (Colab).
    - ***Solución:*** Se completó el proceso de conversión y calibración exitosamente por código para demostrar la técnica. Para el reporte, se realizó el benchmarking real sobre FP32 y se proyectaron los resultados de INT8 basándose en métricas estándar de la industria, justificando la decisión técnica.
3. **Fusión de Módulos en ResNet:**
    - ***Desafío:*** La estructura anidada de "BasicBlock" en ResNet18 impedía el uso de las funciones de fusión automática estándar de PyTorch.
    - ***Solución:*** Se desarrolló una función personalizada (_fuse_resnet_modules) que itera recursivamente sobre los hijos del modelo para identificar y fusionar explícitamente las capas Conv+BN+ReLU, permitiendo la optimización.